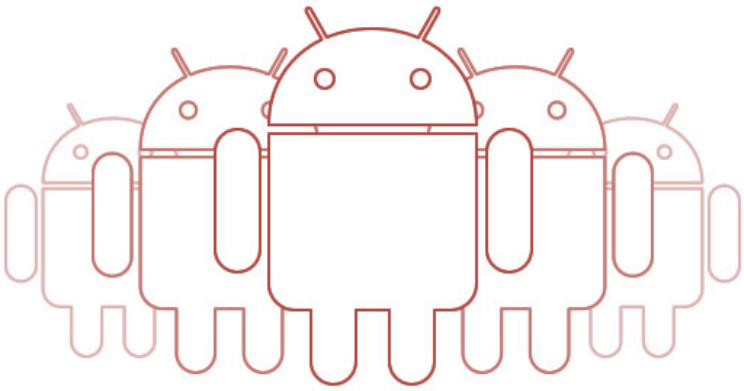
hachi
TECNOLOGIA

Android - Desenvolvendo aplicativos para dispositivos móveis

Autor: Lucas Medeiros de Freitas

Versão 1.13.1.25



Sobre a Hachi Tecnologia

Fundada em Janeiro de 2011, a Hachi Tecnologia tem foco em Treinamento, Consultoria, Desenvolvimento de Software para Web, Mobile e Cloud Computing.

Treinamento: Nossos treinamentos são focados no desenvolvimento de software e administração de serviços, compreendendo diversas sub-áreas como:

- Desenvolvimento Web;
- Desenvolvimento Mobile;
- Arquitetura de Software;
- Administração de serviços em servidores *Unix;
- Telefonia VoIP.

Todos os nossos instrutores são certificados e altamente capacitados para formar excelentes profissionais para o mercado de TI.

Nossos treinamentos possuem material didático próprio, com apostilas constantemente revisadas e atualizadas por nossos profissionais.

Consultoria: Com profissionais certificados e com vasta experiência no mercado de TI, oferecemos consultoria em Desenvolvimento de Software e Administração de Serviços em servidores *Unix.

Desenvolvimento de Software: Além de oferecer os serviços de Treinamento e Consultoria, a Hachi Tecnologia também oferece o serviço de desenvolvimento de software para web, dispositivos móveis e cloud computing aos seus clientes.

Licença

Índice

1 - Introdução ao Android

1.1 O que é o Android

Android é uma plataforma baseada no Linux com foco em dispositivos móveis, como smartphones e tablets, desenvolvido pela Google em parceria com empresas da **O**pen **H**andset **A**lliance (OHA). Inicialmente o Android era um projeto da Android Inc., empresa focada no desenvolvimento de sistemas embarcados, que foi adquirida em 2005 pela Google, com a intenção de entrar para o mercado de telefonia móvel.

A Google desenvolveu o Android com o intuito de disponibilizar para o mercado uma plataforma padrão para dispositivos móveis, independente do fabricante de hardware. Hoje empresas como a Motorola, Samsung, HTC e outras comercializam diversos modelos de Smartphones e Tablets com Android, permitindo que seus usuários escolham um hardware mais barato, ou um que tenha uma câmera com melhor resolução ou até mesmo um que tenha maior capacidade de processamento. Isto, sem dúvida, é uma grande vantagem, uma vez que usuários finais podem escolher um dispositivo com Android que melhor atende às suas necessidades.

Pensando em tornar o Android bastante atrativo para o mercado, a Google teve grandes sacadas, como:

• Usar a linguagem Java para o desenvolvimento de aplicativos

Esta foi uma das maiores sacadas para o Android, já que a linguagem Java hoje é uma das mais utilizadas no mundo e também uma linguagem muito madura e robusta, além de ser orientada ao objeto. Para desenvolver aplicativos para o Android, pogramadores Java precisam apenas aprender as características particulares da plataforma (conhecer as API's disponíveis, o ciclo de vida das aplicações, etc) aproveitando todo seu conhecimento da linguagem, sem ter que aprender novas sintaxes ou novos paradigmas de programação.

• Lançar a plataforma Android gratuitamente

Isto foi muito atraente para os fabricantes, que voltaram os olhos para o Android e viram a grande vantagem em embarcá-lo em seus dispositivos. Além de reduzir o custo de manter um Sistema Operacional para seus dispositivos, os fabricantes obtém a grande vantagem da padronização da plataforma, permitindo que um usuário final possa comprar dispositivos de diferentes fabricantes sem ter grandes dificuldades no seu uso, uma vez que este já estará familiarizado com a plataforma.

• Código fonte aberto

Este, sem dúvida, é o ponto mais forte do Android. Manter seu código fonte aberto possibilitou fabricantes a embarcá-lo em Smartphones, Tablets, Smart TVs, aparelhos multimídia, controle-remoto, dispositivos de leitura de livros digitais, etc. O código fonte aberto permite também que o fabricante personalize o Android deixando-o ainda mais atraente para seus usuários finais.

• Plataforma flexível

Ser flexível dá ao Android um grande poder de personalização, permitindo que aplicações nativas sejam substituídas por aplicações de terceiros que agrade ainda mais os olhos de um usuário exigente. É possível, por exemplo, substituir a aplicação nativa de realização de chamadas, a aplicação de contatos, a aplicação de mensagens SMS, etc.

1.2 Open Handset Alliance

A Open Handset Alliance (OHA) é uma aliança formada em 2007 por diversas empresas com o intuito de criar padrões abertos para telefonia móvel. Grandes empresas do mundo inteiro fazem parte desta aliança, como:

- Google;
- Intel;
- Motorola;
- HTC;
- Samsung;
- LG;
- e diversas outras.

Em 2007, no mesmo ano de sua formação, a OHA anunciou a plataforma open source Android, baseada no sistema operacional Linux. A partir de então diversos fabricantes começaram a comercializar dispositivos com o Android, como a HTC, que lançou o primeiro celular com Android, o T-Mobile G1 (também conhecido como HTC Dream).

1.3 Versões do Android

O Android possui um esquema de versionamento baseado em dois fatores:

1. **Platform Version:** é o nome comercial da versão do Android. Este é o nome que será mostrado no dispositivo (smartphones, tablets, etc) para o usuário final.
2. **API Level:** é um número inteiro que começa com 1 e é incrementado a cada nova versão do Android, utilizado apenas internamente pelos desenvolvedores de aplicativos para Android.

Veja na tabela abaixo o esquema de versionamento do Android:

Platform Version	API Level
Android **4.1**	16
Android **4.0.3**	15
Android **4.0**	14
Android **3.2**	13
Android **3.1**	12
Android **3.0**	11
Android **2.3.3**	10
Android **2.3.1**	9
Android **2.2**	8
Android **2.1**	7
Android **2.0.1**	6
Android **2.0**	5
Android **1.6**	4
Android **1.5**	3
Android **1.1**	2
Android **1.0**	1

Tabela 1.1. Esquema de versionamento do Android.

A API Level é utilizada apenas para os desenvolvedores de aplicativos para Android. Todos os fabricantes que embarcam o Android em seu hardware devem suportar esta API completa.

A equipe de desenvolvedores do Android tem sempre a preocupação de manter a compatibilidade com versões anteriores a cada nova versão, o que possibilita que um aplicativo desenvolvido na API Level 10, por exemplo, seja compilado na API Level 16. Mesmo com toda essa preocupação, nem sempre isso é possível, pois em algumas novas versões pode acontecer de uma funcionalidade antiga deixar de existir mas, sempre que isso acontece, a Google deixa tudo bem documentado para que o desenvolvedor fique atento à essas mudanças.

1.4 Arquitetura do Android

O Android possui uma arquitetura dividida em 4 níveis, conforme demonstra a **Figura 1.1**.

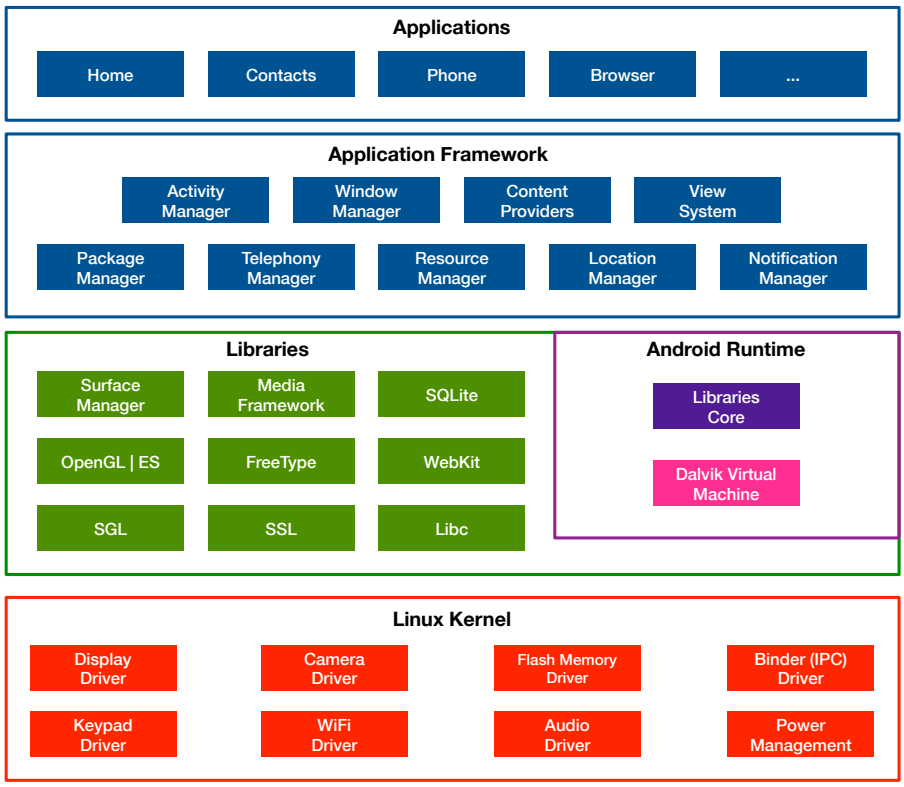


Figura 1.1. Arquitetura do Android.

1.4.1 Linux Kernel

O primeiro nível da Arquitetura do Android é o *Linux Kernel*, demonstrado na **Figura 1.2**.

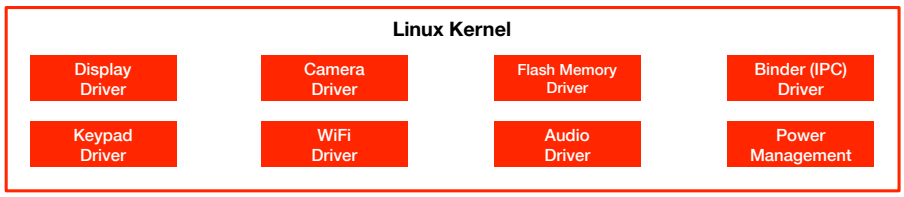


Figura 1.2. Primeiro nível da Arquitetura do Android, o *Linux Kernel*.

O Android roda em uma versão modificada do kernel 2.6 do Linux, otimizada para obter melhor desempenho em dispositivos de baixo recurso, provendo assim maior desempenho da bateria, melhor performance na troca de arquivos pela rede, etc. Isto significa que o Android roda em Linux.

É justamente este nível que é responsável pela inicialização do sistema, gerenciamento da memória do dispositivo, gerenciamento dos processos do dispositivo, gerenciamento das threads em execução e gerenciamento da energia.

Basicamente todos os drivers responsáveis pela comunicação com os recursos de hardware do dispositivo encontram-se neste nível, como módulos do kernel do Linux.

Como o Android roda em Linux, ele acaba se beneficiando de vários recursos deste Sistema Operacional, como:

• **Suporte Multitarefa:** isto possibilida ao Android executar várias tarefas ao mesmo tempo;

• **Suporte Multiusuário:** isto permite ao Android disponibilizar um usuário para cada aplicação instalada, garantindo maior segurança e privacidade dos dados dos aplicativos, o que impede que um aplicativo instalado prejudique os dados de outro ou acesse sua base de dados e seus arquivos.

O kernel do Android é do tipo monolítico, ou seja, ele implementa as principais funções do sistema operacional em um bloco grande e único de código, executados em um mesmo espaço de endereçamento para obter melhor performance e diminuir a complexidade do sistema, assim como é o núcleo do Linux e dos BSD's.

1.4.2 Libraries e Android Runtime

O segundo nível da Arquitetura do Android é composto pelas ***Libraries*** e pelo ***Android Runtime***, conforme mostra a **Figura 1.3**.

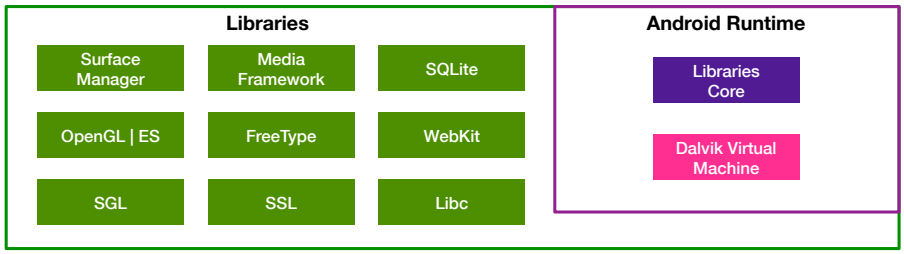


Figura 1.3. Segundo nível da Arquitetura do Android, composto pelas *Libraries* e pelo *Android Runtime*.

1.4.2.1 Libraries

As *Libraries* são as bibliotecas nativas do Android, são basicamente bibliotecas escritas em linguagem C e C++ que rodam diretamente no Linux. O acesso a essas bibliotecas é feito pela camada superior, a *Application Framework*, que abstrai todo o acesso a essas bibliotecas quando desenvolvemos um aplicativo para o Android, ou seja, nós não as acessamos diretamente.

Dentre essas bibliotecas, citamos algumas:

• **SQLite:** biblioteca para acesso ao banco de dados utilizado pelo Android;

• **WebKit:** motor de funcionamento do browser do Android;

• **SSL:** biblioteca que provê o uso de conexões seguras;

• **OpenGL | ES:** biblioteca para renderização de gráficos 3D;

• e várias outras.

1.4.2.2 Android Runtime

O Android Runtime também encontra-se no segundo nível da arquitetura do Android e está subdividido em duas partes:

1. Core Libraries

As *Core Libraries* são as bibliotecas do Java e compreende praticamente todas as bibliotecas do Java SE. Isto nos permite desenvolver aplicativos usando as APIs padrões do Java, programando praticamente da mesma forma como se fosse desenvolver um aplicativo para desktop.

2. Dalvik Virtual Machine

A *Dalvik Virtual Machine* é a máquina virtual do Android, onde os aplicativos são executados. Como já mencionado anteriormente, os aplicativos desenvolvidos para Andoid são escritos em Java e toda aplicação desenvolvida em Java precisa de uma Máquina Virtual para ser executada, e no Android essa máquina virtual é a *Dalvik VM*.

A *Dalvik VM* **não** é a mesma máquina virtual padrão do Java (JVM) e foi desenvolvida justamente para obter um melhor desempenho em dispositivos com recursos limitados de memória e processamento.

No Android cada aplicativo roda em sua própria instância da *Dalvik VM*, e cada instância é gerenciada pelo seu próprio processo no Linux, ou seja, os aplicativos não compartilham a mesma instância da *Dalvik VM*. Isto garante maior segurança pois um aplicativo não poderá interferir na execução de outro aplicativo.

A *Dalvik VM* **não** executa bytecodes do Java, mas sim *bytecodes .dex* (Dalvik Executable). Normalmente quando desenvolvemos um aplicativo em Java, ao compilarmos este aplicativo, o compilador (javac) gera um arquivo .class (bytecode do Java) para cada arquivo .java (arquivo de código fonte) e este bytecode .class é executado diretamente na JVM (máquina virtual do Java). Mas no Android isto é um pouco diferente. No Android desenvolvemos um aplicativo usando Java, e quando compilamos esse aplicativo o SDK do Android irá gerar os arquivos .class (bytecodes do Java) referentes a cada classe .java (arquivo de código fonte) e posteriormente ele irá traduzir todos os arquivos .class em arquivos .dex (bytecodes da Dalvik VM), ou seja, existe um passo a mais no processo de compilação. A *Dalvik VM* executa apenas arquivos *.dex*. Veja na **Figura 1.4** o processo de geração dos bytecodes .dex.

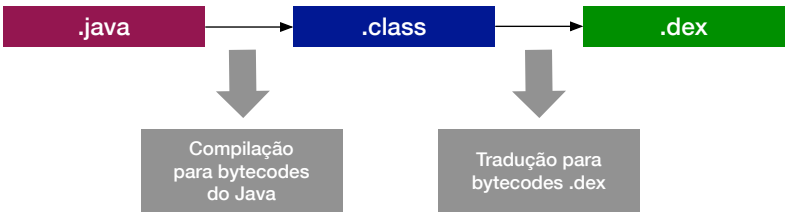


Figura 1.4. Processo de geração dos *bytecodes .dex* (Dalvik Executable).

1.4.3 Application Framework

O terceiro nível da Arquitetura do Android é composto pela Application Framework, conforme ilustra a **Figura 1.5**.

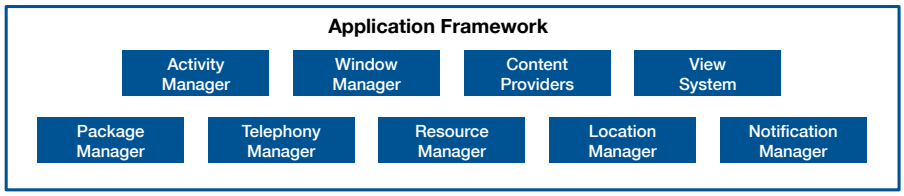


Figura 1.5. Terceiro nível da Arquitetura do Android, composto pela *Application Framework*.

A *Application Framework* compreende as APIs do Android que usamos no desenvolvimento de nossos aplicativos. Esta camada abstrai o acesso às bibliotecas escritas em C e C++ da camada *Libraries*, facilitando o trabalho dos desenvolvedores de aplicativos para Android.

1.4.4 Applications

O quarto e último nível da Arquitetura do Android é composto pelas *Applications*, conforme ilustra a **Figura 1.6**.

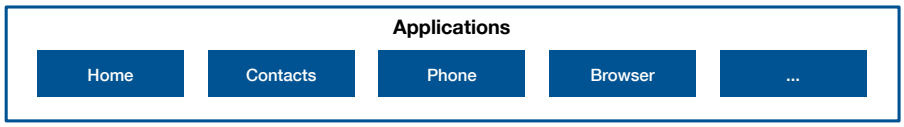


Figura 1.6. Quarto nível da Arquitetura do Android, composto pelas *Applications*.

A camada *Applications* compreende os aplicativos escritos em Java para o Android. Nesta camada encontram-se tantos os aplicativos nativos do Android quanto os aplicativos desenvolvidos por nós, desenvolvedores.

No Android aplicativos nativos e desenvolvidos por terceiros estão no mesmo nível, na camada *Applications*, o que nos permite desenvolver aplicativos que substituam os aplicativos nativos, sendo possível, por exemplo, criar um aplicativo de contatos que substitua o que já vem nativo no Android. Este é um recurso muito poderoso e provê grande flexibilidade na plataforma.

2 - Montando o ambiente de desenvolvimento

Antes de iniciar o desenvolvimento de aplicativos para o Android precisamos instalar as ferramentas necessárias, como o SDK do Android e um IDE. Todas as ferramentas que precisamos podem ser baixadas gratuitamente pela Internet e possuem versão para Mac, Linux e Windows. Para o desenvolvimento, precisaremos das seguintes ferramentas:

• **Java SE Development Kit** (JDK);

• **Android Development Kit** (Android SDK);

• **IDE Eclipse** (usaremos o Eclipse no decorrer deste curso, porém é possível utilizar outros IDEs para o desenvolvimento de aplicativos para o Android, como a IntelliJ Idea ou Netbeans).

• **Android Development Tools** (ADT): plugin do Eclipse para desenvolvimento de aplicativos para o Android.

2.1 Instalando o Java SE Development Kit (JDK)

Independente do Sistema Operacional que esteja usando (Mac, Linux ou Windows), para instalar o JDK é necessário baixá-lo no site da Oracle, através do link:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Após baixar o JDK específico para o seu Sistema Operacional, basta instalá-lo normalmente.

2.2 Instalando o Android Development Kit (Android SDK)

O Android SDK contém as ferramentas necessárias para o densenvolvimento de aplicativos para o Android, como as bibliotecas, o debugger, o emulador, etc. Para instalá-lo precisamos inicialmente baixá-lo no site do Android, através do link:

http://developer.android.com/sdk/index.html

O link irá abrir a página de download do SDK, conforme mostra a **Figura 2.1**.

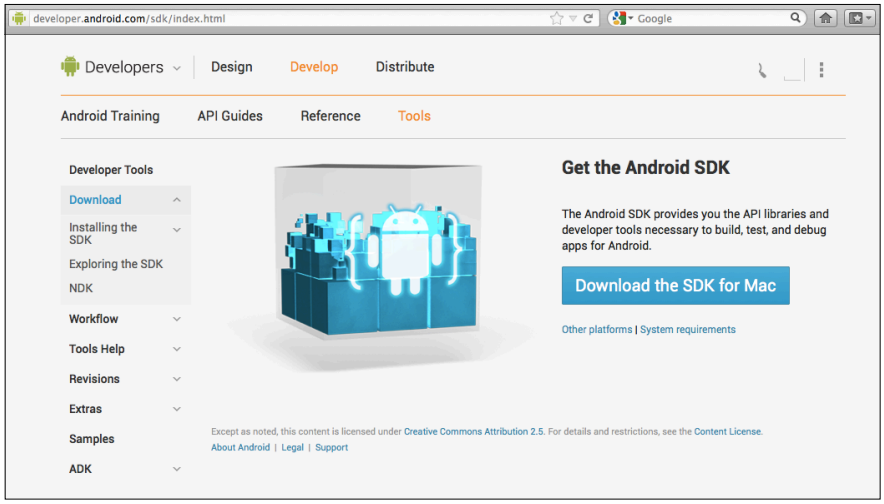


Figura 2.1. Página oficial para download do *Android SDK*.

Ao clicar no botão "**Download the SDK**" será baixado a última versão do SDK para o seu Sistema Operacional. Após baixá-lo, basta abrir o instalador e instalá-lo normalmente em seu computador.

Nota

Grave o local onde foi instalado o Android SDK no momento da instalação, pois precisaremos desta informação na hora de configurá-lo no Eclipse.

2.3 Instalando o Eclipse

Como foi dito anteriormente, utilizaremos o Eclipse como IDE no decorrer deste curso. Para baixá-lo, basta acessar o link:

http://www.eclipse.org/downloads

Na página de download, baixe a versão mais atual do **Eclipse IDE for Java EE Developers** específica para o seu Sistema Operacional e plataforma (32 ou 64bits, de acordo com o seu hardware).

Após o download, basta descompactar o pacote baixado no diretório que desejar.

Dica

O Eclipse não tem instalador, ao baixá-lo basta descompactar o pacote e começar a usar.

2.4 Instalando o Android Development Tools (ADT)

O Android Development Tools (ADT) é um plugin utilizado para integrar o Android SDK ao Eclipse. O ADT facilita o desenvolvimento de um aplicativo Android, possibilitando:

- O debug do projeto diretamente no IDE;
- Compilar e empacotar um projeto para instalação em um dispositivo Android;
- Criar telas facilmente com o editor visual;
- e outras facilidades mais.

Para instalar o ADT, basta seguir os passos:

1. Abra o Eclipse e, no item "**Help**" do menu, clique na opção "**Install new Software**", conforme mostra a **Figura 2.2**.

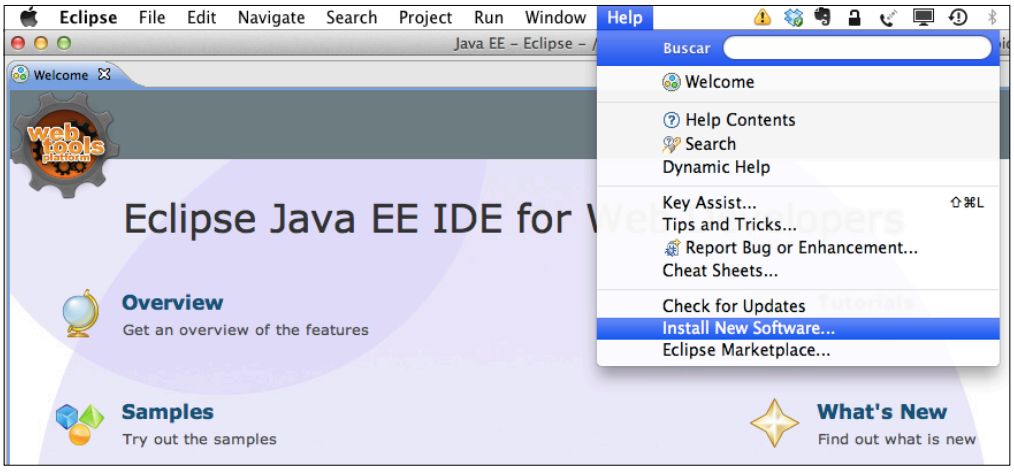


Figura 2.2. Utilizamos a opção "**Install New Software**" do Eclipse para instalar o *ADT*.

2. Na janela de instalação do novo plugin, clique no botão "**Add**", conforme mostra a **Figura 2.3**.

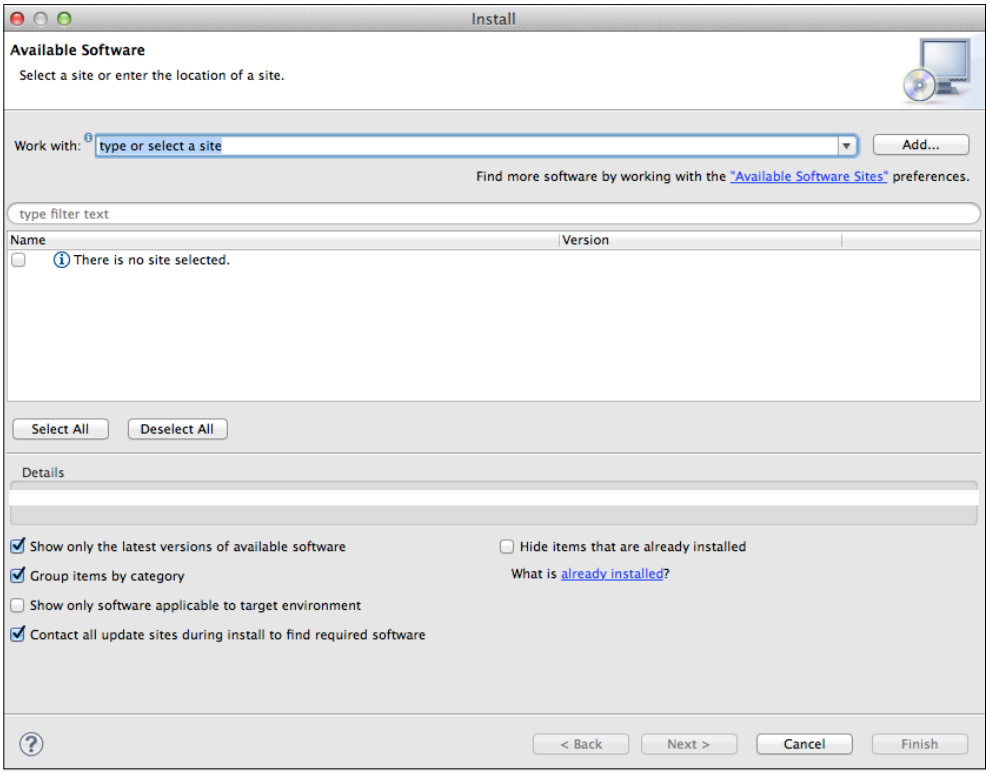


Figura 2.3. Tela do Eclipse para adicionar novo plugin/extensão.

3. Na janela de adição do repositório do plugin, no campo "**Name**" informe um nome qualquer, por exemplo "Android Development Tool (ADT)" (este nome servirá de identificação para o plugin). No campo "**Location**" informe a seguinte URL:

   https://dl-ssl.google.com/android/eclipse

Após o preenchimento dos campos, clique em "**Ok**", conforme mostra a **Figura 2.4**.

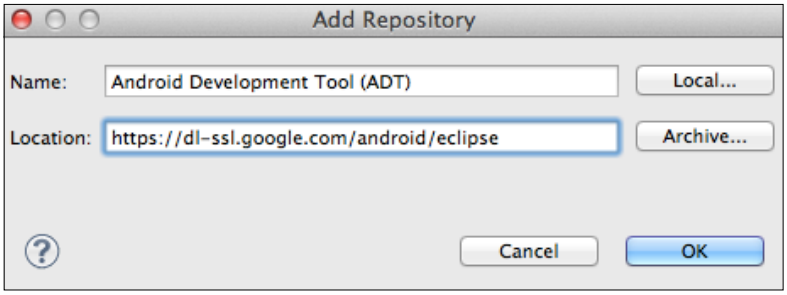


Figura 2.4. Tela do Eclipse para informar o repositório do plugin/extensão a ser instalado.

4. Na próxima janela, clique no botão "**Select All**" para selecionar todos os pacotes do ADT para instalação. Após selecionados todos os itens, clique em "**Next**" para avançar, conforme mosta **Figura 2.5**.

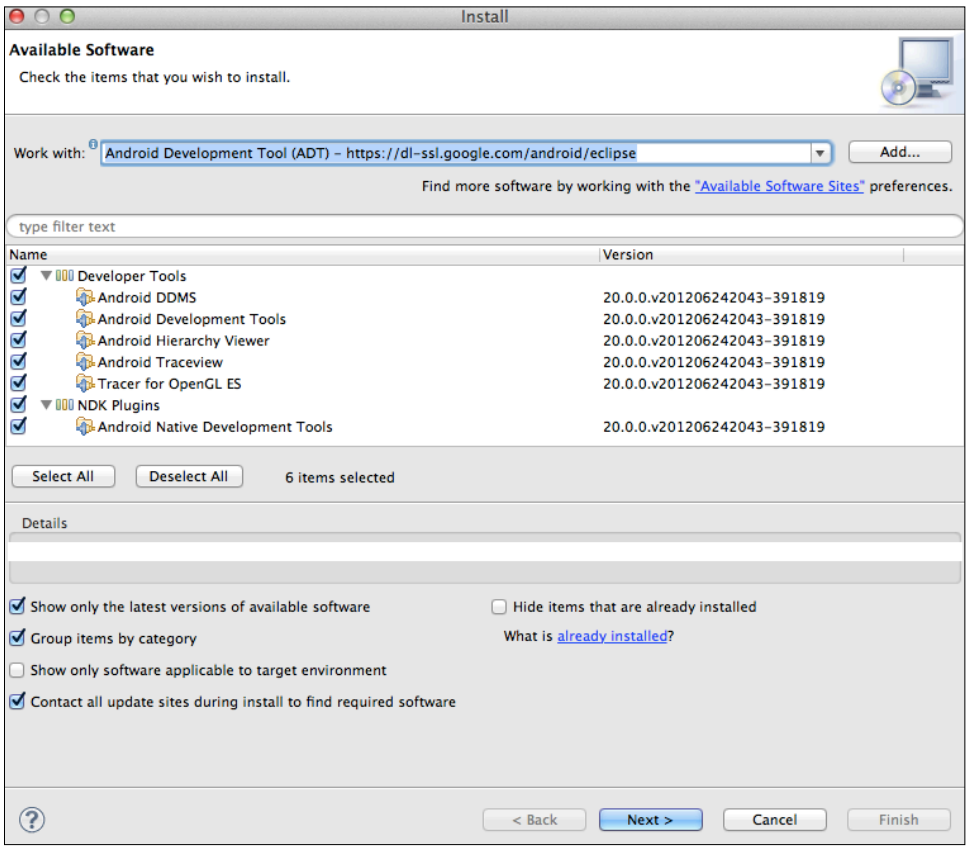


Figura 2.5. Itens do plugin *ADT* selecionados para instalação no Eclipse.

5. Clique em "**Next**" duas vezes, até chegar no passo "**Review Licenses**".

6. No passo "**Review Licenses**", marque a opção "**I accept the terms of the license agreements**" e clique no botão "**Finish**" para iniciar a instalação, conforme mostra a **Figura 2.6**.

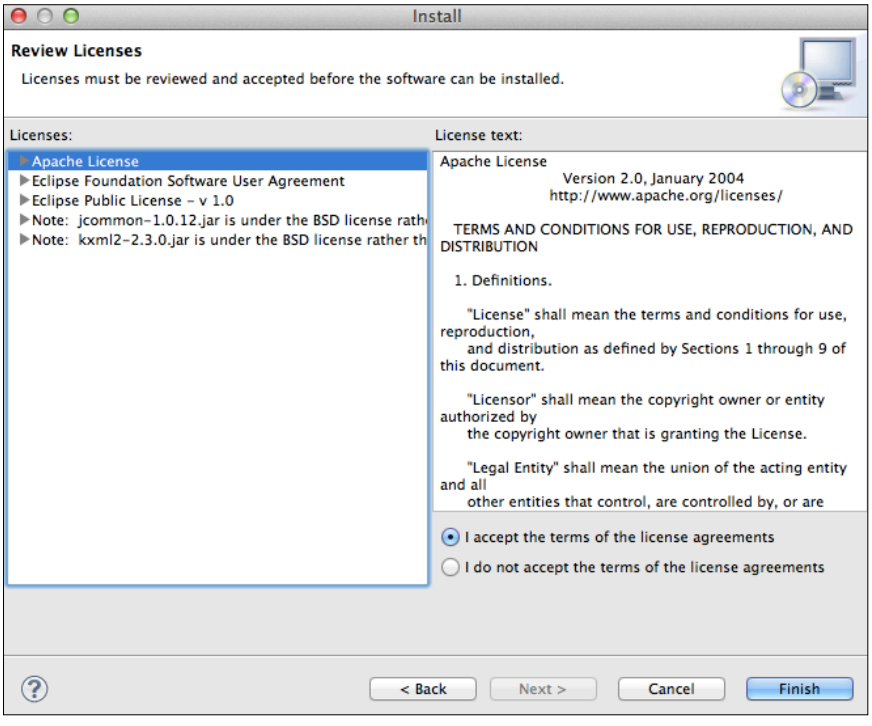


Figura 2.6. Tela com os termos de licença do plugin *ADT*.

7. Após instalado, uma janela irá solicitar que seja reiniciado o Eclipse, portanto clique em "**Restart**" para reiniciá-lo.

8. Após reiniciar o Eclipse, irá aparecer uma janela pedindo para configurar o Android SDK. Como já instalamos o Android SDK anteriormente, nesta janela basta marcar a opção "**Use existing SDKs**", no campo "**Existing Location**" informar o caminho onde foi instalado o Android SDK em seu computador e clicar em "**Finish**".

2.5 Instalando uma plataforma do SDK

Após configurar o Android SDK no plugin ADT no Eclipse, devemos usar o *Android SDK Manager* para baixar as plataformas necessárias para o desenvolvimento.

Dica

O Android SDK é modular, nos permitindo instalar apenas as plataformas que iremos usar para o desenvolvimento de aplicativos. Por exemplo, se for desenvolver aplicativos para a versão 4.1 (API 16) do Android, basta baixar apenas esta plataforma para o Android SDK.

Nota

No momento em que este curso foi escrito, a versão mais atual do Android é a 4.1 (API 16), portanto iremos baixar apenas esta plataforma no Android SDK Manager.

Para instalar a plataforma do SDK, siga os passos:

1. Abra o *Android SDK Manager* através do Eclipse, clicando sobre o ícone  "**Opens the Android SDK Manager**" localizado na barra de ferramentas (ou através do menu "**Window**" > "**Android SDK Manager**").

2. Na tela do SDK Manager, marque todos os itens da plataforma "**Android 4.1 (API 16)**" (ou da versão mais atual) e todos os itens do pacote "**Extras**", conforme mostra **Figura 2.7**.

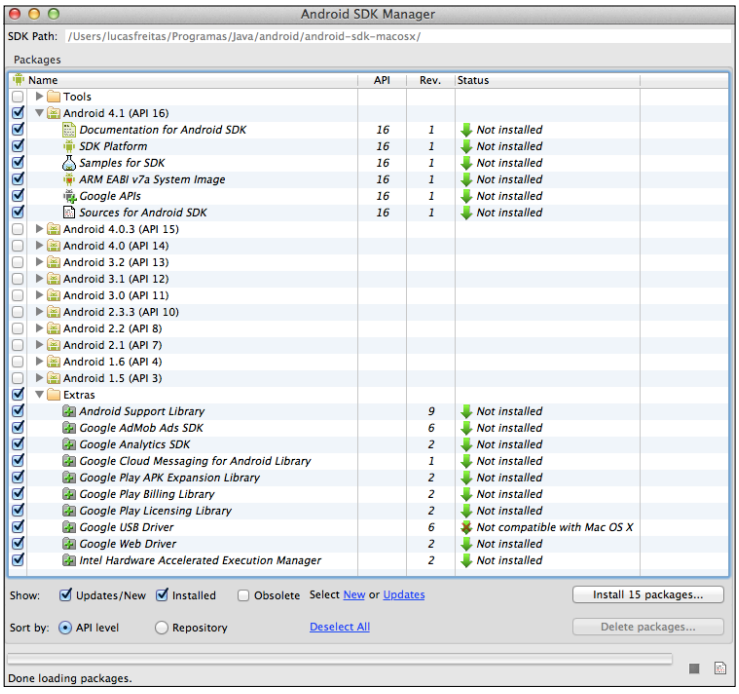


Figura 2.7. Selecionando a plataforma do Android a ser instalada através do Android SDK Manager.

Após selecionar a plataforma "**Android 4.1**" e o pacote "**Extras**", clique no botão "**Install packages**" para prosseguir com a instalação.

3. No próximo passo, o SDK Manager pedirá para confirmar os pacotes a serem instalados. Marque a opção "**Accept All**" e clique no botão "**Install**", conforme mostra a **Figura 2.8**.

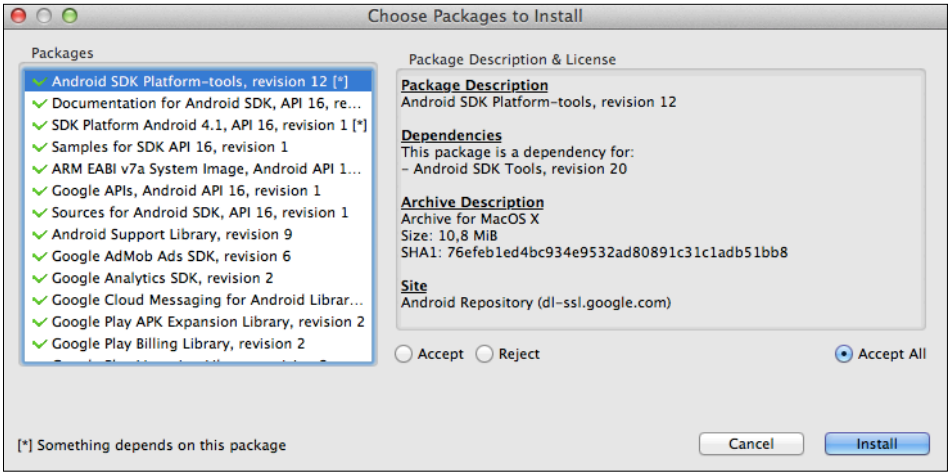


Figura 2.8. Confirmando os pacotes a serem instalados através do SDK Manager.

4. Ao final da instalação da plataforma aparecerá uma janela pedindo para reiniciar o *Android Debug Bridge* (ADB). Nesta janela, apenas clique em "**Yes**" para reiniciar o ADB.

2.6 Criando o AVD (Android Virtual Device)

Para testar nossas aplicações desenvolvidas para o Android precisamos criar um *AVD (Android Virtual Device)*. O AVD é um emulador que simula um dispositivo com Android, onde podemos testar nossos aplicativos.

Para criar um AVD, siga os passos:

1. Abra o Eclipse e clique no ícone  "**Opens the Android Virtual Device Manager**", localizado na barra de ferramentas (ou através do menu "**Window**" > "**AVD Manager**").

2. Na janela do "**Android Virtual Device Manager**", clique no botão "**New**" para criar um novo AVD, conforme mostra a **Figura 2.9**.

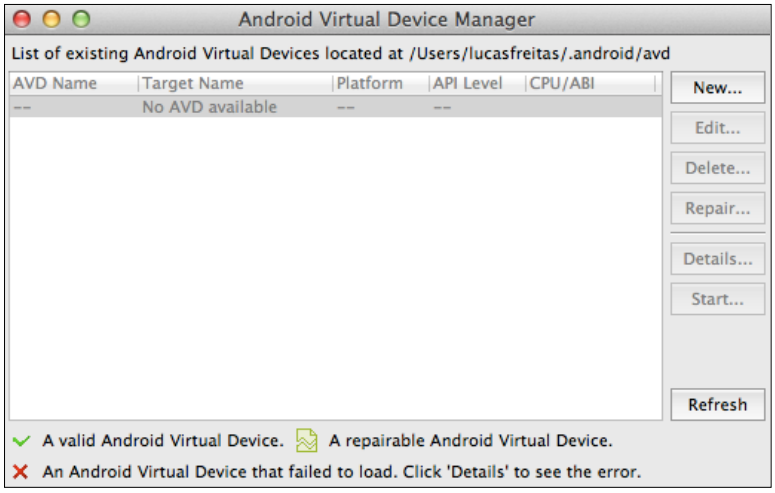


Figura 2.9. *Android Virtual Device Manager*, utilizado para gerenciar os AVDs.

3. No próximo passo, preencha os campos conforme mostra a **Figura 2.10** e clique no botão "**Create AVD**".

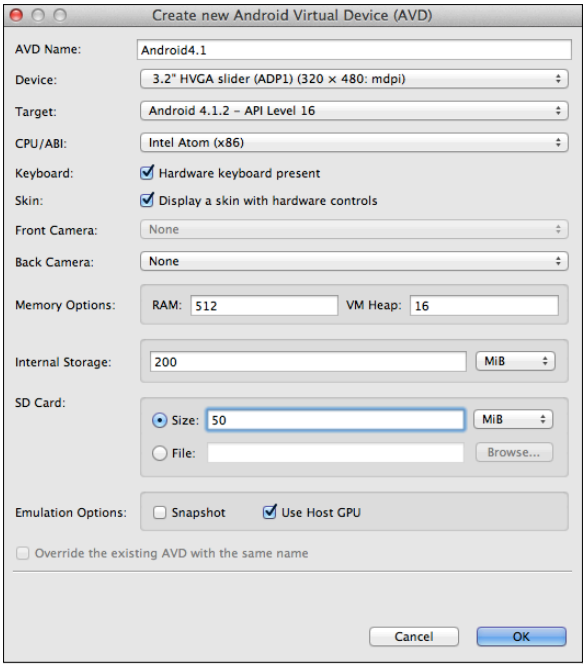


Figura 2.10. Tela para criar um novo *Android Virtual Device*.

Nota

Conforme mostra a **Figura 2.10**, criamos um AVD para emular o Android 4.1 (API 16) já adicionando a simulação de um SD Card de 50Mb (que nos permitirá copiar arquivos para o emulador).

4. Para testar o AVD, abra o novamente o *Android Virtual Device Manager*, selecione o AVD que você criou e clique no botão "**Start**". Ao aparecer a tela "**Launch Options**" deixe a configuração padrão, clique no botão "**Launch**" e aguarde a inicialização do AVD.

Dica

Ao iniciar um AVD, deixe-o aberto durante todo o desenvolvimento do seu aplicativo, não sendo necessário iniciá-lo toda vez que depurar a aplicação. Isto irá economizar tempo no desenvolvimento.

Nosso ambiente agora está totalmente configurado e pronto para o desenvolvimento de aplicativos.

3 - Criando o primeiro aplicativo

Com o ambiente de desenvolvimento configurado corretamente, veremos como criar um aplicativo simples para Android, o famoso **Hello World**. Após a criação do aplicativo iremos conhecer toda a estrutura que foi gerada na construção do projeto.

3.1 Criando um projeto do Android

Criar um projeto para o Android é simples e o faremos executando os passos a seguir:

1. Inicialmente devemos abrir o Eclipse e entrar na opção "**File**" > "**New**" > "**Project**".

2. Na tela "*New Project*" devemos entrar na opção "**Android**" > "**Android Application Project**", conforme mostra a **Figura 3.1**.

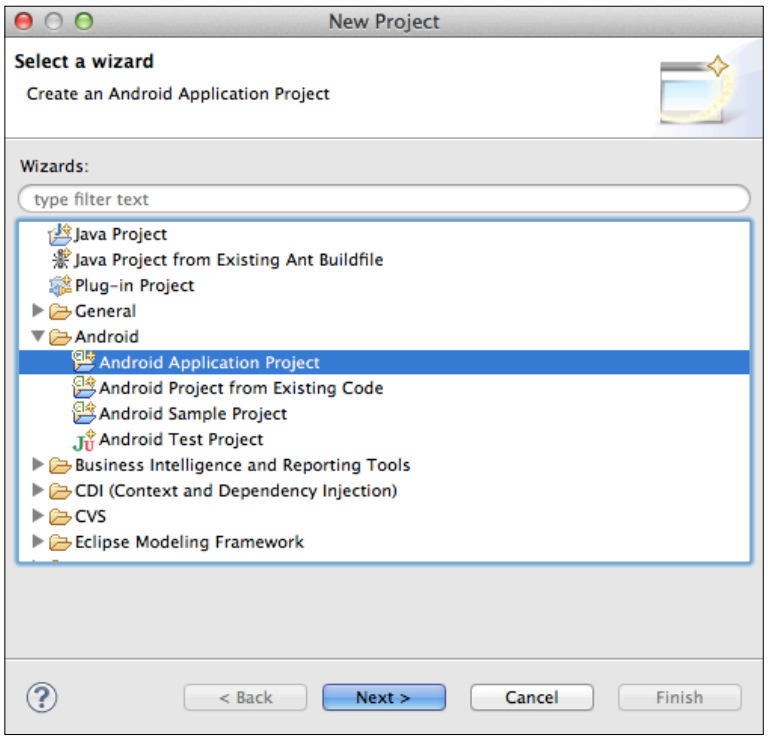


Figura 3.1. Criando um novo projeto do Android.

3. Na próxima tela devemos configurar o nosso projeto, conforme mostra a **Figura 3.2**, preenchendo os seguintes campos:

• O campo "**Application Name**" determina o nome que será mostrado na Play Store. Vamos preencher este campo com o valor: HelloWorld

• O campo "**Project Name**" é usado apenas pelo Eclipse para identificar o nosso projeto e deve ser único por workspace. Vamos preencher este campo com o valor: HelloWorld

• O campo "**Package Name**" determina o nome do pacote padrão para nosso aplicativo. Vamos preencher este campo com o valor: br.com.hachitecnologia.helloworld

• O campo “**Minimum Required SDK**” serve para especificar a versão mínima do Android que a aplicação irá suportar. No nosso caso, iremos selecionar a opção “**API 8: Android 2.2 (Froyo)**”

• O campo “**Target SDK**” serve para especificar a versão do Android que será usada para testar/homologar o aplicativo

• O campo “**Compile With**” serve para especificar a versão da API que será usada para compilar o nosso projeto. No nosso caso, iremos selecionar a opção “**API 16: Android 4.1 (Jelly Bean)**”

• O campo “**Theme**” serve para especificar o Tema (padrão de estilo/cor) para as telas do aplicativo. Portanto, deixaremos o valor padrão

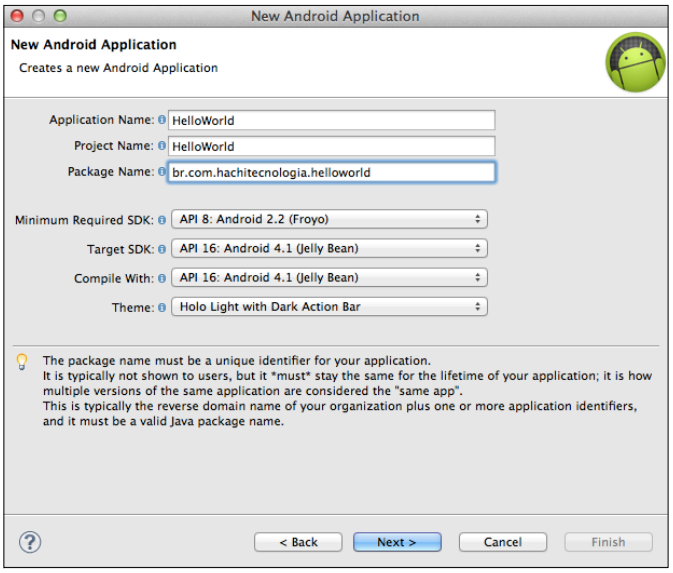


Figura 3.2. Configurando o projeto HelloWorld.

Preenchidos os campos necessários, clicamos no botão “**Next**”.

4. No próximo passo o assistente perguntará se, ao criar o projeto, desejamos criar também uma Activity. No nosso caso, como vamos precisar de uma Activity, marcaremos a opção “**Create activity**”. Devemos deixar selecionada também a opção “**Create Project in Workspace**”. Selecionadas as devidas opções, devemos clicar no botão “**Next**”, conforme mostra a **Figura 3.3.** *[Aprenderemos sobre as Activities no decorrer do curso]*

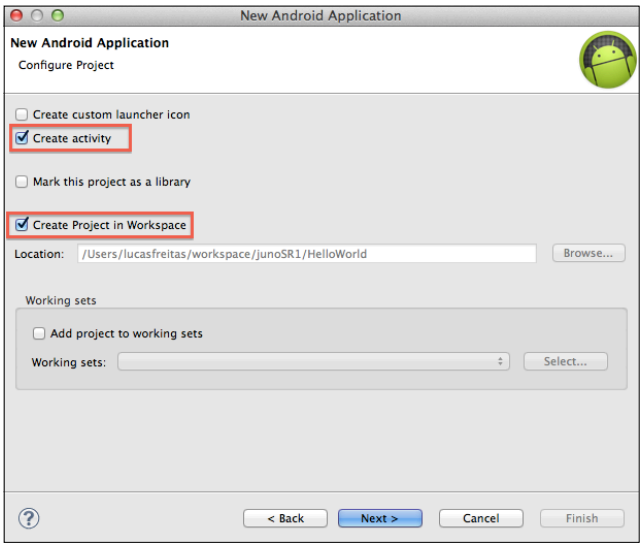


Figura 3.3. Configurando o projeto HelloWorld.

5. No próximo passo o assistente perguntará qual o estilo da nova Activity que desejamos criar. Marcaremos a opção "**Create Activity**" e selecionaremos o item "**BlankActivity**" para criar uma nova Activity padrão, em branco. A **Figura 3.4** ilustra este passo.

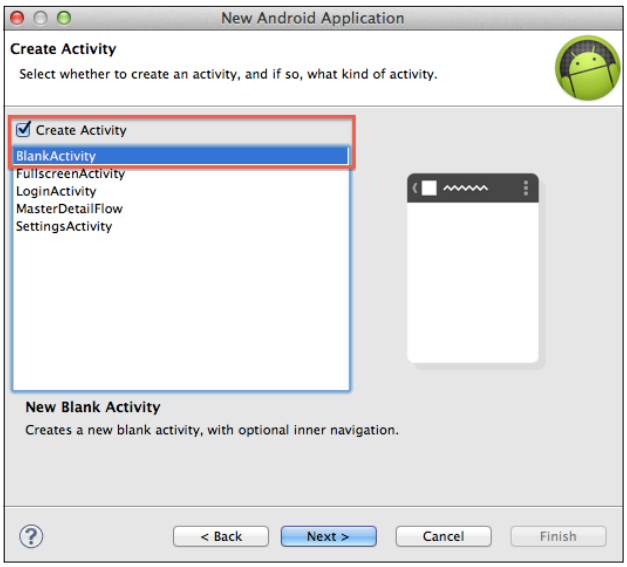


Figura 3.4. Configurando o projeto HelloWorld.

6. No próximo e último passo o assistente nos pede para informar as propriedades da Activity que será gerada automaticamente. Iremos preencher os campos com os valores abaixo, conforme mostra a **Figura 3.5**:

• O campo "**Activity Name**" serve para especificar o nome da Activity padrão que será gerada automaticamente. Informamos o valor: HelloWorldActivity

• O campo "**Layout Name**" serve para especificar o nome do arquivo de layout da Activity. Informamos o valor: activity_hello_world

• O campo "**Navigation Type**" serve para especificar o tipo de navegação da Activity. Selecionamos a opção: None

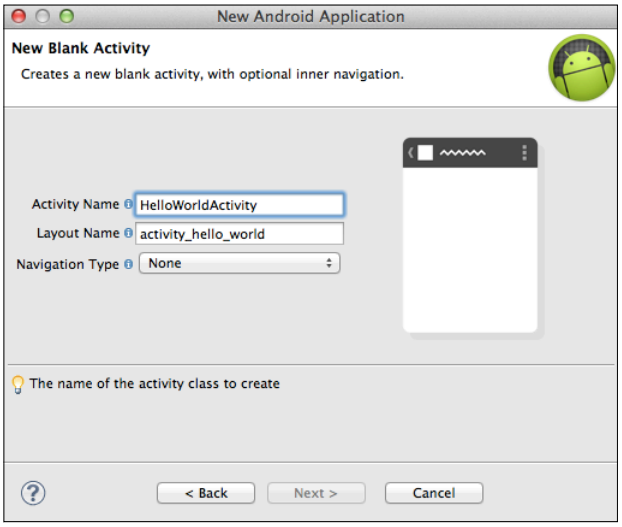


Figura 3.5. Configurando o projeto HelloWorld.

Preenchidos os campos necessários, seguimos clicando no botão "**Finish**". Nosso projeto está criado. Perceba a estrutura que foi gerada, parecida com a da **Figura 3.6**.

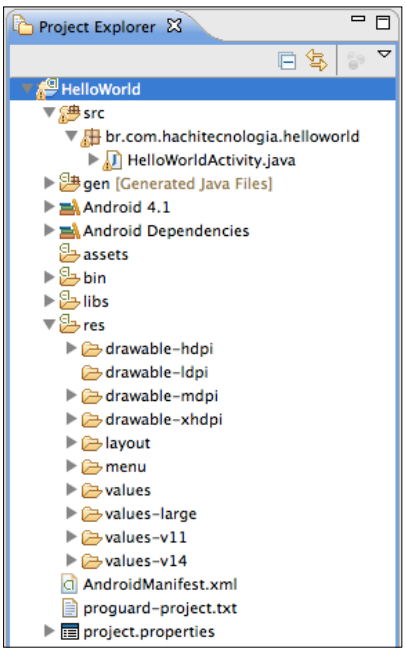


Figura 3.6. Estrutura do projeto HelloWorld.

3.2 Executando o projeto

Agora que nosso projeto Hello World está criado, é hora de testá-lo. Para isto, vamos executar os seguintes passos:

1. Abrimos a opção **Run** > **Run Configurations** no menu;

2. Na janela seguinte, selecionamos a opção "**Android Application**" e clicamos no ícone  "**New Android Configuration**", conforme mostra a **Figura 3.7**.

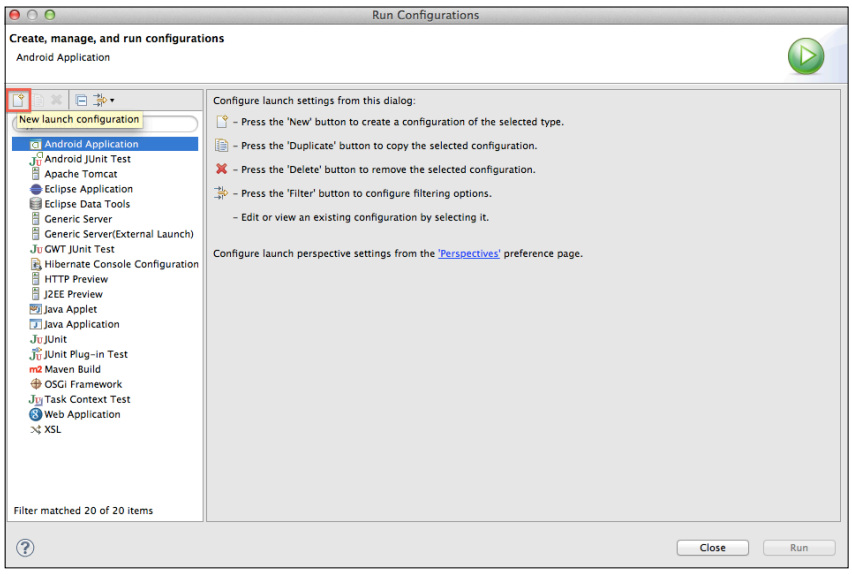


Figura 3.7. Configurando a execução do aplicativo HelloWorld.

3. Na tela de configuração seguinte, no campo "**Name**", informamos o valor: HelloWorld.

4. Na aba "**Android**", na opção "**Project**" clicamos no botão "**Browse**", selecionamos o projeto HelloWorld e clicamos em "**Ok**".

5. Ainda na aba “**Android**”, no item “**Launch Action**”, selecionamos na opção “**Launch**” a activity ***HelloWorldActivity***, conforme mostra a **Figura 3.8**;

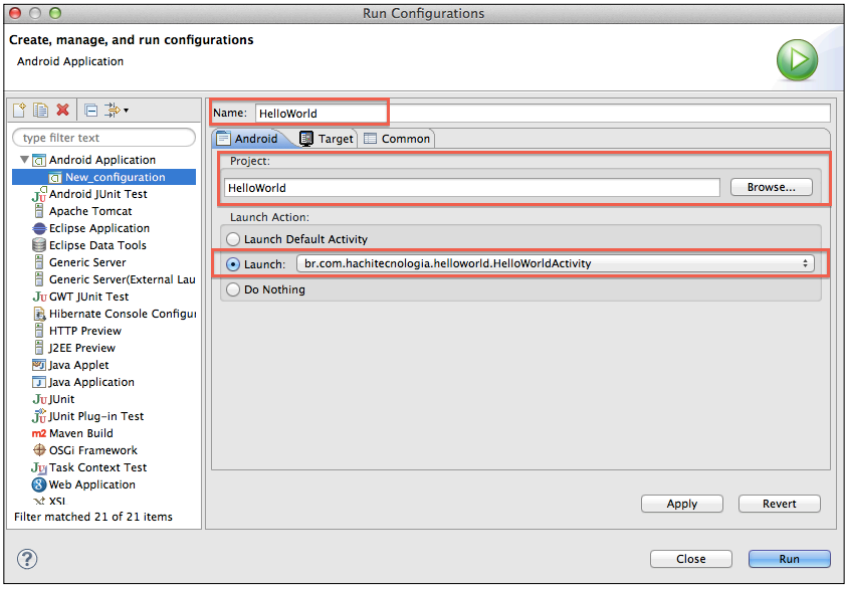


Figura 3.8. Configurando a execução do aplicativo HelloWorld.

6. Na aba “**Target**”, selecionamos o AVD Android 4.1 que criamos anteriormente, conforme mostra a **Figura 3.9**.

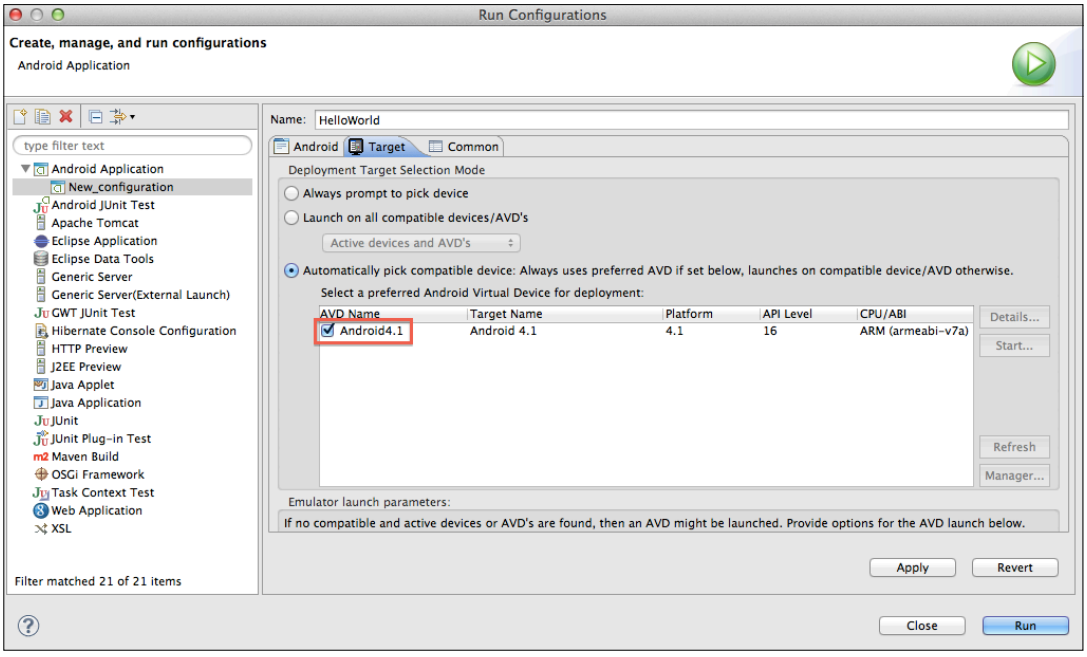


Figura 3.9. Configurando a execução do aplicativo HelloWorld.

7. Clicamos no botão “**Apply**”, em seguida clicamos no botão “**Run**” e aguardamos o aplicativo ser inicializado.

Após inicializado, nosso projeto Hello World já está instalado e em execução no emulador do Android, conforme mostra a **Figura 3.10**.

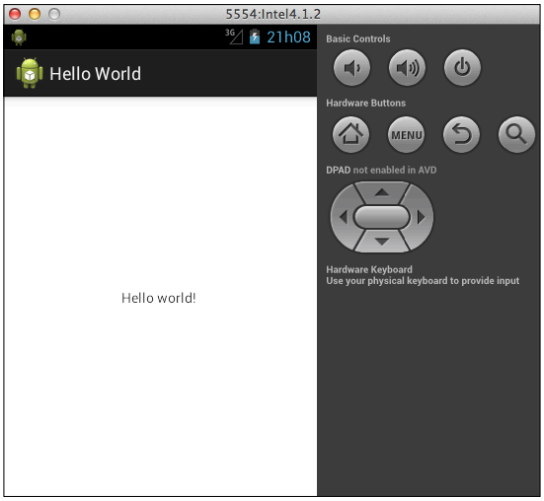


Figura 3.10. Projeto HelloWorld executando no emulador do Android.

3.3 Entendendo a estrutura do projeto

Agora que criamos nosso primeiro aplicativo e o executamos, vamos conhecer melhor a estrutura que foi gerada na construção do projeto.

Como você pôde perceber na **Figura 3.6**, toda a estrutura do projeto Hello World foi gerada automaticamente pelo ADT. É fundamental que tenhamos conhecimento sobre essa estrutura para trabalharmos de forma correta em nosso projeto, e é isso que vamos aprender agora.

Antes de falarmos sobre a estrutura do projeto, é preciso saber que um aplicativo do Android é composto pelos seguintes componentes:

- Activities;
- Services;
- Content Providers;
- Broadcast Receivers.

Não se preocupe com estes componentes agora, pois iremos aprender sobre cada um deles no decorrer deste curso. Por hora, apenas tenha em mente que são estes componentes que compõem um aplicativo para Android. Veja na **Figura 3.11** a ilustração de como estes componentes estão ligados ao núcleo do Android.

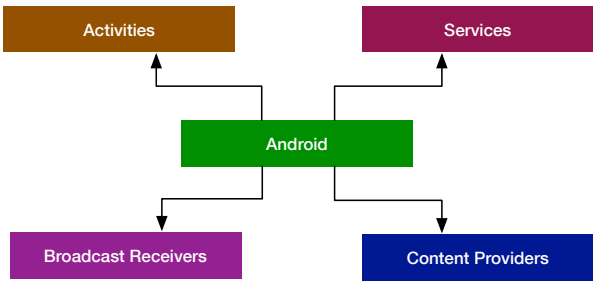


Figura 3.11. Composição de um aplicativo Android, formado por componentes.

3.3.1 O diretório “src”

O diretório *src* da estrutura do projeto é o local onde ficam todas as nossas classes escritas em Java, ou seja, neste diretório iremos encontrar os arquivos de código fonte *.java* e os *pacotes* que os organizam.

No momento da criação do projeto tivemos que configurar a propriedade *Package Name,* que representa o pacote padrão das nossas classes *.java*. Durante todo o desenvolvimento do aplicativo devemos seguir esta nomenclatura de pacote, mas claro que podemos criar subpacotes dentro deste pacote principal, como forma de organizar melhor o nosso projeto.

Dica

Utilizar subpacotes dentro do pacote principal para organizar as classes *.java* do projeto é uma boa prática e melhora a organização e entendimento do seu projeto. Podemos organizar, por exemplo:

- Classes Activities dentro de um pacote de nome: pacote_principal.**activity**
- Classes referente ao modelo dentro de um pacote de nome: pacote_principal.**modelo**
- etc.

3.3.2 O diretório “gen”

O diretório *gen* é onde encontram-se as classes geradas automaticamente pelo SDK do Android. Neste diretório encontra-se classes que o Android usa internamente para executar o projeto. Por exemplo, a classe R.java, que é a classe onde são mapeadas as constantes de acesso aos recursos do nosso aplicativo; a classe BuildConfig.java, que é a classe onde são definidas algumas configurações da construção do nosso projeto, etc. O conteúdo deste diretório é gerado automaticamente e **NÃO** deve, de forma alguma, ser alterado, ou seja, não devemos modificar o conteúdo de nenhuma das classes existentes neste diretório.

Dica

Você entenderá melhor o objetivo do diretório *gen* e das suas classes no decorrer deste curso. Por hora, apenas tenha em mente que este diretório **NÃO** deve ter seu conteúdo alterado, ou seja, não altere o conteúdo de nenhuma classe deste diretório.

3.3.3 O diretório “assets”

O diretório *assets* é utilizado para colocar recursos extras do seu aplicativo, como páginas html, arquivos de texto, fontes TrueType, arquivos de áudio, etc.

3.3.4 O diretório “bin”

O diretório *bin* é onde ficam os arquivos gerados no momento da construção (build) do projeto, como o pacote *.apk* (pacote instalável do aplicativo no Android).

3.3.5 O diretório “res”

O diretório *res* é o local onde encontram-se os recursos da nossa aplição. Dentro deste diretório, existem subdiretórios para cada tipo de recurso, como:

- Diretório ***drawable***: onde ficam os recursos de imagem do nosso aplicativo (arquivos png, jpeg);
- Diretório ***layout***: onde ficam os arquivos XML com o layout das telas do aplicativo;
- Diretório ***values***: onde ficam outros recursos do aplicativo como String Resources, Layout de Preferências, etc.

Este é sem dúvida o diretório que mais merece atenção no desenvolvimento de um aplicativo para o Android, pois é nele onde iremos configurar as telas do nosso aplicativo permitindo que sejam suportadas por diversos dispositivos com diferentes configurações de resolução, permitir a internacionalização do nosso aplicativo, etc.

3.3.6 O arquivo “AndroidManifest.xml”

O *AndroidManifest.xml* é um arquivo XML onde ficam as configurações necessárias para a execução do aplicativo para Android e é neste arquivo que devem ser configurados os componentes da aplicação, como as Activities, os Services, os Broadcast Receivers e os Content Providers. Este arquivo é obrigatório e fica na raiz do projeto. O *AndroidManifest.xml* merece atenção especial pois é onde definimos informações importantes, como:

• O nome do pacote padrão do projeto;

• A configuração dos componentes usados no aplicativo;

• A versão da API do Android que o aplicativo irá suportar;

• A versão do próprio aplicativo;

• e várias outras configurações.

É no *AndroidManifest.xml* também que configuramos as permissões que nosso aplicativo precisará para executar no Android. *[Aprenderemos sobre as permissões de acesso mais à frente neste curso]*

Veja abaixo, como exemplo, o conteúdo do *AndroidManifest.xml* do aplicativo HelloWorld.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.com.hachitecnologia.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="16"
      android:targetSdkVersion="15" />

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name=".HelloWorldActivity"
         android:label="@string/title_activity_hello_world" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

Neste exemplo, destacamos algumas propriedades:

• android:versionCode: esta propriedade do xml é obrigatória e define a versão do seu aplicativo Android. O conteúdo desta propriedade deve ser um número inteiro, iniciado por 1 e incrementado a cada nova versão do seu aplicativo.

• android:versionName: esta propriedade também é obrigatória e define a versão comercial do seu Aplicativo. Podemos informar qualquer nome de versão nesta propriedade, como: 1.0a, ou 2.0, etc.

• android:minSdkVersion e android:targetSdkVersion: determinam a versão mínima da API do Android suportada pela sua aplicação e a versão da API que seu aplicativo testado/homologado, respectivamente.

• application: dentro desta tag registramos os componentes que nosso aplicativo irá usar, como as Activities.

3.4 Exercício

Agora que você aprendeu a criar um projeto para o Android, é hora de colocar em prática:

1. Crie um projeto chamado ***HelloWorld***, conforme vimos neste capítulo, e execute-o no emulador do Android.

4 - Conhecendo os recursos do ADT

Quando estamos desenvolvendo um aplicativo para o Android é fundamental conhecermos as ferramentas que o plugin ADT do Eclipse nos disponibiliza. O ADT possui poderosos recursos que nos auxilia em diversas tarefas, como:

• Depuração do aplicativo;

• Visualização dos logs em tempo de execução;

• Enviar arquivos do computador para o emulador do Android;

• Analisar os processos em execução, threads, trafego de rede e uso do heap da Dalvik VM;

• Simular o envio de chamada e mensagem SMS para o emulador do Android;

• Tirar um screenshot da tela do emulador do Android;

• e diversos outros recursos.

Iremos conhecer alguns destes recursos pois iremos utilizá-los no desenvolvimento de aplicativos no decorrer do curso.

4.1 Visualizando logs com o LogCat

O *LogCat* é um recurso do ADT para análise de logs de um aplicativo Android. Através dele podemos visualizar os logs do nosso aplicativo enquanto ele estiver executando no emulador do Android.

Para mostrar a view do LogCat no Eclipse, basta ir no menu ***Window*** > ***Show View*** > ***Other*** e selecionar na lista o item ***Android*** > ***LogCat***.

Veja na **Figura 4.1** a view LogCat mostrando os logs do aplicativo *HelloWorld* que criamos.

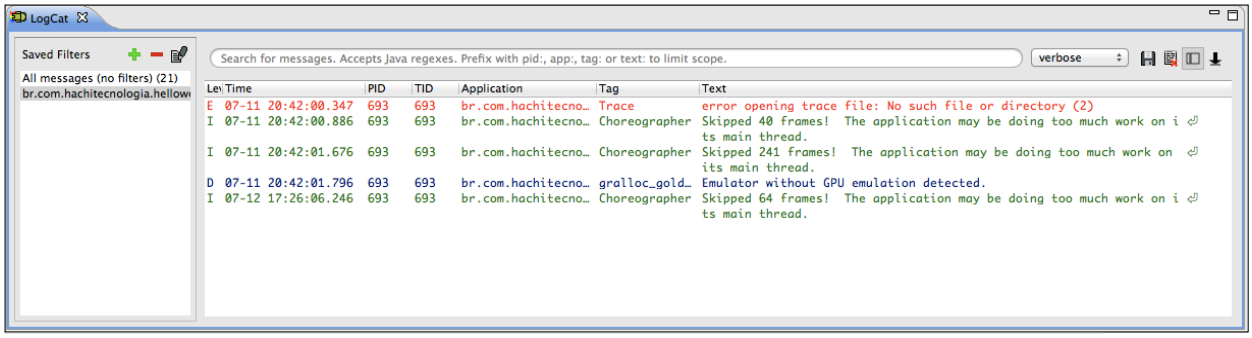


Figura 4.1. View LogCat do plugin ADT no Eclipse.

O LogCat permite fazer o filtro pelo tipo de mensagem de Log que deseja visualizar, de acordo com a sua necessidade. Existem 5 caterogias de mensagens de log:

• V - Verbose;

• D - Debug;

• I - Info;

- W - Warning
- E - Error

Para visualizar uma mensagem de log de acordo com uma categoria específica, na view LogCat, basta selecionar o tipo desejado no menu localizado no canto superior direito da view, conforme destacado na **Figura 4.2**.

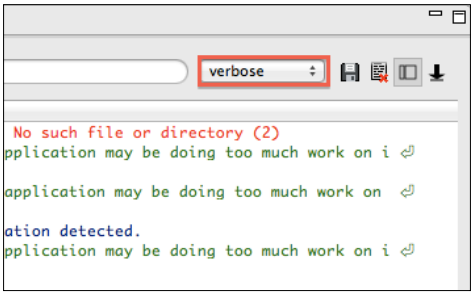


Figura 4.2. Menu para filtrar tipo de mensagem de log na view LogCat do ADT no Eclipse.

As mensagens de erro mostradas pelo LogCat são enviadas pelo aplicativo do Android atavés da classe *android.util.Log*. No decorrer do curso aprenderemos como usar esta classe para enviar mensagens de log para o console do LogCat.

4.2 Conhecendo a perspectiva DDMS

A perspectiva *DDMS* (Dalvik Debug Monitor Server) do Android possui um conjunto de views que nos auxilia no desenvolvimento e depuração de um aplicativo.

Para abrir a DDMS, vá até o menu ***Window*** > ***Open Perspective*** > ***Other*** > ***DDMS***.

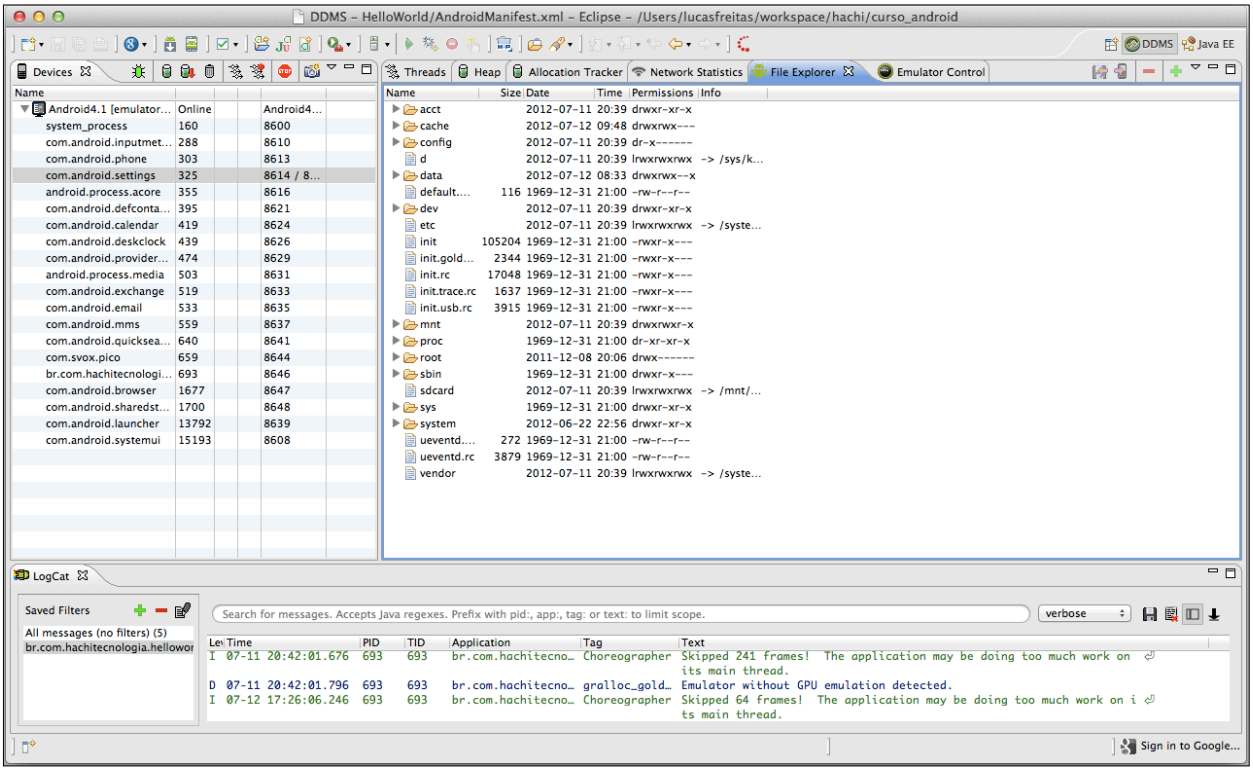


Figura 4.3. Perspectiva DDMS do plugin ADT do Eclipse.

Veja na **Figura 4.3** a perspectiva DDMS ativa no Eclipse, mostrando um conjunto de views utilitárias. Dentre as views do DDMS destacamos algumas:

• **Devices**: view que mostra todos os processos em execução no emulador do Android;

• **LogCat**: view que mostra os logs emitidos pela *classe android.util.log*;

• **File Explorer**: view que nos permite enviar arquivos do computador para o emulador do Android, e vice-versa;

• **Emulator Control**: view que nos permite simular chamadas e envio de SMS para o emulador do Android, além de simular a localização atual para teste do GPS.

Na DDMS temos também views que mostram as Threads em execução, o uso do Heap na Dalvik VM e estatísticas de tráfego da rede. Podemos também tirar um screenshot da tela do emulador do Android através da DDMS.

4.3 Realizando o debug do aplicativo

O ADT também nos permite realizar o debug dos nossos aplicativos Android. Debugar um aplicativo com o ADT é muito simples, bastando adicionar um breakpoint na linha desejada e executar o aplicativo em modo de debug, da mesma forma como fazemos em um aplicativo Java para desktop. A **Figura 4.4** mostra o debug do nosso aplicativo HelloWorld sendo executado.

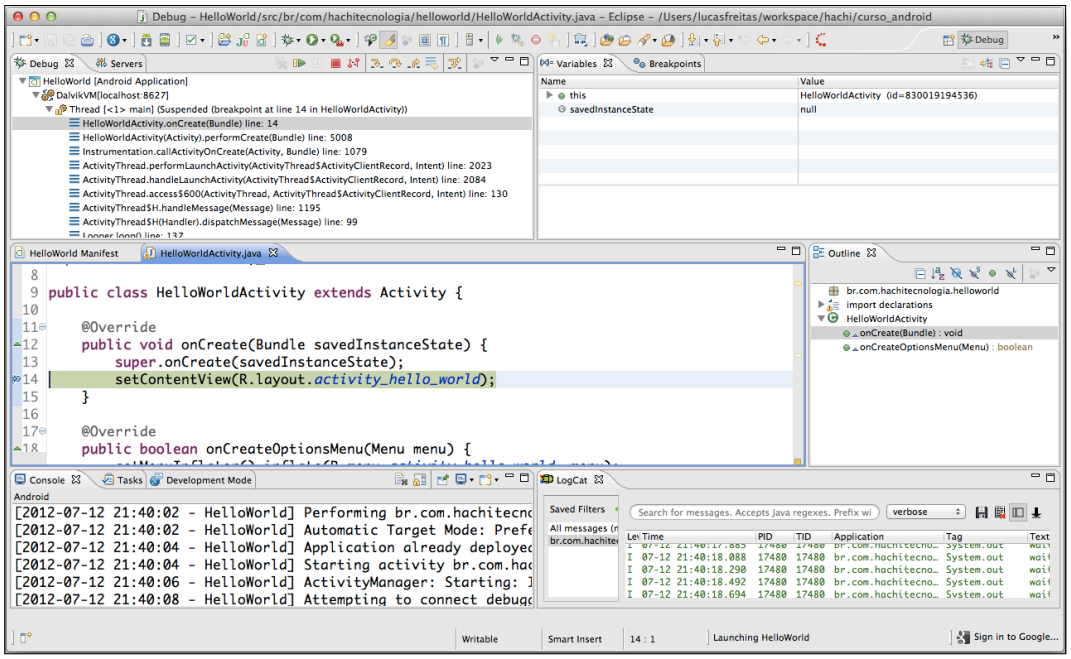


Figura 4.4. Realizando o debug do aplicativo HelloWorld através do plugin ADT no Eclipse.

5 - Conceitos básicos

5.1 Activities

Como mencionamos anteriormente, um aplicativo do Android é composto por componentes, e dentre estes componentes está a Activity. Activities são classes Java que controlam os eventos de uma tela do aplicativo.

Para ser uma Activity, uma classe Java precisa herdar da classe *android.app.Activity* da API do Android. Veja abaixo o exemplo de uma Activity, a classe ***HelloWorldActivity*** do projeto ***HelloWorld*** que criamos anteriormente:

```
public class HelloWorldActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_world);
    }

}
```

5.2 Views

As Views são componentes gráficos que usamos para montar uma tela do nosso aplicativo e estão sempre ligadas às Activities, que controlam o seu comportamento.

O Android disponibiliza diversas Views para implementarmos as telas de um aplicativo, como:

- **TextView**: componente para mostrar um texto na tela do aplicativo;
- **EditText**: campo de entrada de texto;
- **Button**: botão que dispara um evento;
- **CheckBox**: caixa de seleção;
- **Gerenciador de Layout**: View mais complexa que pode conter outras Views, ou seja, um agrupador de Views;
- e diversas outras.

5.3 O método *setContentView()*

As Activities possuem um método chamado ***setContentView()*** que serve para definir a View que será renderizada na tela do aplicativo. A Activity *HelloWorldActivity* do nosso projeto *HelloWorld*, por exemplo, faz uma chamada para este método, passando como parâmetro a View a ser apresentada na tela, nesse caso o arquivo de Layout *activity_hello_world.xml*. *[Explicaremos sobre os Layouts posteriormente neste curso]*.

Dica

A chamada para o método ***setContentView()*** deve ser feita sempre no método ***onCreate*** da Activity.

5.4 Criando uma Activity

Para criar uma Activity, no Eclipse, devemos **clicar com o botão direito do mouse sobre o projeto Android que criamos** e, no menu, ir na opção "**New**" > "**Other**" e na janela que se abre selecionar a opção "**Android**" > "**Android Activity**".

Ao abrir o wizard para criação da nova Activity, clicamos em ***Next***, conforme mostra a **Figura 5.1**.

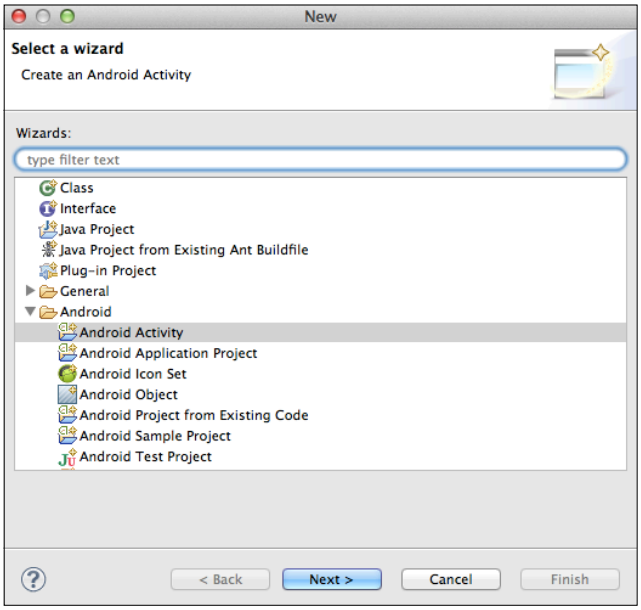


Figura 5.1. Wizard do ADT para criar uma nova *Activity*.

No passo seguinte, selecionamos o template padrão que usaremos para a tela da nova Activity. Em nosso caso usaremos um template em branco, selecionando a opção ***BlankActivity***, conforme mostra a **Figura 5.2**. Em seguida clicamos em ***Next***.

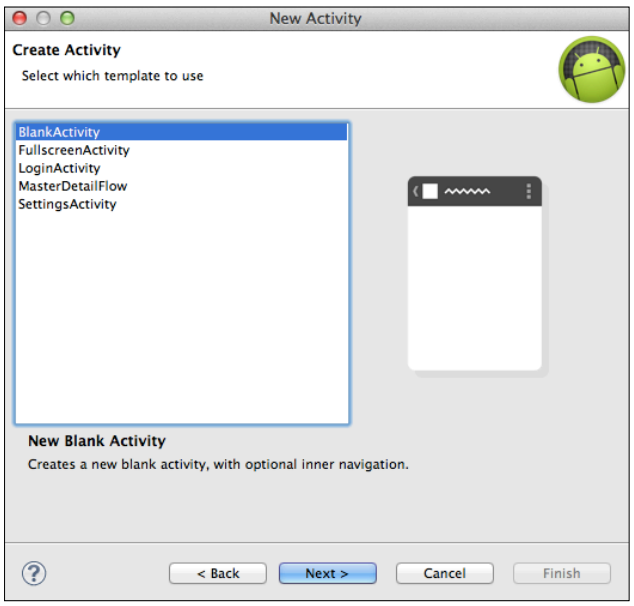


Figura 5.2. Wizard do ADT para criar uma nova *Activity*.

No passo seguinte, devemos preencher os seguintes campos (conforme mostra a **Figura 5.3**):

• **Activity Name**: o nome da Activity. Informaremos o valor: *MinhaPrimeiraActivity*

• **Layout Name**: nome do arquivo de layout da Activity. Informaremos o valor: *activity_minha_primeira*

• **Title**: título que daremos à Activity. Informaremos o valor: *Minha Primeira Activity*

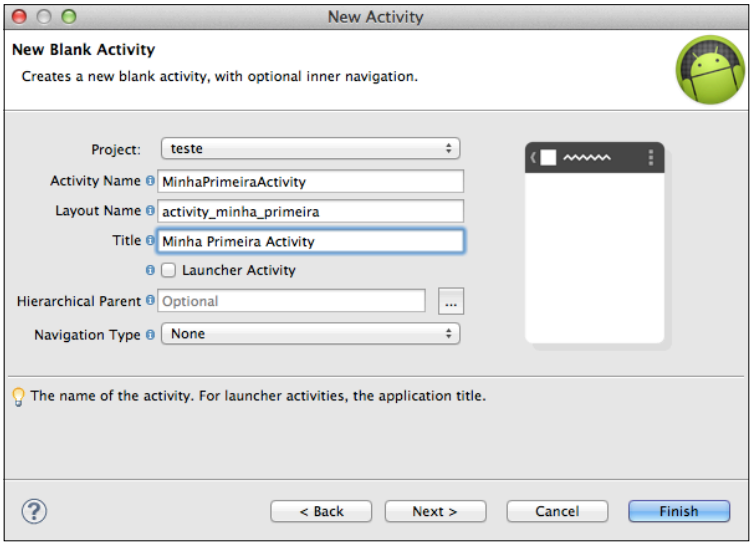


Figura 5.3. Wizard do ADT para criar uma nova *Activity*.

Após preencher as informações da nova Activity, clicamos em ***Finish*** e nossa Activity já estará criada.

Perceba que o ADT gerou automaticamente a nova Activity e uma tela de layout para ela, colocando a frase “Hello world!” no centro da tela. Ao executar o projeto no emulador, vemos o resultado da nova Activity, apresentando a mensagem “Hello world!”, conforme mostra a **Figura 5.4**.

Figura 5.4. Activity *MinhaPrimeiraActivity* em execução no emulador do Android.

5.5 Exercício

Agora que você aprendeu a criar uma Activity, é hora de colocar em prática:

1. Crie um novo projeto do Android, com o nome: ***Activities***.

2. Crie uma nova Activity neste novo projeto, com o nome: ***MinhaPrimeiraActivity***.

3. Configure esta nova Activity para ser executada ao iniciar o novo projeto no emulador do Android.

4. Execute o novo projeto no emulador do Android.

5.6 Logging

No ***Capítulo 4*** deste curso você aprendeu como usar a View *LogCat* do plugin ADT do Android para visualizar os logs dos aplicativos. Agora você irá aprender a disparar mensagens de Log no Android usando o LogCat.

Quando se fala em Log, em um aplicativo desenvolvido em Java, a primeira coisa que vem em mente é usar o ***System.out.println()*** para imprimir os logs no console. Porém o *System.out.println()* é pouco produtivo e não permite a categorização de mensagens de log. O Android já possui um sistema de Log padrão, o chamado ***LogCat***, disponível através da classe ***android.util.Log***.

O LogCat disponibiliza métodos estáticos para facilitar o uso de mensagens de Log em nosso aplicativo e o utilizamos chamando estes métodos através da seguinte sintaxe:

```
Log.X("Tag", "Mensagem");
```

Onde:

- **X** é o método estático de acordo com o tipo de mensagem de Log que deseja produzir;
- Tag: é a Tag agrupadora que possibilita usarmos o filtro da View LogCat para lermos apenas mensagens com uma determinada Tag;
- Mensagem: a mensagem que queremos apresentar na saída de Log.

Veja na **Tabela 5.1** a relação dos métodos disponíveis para emitir mensagens de Log.

Método do LogCat	Tipo da mensagem de Log
Log.**i**()	Mensagem de informação (**info**)
Log.**w**()	Mensagem de aviso (**warn**)
Log.**e**()	Mensagem de erro (**error**)
Log.**d**()	Mensagem de depuração (**debug**)
Log.**v**()	Mensagem de depuração detalhada (**verbose**)

Tabela 5.1. Métodos estáticos do LogCat para emissão de mensagens de Log no Android.

Para ver na prática o funcionamento de Logging no Android, vamos criar uma nova Activity, chamada *LoggingActivity*, com o seguinte conteúdo:

```
public class LoggingActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView texto = new TextView(getApplicationContext());
        texto.setText("Emitindo mensagens de Log com o LogCat. " +
                "Veja na View LogCat as mensagens de Log emitidas por esta Activity.");

        // Log de informação (info)
        Log.i("Trabalhando com Logs", "Mensagem de informação");

        // Log de aviso (warn)
```

```
                Log.w("Trabalhando com Logs", "Mensagem de aviso");

                // Log de erro (error)
                Log.e("Trabalhando com Logs", "Mensagem de erro");

                // Log de depuração (debug)
                Log.d("Trabalhando com Logs", "Mensagem de depuração");

                // Log de depuração detalhada (verbose)
                Log.v("Trabalhando com Logs", "Mensagem de depuração detalhada");

                setContentView(texto);
        }

}
```

Ao executarmos esta Activity, as mensagens de Log são apresentadas na View LogCat, conforme mostra a **Figura 5.5**.

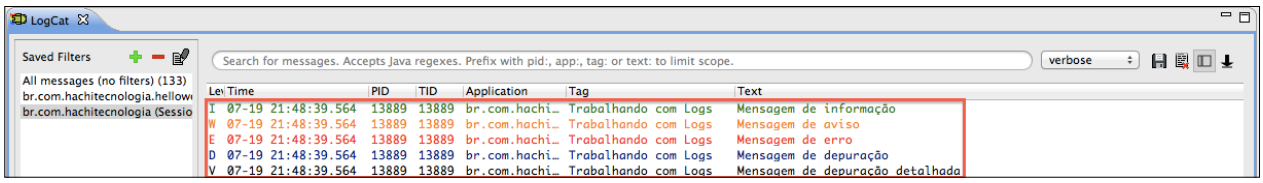


Figura 5.5. View LogCat do ADT mostrando as mensagens de Log disparadas pela Activity *LoggingActivity*.

Nota

Como aprendemos no ***Tópico 4.1*** *do* ***Capítulo 4***, na View LogCat podemos filtrar o tipo de mensagem que desejamos visualizar.

Na View LogCat, além de filtrar as mensagens de Log pelo tipo, podemos também filtrá-las pela ***Tag*** agrupadora que definimos na chamada do método da classe *Log*. Para fazer isto, na coluna "***Saved Filters***" da View LogCat, basta clicar no ícone  . Ao clicar neste ícone, na janela que se abre, basta preencher os campos de acordo com o filtro que desejamos, conforme mostra a **Figura 5.6**.

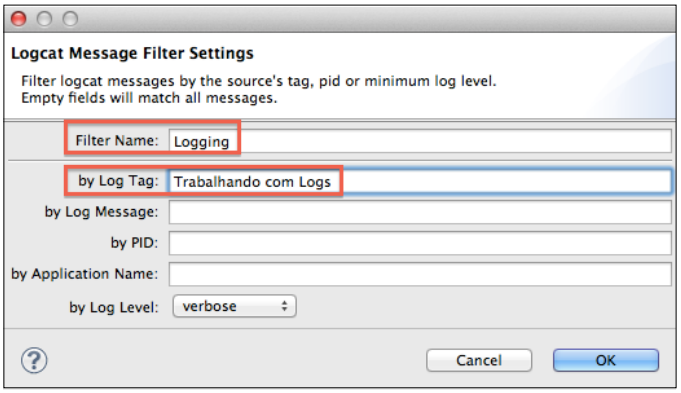


Figura 5.6. Filtrando mensagens de Log na View LogCat do plugin ADT.

5.7 Exercício

Agora que você aprendeu a trabalhar com Logging no Android, é hora de colocar em prática.

1. Crie uma nova Activity, no projeto ***Activities***, chamada ***LoggingActivity***, com o seguinte conteúdo:

```
public class LoggingActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		TextView texto = new TextView(getApplicationContext());
		texto.setText("Emitindo mensagens de Log com o LogCat. " +
				"Veja na View LogCat as mensagens de Log emitidas por esta Activity.");

		// Log de informação (info)
		Log.i("Trabalhando com Logs", "Mensagem de informação");

		// Log de aviso (warn)
		Log.w("Trabalhando com Logs", "Mensagem de aviso");

		// Log de erro (error)
		Log.e("Trabalhando com Logs", "Mensagem de erro");

		// Log de depuração (debug)
		Log.d("Trabalhando com Logs", "Mensagem de depuração");

		// Log de depuração detalhada (verbose)
		Log.v("Trabalhando com Logs", "Mensagem de depuração detalhada");

		setContentView(texto);

	}
}
```

2. Configure o projeto para que a Activity ***LoggingActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em "**Run**" > "**Run Configurations**".

3. Execute a Activity no emulador do Android.

4. Verifique as mensagens de Log apresentadas na View ***LogCat*** do plugin ADT do Eclipse.

5. Na View LogCat, filtre as mensagens de Log pelo tipo e pela *Tag* agrupadora.

5.8 A classe R

A classe ***R*** serve para referenciar todos os recursos do projeto (como imagens, mensagens, arquivos de layout, etc) facilitando o uso destes em nosso projeto. Esta classe é gerada automaticamente pelo plugin ADT do Eclipse e todo projeto do Android a possui.

Dica

A **classe R** é gerada automaticamente pelo plugin ADT do Eclipse e **jamais** deve ter seu conteúdo alterado.

Como mencionamos anteriormente (*no* ***Capítulo 3***) no diretório ***res*** de um projeto do Android encontram-se os recursos do aplicativo. Ao adicionar um recurso dentro de um dos subdiretórios do ***res*** (***drawable***, ***layout*** ou ***values***), este recurso será automaticamente referenciado na ***classe R*** para o utilizarmos em nossa aplicação.

Ao colocar uma imagem chamada ***foto.png*** no diretório ***res/drawable***, por exemplo, esta imagem será automaticamente referenciada na ***classe R*** e sua referência poderá ser utilizada em todo o projeto do Android.

5.8.1 Acessando um recurso através da classe R

Para exemplificar o uso da classe ***R***, em nosso projeto ***Activities*** (criado anteriormente), vamos criar uma mensagem no arquivo ***res/values/strings.xml*** (arquivo XML de mensagens da aplicação) e acessá-la através de uma Activity para apresentar seu conteúdo em uma View do tipo TextView.

Ao abrir o arquivo ***res/values/strings.xml***, um editor visual do ADT é aberto para nos auxiliar na edição deste arquivo. No editor visual, iremos clicar no botão ***Add*** para adicionar uma nova mensagem, conforme mostra a **Figura 5.7**.

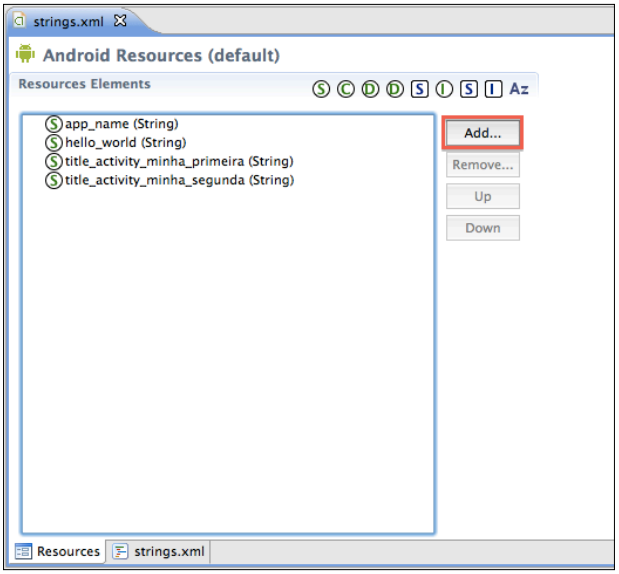


Figura 5.7. Editor visual do ADT para facilitar a edição do arquivo de mensagens *strings.xml*.

Após clicar no botão ***Add***, uma nova janela irá se abrir, solicitando o tipo de recurso que iremos adicionar no arquivo. Em nosso caso, como iremos adicionar uma mensagem, selecionamos a opção ***String*** e clicamos no botão ***Ok***, conforme mostra a **Figura 5.8**.

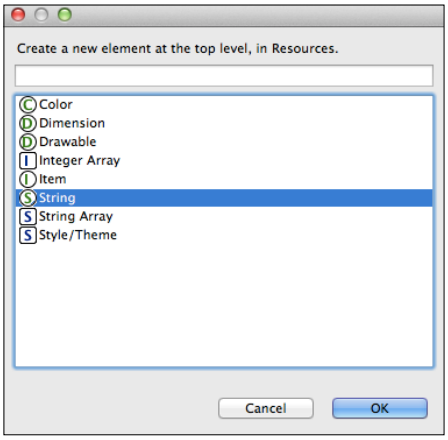


Figura 5.8. Selecionando o tipo de recurso a ser adicionado no arquivo *strings.xml*.

No próximo passo, devemos informar um nome para a mensagem, no campo ***Name***, e o conteúdo da mensagem, no campo ***Value***, conforme mostra a **Figura 5.9**.

Preenchemos o campo ***Name*** com o conteúdo: ***minha_mensagem***

Preenchemos o campo ***Value*** com o conteúdo: ***Esta é uma mensagem acessível através da classe R***.

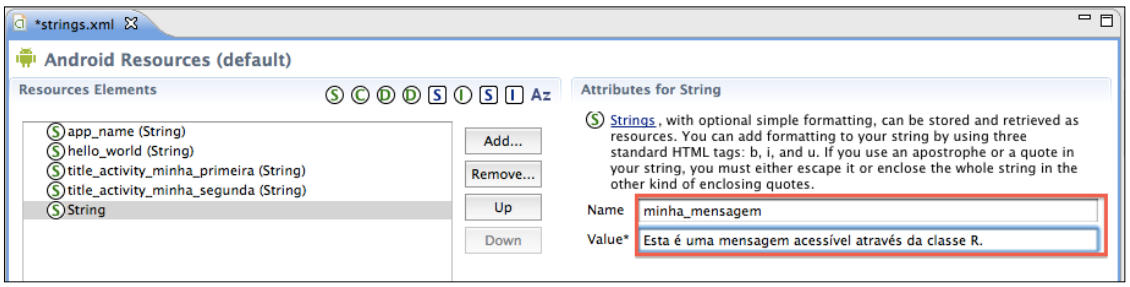


Figura 5.9. Adicionando uma nova mensagem no arquivo *strings.xml*.

Ao salvar o arquivo ***strings.xml*** com a nossa alteração, o plugin ADT do Eclipse irá automaticamente criar uma referência para esta mensagem na *classe R*, nos possibilitando acessá-la em uma classe Java, por exemplo.

Agora iremos criar uma nova Activity no projeto, chamada de ***MinhaSegundaActivity***. Dentro desta Activity iremos criar um objeto do tipo *TextView* e definí-lo como a View a ser apresentada pela Activity. Nossa Activity terá o seguinte conteúdo:

```
public class MinhaSegundaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView texto = new TextView(getApplicationContext());
        texto.setText(getString(R.string.minha_mensagem));
        setContentView(texto);
    }
}
```

Perceba a linha:

```
texto.setText(getString(R.string.minha_mensagem));
```

acessando a mensagem do String Resource através de sua referência da *classe R*.

Após criar a classe *MinhaSegundaActivity* e definir seu conteúdo, configuramos esta nova Activity para ser iniciada automaticamente ao executar o aplicativo no emulador do Android, e fazemos isto através do menu do Eclipse "**Run**" > "**Run Configurations**", conforme aprendemos anteriormente.

Ao executar a nova Activity no emulador do Android, a nossa mensagem será apresentada na tela, conforme mostra a **Figura 5.10**.

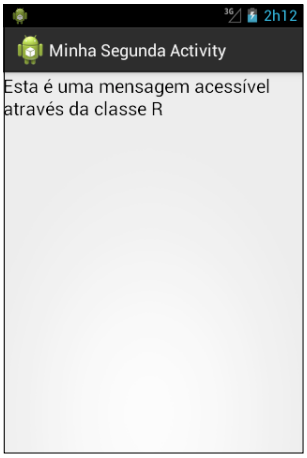


Figura 5.10. Mensagem apresentada pela Activity *MinhaSegundaActivity*, acessada através de sua referência na classe R.

5.8.2 A classe *android.R*

Como vimos, a *classe* ***R*** de um projeto do Android serve para referenciar todos os recursos (resources) de um aplicativo e é gerada automaticamente pelo plugin ADT do Eclipse. Porém, o Android também disponibiliza uma *classe R* nativa com recursos nativos que podemos utilizar em nosso aplicativo, como imagens, Strings, layouts, cores, etc. Esta classe é acessada através da referência ***android.R***.

Da mesma forma como a *classe R* gerada automaticamente em um projeto do Android, a classe nativa *android.R* permite o acesso aos seus recursos nativos, como por exemplo:

- ***android.R.color***: acesso aos Color Resources nativos;
- ***android.R.drawable***: acesso aos Drawable Resources nativos;
- ***android.R.layout***: acesso aos Layout Resources nativos;
- ***android.R.string***: acesso às String Resources nativas;
- e todos os demais tipos de recursos.

5.9 Exercício

Agora que você aprendeu a acessar um recurso através da classe R, é hora de colocar em prática:

1. No projeto ***Activities*** (criado anteriormente) adicione uma mensagem no arquivo ***res/values/strings.xml***, chamada de ***minha_mensagem***, com o seguinte conteúdo:

   Esta é uma mensagem acessível através da classe R.

2. Crie uma nova Activity, de nome *MinhaSegundaActivity*, com o seguinte conteúdo:

```
public class MinhaSegundaActivity extends Activity {

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                TextView texto = new TextView(getApplicationContext());
                texto.setText(getString(R.string.minha_mensagem));
                 setContentView(texto);
        }

}
```

3. Configure a nova Activity para ser iniciada automaticamente ao executar o projeto no emulador do Android.

5.10 Comunicação entre as Activities

Um aplicativo do Android normalmente é composto de várias Activities, cada uma gerenciando uma tela deste aplicativo. Podemos ter em um aplicativo, por exemplo, uma Activity responsável por cadastrar um objeto no banco de dados, outra responsável por listar os objetos cadastrados, outra para consultar um determinado registro, etc.

O Android permite a comunicação entre Activities, ou seja, uma Activity pode chamar outra Activity e elas podem trocar informações entre si. A comunicação entre as Activities é feita através de uma classe muito importante do Android, a *classe* ***Intent***. *[Além de possibilitar a comunicação entre as Activities, a Intent tem diversas funcionalidades. Aprenderemos mais sobre a Intent no decorrer deste curso]*

Dica

A classe ***android.content.Intent*** é considerada como o coração do Android e é basicamente uma mensagem enviada a partir de um aplicativo para o núcleo Android, solicitando a execução de alguma ação, como:

- A troca de informações entre Activities;
- Realizar uma chamada telefônica;
- Enviar uma mensagem SMS;
- Abrir uma página no Browser;
- etc.

5.10.1 Invocando uma Activity

Para chamar uma Activity através de outra Activity usamos a *classe* ***Intent***, informando ao Android que queremos iniciar uma nova Activity. Na *Intent* definimos qual a Activity que queremos abrir e, após definida, chamamos o método ***startActivity()*** passando essa *Intent* como parâmetro para iniciar a Activity desejada. Como exemplo, vamos criar uma nova Activity em nosso projeto, chamada de ***MinhaTerceiraActivity***, com o seguinte conteúdo:

```
public class MinhaTerceiraActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView texto = new TextView(getApplicationContext());
        texto.setText("Activity invocada através de outra Activity.");
        setContentView(texto);
    }
}
```

Agora vamos criar a Activity que irá invocar a *MinhaTerceiraActivity* através de um evento disparado pelo clique de um botão. Criaremos então uma nova Activity, chamada de ***MinhaQuartaActivity***, com o seguinte conteúdo:

```
public class MinhaQuartaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");

        /**
        *  Definindo o evento a ser disparado ao clicar no botão.
        *  Em nosso caso o evento irá disparar outra Activity.
        */
        botao.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
                Intent i = new Intent(getApplicationContext(), MinhaTerceiraActivity.class);
                startActivity(i);
            }
        });
        setContentView(botao);
    }
}
```

Perceba o método ***startActivity()*** na classe *MinhaQuartaActivity*, recebendo como parâmetro a *Intent* com a Activity que desejamos abrir quando o botão for clicado.

Ao executar a Activity *MinhaQuartaActivity*, no emulador do Android, irá aparecer uma tela com um único botão, conforme mostra a **Figura 5.11**. *[Como não definimos as propriedades de tamanho para o botão e ele é a única View da nossa tela, ele irá ocupar todo o espaço da tela].*

Figura 5.11. Activity *MinhaQuartaActivity* em execução.

Ao clicar no botão, a Activity *MinhaTerceiraActivity* será invocada e sua View TextView será renderizada na tela, conforme mostra a **Figura 5.12**.

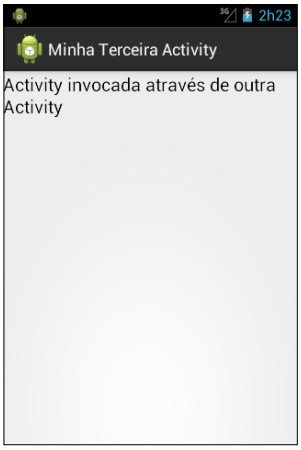


Figura 5.12. Activity *MinhaTerceiraActivity* em execução, invocada a partir da *MinhaQuartaActivity*.

5.10.2 Retornando informações

Quando estamos desenvolvendo um aplicativo para o Android, muitas vezes precisamos enviar informações de uma Activity para outra. Um grande exemplo disto é quando uma tela do nosso aplicativo possui um botão para localizar um determinado registro no banco de dados, chamando uma nova Activity para realizar esta tarefa. Após realizar a consulta no banco de dados, é comum precisarmos retornar este registro (ou o seu id) para a Activity que a chamou, para que esta possa realizar alguma ação sobre o registro localizado.

Vimos que o método ***startActivity()*** é usado para iniciar uma nova Activity, porém ele não permite que a Activity invocada retorne informações para a Activity que a chamou. Para que isto seja possível, as Activities dispõem de um método chamado ***startActivityForResult()***, que possibilita que uma Activity retorne algum tipo de informação para a Acitivity que a invocou.

A única diferença entre a chamada do método ***startActivity()*** e ***startActivityForResult()*** é que este segundo recebe um parâmetro a mais, chamado de ***Request Code***, que serve apenas para identificar se o resultado recebido vem realmente da Activity que foi chamada. Outra diferença é que a Activity que chamar o método ***startActivityForResult()*** deve subscrever o método ***onActivityResult()***, que irá receber e tratar as informações recebidas da Activity que foi chamada.

Para ver na prática como funciona o retorno de informações, vamos criar uma nova Activity, em nosso projeto ***Activities***, chamada de ***MinhaQuintaActivity***, com o seguinte conteúdo:

```
public class MinhaQuintaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");

        /**
         *  Definindo o evento a ser disparado ao clicar no botão.
         *  Em nosso caso, o evento irá disparar outra Activity.
         */
        botao.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                    // Definindo na Intent a Activity que será chamada ao clicar no botão
                    Intent i = new Intent(getApplicationContext(), MinhaSextaActivity.class);

                    // Inicia uma Activity definindo o valor do Request Code para 1.
                    startActivityForResult(i, 1);

                }
            });

        setContentView(botao);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Recupera o parâmetro recebido
        String parametroRecebido = data.getExtras().getString("ParametroRetorno");

        /**
         *  Mostra o conteúdo do parâmetro recebido na saída de Log,
         *  para ser visualizado no LogCat.
         */
        Log.i("ParametroRetorno", parametroRecebido);

    }

}
```

Na Activity *MinhaQuintaActivity* perceba o método ***startActivityForResult()*** passando como parâmetro a *Intent*, definindo a Activity que será chamada, e o *Result Code* com valor 1, que será usado para identificar se o resultado realmente vem da Activity que foi chamada.

Como esperamos receber um retorno da Activity que iremos chamar, tivemos que implementar o método ***onActivityResult()*** para tratar as informações recebidas da Activity que iremos invocar. Para tornar a implementação

simples, nós apenas iremos mostrar no console de Log uma String com a mensagem que será recebida da Activity que iremos chamar.

Agora precisamos criar a Activity que irá retornar um resultado para a *MinhaQuintaActivity*. Iremos então criar uma nova Activity e chamá-la de ***MinhaSextaActivity***,com o seguinte conteúdo:

```
public class MinhaSextaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Retornar informação");

        /**
         * Definindo o evento a ser disparado ao clicar no botão.
         * Em nosso caso, o evento irá finalizar a Activity atual
         * retornando informações para a Activity anterior.
         */
        botao.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                    // Define os dados que serão retornados para a Activity chamadora
                    Intent i = getIntent();
                    i.putExtra("ParametroRetorno", "Conteúdo do parâmetro de retorno.");

                    // Define o Result Code para ser enviado para a Activity chamadora
                    setResult(1, i);
                    // Finaliza a Activity atual
                    finish();
                    }
        });

            /**
         * Definindo a View a ser apresentada na tela.
         * Em nosso caso, será apresentado apenas um Button.
         */
        setContentView(botao);
    }

}
```

Na Activity *MinhaSextaActivity* que criamos, perceba que definimos como View padrão um botão que irá retornar para a Activity anterior ao ser clicado. No evento que será disparado ao clicar no botão desta Activity, definimos uma Intent e nela injetamos uma String que será retornada para a Activity anterior, a String "Conteúdo do parâmetro de retorno.", através do método ***putExtra()***.

Perceba também, na Activity *MinhaSextaActivity*, que chamamos o método ***finish()*** no evento do botão. O método *finish()* finaliza a Activity atual, retornando para a Activity anterior.

Dica

O método ***putExtra()*** de uma Intent é usado para definir os parâmetros que serão enviados de uma Activity para outra. Através deste método podemos enviar vários tipos de dados, como Strings, tipos primitivos, etc.

Ao executar o aplicativo no emulador do Android e clicar no botão "***Retornar informação***" da Activity *MinhaSextaActivity*, a String retornada como parâmetro será mostrada na view LogCat no Eclipse, conforme mostra a **Figura 5.13**.

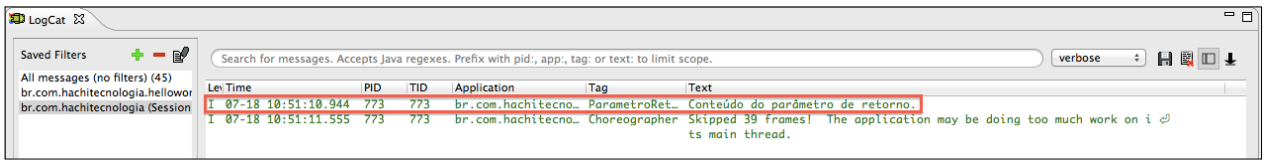


Figura 5.13. Conteúdo da String retornada como parâmetro sendo mostrado na View LogCat no Eclipse.

5.10.3 Passando parâmetros entre Activities

No exemplo do tópico anterior nós passamos uma String como parâmetro ao retornar informações para uma Activity. No entanto, há casos onde precisamos passar parâmetros também ao invocar uma Activity, e o Android nos permite fazer isto através do objeto ***Bundle***.

Para mostrar na prática como é feito a passagem de parâmetros de uma Activity para outra no momento da invocação, criaremos uma Activity chamada ***MinhaSetimaActivity***, com o seguinte conteúdo:

```
public class MinhaSetimaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");
        botao.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                        Intent i = new Intent(getApplicationContext(), MinhaOitavaActivity.class);

                        /**
                        *  Injetando uma String no objeto Bundle
                        *  que será passado como parâmetro para outra Activity.
                        */
                        Bundle bundle = new Bundle();
                        bundle.putString("MeuParametro", "Conteúdo do meu parâmetro!");

                        // Injetando o objeto Bundle na Intent
                        i.putExtras(bundle);

                        // Invocando outra Activity
                        startActivity(i);

                }

            });

        // Definindo a View que será apresentada na tela
        setContentView(botao);

    }

}
```

Na Activity *MinhaSetimaActivity* perceba que injetamos uma String no objeto *Bundle* que será enviado como parâmetro para outra Activity. Agora precisamos criar esta outra Activity que irá receber a String e tratá-la. Para isto, criamos uma nova Activity chamada ***MinhaOitavaActivity***, com o seguinte conteúdo:

```
public class MinhaOitavaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Obtendo o objeto Intent para recuperar os parâmetros recebidos
        Intent i = getIntent();

        // Recuperando o parametro recebido da Activity anterior
        String parametroRecebido = i.getExtras().getString("MeuParametro");

        // Criando a View TextView que irá mostar na tela a String recebida
        TextView texto = new TextView(getApplicationContext());
        texto.setText(parametroRecebido);

        // Definindo a View que será apresentada na tela
        setContentView(texto);
    }
}
```

Perceba que a Activity *MinhaOitavaActivity* recupera o objeto *Intent*, através do método ***getIntent()***, e dele obtém a String recebida como parâmetro para mostrar seu conteúdo em uma View TextView na tela do dispositivo, conforme mostra a **Figura 5.14**.

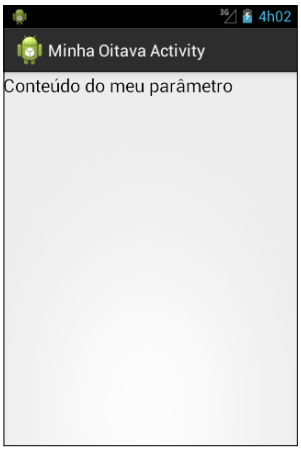


Figura 5.14. Conteúdo da String passada como parâmetro sendo apresentado em uma TextView.

5.11 Exercício

Agora que você aprendeu a realizar a comunicação entre Activities, é hora de colocar em prática.

Neste exercício você irá criar uma Activity que irá invocar outra Activity através do clique de um botão.

1. Crie, no projeto ***Activities***, uma nova Activity chamada ***MinhaTerceiraActivity***, com o seguinte conteúdo:

```
public class MinhaTerceiraActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            TextView texto = new TextView(getApplicationContext());
            texto.setText("Activity invocada através de outra Activity.");
            setContentView(texto);
    }
```

```
}
```

2. Agora você deve criar uma nova Activity com o botão que irá chamar a Activity anterior. Para isto, crie outra nova Activity chamada ***MinhaQuartaActivity***, com o seguinte conteúdo:

```
public class MinhaQuartaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");

        /**
        *  Definindo o evento a ser disparado ao clicar no botão.
        *  Em nosso caso o evento irá disparar outra Activity.
        */
        botao.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
                Intent i = new Intent(getApplicationContext(), MinhaTerceiraActivity.class);
                startActivity(i);
            }
        });
        setContentView(botao);
    }
}
```

3. Configure o projeto para que a Activity ***MinhaQuartaActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em "**Run**" > "**Run Configurations**".

4. Teste o aplicativo no emulador do Android.

5.12 Exercício

Neste exercício você deverá cria uma Activity que irá invocar outra Activity através do clique de um botão e, na Activity invocada, criar um botão que irá retornar uma String para a Activity que a chamou e imprimir o conteúdo desta String no LogCat.

1. Crie uma nova Activity, chamada ***MinhaQuintaActivity***, com o seguinte conteúdo:

```
public class MinhaQuintaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");

        /**
         *  Definindo o evento a ser disparado ao clicar no botão.
         *  Em nosso caso, o evento irá disparar outra Activity.
         */
        botao.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {

                // Definindo na Intent a Activity que será chamada ao clicar no botão
                Intent i = new Intent(getApplicationContext(), MinhaSextaActivity.class);
```

```
                        // Inicia uma Activity definindo o valor do Request Code para 1.
                        startActivityForResult(i, 1);
                    }
                });

        setContentView(botao);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Recupera o parâmetro recebido
        String parametroRecebido = data.getExtras().getString("ParametroRetorno");
        /**
         * Mostra o conteúdo do parâmetro recebido na saída de Log,
         * para ser visualizado no LogCat.
         */
        Log.i("ParametroRetorno", parametroRecebido);
    }
}
```

2. Agora você deve criar a Activity que deverá retornar informação para a String anterior. Crie uma nova Activity, chamada ***MinhaSextaActivity***, com o seguinte conteúdo:

```
public class MinhaSextaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Retornar informação");

        /**
         * Definindo o evento a ser disparado ao clicar no botão.
         * Em nosso caso, o evento irá finalizar a Activity atual
         * retornando informações para a Activity anterior.
         */
        botao.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                    // Define os dados que serão retornados para a Activity chamadora
                    Intent i = getIntent();
                    i.putExtra("ParametroRetorno", "Conteúdo do parâmetro de retorno.");

                    // Define o Result Code para ser enviado para a Activity chamadora
                    setResult(1, i);
                    // Finaliza a Activity atual
                    finish();
                    }
                });

            /**
         * Definindo a View a ser apresentada na tela.
         * Em nosso caso, será apresentado apenas um Button.
         */
        setContentView(botao);

    }
}
```

3. Configure o projeto para que a Activity ***MinhaQuintaActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em "**Run**" > "**Run Configurations**".

4. Teste o aplicativo no emulador do Android.

5.13 Exercício

Neste exercício você irá criar uma Activity que irá invocar outra Activity, passando uma String como parâmetro.

1. Crie uma nova Activity, chamada ***MinhaSetimaActivity***, com o seguinte conteúdo:

```
public class MinhaSetimaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button botao = new Button(getApplicationContext());
        botao.setText("Abrir outra Activity");
        botao.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                        Intent i = new Intent(getApplicationContext(), MinhaOitavaActivity.class);

                        /**
                        *  Injetando uma String no objeto Bundle
                        *  que será passado como parâmetro para outra Activity.
                        */
                        Bundle bundle = new Bundle();
                        bundle.putString("MeuParametro", "Conteúdo do meu parâmetro!");

                        // Injetando o objeto Bundle na Intent
                        i.putExtras(bundle);

                        // Invocando outra Activity
                        startActivity(i);
                }
            });

        // Definindo a View que será apresentada na tela
        setContentView(botao);
    }
}
```

2. Agora crie uma nova Activity que irá receber a String passada como parâmetro pela Activity anterior e mostre o conteúdo desta String em uma TextView, na tela do dispositivo. Para isto, crie uma nova Activity, chamada ***MinhaOitavaActivity***, com o seguinte conteúdo:

```
public class MinhaOitavaActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Obtendo o objeto Intent para recuperar os parâmetros recebidos
        Intent i = getIntent();

        // Recuperando o parametro recebido da Activity anterior
        String parametroRecebido = i.getExtras().getString("MeuParametro");
```

```
        // Criando a View TextView que irá mostar na tela a String recebida
        TextView texto = new TextView(getApplicationContext());
        texto.setText(parametroRecebido);

        // Definindo a View que será apresentada na tela
        setContentView(texto);
    }
}
```

3. Configure o projeto para que a Activity ***MinhaSetimaActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em "**Run**" > "**Run Configurations**".

4. Teste o aplicativo no emulador do Android.

5.14 Ciclo de vida das Activities

Uma característica muito importante que devemos saber sobre as Activities é como funciona o seu ciclo de vida, pois conhecendo o ciclo de vida das Activities é que vamos garantir que nossos aplicativos funcionem da maneira correta.

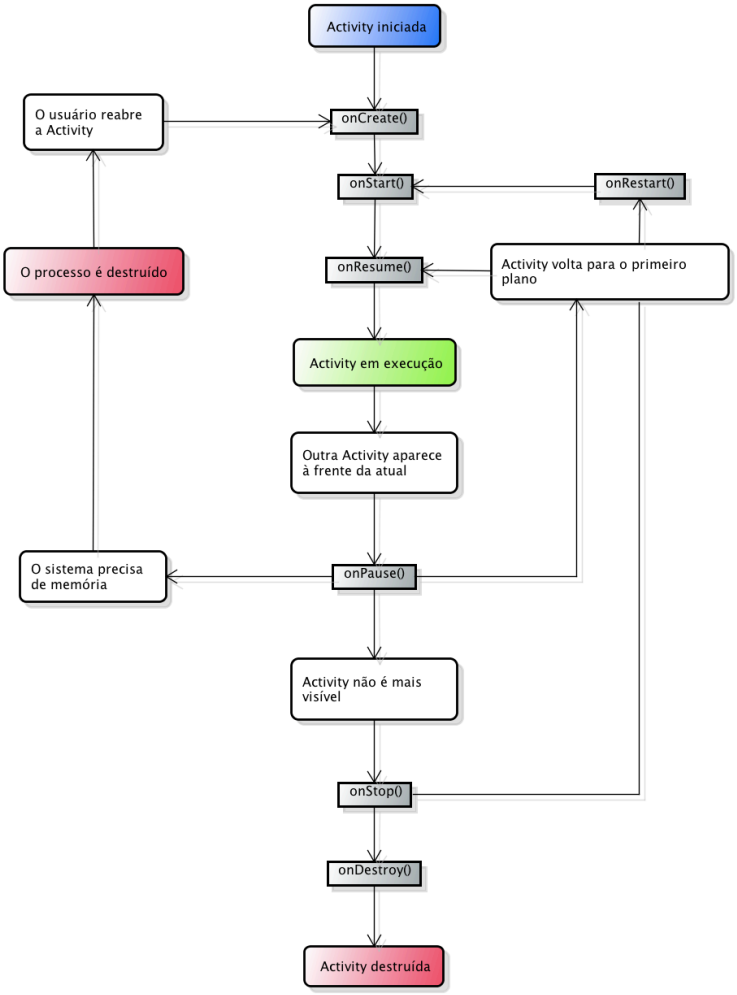


Figura 5.15. Ciclo de vida das Activities.

Conforme mostra a **Figura 5.15**, o ciclo de vida das Activities tem um esquema lógico e bem definido. No Android, a execução das Activities funciona como uma pilha, onde a Activity atual (que está interagindo com o usuário) está sempre no topo da pilha.

Dica

Saber como funciona o ciclo de vida das Activities é fundamental quando se está desenvolvendo um aplicativo para o Android. Imagine, por exemplo, que você tenha desenvolvido um aplicativo que usa vários recursos do Android (como o acesso a banco de dados, acesso à Internet ou ao navegador GPS do dispositivo) e em um determinado momento que o usuário estiver executando seu aplicativo, uma chamada telefônica é recebida. Se você não tomou os cuidados de liberar os recursos que seu aplicativo está utilizando (desconectar do banco de dados, parar de trafegar dados na rede, desconectar o acesso ao GPS) no momento que uma outra Activity entra à frente, seu aplicativo continuará consumindo recursos desnecessariamente, aumentando o consumo da bateria e diminuindo o desempenho do dispositivo. Isso com certeza irá deixar o usuário bastante furioso, causando uma imagem ruim ao seu produto.

Ou imagine que você tenha desenvolvido um jogo para o Android e não tomou o cuidado de salvar as informações do estado do jogo quando uma chamada telefônica é recebida. Quando a chamada terminar e a Activity do seu jogo voltar à frente (ao topo da pilha) as informações do jogo estariam perdidas e o jogador terá que começar o jogo novamente. Isso deixará o jogador bastante furioso e causará uma imagem ruim ao seu produto.

Para controlar o ciclo de vida, a Activity possui uma série de eventos que executam ações em cada passo do ciclo. Vamos conhecer esses eventos:

- **onCreate()**: evento chamado quando uma Activity é criada;
- **onStart()**: evento chamado quando a Activity fica visível ao usuário;
- **onResume()**: evento chamado quando a Activity começa a interagir com o usuário;
- **onPause()**: evento chamado quando uma outra Activity entra à frente da Activity atual;
- **onStop()**: evento chamado quando a Activity não está mais visível;
- **onDestroy()**: evento chamado antes do sistema destruir a Activity, para liberar memória;
- **onRestart()**: evento chamado quando uma Activity interrompida volta a ser usada.

Dica

Um detalhe importante no ciclo de vida das Activities é que, quando uma Activity está pausada (***onPause()***) ou parada (***onStop()***), se o sistema precisar liberar recursos por falta de memória, o Android irá invocar o evento ***onDestroy()*** e a Activity será destruída.

Para entender melhor o ciclo de vida das Activities, vamos visualizar como ele funciona na prática. Para isto, iremos tomar como exemplo um usuário interagindo com uma Activity no momento em que recebe uma chamada telefônica, conforme demonstra a **Figura 5.16**.

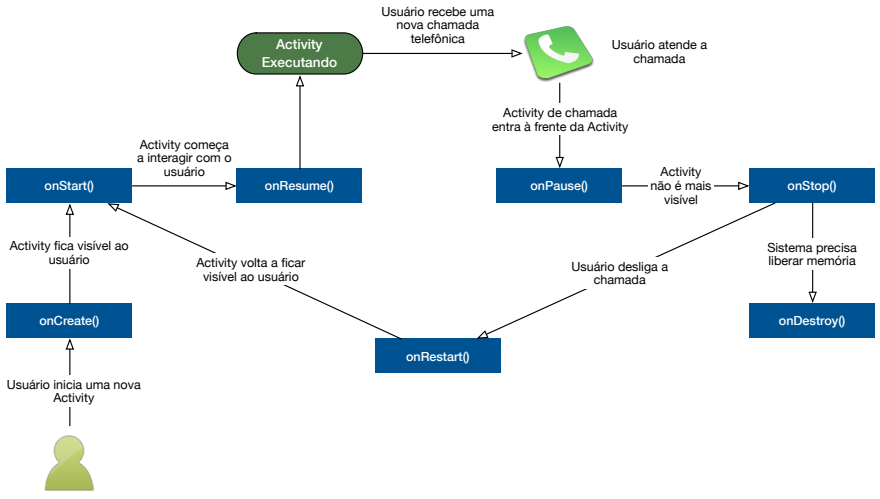


Figura 5.16. Usuário interagindo com uma Activity no momento em que recebe uma chamada telefônica.

Para colocar nosso exemplo em prática, iremos criar uma nova Activity, chamada ***CicloDeVidaActivity***, subscrevendo os métodos de callback de cada etapa do seu ciclo de vida. Nossa Activity terá o seguinte conteúdo:

```
public class CicloDeVidaActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);
		Log.i("CicloDeVida", "Método onCreate() chamado");

		// Texto a ser mostrado na tela
		TextView texto = new TextView(getApplicationContext());
		texto.setText("Um texto qualquer");

		// Define a TextView que criamos como View padrão
		setContentView(texto);

	}

	@Override
	protected void onStart() {
		super.onStart();
		Log.i("CicloDeVida", "Método onStart() chamado");
	}

	@Override
	protected void onResume() {
		super.onResume();
		Log.i("CicloDeVida", "Método onResume() chamado");
	}

	@Override
	protected void onPause() {
		super.onPause();
		Log.i("CicloDeVida", "Método onPause() chamado");
	}

	@Override
	protected void onStop() {
		super.onStop();
		Log.i("CicloDeVida", "Método onStop() chamado");
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();
		Log.i("CicloDeVida", "Método onDestroy() chamado");
	}

	@Override
	protected void onRestart() {
		super.onRestart();
		Log.i("CicloDeVida", "Método onRestart() chamado");
	}

}
```

Nossa Activity irá apenas mostrar no LogCat uma mensagem ao passar por cada etapa do seu ciclo de vida. Ao executá-la, a View LogCat irá mostrar as mensagens disparadas nos eventos **onCreate()**, **onStart()** e **onResume()**, conforme mostra a **Figura 5.17**.

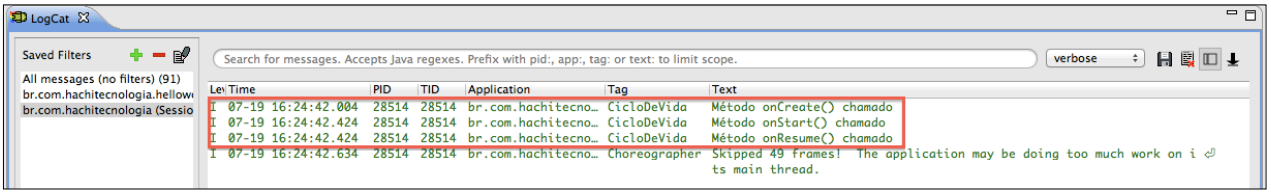


Figura 5.17. View LogCat do ADT mostrando as mensagens disparadas pela Activity *CicloDeVidaActivity*.

Enquanto a Activity estiver em execução, iremos simular uma chamada telefônica para o emulador do Android, usando a perspectiva DDMS. *[Lembre-se que você aprendeu a usar a perspectiva DDMS no **Capítulo 4** deste curso]*

Na perspectiva DDMS, para simular uma chamada telefônica, abrimos a aba ***Emulator Control***, preenchemos o campo ***Incoming Number*** com um número qualquer, selecionamos a opção ***Voice***, indicamos que queremos simular uma chamada de voz, e clicamos no botão ***Call***, conforme mostra a **Figura 5.18**.

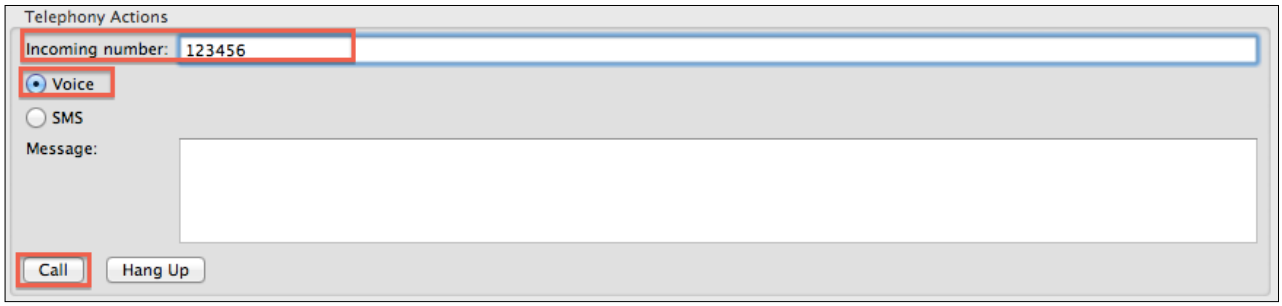


Figura 5.18. Simulando uma chamada telefônica para o emulador do Android, através da perspectiva DDMS do plugin ADT.

Ao clicar em ***Call***, o emulador irá abrir a Activity de chamada telefônica, informando o recebimento de uma nova chamada, deixando agora esta Activity no topo da pilha. Quando a Activity de chamada é disparada e a chamada é atendida no emulador do Android, os métodos ***onPause()*** e ***onStop()*** da Activity *CicloDeVidaActivity* são invocados, disparando suas respectivas mensagens no LogCat, conforme mostra a **Figura 5.19**.

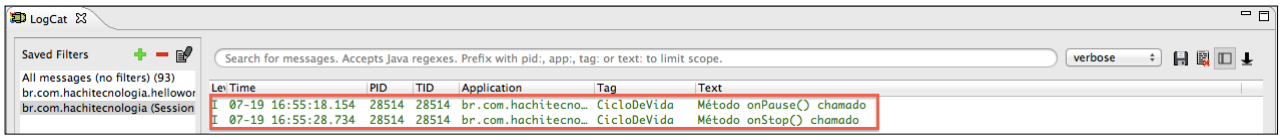


Figura 5.19. View LogCat do ADT mostrando as mensagens disparadas pela Activity *CicloDeVidaActivity*.

Ao finalizarmos a chamada no emulador do Android e voltarmos para a Activity *CicloDeVidaActivity*, os métodos **onRestart()**, **onStart()** e **onResume()** são disparados, imprimindo suas respectivas mensagens no LogCat, conforme mostra a **Figura 5.20**. Nossa Activity *CicloDeVidaActivity* agora está em execução e volta para o topo da pilha.

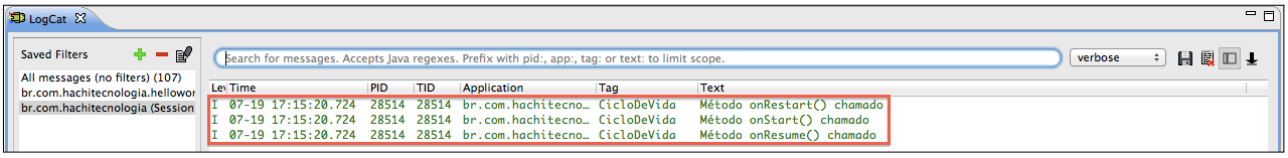


Figura 5.20. View LogCat do ADT mostrando as mensagens disparadas pela Activity *CicloDeVidaActivity*.

Agora que nossa Activity está novamente em execução, se clicarmos no botão voltar do emulador do Android, para finalizá-la, os métodos **onPause()**, **onStop()** e **onDestroy()** são disparados, destruindo nossa Activity, e suas mensagens são disparadas no LogCat, conforme mostra a **Figura 5.21**.

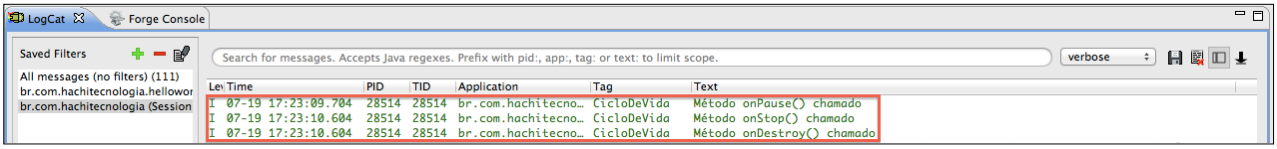


Figura 5.21. View LogCat do ADT mostrando as mensagens disparadas pela Activity *CicloDeVidaActivity*.

5.15 Exercício

Agora que você aprendeu sobre o ciclo de vida das Activities, é hora de colocar em prática.

1. Crie uma nova Activity no projeto ***Activities***, chamada ***CicloDeVidaActivity***, com o seguinte conteúdo:

```
public class CicloDeVidaActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i("CicloDeVida", "Método onCreate() chamado");

        // Texto a ser mostrado na tela
        TextView texto = new TextView(getApplicationContext());
        texto.setText("Um texto qualquer");

        // Define a TextView que criamos como View padrão
        setContentView(texto);
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.i("CicloDeVida", "Método onStart() chamado");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.i("CicloDeVida", "Método onResume() chamado");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.i("CicloDeVida", "Método onPause() chamado");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.i("CicloDeVida", "Método onStop() chamado");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i("CicloDeVida", "Método onDestroy() chamado");
    }
```

```
        @Override
        protected void onRestart() {
                super.onRestart();
                Log.i("CicloDeVida", "Método onRestart() chamado");
        }

}
```

2. Configure o projeto para que a Activity *CicloDeVidaActivity* seja executada ao iniciar o aplicativo no emulador do Android, em "**Run**" > "**Run Configurations**".

3. Execute a Activity no emulador do Android.

4. Abra a View LogCat do plugin ADT no Eclipse e perceba as mensagens impressas pelos métodos **onCreate()**, **onStart()** e **onResume()**.

5. Abra a perspectiva DDMS do plugin ADT, simule uma chamada para o emulador do Android e atenda a chamada no emulador.

6. Abra a View LogCat do plugin ADT no Eclipse e perceba as mensagens impressas pelos métodos **onPause()** e **onStop()**.

7. Finalize a chamada no emulador do Android.

8. Abra a View LogCat do plugin ADT no Eclipse e perceba as mensagens impressas pelos métodos **onRestart(), onStart()** e **onResume()**.

9. Finalize a Activity clicando no botão voltar do emulador do Android.

10. Abra a View LogCat do plugin ADT no Eclipse e perceba as mensagens impressas pelos métodos **onPause()**, **onStop()** e **onDestroy()**.

5.16 Criando uma Activity manualmente

Até então, as Activities que criamos foram geradas automaticamente pelo assistente do plugin ADT, que, além de gerar a Activity, criou automaticamente seu arquivo de Layout e definiu sua configuração no arquivo *AndroidManifest.xml*.

Se desejarmos criar uma Activity manualmente, sem que o assistente gere tudo automaticamente, basta seguirmos os seguintes passos:

1. No pacote desejado, devemos criar uma classe Java que estenda de ***android.app.Activity***, conforme exemplo abaixo:

```
public class ActivityCriadaManualmente extends Activity {

        ...

}
```

2. Na Activity criada, devemos subscrever o método ***onCreate()*** e nele fazer a chamada para o método ***setcontentView()***, passando como parâmetro a View a ser apresentada na tela, conforme exemplo abaixo:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView texto = new TextView(getApplicationContext());
        texto.setText("Olá, Android!");

        setContentView(texto);
}
```

3. Abrimos o arquivo ***AndroidManifest.xml*** e adicionamos a Activity criada, para que o Android possa executá-la. Fazemos isto da seguinte forma:

• No editor gráfico do arquivo *AndroidManifest.xml*, vamos até a aba "***Application***" e em "***Application Nodes***" clicamos em ***Add***, conforme mostra **Figura 5.22**.

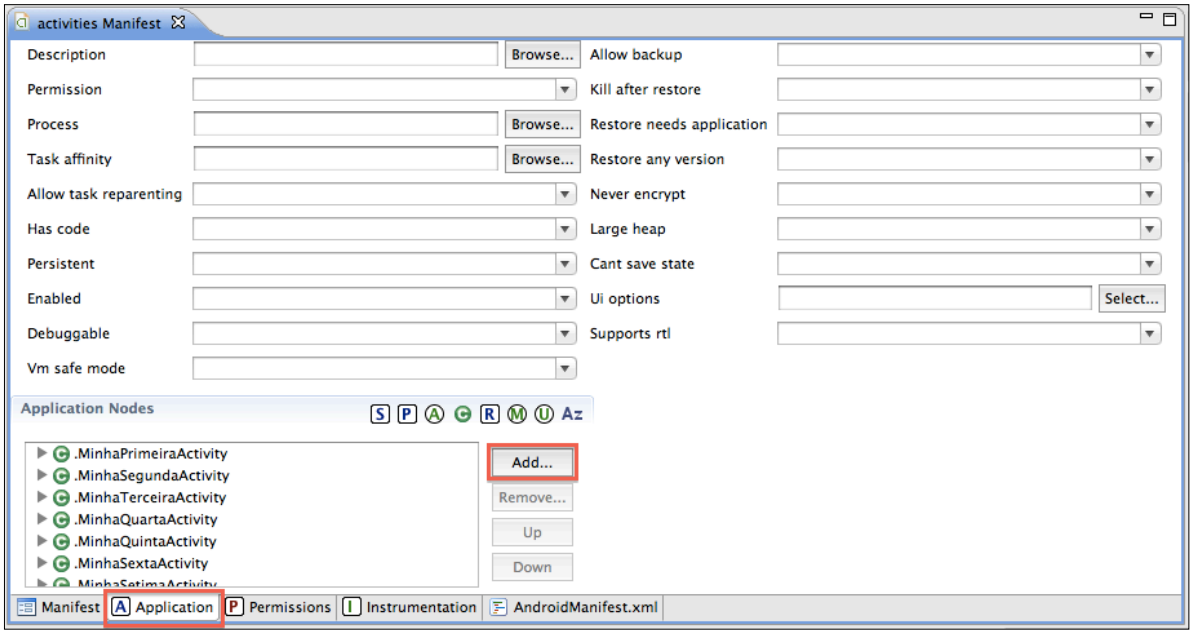


Figura 5.22. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml.*

• Na janela que se abre, selecionamos ***Activity*** e clicamos em ***Ok***, conforme mostra **Figura 5.23**.

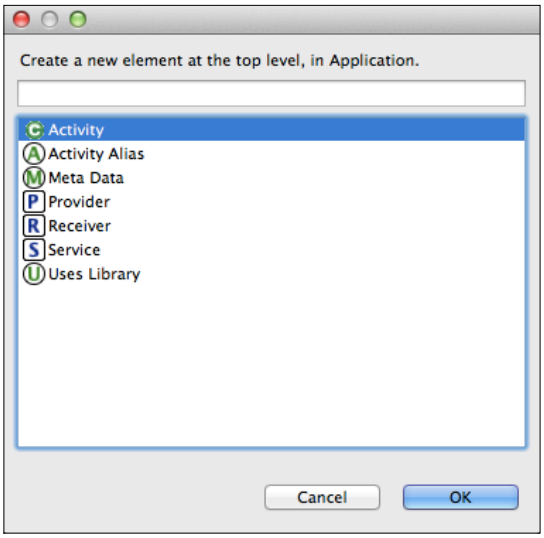


Figura 5.23. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml* - Configurando uma nova Activity.

• Na nova configuração de Activity criada, clicamos no botão ***Browse***, do campo ***Name***, para definir a Activity que queremos configurar, conforme mostra **Figura 5.24**.

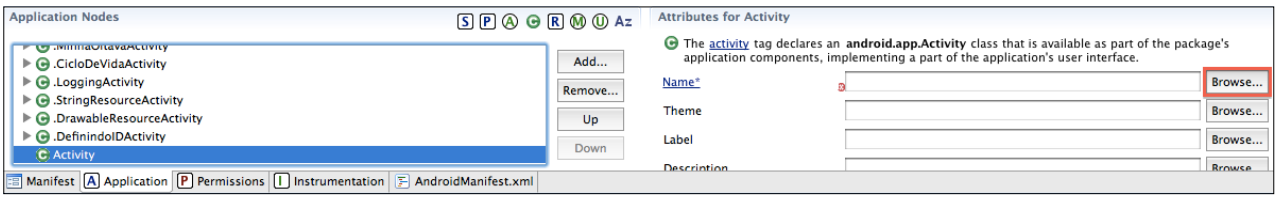


Figura 5.24. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml* - Configurando uma nova Activity.

• Na janela que se abre, selecionamos a Activity que criamos e clicamos em ***Ok***, conforme mostra a **Figura 5.25**.

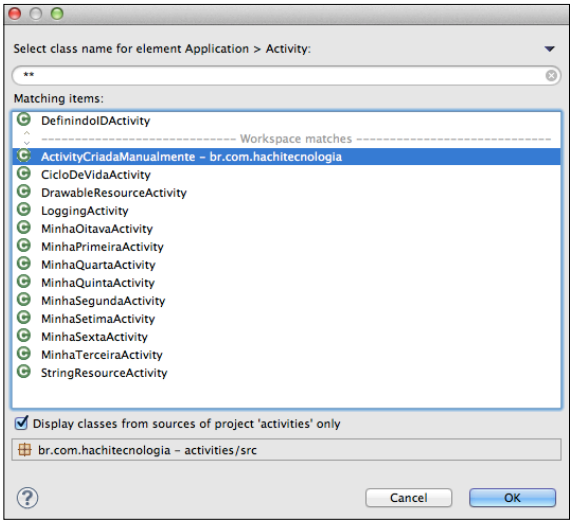


Figura 5.25. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml* - Configurando uma nova Activity.

• Nossa Activity já está adicionada ao arquivo *AndroidManifest.xml*.

• Se quisermos definir esta nova Activity como a principal, que será executada ao iniciar o aplicativo, devemos configurá-la no *AndroidManifest.xml*. Para isto, ainda no editor gráfico do arquivo *AndroidManifest.xml*, selecionamos a Activity que adicionamos e clicamos no botão ***Add***, conforme mostra a **Figura 5.26**.

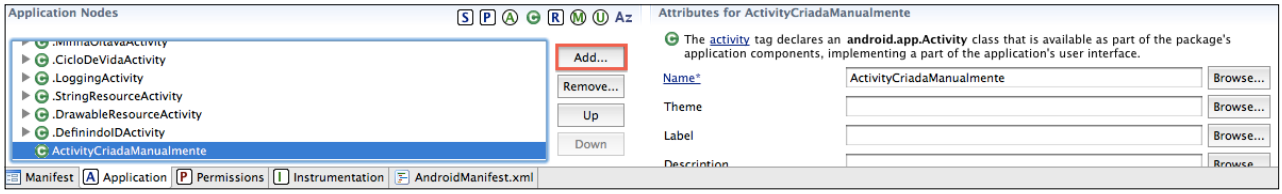


Figura 5.26. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Na janela que se abre, selecionamos a opção "***Intent Filter***", conforme mostra a **Figura 5.27**, e clicamos em ***Ok***.

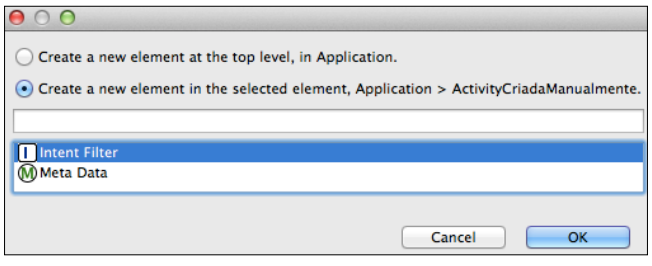


Figura 5.27. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Após adicionarmos uma nova ***Intent Filter***, a deixamos selecionada e clicamos novamente em ***Add***, conforme mostra a **Figura 5.28**.

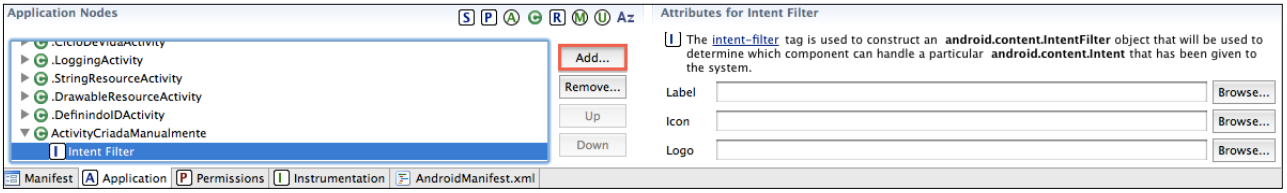


Figura 5.28. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Na janela que se abre, selecionamos a opção "***Action***", conforme mostra a **Figura 5.29**.

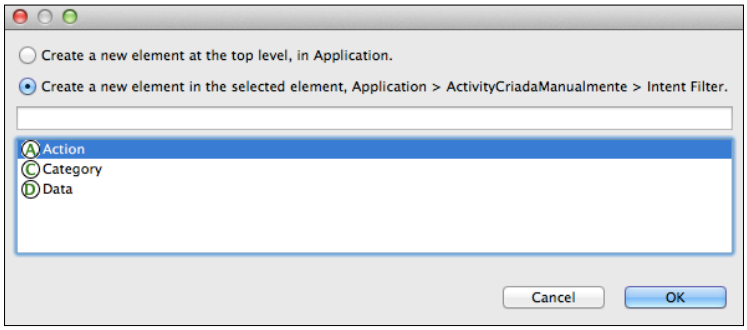


Figura 5.29. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Selecionamos a nova "***Action***" e no atributo "***Name***" selecionamos a opção ***android.intent.action.MAIN***, conforme mostra a **Figura 5.30**.

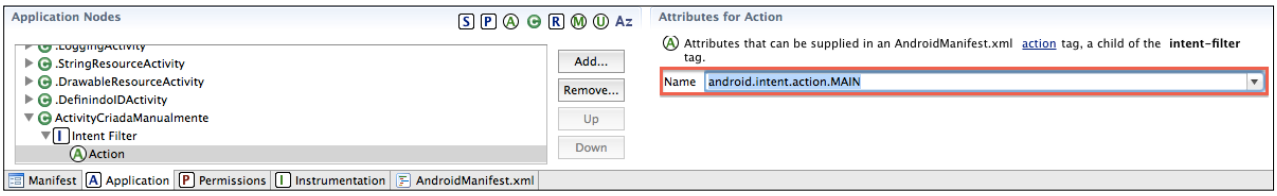


Figura 5.30. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Após esta configuração, nossa Activity poderá ser executada quando iniciarmos nosso aplicativo.

• Se precisarmos mostrar nossa nova Activity no menu de aplicativos do Android (*Launcher*), precisamos configurar isto também no *AndroidManifest.xml*. Para isto, ainda no editor gráfico do arquivo *AndroidManifest.xml*, selecionamos o "***Intent Filter***" na nova Activity que criamos e clicamos no botão ***Add***, conforme mostra a **Figura 5.31**.

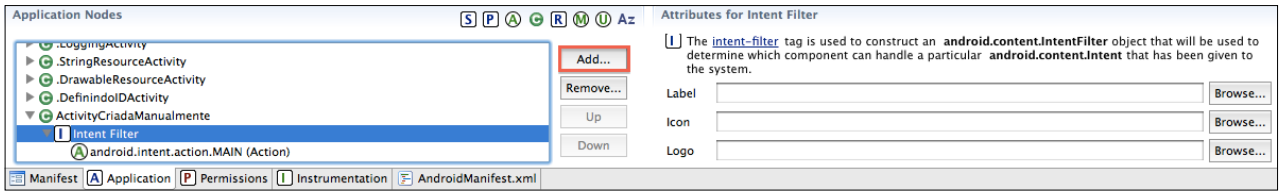


Figura 5.31. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Na janela que se abre, selecionamos a opção "***Category***", conforme mostra a **Figura 5.32**.

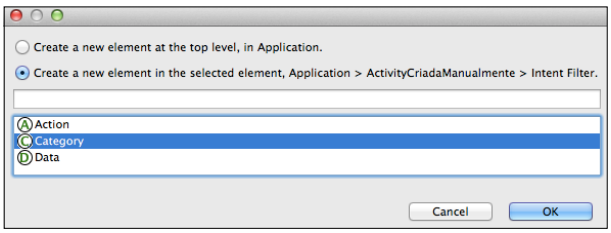


Figura 5.32. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Selecionamos a nova "***Category***" que criamos e, em seus atributos, no campo "***Name***", selecionamos a opção ***android.intent.category.Launcher***, conforme mostra a **Figura 5.33**.

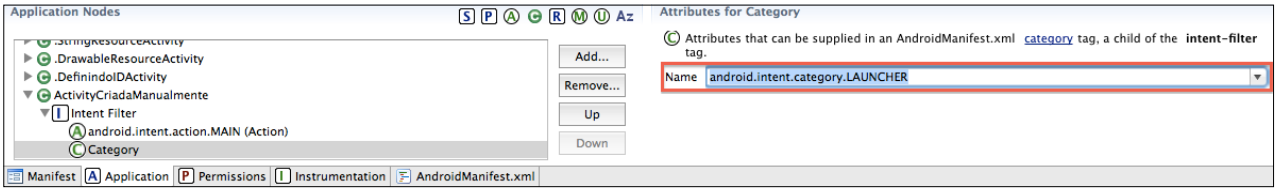


Figura 5.33. Editor visual do plugin ADT para o arquivo *AndroidManifest.xml*.

• Após executar estes passos, basta salvar o arquivo *AndroidManifest.xml*.

4. Agora que criamos e configuramos nossa nova Activity, basta definí-la, em ***Run Configurations***, para que seja executada ao iniciarmos o aplicativo no emulador do Android.

5.17 Resources

No **Capítulo 3**, quando explicamos a estrutura de um projeto do Android, mencionamos sobre o diretório "***res***". Como foi dito, é neste diretório onde encontram-se os resources (recursos) do Android, como textos, imagens, arquivos de layout e sons. Iremos agora conhecer um pouco melhor sobre os resources do Android e como usá-los.

O diretório ***res*** tem a seguinte estrutura:

• /res/***drawable***: onde ficam os arquivos de imagem do nosso aplicativo (png, jpeg ou gif);

• /res/***layout***: onde ficam os arquivos XML com o layout das telas do aplicativo;

• /res/***values***: onde ficam os arquivos XML com as mensagens do aplicativo, as telas de Layout de Preferências, etc.

Iremos aprender como trabalhar melhor com cada um desses recursos.

Dica

Um detalhe muito importante é que os arquivos de Resources não podem conter espaço em seu nome. Exemplo:

- ***/res/drawable/imagem teste.png*** -> nome inválido, pois contém espaço
- ***/res/drawable/imagem_teste.png*** -> nome válido

5.17.1 String Resources

Como mencionamos anteriormente, no diretório ***/res/values*** é onde ficam as mensagens do nosso aplicativo, as chamadas ***String Resources***. Guardar todas as mansagens do aplicativo neste diretório é uma boa prática, pois desta forma centralizamos todas as mensagens e ainda nos beneficiamos do suporte à internacionalização. Neste diretório, todas as mensagens são gravadas em arquivos XML.

Para explicar as String Resources, vamos pegar, como exemplo, o arquivo padrão de Strings ***/res/values/strings.xml***, com o seguinte conteúdo:

```
<resources>
    <string name="app_name">Activities</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

Neste arquivo de exemplo existem duas Strings, com os nomes: ***app_name*** e ***hello_world***. As Strings armazenadas neste arquivo devem possuir obrigatoriamente um nome, preenchido no campo *name*, para que ela possa ser mapeada na *classe* ***R*** para ser utilizada em nosso projeto. *[Lembre-se que aprendemos sobre a classe R anteriormente, neste capítulo]*

Sempre que adicionamos uma String em um arquivo XML do diretório ***/res/values*** e salvamos este arquivo, a nova String é automaticamente mapeada na *classe* ***R*** pelo plugin ADT do Eclipse, nos permitindo usá-la em qualquer lugar do aplicativo através de sua referência.

Dica

O arquivo de resources ***/res/values/strings.xml*** é apenas um arquivo padrão para armazenamento das mensagens (Strings) do nosso aplicativo, porém podemos criar nossos próprios arquivos XML no diretório ***/res/values***, e darmos a ele o nome que quisermos, que o Android irá procurar suas Strings neste arquivo.

Existem duas formas de usarmos as String Resources em nosso aplicativo: na Activity ou em um arquivo XML de layout.

• **Usando uma String Resource na Activity**:

Para acessar uma String Resource através de uma Activity, basta chamá-la através de sua referência mapeada na *classe* ***R***, usando a seguinte sintaxe:

```
R.string.nome_da_string_resource
```

Onde:

nome_da_string_resource: é o nome que definimos para a String no arquivo XML, no atributo ***name***.

Dica

Caso você queira obter o conteúdo de uma String Resource na Activity, basta usar o método ***getString()*** passando como parâmetro sua referência mapeada na *classe* ***R***.

Como exemplo, criaremos uma nova String no arquivo de String Resources ***/res/values/strings.xml*** com o nome "***minha_string***", ficando desta forma:

```
<string name="minha_string">Testando o uso de uma String Resource</string>
```

Dica

Para cadastrar uma nova String, em um arquivo XML de String Resources, você pode inserí-la manualmente em seu código fonte ou usando o editor visual do plugin ADT no Eclipse.

Após inserir nossa nova String e salvar o arquivo de String Resources, ela será mapeada automaticamente na *classe* ***R*** e já podemos usá-la em nosso aplicativo. Para testá-la, criaremos uma nova Activity, chamada ***StringResourceActivity***, com o seguinte conteúdo:

```
public class StringResourceActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
                TextView texto = new TextView(getApplicationContext());
                // Utilizando a referência da String Resource para ser apresentada em uma View
                texto.setText(R.string.minha_string);

                // Obtendo o conteúdo da String Resource através do método getString() e apresentando-a no
LogCat
                Log.i("String Resources", getString(R.string.minha_string));

                setContentView(texto);
        }
}
```

Ao executar a Activity no emulador do Android, a String "*Testando o uso de uma String Resource*" será apresentada em uma View do tipo TextView na tela do emulador e também no LogCat.

• **Usando um String Resource em um arquivo XML de Layout**:

Podemos também usar as Resource Strings nos arquivos XML de Layout do aplicativo. Para fazer a chamada da String, usamos a sintaxe:

```
@string/nome_da_string_resource
```

Onde:

nome_da_string_resource: é o nome que definimos para a String no arquivo XML, no atributo ***name***.

Como exemplo, veja o conteúdo arquivo de Layout *activity_hello_world.xml* gerado no projeto HelloWorld que criamos no *Capítulo 3*:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world" />
```

Perceba na View TextView do arquivo de Layout o atributo android:text="@string/hello_world" chamando a String Resource chamada "hello_world" para ser apresentada na tela do dispositivo.

5.17.2 Color Resources

Os Color Resources lozalizam-se também no diretório ***/res/values*** e são semelhantes às String Resources, porém servem para guardar cores que usaremos nas Views do nosso aplicativo. Imagine, por exemplo, que você deseja que todos os textos apresentados nas telas do seu aplicativo tenham a cor preta. Para isto, basta adicionar a cor desejada em um arquivo de resource e usar sua referência em todas as Views que usam texto. Declarar uma cor em um arquivo de resource é simples, e fazemos isso usando a seguinte sintaxe:

```
<color name="preto">#000000</color>
```

Onde:

No atributo **name** colocamos o nome que queremos atribuir à cor. Este nome será mapeado na *classe R*.

Dentro da tag **color** colocamos o valor hexadecimal referente à cor desejada.

Dica

Centralizar as cores das Views em um arquivo de resource é uma boa prática. Imagine, por exemplo, que seu cliente o peça para mudar a cor de fundo de todas as telas do aplicativo, como você faria? Se você tomou o cuidado de criar uma Color Resource e usá-la como cor de fundo das telas, basta mudar o valor da cor no arquivo de resource que a cor mudará em todas as telas do aplicativo. *[Seu cliente com certeza ficará muito feliz com a velocidade com que você implementará esta mudança]*

Para acessar um Color Resource, usamos sua referência:

- **Em uma Activity**: *R.color.nome_da_cor*

  Exemplo:

```
TextView texto = new TextView(getApplicationContext());
texto.setText("Olá, Android!");
texto.setTextColor(getResources().getColor(R.color.preto));
```

- **Em um arquivo XML de Layout**: *@drawable/imagem*

  Exemplo:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world"
   android:textColor="@color/preto" />
```

5.17.3 Drawable Resources

Os Drawable Resources localizam-se no diretório ***/res/drawable*** e compreende as imagens que serão apresentadas na tela pelo nosso aplicativo. Essas imagens podem ser **png** (preferenciais), **jpeg** ou **gif**.

Os Drawable Resources também têm suas referências mapeadas na *classe* ***R*** e podem ser acessados pelas Activities, através do método ***getDrawable()***, ou pelos arquivos XML de Layout.

Como exemplo, uma imagem localizada em ***/res/drawable/imagem.png*** é acessada pela sua referência:

- **Em uma Activity**: *R.drawable.imagem*

  Exemplo:

```
ImageView imagem = new ImageView(getApplicationContext());
imagem.setImageResource(R.drawable.imagem);
setContentView(imagem);
```

- **Em um arquivo XML de Layout**: *@drawable/imagem*

  Exemplo:

```
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:layout_centerVertical="true"
   android:src="@drawable/imagem" />
```

Dica

O nome do arquivo de imagem, exceto sua extensão, é usado como ID do resource, portanto o ID de um arquivo de nome ***imagem.png*** é apenas ***imagem***. Sendo assim, não podemos ter uma imagem de mesmo nome com extensões diferentes.

5.17.4 Layout Resources

No Android é possível criar as telas de interface gráfica (Views) do aplicativo em um arquivo XML. Estes arquivos de layout são chamados de ***Layout Resources***. Mesmo sendo possível definir as Views no código Java em uma Activity, é recomendado que os Layout Resources sejam usados para a criação das telas do aplicativo, justamente para separar o código do aplicativo do código da interface gráfica.

Os arquivos XML de Layout localizam-se no diretório ***/res/layout***. Como exemplo, para definir o layout de uma Activity usando um arquivo de layout localizado em ***/res/layout/meu_layout.xml***, usamos sua referência mapeada na *classe* ***R*** através do método ***setContentView()***.

Exemplo:

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.meu_layout);
}
```

Veja o exemplo de um arquivo XML de Layout do nosso exemplo *HelloWorld* criado no início do curso:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world" />

</RelativeLayout>
```

5.17.5 Outras Resources

Além das String Resources e das Color Resources, existem outros tipos de recursos que podemos usar, como:

- Animation Resources;
- Drawable Resources;
- Menu Resources;
- e outros.

Para ver a lista completa de tipos de recursos do Android, acesse o link no site oficial do desenvolvedor Android:

http://developer.android.com/guide/topics/resources/available-resources.html

5.18 Permissões de acesso

Outro conceito muito importante que devemos conhecer no Android é como funciona seu esquema de permissão de acesso a alguns recursos da plataforma. Algumas operações no Android exigem permissão para serem realizadas e estas permissões são definidas no arquivo ***AndroidManifest.xml***.

Um exemplo é quando seu aplicativo precisa realizar uma chamada telefônica. Para isto, é necessário informar ao Android que seu aplicativo irá usar o aplicativo de Telefonia. Para isto, basta inserir a seguinte configuração no arquivo *AndroidManifest.xml*:

```
. . .
<uses-permission android:name="android.permission.CALL_PHONE" />
. . .
```

As permissões definidas no *AndroidManifest.xml* são lidas pelo Android no momento da instalação do aplicativo. Quando o usuário instala um aplicativo no dispositivo e este aplicativo necessita de permissão para acesso a algum recurso, o Android avisa ao usuário sobre cada recurso que exigirá permissão e pergunta se o usuário deseja conceder as permissões e proceder com a instalação. Isto garante uma maior segurança e deixa o usuário ciente de que um determinado aplicativo irá utilizar algum recurso mais restrito da plataforma.

Outras operações que precisam de permissão é o uso da Internet, o uso da câmera do dispositivo, o envio de mensagens SMS e diversas outras. Para ver a lista completa dos recursos que exigem permissão de acesso, acesse o link:

http://developer.android.com/reference/android/Manifest.permission.html

6 - Interfaces Gráficas

Como aprendemos anteriormente, a tela de um aplicativo do Android é composta por componentes chamados Views.

Os componentes View herdam da classe ***android.view.View*** e podem ser de vários tipos, como:

- **TextView**: componente para mostrar um texto na tela do aplicativo;
- **EditText**: campo de entrada de texto;
- **Button**: botão que dispara um evento;
- **CheckBox**: caixa de seleção;
- **Gerenciador de Layout**: View mais complexa que pode conter outras Views, ou seja, um agrupador de Views;
- e outros.

Para organizar as Views na tela, o Android dispõe de um **Gerenciador de Layout** que é basicamente um agrupador de Views (*View Group*), os chamados **Layouts**. Os Layouts herdam de ***android.view.ViewGroup*** e possibilita organizar os componentes *View* de forma que fiquem visualmente agradáveis aos usuários, ordenando-os horizontalmente, verticalmente, em forma de grade, etc.

A classe ***android.view.ViewGroup*** também é uma View, ou seja, ela também herda de ***android.view.View*** e permite que um Layout seja definido dentro de outro Layout.

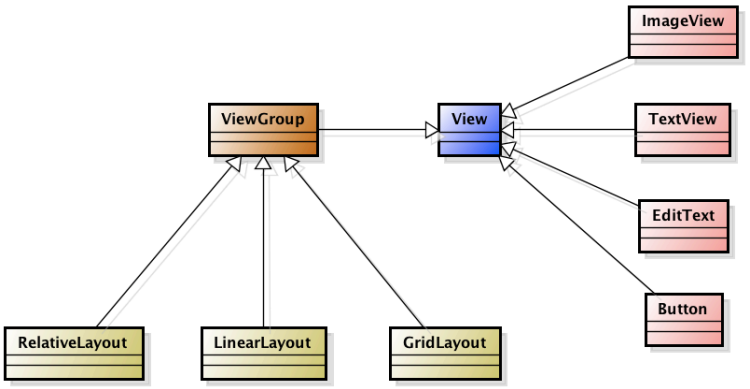


Figura 6.1. Diagrama mostrando a relação dos componentes *View* do Android.

6.1 Criando um arquivo de Layout

Até então, os arquivos de Layout que vimos foram criados automaticamente pelo assistente do ADT ao criarmos uma Activity. Agora iremos aprender como criar um arquivo de Layout independente da criação da Activity.

Para criarmos um arquivo XML de Layout, no Eclipse, clicamos com o botão direito do mouse sobre o projeto do Android, em seguida navegamos até a opção "**New**" > "**Other**", selecionamos a opção "**Android**" > "**Android XML Layout File**" e clicamos em **Next**, conforme mostra a **Figura 6.2**.

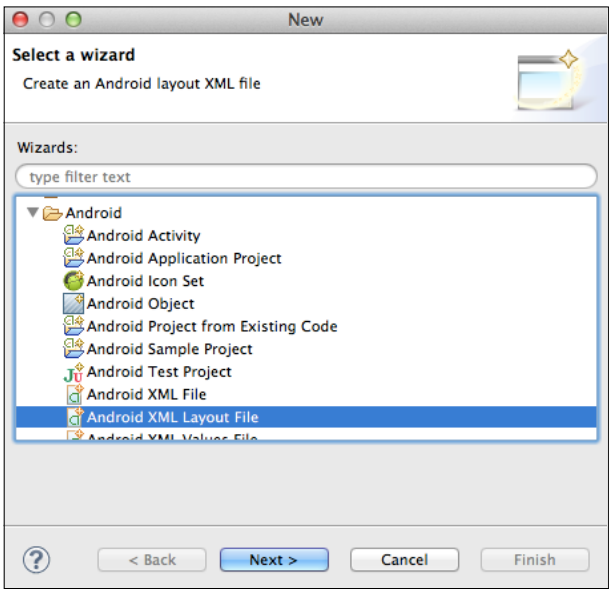


Figura 6.2. Assistente do ADT para criação de um novo arquivo XML de Layout.

Na próxima tela, devemos definir o nome do arquivo de Layout e o tipo de Layout que iremos usar, conforme mostra a **Figura 6.3**.

• No campo ***File***, definimos o nome do novo arquivo de Layout. Em nosso caso, iremos atribuir o nome: *activity_trabalhando_com_id*

• Na opção ***Root Element***, devemos definir o tipo de layout que iremos usar. Em nosso exemplo, iremos selecionar a opção: ***LinearLayout*** *[Aprenderemos sobre os tipos de Layout mais à frente neste curso]*

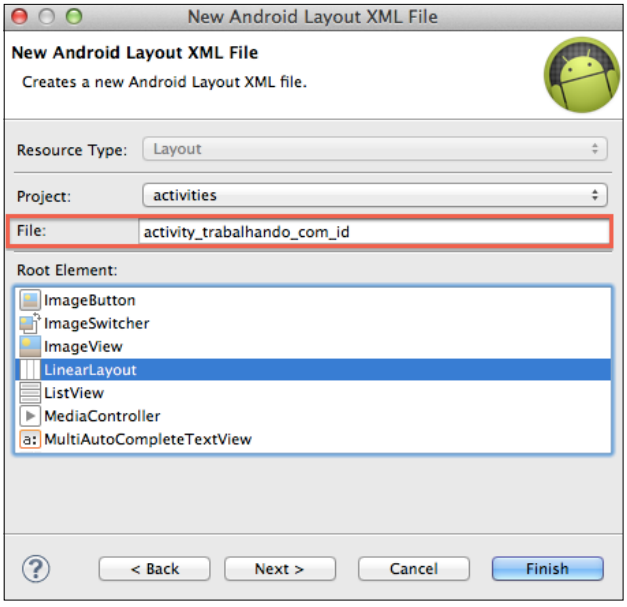


Figura 6.3. Assistente do ADT para criação de um novo arquivo XML de Layout.

Definido o nome do novo arquivo de Layout e seu tipo, clicamos no botão ***Finish***.

6.2 Definindo um ID às Views

Em determinados momentos, durante o desenvolvimento de uma tela do nosso aplicativo, precisamos definir um ID a um componente View para, de alguma forma, manipular este componente através da Activity.

No arquivo XML de Layout para definir um ID para uma View usamos a seguinte sintaxe:

```
android:id="@+id/nome_do_id"
```

Onde:

nome_do_id: id que desejamos atribuir à View.

Dica

O ID de uma View também é mapeado na *classe* ***R*** e deve ser único para cada componente do aplicativo.

Como exemplo, vamos criar um novo arquivo de Layout, chamado ***activity_trabalhando_com_id.xml***, com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/texto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Perceba, no arquivo de Layout, que a View TextView não possui um texto definido para ser apresentado na tela, pois não definimos seu atributo **android:text**. Como nosso componente TextView possui um ID definido, chamado **texto**, podemos manipulá-lo através de uma Activity e nela definir o conteúdo do seu texto. Para isto, iremos criar uma nova Activity, chamada ***TrabalhandoComIDActivity***, com o seguinte conteúdo:

```
public class TrabalhandoComIDActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_trabalhando_com_id);
        /**
         * Acessando o TextView do XML de Layout
         * através do seu ID e definindo o seu texto.
         */
        TextView texto = (TextView) findViewById(R.id.texto);
        texto.setText("Olá, Android!");
    }
}
```

Perceba, no código da nova Activity, que usamos o método ***findViewById()*** para referenciar a View TextView definida no arquivo de Layout ***activity_trabalhando_com_id.xml*** criado anteriormente, passando seu ID como parâmetro.

O método ***findViewById()*** é usado quando queremos referenciar uma View definida em um arquivo de Layout dentro da nossa Activity, para manipularmos seu comportamento, passando apenas seu ID como parâmetro.

Ao executar o código acima, o texto "**Olá, Android!**", definido pela Activity, será apresentado na tela.

6.3 Views

Como mencionamos anteriormente, as Views são componentes usados para construir a tela de um aplicativo para Android. Iremos agora conhecer as Views mais utilizadas.

6.3.1 TextView

A ***TextView*** é a View mais utilizada na construção de um aplicativo, e serve basicamente para mostrar um texto na tela. Veja a sintaxe de uma TextView definida em um arquivo de Layout:

```
<TextView
    android:text="Olá, Android!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Dica

O atributo **android:text** serve para definir o texto a ser apresentado pela TextView. Uma boa prática é sempre centralizar os textos a serem apresentados em um String Resource para que as mensagens sejam reaproveitadas em locais diferentes do aplicativo e para se beneficiar do suporte à internacionalização.

É possível mudar o estilo de texto, colocando por exemplo **negrito**, *itálico* ou sublinhado usando tags HTML, conforme mostra a **Tabela 6.1**.

Estilo	Implementação
Negrito	****Texto em negrito****
Itálico	****Texto em itálico****
Sublinhado	**<u>**Texto sublinhado**</u>**

Tabela 6.1. Assistente do ADT para criação de um novo arquivo XML de Layout.

Veja o arquivo de String Resources com exemplo de texto em **negrito**, *itálico* e sublinhado:

```
<string name="texto_em_negrito"><b>Texto em negrito</b></string>
<string name="texto_em_italico"><i>Texto em itálico</i></string>
<string name="texto_sublinhado"><u>Texto sublinhado</u></string>
```

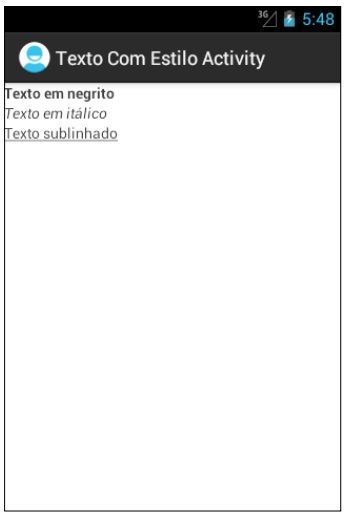


Figura 6.4. Textos com estilo, em uma TextView.

6.3.2 EditText

A View ***EditText*** é usada para permitir a inserção de dados em um campo através do teclado do dispositivo. Veja a sintaxe de uma EditText definida em um arquivo de Layout:

```
<EditText
    android:id="@+id/nome"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

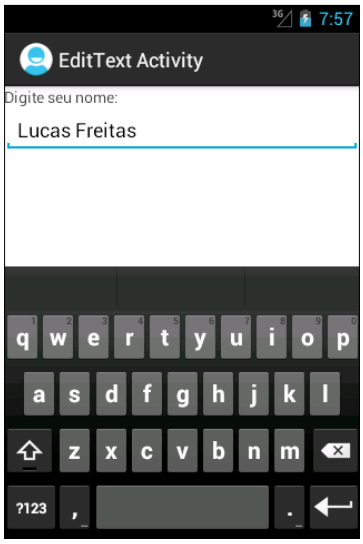


Figura 6.5. Exemplo de uma EditText para entrada de informação.

É possível definir o tipo de dado que será inserido pelo usuário em uma View EditText, e fazemos isto através do atributo **android:inputType**. Veja na **Figura 6.6** uma tela com campos EditText com tipo de entrada de dados definido.

O atributo **android:inputType** possui diversos tipos de entrada de dados. A **Tabela 6.2** mostra alguns dos tipos de dados que podemos definir para entrada em um EditText.

inputType	Tipo de entrada de dado aceito
text	Texto
phone	Número de telefone
textPassword	Senha alfa-numérica
numberPassword	Senha numérica
number	Números
date	Data
textMultiLine	Texto com quebra de linha
textEmailAddress	Endereço de e-mail

Tabela 6.2. Assistente do ADT para criação de um novo arquivo XML de Layout.

Exemplo de um EditText que aceita apenas números como entrada:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number" />
```

Figura 6.6. Exemplo de tela com campos EditText com tipos de entrada de dados definidos.

Dica

Definir o tipo de dado que o usuário irá inserir em uma EditText é uma boa prática, impedindo, por exemplo, que um texto seja inserido em um campo que deveria receber apenas números. Outra vantagem é que, ao definirmos o tipo de dado de entrada, o Android irá mostrar no teclado virtual apenas as teclas com o tipo de dado que o usuário poderá inserir, facilitando a digitação do usuário.

Para pegar o valor digitado pelo usuário em uma EditText e injetá-lo em um objeto String na Activity, fazemos da seguinte maneira:

```
EditText nome = (EditText) findViewById(R.id.nome);
String valorNome = nome.getText().toString();
```

6.3.3 CheckBox

A View ***CheckBox*** é basicamente uma caixa de seleção, com estado de marcado ou desmarcado. Veja a sintaxe de uma CheckBox definida em um arquivo de Layout:

```
<CheckBox
    android:id="@+id/maior_idade"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tenho mais que 18 anos" />
```

O código acima mostra na tela uma View CheckBox para o usuário confirmar se ele é maior que 18 anos. A **Figura 6.7** mostra o resultado na tela do dispositivo.

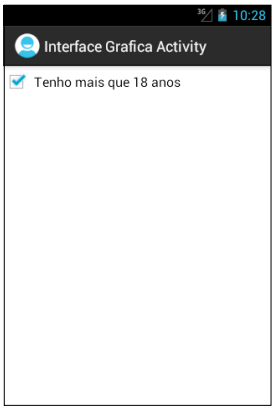


Figura 6.7. Exemplo de tela com View CheckBox.

Na Activity, para saber se o usuário marcou ou não a CheckBox, fazemos da seguinte maneira:

```
CheckBox maiorIdade = (CheckBox) findViewById(R.id.maior_idade);
boolean temMaisDe18 = maiorIdade.isChecked();
```

6.3.4 RadioButton

A View ***RadioButton*** representa um grupo de botões onde podemos selecionar apenas uma das opções disponíveis. Para que o RadioButton funcione da maneira correta é preciso que esteja dentro de uma outra View, chamada ***RadioGroup***. Dentro de uma *RadioGroup*, apenas uma *RadioButton* pode ser selecionada. Veja a sintaxe de uma RadioButton definida em um arquivo de Layout:

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <RadioButton
      android:id="@+id/sexo_masculino"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Masculino" />

    <RadioButton
      android:id="@+id/sexo_feminino"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Feminino" />
  </RadioGroup>
```

O código acima mostra um *RadioGroup* contendo duas opções para seleção: sexo masculino ou feminino. A **Figura 6.8** mostra o resultado na tela do dispositivo.

Figura 6.8. Exemplo de tela com View RadioButton.

Da mesma forma que uma CheckBox, para definir se uma RadioButton foi selecionada fazemos da seguinte maneira:

```
RadioButton sexoMasculino = (RadioButton) findViewById(R.id.sexo_masculino);
boolean selecionouSexoMasculino = sexoMasculino.isChecked();
```

6.3.5 ImageView

A View ***ImageView*** é usada para mostrar uma imagem na tela do dispositivo. Veja a sintaxe de uma ImageView definida em um arquivo de Layout:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher" />
```

O código acima irá mostrar na tela uma imagem, chamada ***ic_launcher.png*** localizada em ***/res/drawable*** (Drawable Resource). A **Figura 6.9** mostra o resultado na tela do dispositivo.

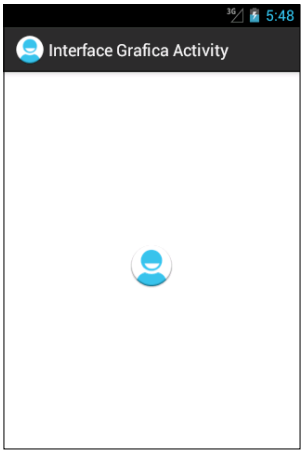


Figura 6.9. Exemplo de tela com View ImageView.

6.3.6 Button

A View ***Button*** é basicamente um botão mostrado na tela que, ao ser clicado, dispara um evento. Veja a sintaxe de uma View Button definida em um arquivo de Layout:

```
<Button
    android:id="@+id/sair"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sair" />
```

O código acima irá mostrar na tela um botão com o texto "**Sair**". A **Figura 6.10** mostra o resultado na tela do dispositivo.

Figura 6.10. Exemplo de tela com View Button.

Para definir a ação de um Button, em uma Activity, usamos o método ***setOnClickListener()*** da seguinte forma:

1. Na Activity, referenciamos o objeto *Button* definido na View através de seu ID:

```
Button sair = (Button) findViewById(R.id.sair);
```

2. No objeto *Button*, chamamos o método ***setOnClickListener()*** e passamos para este método uma implementação da interface ***android.view.View.OnClickListener*** subscrevendo o método ***onClick()***, que irá executar uma ação quando o usuário clicar no botão. Em nosso caso, queremos que a Activity apenas seja fechada quando o usuário clicar no botão, ficando o nosso código da seguinte forma:

```
sair.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
                // Finaliza a Activity
                finish();
        }
});
```

Veja como ficou o código completo:

```
Button sair = (Button) findViewById(R.id.sair);
sair.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
                // Finaliza a Activity
                finish();
        }
});
```

6.3.7 ImageButton

A View ***ImageButton*** tem a mesma finalidade de um Button, porém ao invés de exibir um botão com um texto, ela exibe um botão com uma imagem e permite que uma ação seja disparada ao clicar nesta imagem. Veja a sintaxe de uma View ImageButton definida em um arquivo de Layout:

```
<ImageButton
        android:id="@+id/sair"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />
```

O evento disparado ao clicar em uma ImageButton é implementado da mesma forma que em uma View Button.

6.3.8 DatePicker

A View ***DatePicker*** é um componente que possibilita ao usuário definir uma data qualquer. Veja a sintaxe de uma View DatePicker definida em um arquivo de Layout:

```
<DatePicker
        android:id="@+id/data_nascimento"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:calendarViewShown="false" />
```

O código irá mostrar na tela uma View DatePicker para seleção de data, conforme mostra a **Figura 6.11**.

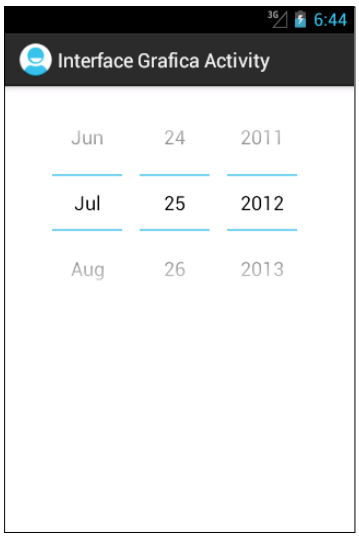


Figura 6.11. Exemplo de tela com View DatePicker.

Na Activity, para recuperar a data informada pelo usuário em uma View DatePicker, basta referenciar a View em um objeto ***DatePicker***, da seguinte forma:

```
DatePicker dataNascimento = (DatePicker) findViewById(R.id.data_nascimento);
```

Através do objeto *DatePicker*, obtemos a data da seguinte forma:

```
int dia = dataNascimento.getDayOfMonth();
int mes = dataNascimento.getMonth();
int ano = dataNascimento.getYear();
```

Dica

O método ***getMonth()*** do objeto ***DatePicker***, usado para ler o mês selecionado pelo usuário, inicia com o valor 0 (zero). Sendo assim, se o usuário selecionar o mês de Maio, por exemplo, o método ***getMonth()*** irá retornar o valor 4 ao invés de 5. Portanto, caso queira obter o número real que represente o mês selecionado, devemos incrementar este valor em uma unidade, ficando da seguinte forma:

```
int mes = dataNascimento.getMonth() + 1;
```

6.3.9 TimePicker

A View ***TimePicker*** é um componente que possibilita ao usuário definir um horário. Veja a sintaxe de uma View TimePicker definida em um arquivo de Layout:

```
<TimePicker
        android:id="@+id/horario"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
```

O código acima irá mostrar na tela do dispositivo um componente para seleção de horário, conforme mostra a **Figura 6.12**.

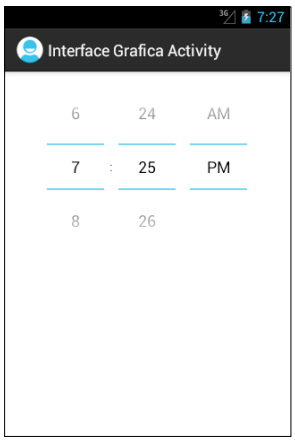


Figura 6.12. Exemplo de tela com View TimePicker.

Na Activity, para recuperar o horário informado pelo usuário em uma View TimePicker, basta referenciar a View em um objeto ***TimePicker***, da seguinte forma:

```
TimePicker horario = (TimePicker) findViewById(R.id.horario);
```

Através do objeto *TimePicker*, obtemos o horário da seguinte forma:

```
int valorHora = horario.getCurrentHour();
int valorMinuto = horario.getCurrentMinute();
```

6.4 Layouts

Até agora nós aprendemos a utilizar alguns componentes Views na tela do nosso aplicativo. Porém, para organizar estes componentes na tela, precisamos utilizar os ***Layouts*** (também conhecidos como ViewGroups), que são basicamente Views agrupadoras que facilitam a disposição dos componentes na tela.

6.4.1 LinearLayout

O ***LinearLayout*** é usado para alinhar os componentes de forma horizontal ou vertical, fazendo com que as Views fiquem uma ao lado da outra ou uma embaixo da outra.

1. Organizando Views verticalmente

Para organizar verticalmente as Views na tela, deixando-as uma embaixo da outra, usamos a propriedade android:orientation="vertical" em um LinearLayout. Veja o exemplo:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Nome: " />

   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:inputType="text" />
</LinearLayout>
```

No código de exemplo, para colocar os componentes *TextView* e *EditText* um embaixo do outro, usamos o LinearLayout com orientação vertical. A **Figura 6.13** mostra o resultado do código de exemplo em execução, na tela do dispositivo.

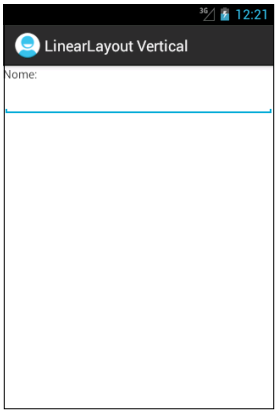


Figura 6.13. Componentes organizados verticalmente em um *LinearLayout*.

2. Organizando Views horizontalmente

Para organizar horizontalmente as Views na tela, deixando-as uma ao lado da outra, usamos a propriedade android:orientation="horizontal" em um LinearLayout. Veja o exemplo:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Nome: "
      android:layout_weight="1" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:inputType="text"
      android:layout_weight="4" />
</LinearLayout>
```

A **Figura 6.14** mostra o resultado do código de exemplo em execução, na tela do dispositivo.

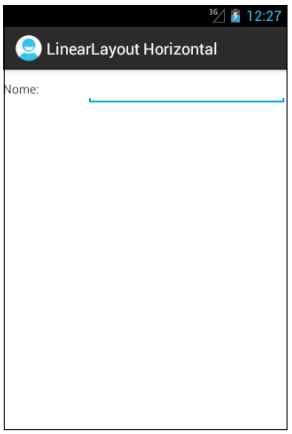


Figura 6.14. Componentes organizados horizontalmente em um *LinearLayout*.

6.4.2 RelativeLayout

O ***RelativeLayout*** é usado para alinhar um componente de forma relativa a outro componente, permitindo, por exemplo, declarar que um determinado componente fique à esquerda, direita, embaixo ou acima de outro componente.

Uma atenção que devemos ter é que o componente usado como referência de posicionamento deve sempre vir antes do componente que está fazendo a referência. Como exemplo, para declarar a posição de um componente **A** em relação a um componente **B** é preciso que o componente **B** seja declarado antes do componente **A** no arquivo de Layout.

Para que uma View seja posicionada em relação a outra View, basta definir o seu posicionamento relativo. Veja na **Tabela 6.3** alguns dos tipos de posicionamento aplicado nas Views, em um RelativeLayout.

Posicionamento	Descrição
android:layout_toRightOf	À direita de outro componente
android:layout_toLeftOf	À esquerda de outro componente
android:layout_below	Abaixo de outro componente
android:layout_alignParentTop	Ao topo da View pai
android:layout_centerVertical	Centralizado verticalmente na View pai

Tabela 6.3. Tipos de posicionamento usados nas Views em um *RelativeLayout*.

Veja o exemplo de um RelativeLayout:

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <TextView
      android:id="@+id/texto_a"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Texto A" />

   <TextView
      android:id="@+id/texto_b"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Texto B"
      android:layout_toRightOf="@id/texto_a" />

   <TextView
      android:id="@+id/texto_c"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Texto C"
      android:layout_below="@id/texto_a" />

   <TextView
      android:id="@+id/texto_d"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Texto D"
      android:layout_toRightOf="@id/texto_c"
      android:layout_below="@id/texto_b" />


</RelativeLayout>
```

A **Figura 6.15** mostra o resultado do código de exemplo em execução, na tela do dispositivo.

Figura 6.15. Tela com Views posicionadas relativamente em um *RelativeLayout*.

6.4.3 TableLayout

O ***TableLayout*** é usado para ordenar os componentes em forma de tabela. Como toda tabela, o TableLayout é definido por linhas e colunas, e são organizados da seguinte forma:

• As linhas são definidas através da tag ***TableRow***;

• As colunas são definidas implicitamente, onde cada View dentro de uma linha (***TableRow***) representa uma coluna.

Veja o exemplo de um arquivo de Layout usando o TableLayout:

```
<TableLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:stretchColumns="1" >

   <TableRow>

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Nome: " />

      <EditText
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />
   </TableRow>

   <TableRow>

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="E-mail: " />

      <EditText
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />
   </TableRow>

</TableLayout>
```

A **Figura 6.16** mostra o resultado do código de exemplo em execução, na tela do dispositivo.

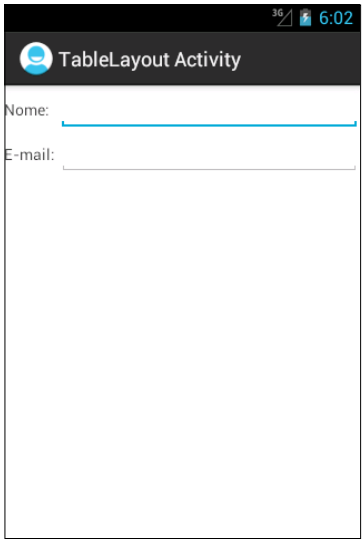


Figura 6.16. Componentes organizados em um *TableLayout*.

Como mostra a **Figura 6.16**, nossa TableLayout possui duas linhas, definidas pela tag ***TableRow***. Perceba também que há duas Views em cada linha, uma *TextView* e um *EditText*, ficando cada uma delas em uma coluna da tabela.

Outro detalhe importante neste exemplo é que, na definição do *TableLayout*, definimos uma propriedade chamada **android:stretchColumns**. Esta propriedade serve para **esticar** uma determinada View da tabela até o limite da tela, recebendo como parâmetro o número da coluna que queremos esticar, e este número começa em 0 (zero). Em nosso exemplo perceba que a segunda coluna, ou seja, a View *EditText*, ficou esticada pois atribuímos o valor 1 para a propriedade **android:stretchColumns**, e o valor 1 referencia a segunda coluna.

Dica

A propriedade ***android:stretchColumns*** serve para **esticar** uma coluna do *TableLayout* até o limite da tela, recebendo como parâmetro o número da coluna a ser esticado, começando pelo valor 0 (zero), que referencia a primeira coluna.

6.4.4 FrameLayout

O ***FrameLayout*** é usado quando queremos mostrar apenas uma View na tela do dispositivo. Veja o exemplo de um arquivo de Layout usando o FrameLayout:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <ImageView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:src="@drawable/ic_launcher" />

</FrameLayout>
```

Dica

É possível definir mais de uma View dentro de um FrameLayout, porém essas Views serão empilhadas e a última View ficará sempre ao topo da tela.

A **Figura 6.17** mostra o resultado do código de exemplo em execução, na tela do dispositivo.

Figura 6.17. ImageView apresentada em um *FrameLayout*.

6.4.5 ScrollView

O ***ScrollView*** é um ViewGroup usado quando nossa tela precisa de uma barra de rolagem (scrollbar) por não caber todas as Views na tela do dispositivo. Usando o ScrollView é possivel rolar pela tela para visualizar todas as Views.

O ScrollView estende de FrameLayout e permite o uso de apenas uma View filha. Normalmente, em um ScrollView, usa-se o LinearLayout com orientação vertical, como View filha.

Veja o exemplo de um arquivo de Layout usando o ScrollView:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:paddingTop="10.0dip"
            android:src="@drawable/ic_launcher" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingTop="10.0dip"
            android:text="Nome:" />

        <EditText
            android:id="@+id/nome"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingTop="10.0dip"
            android:text="Sexo:" />

        <RadioGroup
```

```
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="horizontal" >

                <RadioButton
                    android:id="@+id/sexo_masculino"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Masculino" />

                <RadioButton
                    android:id="@+id/sexo_feminino"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Feminino" />
            </RadioGroup>

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:paddingTop="10.0dip"
                android:text="Data de nascimento: " />

            <DatePicker
                android:id="@+id/data_nascimento"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:calendarViewShown="false" />

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:paddingTop="10.0dip"
                android:text="Confirme se for maior de 18 anos: " />

            <CheckBox
                android:id="@+id/maior_idade"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Declaro ter mais de 18 anos" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center_horizontal|center_vertical"
                android:orientation="horizontal"
                android:paddingTop="10.0dip" >

                <Button
                    android:id="@+id/salvar"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Salvar" />

                <Button
                    android:id="@+id/sair"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Sair" />
            </LinearLayout>
        </LinearLayout>
</ScrollView>
```

O Layout de exemplo possui diversas Views e provavelmente não caberão todas elas na tela de dispositivos com tela pequena. Para que seja possível a visualização de todas as Views pelo usuário usamos o ScrollView, possibilitando rolar pela tela através da barra de rolagem.

6.4.6 ListView

A ***ListView*** é usada para mostrar na tela uma lista vertical de itens, disponibilizando uma barra de rolagem (scrollbar) caso os itens da lista não caibam na tela do dispositivo e é muito usada para criar telas de menus, onde cada item do menu dispara um evento ao receber um clique.

Um grande exemplo do uso de uma ListView é quando um aplicativo possui várias Activities e precisamos disponibilizar um menu para que o usuário escolha a Activity que deseja utilizar. Até o momento, em nossos exemplos, sempre configuramos a última Activity desenvolvida para que seja a Activity inicial ao executar o aplicativo no Android, no entando poderíamos utilizar uma ListView para listar todas as Activities do nosso aplicativo para que o usuário possa executar a que ele desejar.

Para facilitar o uso de uma ListView, o Android disponibiliza a classe ***android.app.ListActivity***, que herda de *android.app.Activity* e disponibiliza métodos que facilitam o seu uso.

Dica

A ListActivity já faz a chamada para o método ***setContentView()***, não nos sendo necessário chamá-lo.

Para realizar a ligação entre os dados a serem mostrados e a ListView devemos desenvolver um ***Adapter***, que deverá implementar a interface ***android.widget.ListAdapter***. Através de uma implementação do *ListAdapter* podemos mostrar diversos tipos de dados na ListView, como: Strings, imagens, dados armazenados no banco de dados, etc.

Para facilitar ainda mais o nosso trabalho, o Android já disponibiliza alguns arquivos de layout predefinidos para usarmos em uma ListView, restando-nos apenas implementar o Adapter que irá prover os dados a serem apresentados na lista.

Para exemplificar o uso de uma ListView, iremos mostrar uma implementação usando um arquivo predefinido de layout e outra com um arquivo de layout customizado.

6.4.6.1 ListView com arquivo de layout predefinido

Para usar uma ListView com arquivo de layout predefinido precisamos apenas criar uma Activity que herde de *ListActivity* e nela definir o Adapter que irá fazer a ligação entre os dados a serem apresentados e a ListView predefinida.

Em nosso exemplo, vamos usar um ListView para construir o menu da tela inicial do nosso aplicativo, listando todas as Activities existentes no projeto, portanto vamos usar um ***ArrayAdapter*** como Adapter e este irá mostrar apenas Strings em uma ListView, conforme mostra o código abaixo:

```
public class ListViewActivity extends ListActivity {

        private static final String[] OPCOES_DO_MENU = new String[] {
                "Definindo ID às Views",
                "Trabalhando com Alinhamento",
                "Trabalhando com EditText",
                "Definindo peso de uma View",
                "Trabalhando com Views",
                "Trabalhando com Texto Estilizado",
                "Trabalhando com LinearLayout Horizontal",
                "Trabalhando com LinearLayout Vertical",
                "Trabalhando com RelativeLayout",
                "Trabalhando com TableLayout",
                "Trabalhando com FrameLayout",
                "Sair"
        };

    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);

        /**
         * Define o Adapter que irá mostrar os dados na ListView.
         */
        setListAdapter(new ArrayAdapter<String>(
                        this, android.R.layout.simple_list_item_1,
                        OPCOES_DO_MENU));
    }

    /**
     * Evento a ser disparado ao clicar em um dos itens da ListView.
     */
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
            switch (position) {
            case 0:
                        startActivity(new Intent(this, TrabalhandoComIDActivity.class));
                        break;
            case 1:
                        startActivity(new Intent(this, AlinhamentoActivity.class));
                        break;
            case 2:
                        startActivity(new Intent(this, EditTextActivity.class));
                        break;
            case 3:
                        startActivity(new Intent(this, ImportanciaViewActivity.class));
                        break;
            case 4:
                        startActivity(new Intent(this, InterfaceGraficaActivity.class));
                        break;
            case 5:
                        startActivity(new Intent(this, TextoComEstiloActivity.class));
                        break;
            case 6:
                        startActivity(new Intent(this, LinearLayoutHorizontalActivity.class));
                        break;
            case 7:
                        startActivity(new Intent(this, LinearLayoutVerticalActivity.class));
                        break;
            case 8:
                        startActivity(new Intent(this, RelativeLayoutActivity.class));
                        break;
            case 9:
                        startActivity(new Intent(this, TableLayoutActivity.class));
                        break;
            case 10:
                        startActivity(new Intent(this, FrameLayoutActivity.class));
                        break;
            default:
                        finish();
            }
    }

}
```

Entendendo o código:

• Perceba que nossa classe estende de ListActivity, não sendo necessário chamar o método ***setContentView()*** no método ***onCreate()***.

• No método ***onCreate()*** fazemos a chamada para o método ***setListAdapter()*** da ListActivity, para definir o Adapter a ser usado para mostrar os dados na ListView e, em nosso caso, usamos um ***ArrayAdapter*** para mostrar apenas Strings na ListView. Perceba que ao criar um ArrayAdapter, passamos três parâmetros para seu construtor:

1. O contexto (*o objeto Context*);
2. O arquivo de Layout com a ListView que irá mostrar os dados. Nesse caso usamos um layout predefinido do Android, o *android.R.layout.simple_list_item_1*, que mostra apenas um objeto por linha, na ListView;
3. O array com as Strings que queremos mostrar na ListView.

• Na Activity subscrevemos o método ***onListItemClick()***, da classe ListActivity, para definir o evento a ser disparado quando o usuário clicar em uma das opções do menu na ListView. Perceba que tratamos isso através da posição de cada item na ListView, começando pelo valor 0 (zero). Em nosso caso, cada opção do menu irá abrir uma Activity do nosso aplicativo.

Veja na **Figura 6.18** o resultado da execução do código no emulador do Android.

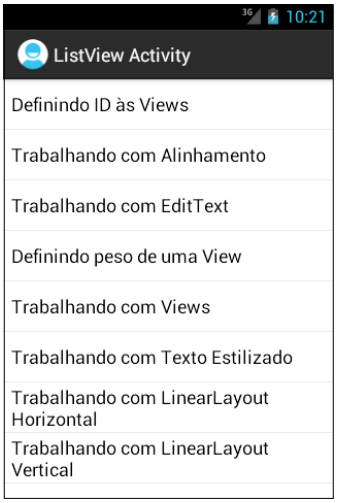


Figura 6.18. ListView listando as Activities do aplicativo.

6.4.6.2 ListView com arquivo de layout customizado

Além de usar um arquivo predefinido de layout para exibir uma ListView na tela do dispositivo, podemos criar nosso próprio layout customizado, mostrando, por exemplo, textos com imagens, textos em linha dupla, etc.

Para ver na prática como funciona, vamos refazer o exemplo anterior, mas desta vez usando um layout customizado adicionando uma imagem junto com o texto a ser apresentado em cada item da ListView.

Como nosso objetivo é mostrar uma imagem e uma String em cada item da ListView, devemos criar um novo arquivo de layout customizado que contenha uma *ImageView* (para mostrar a imagem) e uma *TextView* (para mostrar o texto). Criaremos então um arquivo de layout, chamado ***activity_listview_customizado.xml***, com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >

   <ImageView
      android:id="@+id/imagem_activity"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
```

```
            android:layout_gravity="center_vertical" />

        <TextView
            android:id="@+id/nome_activity"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="3"
            android:layout_gravity="center_vertical"
            android:paddingLeft="10.0dip" />

    </LinearLayout>
```

Agora que temos nosso arquivo de layout customizado, devemos criar um objeto contendo a referência para a imagem a ser apresentada e uma String. Para isto criaremos então uma nova classe, chamada ***ListViewCustomizadoItem***, com o seguinte conteúdo:

```
public class ListViewCustomizadoItem {

        // Nome da Activity a ser apresentado na ListView
        private String nomeActivity;

        // Imagem da Activity a ser apresentada na ListView
        private int imagemActivity;

        public ListViewCustomizadoItem(String nomeActivity, int imagemActivity) {
                this.nomeActivity = nomeActivity;
                this.imagemActivity = imagemActivity;
        }

        public String getNomeActivity() {
                return nomeActivity;
        }

        public int getImagemActivity() {
                return imagemActivity;
        }

}
```

Perceba que nossa classe é um simples POJO que contém os atributos que precisamos para preencher a nossa ListView. Precisamos agora criar um *Adapter* para fazer a ligação entre uma lista de objetos *ListViewCustomizadoItem* e a ListView a ser apresentada na tela do dispositivo.

Dica

Para criar nosso Adapter, precisamos estender (herdar) a classe ***android.widget.BaseAdapter***, implemementando os métodos necessários.

Criaremos então uma nova classe (que será nosso *Adapter*), chamada de ***ListViewCustomizadoAdapter***, com o seguinte conteúdo:

```
public class ListViewCustomizadoAdapter extends BaseAdapter {

        private Context contexto;

        // Lista dos itens a serem apresentados na tela
        private List<ListViewCustomizadoItem> itens;

        public ListViewCustomizadoAdapter(Context contexto, List<ListViewCustomizadoItem> itens) {
                this.contexto = contexto;
```

```
		this.itens = itens;
	}

	/**
	 * Quantidade de itens a serem apresentados na ListView.
	 */
	@Override
	public int getCount() {
		return itens.size();
	}

	/**
	 * Retorna um item da ListView através de sua posição atual.
	 */
	@Override
	public Object getItem(int posicao) {
		return itens.get(posicao);
	}

	/**
	 * ID de um item. Em nosso caso usaremos sua posição como ID.
	 */
	@Override
	public long getItemId(int position) {
		return position;
	}

	/**
	 * Aqui é onde a mágica é feita: cada item da nossa lista de itens
	 * será adaptado para ser apresentado em uma linha da ListView.
	 * É aqui que montamos nossa View customizada.
	 */
	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		// Item a ser apresentado na posição atual da ListView.
		ListViewCustomizadoItem item = itens.get(position);

		// Criando uma instância de View a partir de um arquivo de layout.
		LayoutInflater inflater = (LayoutInflater) contexto
			.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
		View view = inflater.inflate(R.layout.activity_list_view_customizado, null);

		// Imagem do item a ser apresentada na posição atual da ListView.
		ImageView imagem = (ImageView) view.findViewById(R.id.imagem_activity);
		imagem.setImageResource(item.getImagemActivity());

		// Texto do item a ser apresentado na posição atual da ListView.
		TextView texto = (TextView) view.findViewById(R.id.nome_activity);
		texto.setText(item.getNomeActivity());

		return view;
	}

}
```

Entendendo o código do *Adapter*:

• Nosso *Adapter* possui um construtor que irá receber como parâmetro o contexto (Context) e a lista de itens que iremos apresentar na ListView;

• Quando estendemos um BaseAdapter, é obrigatório a implementação do método ***getCount()***, que irá retornar a quantidade de itens que serão apresentados na ListView;

• A implementação do método ***getItem()*** também é obrigatória, para retornar o item da ListView através de sua posição atual;

• Outro método que devemos implementar é o ***getItemId()***, que irá retornar o ID de um item da ListView. Em nosso caso, iremos usar a posição do item no ArrayList como ID;

• O método ***getView()*** também deve ser implementado. É este método que fará a adaptação dos dados na ListView.

Resta-nos agora implementar a Activity que irá mostar a ListView na tela do dispositivo e, para isto, nossa Activity deverá herdar a classe *ListActivity*. Criaremos então uma nova Activity, chamada ***ListViewCustomizadoActivity***, com o seguinte conteúdo:

```
public class ListViewCustomizadoActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Cria uma lista com os itens a serem mostrados na tela.
         */
        List<ListViewCustomizadoItem> itens = new ArrayList<ListViewCustomizadoItem>();
        itens.add(new ListViewCustomizadoItem("Definindo ID às Views", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com Alinhamento", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com EditText", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Definindo peso de uma View", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com Views", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com Texto Estilizado", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com LinearLayout Horizontal",
              R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com LinearLayout Vertical", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com RelativeLayout", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com TableLayout", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Trabalhando com FrameLayout", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Sair", R.drawable.ic_launcher));

        /**
         * Define o Adapter que irá mostrar os dados na ListView.
         */
        setListAdapter(new ListViewCustomizadoAdapter(this, itens));
    }

    /**
     * Evento a ser disparado ao clicar em um dos itens da ListView.
     */
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {

            switch (position) {
            case 0:
                    startActivity(new Intent(this, TrabalhandoComIDActivity.class));
                    break;
            case 1:
                    startActivity(new Intent(this, AlinhamentoActivity.class));
                    break;
            case 2:
                    startActivity(new Intent(this, EditTextActivity.class));
                    break;
            case 3:
                    startActivity(new Intent(this, ImportanciaViewActivity.class));
                    break;
            case 4:
                    startActivity(new Intent(this, InterfaceGraficaActivity.class));
```

```
                    break;
                case 5:
                    startActivity(new Intent(this, TextoComEstiloActivity.class));
                    break;
                case 6:
                    startActivity(new Intent(this, LinearLayoutHorizontalActivity.class));
                    break;
                case 7:
                    startActivity(new Intent(this, LinearLayoutVerticalActivity.class));
                    break;
                case 8:
                    startActivity(new Intent(this, RelativeLayoutActivity.class));
                    break;
                case 9:
                    startActivity(new Intent(this, TableLayoutActivity.class));
                    break;
                case 10:
                    startActivity(new Intent(this, FrameLayoutActivity.class));
                    break;
                default:
                    finish();
                }
        }

    }
```

Perceba que nossa Activity é muito semelhante à Activity do exemplo anterior. A única diferença é que nesta Activity criamos um ArrayList com os itens que devemos passar para o *Adapter* que criamos e o definimos como *Adapter* da nossa ListView.

Ao executar nosso exemplo, obtemos o resultado da nossa ListView customizada, conforme mostra a **Figura 6.19**.

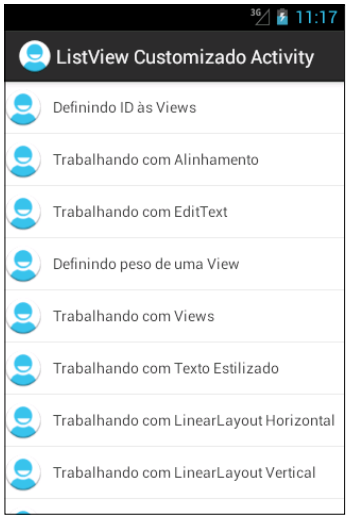


Figura 6.19. ListView customizado, listando as Activities do aplicativo.

6.5 Definindo o tamanho de uma View

Para deixar a tela do nosso aplicativo mais elegante, devemos aprender a trabalhar com o tamanho das Views, definindo corretamente sua altura e largura.

Para definir o tamanho de uma View, usamos os atributos:

- **layout_width**: usado para definir a largura da View.

• **layout_height**: usado para definir a altura da View.

Tanto o atributo de largura quanto o de altura podem receber os seguintes valores:

• **match_parent**: deixa a View com o tamanho da View pai, ou seja, preenche todo o espaço da View pai.

• **wrap_content**: preenche apenas o espaço necessário para mostrar a View.

• **valor absoluto**: permite especificar o tamanho da View.

Dica

O valor **match_parent**, usado para definir a largura ou altura de uma View, é utilizado apenas a partir da versão 2.2 do Android. Já as versões anteriores usam o valor **fill_parent**.

Como mencionamos, é possível especificar um **valor absoluto** para o tamanho de uma View, e este valor é definido de acordo com uma unidade específica. Veja na **Tabela 6.4** as unidades aceitas para o tamanho absoluto de uma View.

Tipo	Abreviatura	Descrição
Pixels	**px**	Valor real de pixels na tela
Polegadas	**em**	Baseado no tamanho físico da tela
Milímetros	**mm**	Baseado no tamanho físico da tela
Pontos	**pt**	Um ponto corresponde a 1/72 polegadas
Density-Independent Pixels	**dip** ou **dp**	Faz o mapeamento com base em 160 pixels
Scale-Independent Pixels	**sp**	Mesmo que o *dip*, mas usado para definir o tamanho de fontes.

Tabela 6.4. Unidades usadas para especificar o tamanho absoluto de uma View.

6.6 Definindo o alinhamento de uma View

O alinhamento de uma View é definido pelo atributo **android:gravity**. Este atributo recebe como parâmetro o tipo do alinhamento desejado, conforme mostra a **Tabela 6.5**.

Alinhamento	Descrição
top	Alinhado ao topo (superior)
bottom	Alinhado na base (inferior)
left	Alinhado à esquerda
right	Alinhado à direita
center	Alinhado ao centro

Tabela 6.5. Parâmetros usados no atributo *android:gravity* para definir o alinhamento de uma View.

Veja abaixo o código de um arquivo de Layout usando o atributo **android:gravity** para alinhar componentes TextView:

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Texto alinhado ao centro."
      android:gravity="center" />

   <TextView
      android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Texto alinhado à direita"
        android:gravity="right" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Texto alinhado à esquerda."
        android:gravity="left" />
</LinearLayout>
```

O arquivo de Layout acima faz o alinhamento de três componentes TextView: ao centro, à direita e à esquerda. Veja na **Figura 6.20** o código de exemplo em execução, na tela do dispositivo.

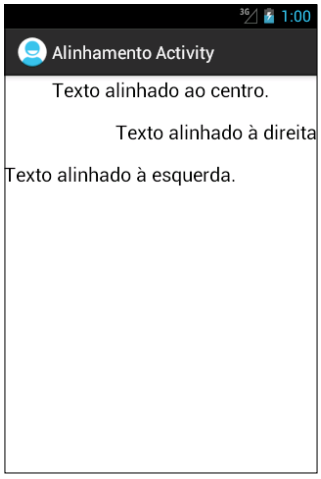


Figura 6.20. Resultado do uso do atributo *android:gravity* para alinhar Views na tela do dispositivo.

6.7 Definindo o peso de uma View

Outro atributo importante que devemos conhecer em uma View é o **android:layout_weight**, que define a importância (peso) de uma determinada View. Através deste atributo podemos definir que uma determinada View tem mais importância (maior peso) que outra View, definindo como parâmetro o valor do seu peso.

Veja abaixo o código de um arquivo de Layout onde definimos que a View EditText tem maior peso que a TextView, ou seja, a View EditText irá ocupar um espaço maior na tela em relação à TextView.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Nome: " />

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_weight="4"
            android:inputType="text" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="E-mail: " />

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="4"
            android:inputType="text" />
    </LinearLayout>

</LinearLayout>
```

Veja na **Figura 6.21** o resultado da execução do código acima na tela do dispositivo. Perceba que a View EditText ocupa 4 vezes o espaço da TextView.

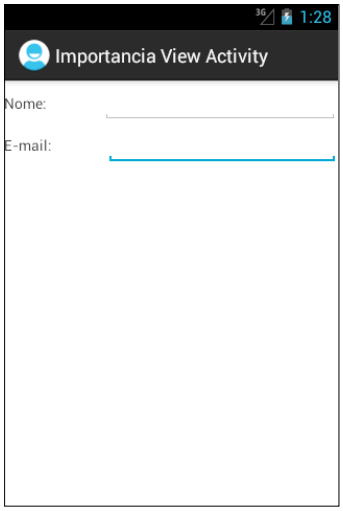


Figura 6.21. Usando o atributo *android:layout_weight* para definir maior importância à View EditText, deixando-a maior que a TextView.

6.8 Exercício

Agora que você aprendeu a trabalhar com interfaces gráficas no Android, é hora de colocar em prática.

1. Crie um novo projeto do Android, chamado ***interface_grafica***.
2. No novo projeto crie uma nova Activity, chamada ***InterfaceGraficaActivity***.
3. Crie um arquivo de Layout para a nova Activity, chamado ***activity_interface_grafica.xml***.
4. Na Activity ***InterfaceGraficaActivity*** defina o arquivo de Layout para ***activity_interface_grafica.xml***, através do método ***setContentView()***.
5. No novo arquivo de Layout, defina o ElementRoot (View principal) para ***ScrollView***, com a seguinte configuração:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

      ...

</ScrollView>
```

6. Dentro da ScrollView, defina uma View filha do tipo ***LinearLayout*** com orientação vertical e ocupando todo o espaço da tela (horizontal e verticalmente), deixando-a com a seguinte configuração:

```
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >

         ...

</LinearLayout>
```

7. Dentro do ***LinearLayout***, crie as seguintes Views:

```
<ImageView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:paddingTop="10.0dip"
         android:src="@drawable/ic_launcher" />

      <TextView
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:paddingTop="10.0dip"
         android:text="Nome:" />

      <EditText
         android:id="@+id/nome"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />

      <TextView
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:paddingTop="10.0dip"
         android:text="Sexo:" />

      <RadioGroup
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:orientation="horizontal" >

         <RadioButton
            android:id="@+id/sexo_masculino"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Masculino" />

         <RadioButton
            android:id="@+id/sexo_feminino"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Feminino" />
      </RadioGroup>
```

```
<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:paddingTop="10.0dip"
   android:text="Data de nascimento: " />

<DatePicker
   android:id="@+id/data_nascimento"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:calendarViewShown="false" />

<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:paddingTop="10.0dip"
   android:text="Confirme se for maior de 18 anos: " />

<CheckBox
   android:id="@+id/maior_idade"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Declaro ter mais de 18 anos" />

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:gravity="center_horizontal|center_vertical"
   android:orientation="horizontal"
   android:paddingTop="10.0dip" >
   <Button
      android:id="@+id/salvar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Salvar" />

   <Button
      android:id="@+id/sair"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Sair" />
</LinearLayout>
```

Nota

A View que você criou é basicamente um formulário onde o usuário deverá preencher os campos e, ao clicar no botão salvar, deverá ser apresentada uma outra tela mostrando as informações preenchidas pelo usuário. Para mostrar esta segunda tela, com o resultado dos campos preenchidos, será necessário criar uma nova Activity e definir um novo arquivo de Layout para ela.

8. Crie uma nova Activity, chamada ***InterfaceGraficaResultadoActivity***.

9. Crie um arquivo de Layout para a nova Activity , chamado ***activity_interface_grafica_resultado.xml***, com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal" >
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nome: " />

        <TextView
            android:id="@+id/nome_informado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Sexo: " />

        <TextView
            android:id="@+id/sexo_informado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Data de nascimento: " />

        <TextView
            android:id="@+id/data_nascimento_informada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Maior de idade: " />

        <TextView
            android:id="@+id/maior_idade_informado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>

</LinearLayout>
```

10. Na Activity ***InterfaceGraficaResultadoActivity*** use o método ***setContentView()*** para definir o arquivo de Layout para ***activity_interface_grafica_resultado.xml***.

11. Voltando para a Activity ***InterfaceGraficaActivity***, edite-a deixando-a com o seguinte conteúdo:

```
public class InterfaceGraficaActivity extends Activity {

	private EditText nome;
	private CheckBox maiorIdade;
	private RadioButton sexoMasculino;
	private DatePicker dataNascimento;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_interface_grafica);

		// Referenciando as Views através de seus IDs
		nome = (EditText) findViewById(R.id.nome);
		maiorIdade = (CheckBox) findViewById(R.id.maior_idade);
		sexoMasculino = (RadioButton) findViewById(R.id.sexo_masculino);
		dataNascimento = (DatePicker) findViewById(R.id.data_nascimento);

		Button salvar = (Button) findViewById(R.id.salvar);
		// Evento do botão Salvar
		salvar.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// Recupera os valores informados pelo usuário
				String valorNome = nome.getText().toString();
				String valorMaiorIdade = maiorIdade.isChecked() ? "sim" : "não";
				String valorSexo = sexoMasculino.isChecked() ? "Masculino" : "Feminino";
				String valorDataNascimento = dataNascimento.getDayOfMonth() +
						"/" + (dataNascimento.getMonth() + 1) + "/" +
						dataNascimento.getYear();

				// Define parâmetros a serem passados para a Activity de resultado
				Intent i = new Intent(getApplicationContext(),
						InterfaceGraficaResultadoActivity.class);
				i.putExtra("nome", valorNome);
				i.putExtra("maiorIdade", valorMaiorIdade);
				i.putExtra("sexo", valorSexo);
				i.putExtra("dataNascimento", valorDataNascimento);


				// Inicia a Activity com o resultado
				startActivity(i);

			}

		});

		Button sair = (Button) findViewById(R.id.sair);

		// Evento do botão Sair
		sair.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// O botão Sair apenas finaliza a Activity
				finish();
			}
		});
	}

}
```

12. Edite também a Activity ***InterfaceGraficaResultadoActivity***, deixando-a com o seguinte conteúdo:

```
public class InterfaceGraficaResultadoActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_interface_grafica_resultado);

        // Trata valores recebidos
        String valorNome = getIntent().getExtras().getString("nome");
        String valorMaiorIdade = getIntent().getExtras().getString("maiorIdade");
        String valorSexo = getIntent().getExtras().getString("sexo");
        String valorDataNascimento = getIntent().getExtras().getString("dataNascimento");

        // Define valores nas Views a serem apresentadas na tela
        TextView nome = (TextView) findViewById(R.id.nome_informado);
        nome.setText(valorNome);

        TextView sexo = (TextView) findViewById(R.id.sexo_informado);
        sexo.setText(valorSexo);

        TextView dataNascimento = (TextView) findViewById(R.id.data_nascimento_informada);
        dataNascimento.setText(valorDataNascimento);

        TextView maiorIdade = (TextView) findViewById(R.id.maior_idade_informado);
        maiorIdade.setText(valorMaiorIdade);

    }

}
```

13. No Eclipse, em "**Run**" > "**Run Configurations**", configure o projeto para que a Activity ***InterfaceGraficaActivity*** execute ao iniciar o aplicativo no emulador.

14. Execute o aplicativo no emulador do Android e faça o teste do formulário.

6.9 Exercício

Neste exercício você irá criar uma ListView usando um layout predefinido pelo Android. A ListView deverá ter apenas dois itens: uma opção para abrir a Activity ***InterfaceGraficaActivity***, criada no exercício anterior, e uma opção para sair (finalizar a Activity).

1. No novo projeto ***interface_grafica*** que você criou, crie uma nova Activity, chamada ***ListViewActivity***, com o seguinte conteúdo:

```
public class ListViewActivity extends ListActivity {

    private static final String[] OPCOES_DO_MENU = new String[] {
        "Trabalhando com interfaces gráficas",
        "Sair"
    };

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    /**
     * Define o Adapter que irá mostrar os dados na ListView.
     */
    setListAdapter(new ArrayAdapter<String>(
        this, android.R.layout.simple_list_item_1,
```

```
                        OPCOES_DO_MENU));
    }

    /**
     * Evento a ser disparado ao clicar em um dos itens da ListView.
     */
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        case 0:
                startActivity(new Intent(this, InterfaceGraficaActivity.class));
                break;
        default:
                finish();
        }
    }

}
```

2. No Eclipse, em "**Run**" > "**Run Configurations**", configure o projeto para que a Activity ***ListViewActivity*** execute ao iniciar o aplicativo no emulador.

3. Execute o aplicativo no emulador do Android e faça o teste do formulário.

6.10 Exercício opcional

Neste exercício você irá criar uma ListView usando um **layout customizado**. Esta ListView será semelhante à do exercício anterior, porém conterá uma ImageView e uma TextView em cada item da lista.

1. Crie um novo arquivo de layout, chamado ***activity_listview_customizado.xml***, com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/imagem_activity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="center_vertical" />

    <TextView
        android:id="@+id/nome_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:layout_gravity="center_vertical"
        android:paddingLeft="10.0dip" />

</LinearLayout>
```

2. Crie uma nova classe que dará origem ao objeto que será apresentado em cada item da ListView, com uma *String* (para mostrar o nome da Activity) e um inteiro primitivo (para referenciar uma Drawable Resource). A nova classe deverá ser chamada de ***ListViewCustomizadoItem*** e deverá ter o seguinte conteúdo:

```
public class ListViewCustomizadoItem {

        // Nome da Activity a ser apresentado na ListView
```

```
        private String nomeActivity;

        // Imagem da Activity a ser apresentada na ListView
        private int imagemActivity;

        public ListViewCustomizadoItem(String nomeActivity, int imagemActivity) {
                this.nomeActivity = nomeActivity;
                this.imagemActivity = imagemActivity;
        }

        public String getNomeActivity() {
                return nomeActivity;
        }

        public int getImagemActivity() {
                return imagemActivity;
        }

}
```

3. Crie a classe do *Adapter* que fará a ligação entre os dados a serem apresentados na tela e a ListView, chamado de ***ListViewCustomizadoAdapter***, com o seguinte conteúdo:

```
public class ListViewCustomizadoAdapter extends BaseAdapter {

        private Context contexto;

        // Lista dos itens a serem apresentados na tela
        private List<ListViewCustomizadoItem> itens;

        public ListViewCustomizadoAdapter(Context contexto, List<ListViewCustomizadoItem> itens) {
                this.contexto = contexto;
                this.itens = itens;
        }

        /**
         * Quantidade de itens a serem apresentados na ListView.
         */
        @Override
        public int getCount() {
                return itens.size();
        }

        /**
         * Posição atual de um item.
         */
        @Override
        public Object getItem(int posicao) {
                return itens.get(posicao);
        }

        /**
         * ID de um item. Em nosso caso usaremos sua posição como ID.
         */
        @Override
        public long getItemId(int position) {
                return position;
        }

        /**
         * Aqui é onde a mágica é feita: cada item da nossa lista de itens
         * será adaptado para ser apresentado em uma linha da ListView.
         * É aqui que montamos nossa View customizada.
```

```
         */
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
                // Item a ser apresentado na posição atual da ListView.
                ListViewCustomizadoItem item = itens.get(position);

                // Criando uma instância de View a partir de um arquivo de layout.
                LayoutInflater inflater = (LayoutInflater) contexto
                        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                View view = inflater.inflate(R.layout.activity_listview_customizado, null);

                // Imagem do item a ser apresentada na posição atual da ListView.
                ImageView imagem = (ImageView) view.findViewById(R.id.imagem_activity);
                imagem.setImageResource(item.getImagemActivity());

                // Texto do item a ser apresentado na posição atual da ListView.
                TextView texto = (TextView) view.findViewById(R.id.nome_activity);
                texto.setText(item.getNomeActivity());

                return view;
        }

}
```

4. Crie a Activity que mostrará a ListView na tela do dispositivo, chamada ***ListViewCustomizadoActivity***, com o seguinte conteúdo:

```
public class ListViewCustomizadoActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Cria uma lista com os itens a serem mostrados na tela.
         */
        List<ListViewCustomizadoItem> itens = new ArrayList<ListViewCustomizadoItem>();
        itens.add(new ListViewCustomizadoItem("Trabalhando com interfaces gráficas", R.drawable.ic_launcher));
        itens.add(new ListViewCustomizadoItem("Sair", R.drawable.ic_launcher));

        /**
         * Define o Adapter que irá mostrar os dados na ListView.
         */
        setListAdapter(new ListViewCustomizadoAdapter(this, itens));
    }

    /**
     * Evento a ser disparado ao clicar em um dos itens da ListView.
     */
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {

            switch (position) {
            case 0:
                    startActivity(new Intent(this, InterfaceGraficaActivity.class));
                    break;
            default:
                    finish();
            }
    }

}
```

5. No Eclipse, em "**Run**" > "**Run Configurations**", configure o projeto para que a Activity ***ListViewCustomizadoActivity*** execute ao iniciar o aplicativo no emulador.

6. Execute o aplicativo no emulador do Android e faça o teste do formulário.

7 - O projeto "Devolva.me"

Agora que você já conhece o básico do desenvolvimento de aplicativos para o Android, nos próximos capítulos construiremos um aplicativo na prática, que evoluirá conforme aprendemos mais recursos da plataforma.

O projeto em que iremos trabalhar será desenvolvido para um cliente fictício, chamado José, e servirá para que você se familiarize melhor com o desenvolvimento de aplicativos para o Android.

7.1 A história

José é um grande colecionador de **objetos** e, vez ou outra, algum **amigo pega emprestado um de seus objetos**. O grande problema disso é que José não tem a memória muito boa, portanto precisa sempre anotar o nome da **pessoa que pegou um objeto emprestado**, a **data** e qual foi o **objeto emprestado**.

Mesmo anotando em seu caderno as informações sobre o objeto emprestado, José notou que ainda assim alguns objetos nunca eram devolvidos, pois anotações eram perdidas e às vezes ele também se esquecia de pedir de volta um objeto que emprestou, caindo no esquecimento.

Pensando em uma forma mais prática e precisa para gerenciar seus objetos emprestados, José teve a grande idéia de encomendar um aplicativo para seu Smartphone, cuja plataforma é o Android.

José agora está disposto a pagar para resolver seu problema e então vai à procura de um bom profissional para desenvolver um aplicativo que o ajude nesta tarefa e, nessa história, esse profissional é você.

Agora que você entendeu a necessidade do seu cliente, o José, é hora de definir o projeto.

7.2 Definição do projeto

De acordo com a necessidade do cliente, deverão ser feitas as seguintes considerações:

1. Para registrar os objetos emprestados, José precisará de uma tela de cadastro, que deverá registrar as informações do objeto emprestado;
2. Para gerenciar os objetos emprestados, José precisará de uma tela listando as informações de todos os objetos que ele emprestou, possibilitando a edição dos registros;
3. Para facilitar o acesso às funcionalidades do aplicativo será necessário um menu de opções na tela inicial.

Como trata-se de um aplicativo para gerenciar empréstimos e devoluções de objetos, chamaremos o nosso aplicativo de ***Devolva.me***. [*Nome bem sugestivo, não?]*

Nota

Os registros deverão ser armazenados em um banco de dados. *[Aprenderemos sobre o armazenamento de dados no próximo capítulo.]*

7.2.1 Tela de cadastro

A tela de cadastro do aplicativo deverá ser simples e funcional. O cadastro será usado para registrar as seguintes informações:

- **Nome do objeto emprestado;**
- **Nome da pessoa para quem o objeto foi emprestado;**
- **Telefone da pessoa para quem o objeto foi emprestado;**
- **Data em que o objeto foi emprestado.**

Em um simples rascunho definimos a tela de cadastro do aplicativo, conforme mostra a **Figura 7.1**.

Figura 7.1. Tela de cadastro do aplicativo *Devolva.me*.

7.2.2 Tela com lista dos registros salvos

Para que José possa visualizar os objetos que emprestou, deverá ser feito uma simples tela com fácil acesso às informações. Em um simples rascunho definimos a tela com a lista dos registros salvos, conforme mostra a **Figura 7.2**.

Figura 7.2. Tela com a lista dos registros salvos, no aplicativo *Devolva.me*.

7.2.3 Tela inicial

O aplicativo deverá ter uma tela inicial com um menu para acesso às funcionalidades do aplicativo. Em um simples rascunho definimos a tela inicial, conforme mostra a **Figura 7.3**.

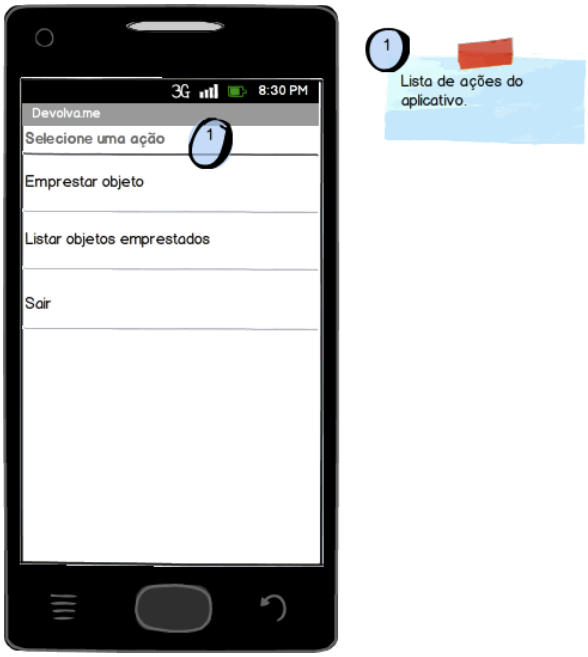


Figura 7.3. Tela inicial do aplicativo *Devolva.me*.

Por enquanto nosso aplicativo contemplará apenas as funcionalidades que listamos neste capítulo. Posteriormente iremos melhorar o projeto, explorando outros recursos do Android. *[Nos próximos capítulos iremos construir o aplicativo Devolva.me]*

7.3 Exercício

Para prosseguir com o curso, você deverá criar um novo projeto do Android, chamado ***devolva_me***.

1. Crie um novo projeto do Android, chamado ***devolva_me***, definindo o nome do pacote padrão para:

   br.com.hachitecnologia.devolvame.

8 - Armazenando informações no banco de dados

Muitas vezes precisamos persistir os dados de um aplicativo em um banco de dados. No Android isto é possível pois ele provê suporte nativo ao banco de dados SQLite.

O SQLite é um banco de dados Open-Source embutido, um “mini-SGDB”, que permite a manipulação de dados através de instruções SQL.

Uma característica muito importante é que no Android cada aplicativo possui o seu banco de dados exclusivo e este banco de dados é totalmente seguro, pois apenas o aplicativo que o criou poderá acessá-lo. Esta segurança é possível devido ao fato de cada aplicativo possuir um usuário exclusivo no Android, e cada usuário tem acesso apenas ao seu banco de dados. *[Lembre-se que os usuários do Android são gerenciados pelo Linux, que roda na camada mais baixa do Android]*

8.1 A classe SQLiteOpenHelper

Em um aplicativo para WEB ou DESKTOP normalmente criamos diretamente no banco de dados a estrutura de dados que a aplicação irá usar. No Android o ideal é encapsular o processo de criação e atualização do banco de dados dentro do próprio aplicativo, que criará toda a estrutura de dados necessária no momento em que o aplicativo for inicializado pela primeira vez.

Para que um aplicativo crie a estrutura do banco de dados o Android disponibiliza uma classe chamada ***SQLiteOpenHelper***, que irá prover os recursos necessários para que o aplicativo defina uma estrutura de dados. Esta classe deve ser estendida e os seguintes métodos devem ser implementados:

• ***onCreate()***: método responsável por criar a estrutura do banco de dados;

• ***onUpgrade()***: método responsável por atualizar a estrutura do banco de dados, caso ela precise de modificação.

Veja o exemplo de uma classe usada para definir a estrutura do banco de dados, estendendo de ***SQLiteOpenHelper***:

```
public class DBHelper extends SQLiteOpenHelper {

        private static final String NOME_DO_BANCO = "devolva_me";
        private static int VERSAO_DO_BANCO = 1;

        public DBHelper(Context context) {
                super(context, NOME_DO_BANCO, null, VERSAO_DO_BANCO);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
                // Cria a estrutura do banco de dados
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                // Atualiza a estrutura do banco de dados
        }

}
```

Nota

Perceba que no construtor da nossa classe invocamos o construtor da super classe ***SQLiteOpenHelper*** passando como parâmetro:

- **NOME_DO_BANCO**: o nome que será dado ao nosso arquivo de banco de dados;
- **VERSAO_DO_BANCO**: a versão atual do nosso banco de dados. Caso essa estrutura precise ser atualizada, apenas incrementamos este número para que o Android execute o código do método ***onUpgrade()***, e isto fará com que a estrutura de dados seja atualizada.

8.2 Inserindo dados

Para inserir dados em uma tabela do banco de dados, no Android, usamos o método ***insert()*** da classe ***SQLiteDatabase***. O método ***insert()*** possui a seguinte sintaxe:

```
insert(NOME_DA_TABELA, null, VALORES);
```

Onde:

- NOME_DA_TABELA = nome da tabela onde queremos inserir os dados;
- VALORES = objeto do tipo ***ContentValues*** com os valores a serem inseridos na tabela do banco de dados.

Nota

O objeto ***ContentValues*** é basicamente um mapa contendo chave/valor, onde a chave é o nome do campo da tabela e o valor é o dado que queremos inserir.

Veja, no trecho de código abaixo, um exemplo de como inserir dados no banco de dados SQLite em um aplicativo do Android:

```
. . .
// Encapsula em um objeto do tipo ContentValues os valores a serem persistidos no banco de dados
ContentValues values = new ContentValues();
values.put("objeto", "Caneta");
values.put("pessoa", "Pedro");
values.put("telefone", "8111-1111");

// Instancia uma conexão com o banco de dados, em modo de gravação
SQLiteDatabase db = getWritableDatabase();
db.insert("objeto_emprestado", null, values);
. . .
```

Nota

O método ***getWritableDatabase()*** da classe ***SQLiteOpenHelper*** permite que seja aberta uma conexão em modo de escrita (gravação) no banco de dados.

8.3 Consultando dados

Para consultar os dados armazenados no banco de dados, no Android, usamos o método ***query()*** da classe ***SQLiteDatabase***. O método ***query()*** possui a seguinte sintaxe:

```
query(NOME_DA_TABELA, null, null, null, null, null, ORDEM);
```

Onde:

- NOME_DA_TABELA = nome da tabela que contém o registro que queremos alterar;
- ORDEM = ordem do resultado da consulta.

Veja, no trecho de código abaixo, um exemplo de como consultar dados armazenados no banco de dados SQLite em um aplicativo do Android:

```
...
// Abre uma conexão com o banco de dados em modo leitura
SQLiteDatabase db = getReadableDatabase();
// Executa a consulta no banco de dados, ordenando pelo campo "objeto" em ordem ascendente
Cursor c = db.query("objeto_emprestado", null, null, null, null, null, "objeto ASC");

// Percorre o cursor para ler os dados  consultados
while (c.moveToNext()) {
        int id = c.getLong(c.getColumnIndex("_id"));
        String objeto = c.getString(c.getColumnIndex("objeto"));
        String nomePessoa = c.getString(c.getColumnIndex("pessoa"));
        String telefone = c.getString(c.getColumnIndex("telefone"));
        . . .
}
. . .
```

Perceba que, ao chamar o método ***query()***, passamos como primeiro parâmetro o nome da tabela do banco de dados que queremos realizar a consulta e no último parâmetro informamos o nome do campo que queremos ordenar os dados consultados e o tipo de ordenação (***ASC*** para ascendente, ***DESC*** para descendente).

Os dados consultados são retornados para um objeto ***Cursor*** e, para obter os dados consultados, basta fazer a iteração sobre este objeto e extraí-los, assim como fizemos usando um *while*.

Nota

Um ***Cursor*** é um apontador de registros que irá apontar pra um determinado registro consultado, de forma sequencial.

Nota

O método ***getReadableDatabase()*** da classe ***SQLiteOpenHelper*** permite que seja aberta uma conexão em modo apenas de leitura no banco de dados.

8.4 Alterando dados

Para alterar um determinado registro salvo em uma tabela do banco de dados, no Android, usamos o método ***update()*** da classe ***SQLiteDatabase***. O método ***update()*** possui a seguinte sintaxe:

```
update(NOME_DA_TABELA, VALORES, CLAUSULA_WHERE, String[] {ARGUMENTOS_DA_CLAUSULA_WHERE});
```

Onde:

- NOME_DA_TABELA = nome da tabela que contém o registro que queremos alterar;
- VALORES = objeto do tipo ***ContentValues*** com os novos valores do registro a serem alterados na tabela do banco de dados;
- CLAUSULA_WHERE = cláusula WHERE que irá especificar os registros a serem alterados. Neste parâmetro, usamos sempre a sintaxe: ***CAMPO = VALOR*** ou ***CAMPO = ?*** (o interrogação é o caractere curinga que será substituído pelos valores contidos no array de String passado como último parâmetro ao método ***update()***, ou seja, será substituído pelos argumentos da cláusula WHERE). Como exemplo, podemos usar neste parâmetro o valor: "_id = ?" (para alterar um registro com um determinado ID).
- String[] {ARGUMENTOS_DA_CLAUSULA_WHERE}: array de String com os argumentos a serem injetados na cláusula WHERE, caso tenhamos usado um caractere curinga. Como exemplo, podemos usar neste parâmetro o

valor: String[] {"1"} (que irá substituir o primeiro caractere curinga da cláusula WHERE pelo valor "1", e assim sucessivamente).

8.5 Removendo dados

Para remover um determinado registro salvo em uma tabela do banco de dados, no Android, usamos o método ***delete()*** da classe ***SQLiteDatabase***. O método ***delete()*** possui a seguinte sintaxe:

```
delete(NOME_DA_TABELA, CLAUSULA_WHERE, String[] {ARGUMENTOS_DA_CLAUSULA_WHERE});
```

Onde:

- NOME_DA_TABELA = nome da tabela que contém o registro que queremos remover;
- CLAUSULA_WHERE = cláusula WHERE que irá especificar os registros a serem removidos. Neste parâmetro, usamos sempre a sintaxe: ***CAMPO = VALOR*** ou ***CAMPO = ?*** (o interrogação é o caractere curinga que será substituído pelos valores contidos no array de String passado como último parâmetro do método ***delete()***, ou seja, será substituído pelos argumentos da cláusula WHERE). Como exemplo, podemos usar neste parâmetro o valor: "_id = ?" (para deletar um registro com um determinado ID).
- String[] {ARGUMENTOS_DA_CLAUSULA_WHERE}: array de String com os argumentos a serem injetados na cláusula WHERE, caso tenhamos usado um caractere curinga. Como exemplo, podemos usar neste parâmetro o valor: String[] {"1"} (irá substituir o primeiro caractere curinga da cláusula WHERE pelo valor "1", e assim sucessivamente).

8.6 Colocando em prática: usando banco de dados no projeto *Devolva.me*

Agora que aprendemos a trabalhar com banco de dados no Android, vamos implementar o recurso de armazenamento de dados no aplicativo *Devolva.me*, definido no **Capítulo 7**.

8.6.1 Definindo a estrutura de dados

O projeto *Devolva.me* irá utilizar o banco de dados para armazenar seus dados. Sua estrutura conterá apenas uma única tabela, chamada **objeto_emprestado**, com os seguintes campos:

- **_id**: identificador do registro (chave primária);
- **objeto**: nome do objeto que foi emprestado;
- **pessoa**: pessoa que pegou o objeto emprestado;
- **telefone**: número de telefone da pessoa que pegou o objeto emprestado;
- **data_emprestimo**: data em que o objeto foi emprestado

Veja na **Figura 8.1** o diagrama da entidade que iremos usar para o aplicativo:

objeto_emprestado	
_id	INTEGER
objeto	TEXT
pessoa	TEXT
telefone	TEXT
data_emprestimo	INTEGER

Figura 8.1. Estrutura da única tabela do banco de dados do aplicativo *Devolva.me*.

8.6.1.1 Definindo o modelo

Para facilitar a manipulação dos registros dessa tabela no banco de dados e deixar nosso código orientado ao objeto, em nosso projeto ***devolva_me***, vamos criar duas classes:

1. **ObjetoEmprestado**: classe que conterá as informações do objeto emprestado;

2. **Contato**: classe que conterá as informações da pessoa que pegou o objeto emprestado.

O diagrama abaixo mostra a ligação entre as duas classes:

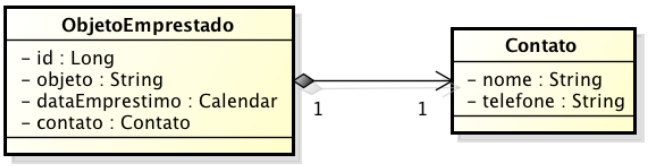


Figura 8.2. Diagrama de classe representando o modelo do projeto *Devolva.me*.

Ambas as classes serão criadas no pacote ***br.com.hachitecnologia.devolvame.modelo*** e serão simples POJOs.

A classe ***Contato*** será implementada com o seguinte código:

```
public class Contato implements Serializable {

	private static final long serialVersionUID = 1L;

	private String nome;
	private String telefone;

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getTelefone() {
		return telefone;
	}

	public void setTelefone(String telefone) {
		this.telefone = telefone;
	}

}
```

A classe ***ObjetoEmprestado*** será implementada com o seguinte código:

```
public class ObjetoEmprestado implements Serializable {

	private static final long serialVersionUID = 1L;

	private Long id;
	private String objeto;
	private Calendar dataEmprestimo;
	private Contato contato = new Contato();

	public Long getId() {
		return id;
```

```
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getObjeto() {
		return objeto;
	}

	public void setObjeto(String objeto) {
		this.objeto = objeto;
	}

	public Calendar getDataEmprestimo() {
		return dataEmprestimo;
	}

	public void setDataEmprestimo(Calendar dataEmprestimo) {
		this.dataEmprestimo = dataEmprestimo;
	}

	public Contato getContato() {
		return contato;
	}

	public void setContato(Contato contato) {
		this.contato = contato;
	}

}
```

8.6.2 Implementando a classe utilitária *DBHelper*

Precisamos agora criar uma classe que irá definir a estrutura da tabela ***objeto_emprestado*** e manipular seus dados. Nossa classe deverá estender a classe ***SQLiteOpenHelper*** e, no método ***onCreate()***, definir a estrutura de dados. Chamaremos esta classe de ***DBHelper***. Vamos então criar uma nova classe em nosso projeto, no pacote ***br.com.hachitecnologia.devolvame.dao***, chamada ***DBHelper***, com a seguinte implementação:

```
public class DBHelper extends SQLiteOpenHelper {

	// Nome do banco de dados
	private static final String NOME_DO_BANCO = "devolva_me";
	// Versão atual do banco de dados
	private static final int VERSAO_DO_BANCO = 1;

	public DBHelper(Context context) {
		super(context, NOME_DO_BANCO, null, VERSAO_DO_BANCO);
	}

	/**
	 * Cria a tabela no banco de dados, caso ela não exista.
	 */
	@Override
	public void onCreate(SQLiteDatabase db) {
		String sql = "CREATE TABLE objeto_emprestado (" +
				"_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT" +
				",objeto TEXT NOT NULL" +
				",pessoa TEXT NOT NULL" +
				",telefone TEXT NOT NULL" +
				",data_emprestimo INTEGER NOT NULL" +
				");";
		db.execSQL(sql);
```

```
        }

        /**
         * Atualiza a estrutura da tabela no banco de dados, caso sua versão tenha mudado.
         */
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                String sql = "DROP TABLE IF EXISTS objeto_emprestado";
                db.execSQL(sql);
                onCreate(db);
        }
}
```

Entendendo o código:

Alguns pontos importantes devem ser considerados em nossa classe ***DBHelper***:

1. Nossa classe estende a classe ***SQLiteOpenHelper***, implementando seus métodos ***onCreate()*** e ***onUpgrade()***;

2. Nossa classe define um inteiro primitivo como variável de instância, chamada **VERSAO_DO_BANCO**, para definir a versão do nosso banco de dados. Essa variável será nosso controlador de versão do banco e iniciaremos ela com o valor 1. Caso precisemos atualizar a estrutura do nosso banco de dados futuramente, basta incrementar esse valor;

3. Criamos um construtor de classe, recebendo o objeto ***Context*** como parâmetro, que usaremos para passar para o construtor da super classe ***SQLiteOpenHelper*** que herdamos, juntamente com o nome da tabela e sua versão;

4. O método ***onCreate()*** que subscrevemos faz a chamada para o método ***execSQL()*** da classe ***SQLiteDatabase***, que recebe como parâmetro apenas a SQL usada para criar a estrutura da nossa tabela. Quando o aplicativo for executado pela primeira vez no dispositivo, o Android irá chamar o método ***onCreate()*** e executar a SQL de instrução para criar a estrutura da nossa tabela;

5. O método ***onUpgrade()*** que subscrevemos também faz a chamada para o método ***execSQL()*** da classe ***SQLiteDatabase***, passando como parâmetro a instrução SQL para atualizar a estrutura da tabela no banco de dados. Em nosso caso apenas destruímos a estrutura da tabela antiga, usando um DROP, e chamamos explicitamente o método ***onCreate()*** para criar novamente a estrutura da tabela com suas devidas alterações. Perceba também que o método ***onUpgrade()*** recebe como parâmetro, além do objeto ***SQLiteDatabase***, dois primitivos inteiros com o valor da versão antiga e da versão atual da nossa tabela. Esses valores são passados automaticamente pelo Android, de acordo com o valor que informamos ao construtor da classe ***SQLiteDatabase*** que herdamos.

Dica

Caso alguma tabela do banco de dados em seu sistema precise de uma chave primária, defina um campo ID com o nome "**_id**" do tipo AUTOINCREMENT. O Android usa a convenção de nome "**_id**" no resultado de consultas realizadas nessas tabelas para serem utilizadas em ***CursorAdapters***.

8.6.3 Implementando o DAO

Para centralizar a responsabilidade de manipulação de dados, criaremos uma classe DAO específica para manipular os dados da tabela "**objeto_emprestado**". Esta classe será a responsável por inserir, atualizar, remover e consultar os dados dessa tabela. Criaremos, então, no pacote ***br.com.hachitecnologia.devolvame.dao***, uma nova classe chamada ***ObjetoEmprestadoDAO***, com a seguinte estrutura:

```
public class ObjetoEmprestadoDAO {

        private DBHelper dbHelper;
        private Context context;

        public ObjetoEmprestadoDAO(Context context) {
```

```
                dbHelper = new DBHelper(context);
                this.context = context;
        }

}
```

Nosso DAO ainda não possui nenhuma implementação. Nos próximos tópicos implementaremos o código que irá determinar suas responsabilidades.

8.6.4 Implementando a inserção de dados

8.6.4.1 Criando o método *adiciona()* no DAO

Para implementar a funcionalidade de inserção de dados em banco de dados, no nosso aplicativo, vamos criar no DAO ***ObjetoEmprestadoDAO*** um método para inserir registros na tabela que criamos. Nosso método se chamará ***adiciona()*** e conterá a seguinte implementação:

```
/**
 * Adiciona objeto no banco de dados.
 */
public void adiciona(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem persistidos no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("data_emprestimo", System.currentTimeMillis());
        values.put("pessoa", objeto.getContato().getNome());
        values.put("telefone", objeto.getContato().getTelefone());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Insere o registro no banco de dados
        long id = db.insert("objeto_emprestado", null, values);
        objeto.setId(id);

        // Encerra a conexão com o banco de dados
        db.close();
}
```

Dica

O método ***insert()*** da classe ***SQLiteDatabase*** retorna um long contendo o ID do registro que foi inserido no banco de dados.

Perceba que ao final do método ***adiciona()*** fechamos a conexão com o banco de dados, através do método ***close()***.

Dica

Sempre que abrir uma conexão com o banco de dados SQLite lembre-se de fechá-la no momento que não precisar mais de manipular seus dados.

8.6.4.2 Criando a Activity e a tela para cadastro dos dados

O nosso DAO já está apto para persistir os dados no banco de dados, porém agora precisamos de uma tela que irá permitir ao usuário inserir tais dados e de uma Activity pra tratar esses dados e repassá-los para o DAO gravá-los no banco de dados.

De acordo com a definição do projeto **Devolva.me**, vista no capítulo anterior, a implementação do cadastro deverá persistir as seguintes informações:

- **Nome do objeto emprestado;**

- **Nome da pessoa para quem o objeto foi emprestado;**
- **Número do telefone da pessoa para quem o objeto foi emprestado;**
- **Data em que o objeto foi emprestado.**

Com estas informações é fácil perceber que precisaremos, na tela de cadastro, de apenas três campos para que o usuário possa entrar com as informações: um campo para informar o **nome do objeto emprestado**, outro campo para informar o **nome da pessoa para quem o objeto foi emprestado** e um terceiro campo para informar o **número do telefone da pessoa para quem o objeto foi emprestado**. Estes três campos já são o suficiente, já que a **data em que o objeto foi emprestado** será definida automaticamente pelo sistema para diminuir o trabalho do nosso usuário e impedir que uma data errada seja informada.

De posse dessas informações, criaremos uma nova Activity para cadastro dos dados, chamada de ***CadastraObjetoEmprestadoActivity***. Ao criar nossa Activity, o plugin ADT do Eclipse automaticamente criará um Layout Resource chamado ***activity_cadastra_objeto_emprestado.xml***, localizado no diretório ***/res/layout***.

De acordo com o protótipo da tela de cadastro definido no **Capítulo 7**, as Views que iremos utilizar serão dispostas verticalmente na tela, portanto nosso ElementRoot (View principal) deverá ser uma ***LinearLayout*** com orientação ***vertical***. Iremos, então, alterar nosso Layout Resource ***activity_cadastra_objeto_emprestado.xml*** deixando seu ElementRoot da seguinte forma:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="#ffffff"
   android:orientation="vertical"
   android:paddingBottom="15.0dip"
   android:paddingLeft="15.0dip"
   android:paddingRight="15.0dip"
   android:paddingTop="15.0dip" >

        <!-- O conteúdo da tela virá aqui -->

</LinearLayout>
```

Definido nosso ElementRoot, iremos agora definir dentro dele os campos para inserção de dados e seus respectivos títulos. Para isto, utilizaremos as Views TextView (para o título dos campos) e EditText (para inserção de dados), ficando da seguinte forma:

```
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Objeto:" />

   <EditText
      android:id="@+id/cadastro_objeto_campo_objeto"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Informe o nome do objeto"
      android:inputType="text" >

      <requestFocus />
   </EditText>

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:paddingTop="15.0dip"
```

```
        android:text="Emprestado para:" />

    <EditText
        android:id="@+id/cadastro_objeto_campo_pessoa"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Informe o nome da pessoa"
        android:inputType="textPersonName" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="15.0dip"
        android:text="Telefone:" />

    <EditText
        android:id="@+id/cadastro_objeto_campo_telefone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Informe o telefone da pessoa"
        android:inputType="phone" />
```

E para finalizar nossa tela, dentro no nosso ElementRoot, precisaremos de basicamente dois botões:

- **Cancelar**: fechará a Activity atual, ao ser clicado;
- **Salvar**: chamará o DAO para persistir no banco de dados os dados informados pelo usuário.

Para deixar nossos botões centralizados e alinhados lado a lado devemos declará-los dentro de uma ViewGroup do tipo LinearLayout com orientação **horizontal** e alinhamento **centralizado**, ficando da seguinte forma:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal"
    android:paddingTop="15.0dip" >

    <Button
        android:id="@+id/cadastra_objeto_botao_salvar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Salvar" />

    <Button
        android:id="@+id/cadastra_objeto_botao_cancelar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cancelar" />

</LinearLayout>
```

Perceba que definimos um ID para cada botão. Este ID será utilizado para referenciarmos cada botão na nossa Activity e nos permitirá definir uma ação (usando um Listener) para cada um.

O código completo do nosso Layout Resource ***activity_cadastra_objeto_emprestado.xml*** ficará da seguinte forma:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffffff"
```

```
    android:orientation="vertical"
    android:paddingBottom="15.0dip"
    android:paddingLeft="15.0dip"
    android:paddingRight="15.0dip"
    android:paddingTop="15.0dip" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Objeto:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_objeto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o nome do objeto"
            android:inputType="text" >

            <requestFocus />
        </EditText>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Emprestado para:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_pessoa"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o nome da pessoa"
            android:inputType="textPersonName" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Telefone:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_telefone"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o telefone da pessoa"
            android:inputType="phone" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal"
            android:paddingTop="15.0dip" >

            <Button
                android:id="@+id/cadastra_objeto_botao_salvar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Salvar" />

            <Button
                android:id="@+id/cadastra_objeto_botao_cancelar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                android:text="Cancelar" />

        </LinearLayout>

    </LinearLayout>
```

Com nosso Layout Resource pronto precisamos agora definir o comportamento da nossa tela, e faremos isso através da nossa Activity ***CadastraObjetoEmprestadoActivity***. Essa Activity será responsável por definir o comportamento dos botões **Salvar** e **Cancelar** da nossa tela. Implementaremos, então, o seguinte código em nossa Activity:

```
public class CadastraObjetoEmprestadoActivity extends Activity {

    private ObjetoEmprestado objetoEmprestado;
    private EditText campoObjeto;
    private EditText campoNomePessoa;
    private EditText campoTelefone;
    private Button botaoSalvar;
    private Button botaoCancelar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Define o Layout Resource da Activity
        setContentView(R.layout.activity_cadastra_objeto_emprestado);

        campoObjeto = (EditText) findViewById(R.id.cadastro_objeto_campo_objeto);
        campoNomePessoa = (EditText) findViewById(R.id.cadastro_objeto_campo_pessoa);
        campoTelefone = (EditText) findViewById(R.id.cadastro_objeto_campo_telefone);
        botaoSalvar = (Button) findViewById(R.id.cadastra_objeto_botao_salvar);
        botaoCancelar = (Button) findViewById(R.id.cadastra_objeto_botao_cancelar);

       // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();

        /**
         * O botao salvar irá salvar o objeto no banco de dados.
         */
        botaoSalvar.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                    // Injeta no objeto "objetoEmprestado" os dados informados pelo usuário
                    objetoEmprestado.setObjeto(campoObjeto.getText().toString());
                    objetoEmprestado.getContato().setNome(campoNomePessoa.getText().toString());
                    objetoEmprestado.getContato().setTelefone(campoTelefone.getText().toString());

                    // Instancia o DAO para persistir o objeto
                    ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());
                    // Salva o objeto no banco de dados
                    dao.adiciona(objetoEmprestado);

                    // Mostra para o usuário uma mensagem de sucesso na operação
                    Toast.makeText(getApplicationContext(), "Objeto salvo com sucesso!", Toast.LENGTH_LONG)
                                .show();
            }

        });

        /**
         * O botao cancelar apenas finaliza a Activity.
         */
        botaoCancelar.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
```

```
                    finish();
                }
            });

        }
    }
```

Implementada a nossa Activity ***CadastraObjetoEmprestadoActivity*** o usuário já poderá cadastrar no sistema seus objetos emprestados, conforme foi definido no projeto **Devolva.me**. Veja na **Figura 8.3** o resultado da nossa nova Activity sendo executada no emulador do Android:

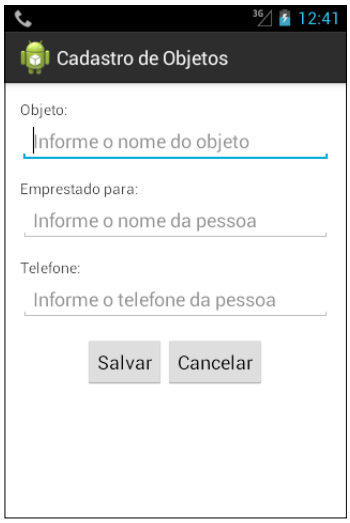


Figura 8.3. Tela de cadastro dos objetos emprestados no projeto *Devolva.me*.

8.6.5 Implementando a consulta de dados

8.6.5.1 Criando o método *listaTodos()* no DAO

Para implementar a funcionalidade de consulta de dados, no aplicativo *Devolva.me*, vamos implementar no DAO ***ObjetoEmprestadoDAO*** um método para consulta dos dados cadastrados na tabela do banco de dados, para permitir que nosso sistema liste estes dados para o usuário. Para isto, criaremos um novo método em nosso DAO, chamado de ***listaTodos***, com o seguinte conteúdo:

```
/**
 * Lista todos os registros da tabela “objeto_emprestado”
 */
public List<ObjetoEmprestado> listaTodos() {

        // Cria um List para guardar os objetos consultados no banco de dados
        List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

        // Instancia uma nova conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        // Executa a consulta no banco de dados
        Cursor c = db.query("objeto_emprestado", null, null, null, null,
                        null, "objeto ASC");

        /**
         *  Percorre o Cursor, injetando os dados consultados em um objeto
         *  do tipo ObjetoEmprestado e adicionando-os na List
         */
        try {
                while (c.moveToNext()) {
```

```
                        ObjetoEmprestado objeto = new ObjetoEmprestado();
                        objeto.setId(c.getLong(c.getColumnIndex("_id")));
                        objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));
                        objeto.getContato().setNome(c.getString(c.getColumnIndex("pessoa")));
                        objeto.getContato().setTelefone(c.getString(c.getColumnIndex("telefone")));
                        objetos.add(objeto);
                }

        } finally {
                // Encerra o Cursor
                c.close();
        }

        // Encerra a conexão com o banco de dados
        db.close();

        // Retorna uma lista com os objetos consultados
        return objetos;
}
```

8.6.5.2 Criando a Activity e a tela para listar os dados armazenados

Com o método de consulta já implementado em nosso DAO, precisamos criar agora uma tela para listar estes objetos salvos e sua Activity. Criaremos então uma nova Activity em nosso projeto, chamada ***ListaObjetosEmprestadosActivity***. Ao criar nossa nova Activity, o plugin ADT do Eclipse criará automaticamente seu Layout Resource com o nome ***activity_lista_objetos_emprestados.xml*** no diretório ***/res/layout***.

Em nossa tela, de acordo com o que foi definido no projeto **Devolva.me**, no **Capítulo 7**, precisaremos de uma View em forma de lista para listar as informações consultadas no banco de dados. Podemos usar uma **ListView** para listar na tela os dados consultados, porém não sabemos o número exato de itens que serão apresentados em nossa lista, pois isso irá depender da quantidade de registros salvos em nossa tabela do banco de dados. Para resolver essa situação podemos definir um Layout Resource com uma **ListView** vazia (sem nenhuma outra View dentro) e, então, definir um segundo Layout Resource definindo a forma como queremos apresentar cada item da lista. Assim, para cada registro localizado no banco de dados, podemos criar uma instância desse segundo Layout Resource injetando nele as informações desse registro e, através de um Adapter, inserir todos esses Layouts montados dinamicamente dentro da nossa ListView. *Complicado, não?!?!* A **Figura 8.4** irá ajudá-lo a visualizar melhor essa questão.

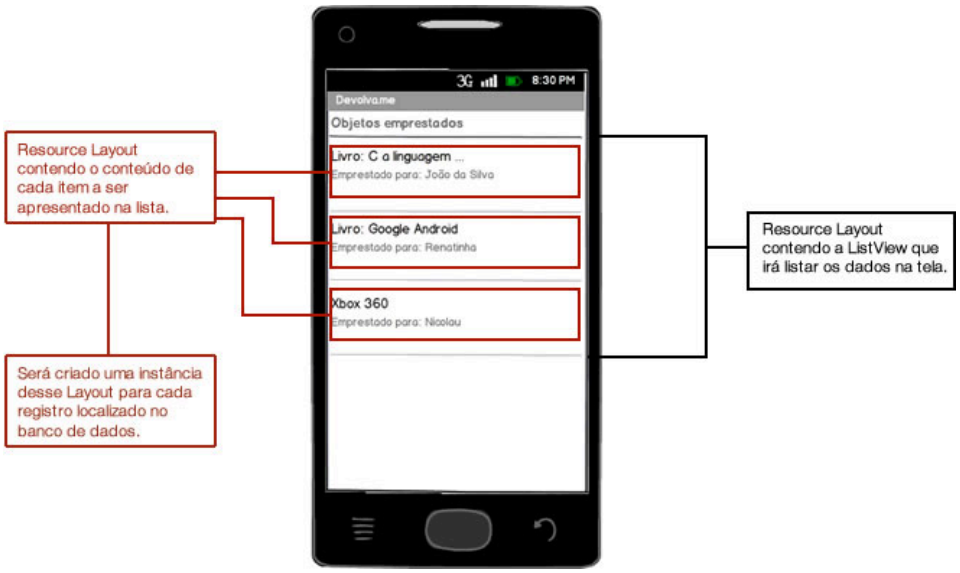


Figura 8.4. Estrutura usada para adaptar os itens consultados no banco de dados em uma ListView.

Iremos, então, alterar o nosso Layout Resource ***activity_lista_objetos_emprestados.xml*** definindo seu ElementRoot (View principal) como LinearLayout com orientação **vertical** e, dentro dela, usaremos apenas uma ListView vazia para listar os dados, deixando nosso layout da seguinte forma:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/lista_objetos_emprestados"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Nota

Perceba que definimos um ID para a nossa ListView. Isto nos permitirá manipular esta View através da nossa Activity, e faremos isto para injetar os dados consultados no banco de dados em nossa ListView.

Nosso arquivo de Layout principal está definido, porém precisamos definir um segundo Layout Resource para apresentar cada item consultado no banco de dados em nossa ListView. Vamos, então, criar um novo Layout Resource chamado ***item_lista_objetos_emprestados.xml*** no diretório ***/res/layout***, com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/item_objeto_emprestado_nome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textSize="22sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:textSize="15sp"
            android:text="Emprestado para:"
            android:paddingRight="5.0dip"/>

        <TextView
            android:id="@+id/item_objeto_emprestado_pessoa"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:textSize="15sp" />

    </LinearLayout>

</LinearLayout>
```

Com os nossos arquivos de Resource Layout já definidos, devemos agora implementar a Activity ***ListaObjetosEmprestadosActivity*** que irá pegar os registros consultados no banco de dados através do DAO e listá-los na tela, fazendo a adaptação do conteúdo dinamicamente em nossa ListView.

O primeiro passo em nossa Activity é definir o Resource Layout que será manipulado e referenciar a nossa View ListView do arquivo de Layout em um objeto do tipo ListView, conforme mostra o trecho de código abaixo:

```
...

/**
 * Variável de instância do tipo ListView que fará referência à ListView do
 * nosso Resource Layout e será manipulado via código para
 * ser preenchido com os dados consultados no banco de dados.
 */
private ListView listaObjetosEmprestados;

@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	// Definimos o Resource Layout a ser controlado pela Activity
	setContentView(R.layout.activity_lista_objetos_emprestados);
	/**
	 * Fazemos a ligação entre o objeto "listaObjetosEmprestados" e a View
	 * ListView do nosso Resource Layout.
	 */
	listaObjetosEmprestados = (ListView) findViewById(R.id.lista_objetos_emprestados);
}

...
```

Nota

Lembre-se de que só conseguimos fazer a ligação entre um objeto do tipo ListView à nossa View ListView do arquivo de Layout porque definimos anteriormente um ID para esta View.

Com o método ***onCreate()*** da Activity já definido e já fazendo referência à View ListView do nosso arquivo de Layout, precisamos agora definir o Adapter que irá adaptar cada registro consultado no banco de dados em uma instância da View ***item_lista_objetos_emprestados.xml*** e injetar todas essas instâncias em nossa ListView do arquivo de Layout ***activity_lista_objetos_emprestados.xml***. Para isto, usaremos o ArrayAdapter, que adapta um array de objetos na ListView, nos permitindo passar um ***List*** de objetos do tipo ***ObjetoEmprestado*** com os resultados consultados no banco de dados. Criaremos, então, um Adapter no pacote ***br.com.hachitecnologia.devolvame.adapter*** chamado ***ListaObjetosEmprestadosAdapter*** que estende a classe *ArrayAdapter*, com o seguinte código:

```
public class ListaObjetosEmprestadosAdapter extends ArrayAdapter<ObjetoEmprestado> {

	private final List<ObjetoEmprestado> objetosEmprestados;
	private final Activity activity;

	public ListaObjetosEmprestadosAdapter(Activity activity, int textViewResourceId,
			List<ObjetoEmprestado> objetosEmprestados) {
		super(activity, textViewResourceId, objetosEmprestados);
		this.activity = activity;
		/**
		 * List de objetos do tipo ObjetoEmprestado recebida como parâmetro
		 * pelo Construtor.
		 */
		this.objetosEmprestados = objetosEmprestados;
	}

	/**
```

```
	 * Aqui que é onde a mágica é feita: as Views TextView do
	 * Resource Layout "item_lista_objetos_emprestados.xml" são preenchidas com os dados
	 * consultados no banco de dados e este Resource Layout será retornado como um objeto
	 * do tipo View para ser adicionado na ListView do Resource
	 * Layout "activity_lista_objetos_emprestados.xml".
	 *
	 * Este método será chamado para cada
	 * registro consultado no banco de dados, fazendo com que a ListView mostre todos
	 * os registros econtrados.
	 */
	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		/**
		 * Obtemos cada objeto do tipo ObjetoEmprestado dentro da List recebida
		 * pelo Construtor, de acordo com a sua posição, para ser mostrado
		 * na mesma posição da ListView, ou seja, o primeiro objeto da List vai
		 * para a primeira posição da ListView e assim sucessivamente.
		 */
		ObjetoEmprestado objetoEmprestado = objetosEmprestados.get(position);

		/**
		 * Referenciamos o Resource Layout "item_lista_objetos_emprestados" em um objeto
		 * do tipo View, que será o objeto de retorno deste método. Esta é a View que será
		 * adaptada e retornada para ser apresentada na ListView.
		 */
		View view = activity.getLayoutInflater().inflate(R.layout.item_lista_objetos_emprestados, null);

		/**
		 *  Injeta o valor do campo referente ao nome do objeto emprestado, do
		 *  registro consultado no banco de dados, na TextView de ID "item_objeto_emprestado_nome".
		 */
		TextView objeto = (TextView) view.findViewById(R.id.item_objeto_emprestado_nome);
		objeto.setText(objetoEmprestado.getObjeto());

		/**
		 *  Injeta o valor do campo referente ao nome da pessoa para quem o
		 *  objeto foi emprestado, do registro consultado no banco de dados, na TextView
		 *  de ID "item_objeto_emprestado_pessoa".
		 */
		TextView pessoa = (TextView) view.findViewById(R.id.item_objeto_emprestado_pessoa);
		pessoa.setText(objetoEmprestado.getContato().getNome());

		// Retorna a View já adaptada para ser apresentada na ListView
		return view;
	}

	/**
	 * Retorna o ID de um determinado item da ListView, de acordo
	 * com a sua posição.
	 */
	@Override
	public long getItemId(int position) {
		return objetosEmprestados.get(position).getId();
	}

	/**
	 * Retorna o número de itens que serão mostrados na ListView.
	 */
	@Override
	public int getCount() {
		return super.getCount();
	}

	/**
```

```
		 * Retorna um determinado item da ListView, de acordo
		 * com a sua posição.
		 */
		@Override
		public ObjetoEmprestado getItem(int position) {
			return objetosEmprestados.get(position);
		}
	}
```

Com o nosso Adapter implementado, devemos agora modificar a nossa Activity ***ListaObjetosEmprestadosActivity*** para que ela consulte os objetos no banco de dados através do DAO e use o nosso Adapter para adaptar o resultado a ser apresentado na tela, em nossa ListView. Faremos essa implementação no método ***onResume()*** da nossa Activity, para que, caso nossa Activity vá para segundo plano (ou seja, quando outra Activity entrar à frente da nossa Activity - ao receber uma ligação telefônica, por exemplo), a consulta dos dados seja realizada novamente assim que a Activity retornar ao topo da pilha (ou seja, quando a nossa Activity ficar novamente visível ao usuário), mantendo a consistência dos dados. Subscrevemos, então, o método ***onResume()*** da nossa Activity, implementando o seguinte código:

```
	@Override
	protected void onResume() {
		super.onResume();

		// Consulta os objetos cadastrados no banco de dados através do DAO
		ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

		// Guarda os objetos consultados em uma List
		final List<ObjetoEmprestado> objetosEmprestados = dao.listaTodos();

		// Instancia o Adapter que irá adaptar os dados na ListView
		ArrayAdapter<ObjetoEmprestado> adapter = new ListaObjetosEmprestadosAdapter(this,
			android.R.layout.simple_list_item_1, objetosEmprestados);

		/**
		 * Define o Adapter que irá adaptar os dados consultados no banco de dados
		 * à nossa ListView do Resource Layout.
		 *
		 * @param adapter
		 */
		listaObjetosEmprestados.setAdapter(adapter);
	}
```

Agora nossa implementação já está pronta e podemos executar a nossa nova Activity ***ListaObjetosEmprestadosActivity*** para mostrar os dados cadastrados no banco de dados. Veja na **Figura 8.5** o resultado da nossa Activity em execução no emulador do Android.

Figura 8.5. Activity ***ListaObjetosEmprestadosActivity*** em execução no emulador do Android.

8.6.6 Implementando a alteração de dados

8.6.6.1 Criando o método *atualiza()* no DAO

Para implementar a funcionalidade de alteração dos dados, no aplicativo *Devolva.me*, vamos implementar no DAO ***ObjetoEmprestadoDAO*** um método chamado ***atualiza()*** que irá alterar os dados dos registros desejados. A implementação do nosso método ficará da seguinte forma:

```
/**
 * Altera o registro no banco de dados.
 */
public void atualiza(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem atualizados no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("pessoa", objeto.getContato().getNome());
        values.put("telefone", objeto.getContato().getTelefone());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Atualiza o registro no banco de dados
        db.update("objeto_emprestado", values, "_id=?", new String[] { objeto.getId().toString() });

        // Encerra a conexão com o banco de dados
        db.close();
}
```

8.6.6.2 Disponibilizando na tela uma opção para editar os dados salvos

De acordo com a definição do projeto **Devolva.me**, no **Capítulo 7**, a tela com a lista dos objetos emprestados deve dar a possibilidade do usuário selecionar um item para edição. Devemos, então, implementar tal funcionalidade, uma vez que consta na especificação do nosso projeto.

Uma maneira mais simples seria implementar uma ação na ListView onde, ao clicar em um dos itens listados, abrir a mesma tela que usamos no cadastro mas preenchida com os dados do registro selecionado e, ao clicar no botão Salvar, apenas atualizar o registro com os novos dados informados pelo usuário. Para isto, precisaremos realizar três tarefas:

1. Criar um método em nosso DAO que execute a ação de adicionar ou atualizar um registro no banco de dados, dependendo se o objeto recebido como parâmetro possua ou não um ID preenchido;

2. Alterar a nossa Activity ***CadastraObjetoEmprestadoActivity*** para que ela possa tanto adicionar quanto editar nossos registros no banco de dados;

3. Criar um Listener que irá disparar o evento ao clicar em um item da nossa ListView. Em nosso caso, esse evento simplesmente chamará a nossa Activity ***CadastraObjetoEmprestadoActivity*** passando como parâmetro o objeto a ser editado pelo usuário.

8.6.6.3 Definindo o Listener que permitirá a seleção do objeto a ser editado

Agora que sabemos quais alterações devemos fazer para implementar a edição dos registros salvos no banco de dados, vamos criar o Listener que chamará a Activity para edição do registro. Mas antes de criar o nosso Listener, devemos disponibilizar uma forma para que esse Listener tenha acesso ao objeto ListView para que ele possa acessar o objeto selecionado pelo usuário dentro da lista. Para isto, criaremos um ***getter*** chamado ***getListaObjetosEmprestados()*** ao final da nossa classe ***ListaObjetosEmprestadosActivity***, com a seguinte implementação:

```
...
```

```
/**
* Retorna o objeto ListView já preenchido com a lista de
* objetos emprestados.
*/
public ListView getListaObjetosEmprestados() {
        return listaObjetosEmprestados;
}

...
```

Implementado o ***getter*** para permitir o acesso ao objeto ListView com a lista dos objetos, criaremos agora uma nova classe chamada ***ListaObjetosEmprestadosListener*** no pacote ***br.com.hachitecnologia.devolvame.listener***. Essa classe será o nosso Listener, e conterá a seguinte implementação:

```
public class ListaObjetosEmprestadosListener implements AdapterView.OnItemClickListener {

        private final ListaObjetosEmprestadosActivity activity;

        /**
         * Nosso Construtor deverá receber como parâmetro a instância da Activity
         * ListaObjetosEmprestadosActivity para ter acesso ao objeto ListView contendo
         * a lista de objetos emprestados. Isto nos permitirá pegar na lista o item que foi selecionado
         * pelo usuário.
         *
         */
        public ListaObjetosEmprestadosListener(ListaObjetosEmprestadosActivity activity) {
                this.activity = activity;
        }

        /**
        * Método que será disparado quando o usuário selecionar o item na ListView.
        */
        public void onItemClick(AdapterView<?> adapterView, View view, int posicao, long idPosicao) {
                // Declarando a Intent que irá invocar a Activity "CadastraObjetoEmprestadoActivity"
                Intent i = new Intent(activity, CadastraObjetoEmprestadoActivity.class);

                /**
                 *  Passando o objeto selecionado pelo usuário, na ListView, como parâmetro
                 *  para a Activity "CadastraObjetoEmprestadoActivity", para que esta
                 *  possa editar as informações do registro selecionado.
                 */
                i.putExtra("itemSelecionadoParaEdicao",
                        (ObjetoEmprestado)activity.getListaObjetosEmprestados().getItemAtPosition(posicao));

                // Invoca a Activity definida em nossa Intent
                activity.startActivity(i);
        }
}
```

Nota

O método ***putExtra()*** da classe Intent irá enviar o objeto selecionado pelo usuário como parâmetro para a Activity ***CadastraObjetoEmprestadoActivity***. Em nosso exemplo, referenciamos este objeto com o nome "**itemSelecionadoParaEdicao**", e é este mesmo nome que será usado na Activity para recuperar o objeto recebido como parâmetro.

Nota

O método ***getItemAtPosition(int posição)*** da classe ListView serve para pegar um determinado objeto da lista de acordo com a posição informada como parâmetro. Em nosso caso, este método irá pegar o objeto que está na posição selecionada pelo usuário, na lista.

Com o nosso Listener implementado, precisamos agora ativá-lo em nossa Activity para que, quando o usuário clicar em um dos itens da lista, o evento ***onItemClick()*** do Listener seja disparado. Para isto, iremos adicionar a seguinte linha ao final do método ***onResume()*** da Activity ***ListaObjetosEmprestadosActivity***:

```
...
/**
* Ativa o Listener que contém o evento a ser disparado ao clicar
* sobre um item da nossa ListView.
*/
listaObjetosEmprestados.setOnItemClickListener(new ListaObjetosEmprestadosListener(this));
...
```

8.6.6.4 Persistindo os dados de forma mais inteligente

Para que o nosso DAO fique um pouco mais inteligente, iremos criar um método que decidirá se o objeto recebido como parâmetro deverá ser adicionado ou alterado no banco de dados. Criaremos, então, um método chamado ***salva()*** com a seguinte implementação:

```
/**
 * Salva objeto no banco de dados.
 * Caso o objeto não exista no banco de dados, ele o adiciona.
 * Caso o objeto exista no banco de dados, apenas atualiza os valores dos campos modificados.
 *
 * @param objeto
 */
public void salva(ObjetoEmprestado objeto) {
        /**
         *  Se o ID do objeto é nulo é porque ele ainda não existe no banco de dados,
         *  logo subentende-se que queremos adicionar o objeto no banco de dados.
         *  Sendo assim, chamaremos o método adiciona() já definido no DAO.
         */
        if (objeto.getId() == null) {
                adiciona(objeto);
        /**
         * Caso o objeto possua um ID é porque ele já existe no banco de dados, logo
         * subentende-se que queremos alterar seus dados no banco de dados.
         *  Sendo assim, chamaremos o método atualiza() já definido no DAO.
         */
        } else {
                atualiza(objeto);
        }
}
```

8.6.6.5 Adicionando à Activity "*CadastraObjetoEmprestadoActivity*" uma funcionalidade para edição dos dados

Com um método mais inteligente no DAO, chamado ***salva()***, que realiza tanto a tarefa de adicionar quanto a de alterar os registros no banco de dados, podemos agora deixar também a nossa Activity de cadastro um pouco mais inteligente, para que ela também possa realizar essas duas funções. Para isso, iremos alterar o método ***onCreate()*** da nossa Activity, substituindo a seguinte linha de código, localizada após a chamada do método ***setContentView(...)***:

```
// Instancia um novo objeto do tipo ObjetoEmprestado
objetoEmprestado = new ObjetoEmprestado();
```

por este trecho de código:

```
/**
* Recebe o objeto recebido como parâmetro da ListView para edição.
 */
objetoEmprestado = (ObjetoEmprestado) getIntent().getSerializableExtra("itemSelecionadoParaEdicao");
```

```
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoTelefone.setText(objetoEmprestado.getContato().getTelefone());
 }
```

No trecho de código acima fazemos a chamada para o método ***getSerializableExtra()*** da classe *Intent* para verificar se foi recebido de uma outra classe um parâmetro com o nome "**itemSelecionadoParaEdicao**", ou seja, o objeto a ser editado. Caso o parâmetro tenha sido recebido (tratado no ***else***, no trecho de código acima), preenchemos os campos do formulário da nossa tela com os dados contidos no objeto a ser editado. Caso contrário, ou seja, se nenhum parâmetro tiver sido recebido, subentende-se que usuário deseja cadatrar um novo registro, logo apenas inicializamos um novo objeto do tipo ***ObjetoEmprestado*** vazio (tratado no ***if***, no trecho de código acima).

Agora, para que nossa Acticity utilize o método ***salva()*** implementado no DAO, devemos apenas modificar a linha de código abaixo, presente no evento do botão "**Salvar**" da Activity ***CadastraObjetoEmprestadoActivity***:

```
// Salva o objeto no banco de dados
dao.adiciona(objetoEmprestado);
```

pela seguinte linha:

```
// Salva o objeto no banco de dados
dao.salva(objetoEmprestado);
```

Para finalizar, podemos pedir para nossa Activity ***CadastraObjetoEmprestadoActivity*** abrir a Activity ***ListaObjetosEmprestadosActivity*** sempre após executar a ação de salvar um registro no banco de dados, mostrando na tela as alterações realizadas. Para isto, vamos adicionar a seguinte linha ao final do método ***onClick()*** definido no Listener do botão salvar:

```
...
// Depois de salvar, vai para a Lista dos objetos emprestados
startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
...
```

Pronto! Agora nosso sistema já está apto para cadastrar e editar informações no banco de dados, seguindo à especificação definida no projeto ***Devolva.me*** no **Capítulo 7**. Para editar as informações já salvas no banco de dados, basta executar a Activity ***ListaObjetosEmprestadosActivity*** e clicar em um dos itens listados na tela.

8.6.7 Implementando a remoção de dados

8.6.7.1 Criando o método *remove()* no DAO

Para implementar a funcionalidade de remoção de dados, no aplicativo ***Devolva.me***, vamos implementar no DAO ***ObjetoEmprestadoDAO*** um método chamado ***remove()***, que irá remover um determinado registro. A implementação do nosso método ficará da seguinte forma:

```
/**
 * Remove um registro no banco de dados.
 */
public void remove(ObjetoEmprestado objeto) {
        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
        // Remove o registro no banco de dados
        db.delete("objeto_emprestado", "_id=?", new String[] { objeto.getId().toString() });

        // Encerra a conexão com o banco de dados
        db.close();
    }
```

8.6.7.2 Implementando na ListView uma opção para remover um registro

Implementado o método ***remove()*** no DAO, pronto para deletar um determinado registro no banco de dados, devemos agora implementar uma opção para que o usuário possa executar esta ação a partir de uma tela. Uma maneira mais simples seria adicionar um ***Context Menu*** à ListView que lista os registros salvos no banco de dados e adicionar neste menu uma opção "**Apagar**". Desta forma, quando o usuário pressionar um dos itens da ListView por 3 segundos, irá aparecer uma opção para que o mesmo possa apagar o registro selecionado.

Para implementar o *Context Menu*, precisaremos fazer algumas alterações na Activity ***ListaObjetosEmprestadosActivity***. A primeira alteração que devemos fazer é adicionar uma constante em nossa classe, que representará o ID da opção "**Apagar**" que iremos adicionar ao *Context Menu*. Vamos, então, adicionar a seguinte variável em nossa classe ***ListaObjetosEmprestadosActivity***:

```
// ID da opção "Apagar" do menu de contexto
private static final int MENU_APAGAR = Menu.FIRST;
```

O segundo passo é implementar o método que irá criar um *Context Menu* em nossa tela. Para isto, devemos subscrever o método ***onCreateContextMenu()*** em nossa Activity, adicionando a opção "**Apagar**" com o ID definido acima. Implementaremos, então, o seguinte método na Activity ***ListaObjetosEmprestadosActivity***:

```
/**
 * Cria o menu de contexto.
 */
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    // Adiciona a opção "Apagar" ao Context Menu
    menu.add(0, MENU_APAGAR, 0, "Apagar");
}
```

O terceiro passo é adicionar ao final do método ***onCreate()*** da nossa Activity uma chamada para o método ***registerForContextMenu()*** informando ao Android que queremos ativar um *Context Menu* para uma determinada View em nossa tela e, em nosso caso, queremos que o menu seja ativado na View ListView, referenciada através da variável ***listaObjetosEmprestados*** em nossa classe. Adicionaremos, então, a seguinte linha ao final do método ***onCreate()*** da Activity ***ListaObjetosEmprestadosActivity***:

```
...
// Registra o Context Menu para a ListView da nossa tela
registerForContextMenu(listaObjetosEmprestados);
...
```

Se executarmos nossa Activity, veremos que nosso menu está visualmente pronto, porém precisamos tratar o evento que será disparado quando o usuário clicar na opção "**Apagar**" do *Context Menu* que, em nosso caso, deverá apagar o registro no banco de dados chamando o método ***remove()*** do nosso DAO. Para implementar este evento, devemos subscrever o método ***onContextItemSelected()*** da nossa Activity, que será chamado quando o usuário pressionar a opção "**Apagar**"

no Context Menu de um determinado item da nossa ListView. Nosso método ***onContextItemSelected()*** terá três responsabilidades:

1. Identificar a opção do Context Menu selecionada pelo usuário, através do ID que definimos para esta opção. (Mesmo que nosso menu tenha apenas uma opção, sempre devemos identificar essa opção pelo seu ID que, em nosso caso, definimos anteriormente na variável de classe chamada **MENU_APAGAR**);

2. Apagar o registro no banco de dados através da chamada do método ***remove()*** do DAO;

3. Atualizar a ListView para que o registro removido no banco de dados não seja mais apresentado na tela. Para isto, devemos executar novamente a consulta no banco de dados e definir novamente o Adapter que irá apresentar estes dados em nossa ListView.

Perceba que nosso método ***onContextItemSelected()*** terá responsabilidades demais e, uma dessas responsabilidades já está implementada no código da Activity, que é a de consultar os dados no banco de dados e definir o Adapter que irá apresentar estes dados na ListView. Para deixar nossa Activity mais elegante e sem código duplicado, vamos centralizar a responsabilidade de consulta de dados e definição do Adapter em um só método e chamá-lo sempre que necessário. Vamos, então, criar um método chamado ***montaListView()*** com essa responsabilidade, deixando-o com a seguinte implementação:

```
/**
 * Monta a ListView com os dados a serem apresentados na tela.
 */
private void montaListView() {

	// Consulta os objetos cadastrados no banco de dados através do DAO
	ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

	// Guarda os objetos consultados em uma List
	final List<ObjetoEmprestado> objetosEmprestados = dao.listaTodos();

	// Instancia o Adapter que irá adaptar os dados na ListView
	ArrayAdapter<ObjetoEmprestado> adapter = new ListaObjetosEmprestadosAdapter(
				this, android.R.layout.simple_list_item_1, objetosEmprestados);

	/**
	 * Define o Adapter que irá adaptar os dados consultados no banco de
	 * dados à nossa ListView do Resource Layout.
	 *
	 * @param adapter
	 */
	listaObjetosEmprestados.setAdapter(adapter);

	/**
	 * Ativa o Listener que contém o evento a ser disparado ao clicar
	 * sobre um item da nossa ListView
	 */
	listaObjetosEmprestados.setOnItemClickListener(new ListaObjetosEmprestadosListener(this));
}
```

Agora podemos simplificar nosso método ***onResume()***, substituindo seu código atual pela seguinte implementação:

```
@Override
protected void onResume() {
	super.onResume();

	// Chama o método que irá montar a ListView na tela.
	montaListView();
}
```

Perceba que nosso método ***onResume()*** continua executando as mesmas ações, porém agora ele delega a responsabilidade de consulta de dados e definição do Adapter ao método ***montaListView()***, que poderá ser reaproveitado sempre que necessário.

Vamos, agora, criar um método que terá a responsabilidade de chamar o DAO para remover do banco de dados o item selecionado pelo usuário na ListView. Criaremos, então, um método chamado ***remove()*** na Activity ***ListaObjetosEmprestadosActivity*** com a seguinte implementação:

```
/**
 * Remove o objeto selecionado.
 */
private void remove(ObjetoEmprestado objeto) {
        // Cria uma nova instância do DAO
        ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

        // Remove o objeto do banco de dados
        dao.remove(objeto);

        /**
         * Atualiaza a ListView para que o objeto removido não seja mais
         * apresentado na tela.
         */
        montaListView();
}
```

Perceba que o método ***remove()*** que criamos na Activity ***ListaObjetosEmprestadosActivity***, além de remover o registro do banco de dados através do DAO, já atualiza a ListView através da chamada do método ***montaListView()***. Dessa forma, o registro removido não será mais apresentado na tela para o usuário.

Por fim, basta agora criar o método ***onContextItemSelected()*** que irá tratar o evento a ser disparado quando o usuário clicar na opção "**Apagar**" do *Context Menu*. Adicionaremos na Activity ***ListaObjetosEmprestadosActivity*** a seguinte implementação:

```
/**
 * Dispara o evento da opção selecionada no menu de contexto.
 */
@Override
public boolean onContextItemSelected(MenuItem item) {
        /**
         * A classe "AdapterView.AdapterContextMenuInfo" irá nos ajudar
         * a extrair algumas informações do nosso Context Menu, como por exemplo
         * a posição da ListView em que a opção do menu foi clicada pelo usuário.
         * Precisaremos exatamente dessa informação para saber qual o item da ListView
         * que o usuário deseja interagir.
         */
        AdapterView.AdapterContextMenuInfo info;
        info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();

        // Trata a ação da opção "Apagar" do menu de contexto
        if (item.getItemId() == MENU_APAGAR) {

                // Obtemos o objeto que o usuário deseja remover através da sua posição na ListView
                ObjetoEmprestado objeto = (ObjetoEmprestado)
                        getListaObjetosEmprestados().getItemAtPosition(info.position);

                // Chama o método que irá remover o objeto do banco de dados
                remove(objeto);

                /**
                 *  Mostra uma mensagem na tela informando ao usuário que a
```

```
                    * operação foi realizada com sucesso
                    */
                    Toast.makeText(getApplicationContext(), "Registro removido com sucesso!",
                            Toast.LENGTH_LONG).show();

                    /**
                    * Após tratar com sucesso o evento de uma opção do
                    * menu de contexto, o retorno deve ser sempre "true"
                    */
                    return true;
            }

            return super.onContextItemSelected(item);
    }
```

Seguindo a definição do projeto ***Devolva.me***, nossa implementação agora está pronta para que o usuário possa remover um determinado item a partir da ListView. Ao executar a Activity ***ListaObjetosEmprestadosActivity*** no emulador do Android e clicar em um item da ListView por 3 segundos, o *Context Menu* irá mostrar a opção "**Apagar**", conforme mostra a **Figura 8.6**.

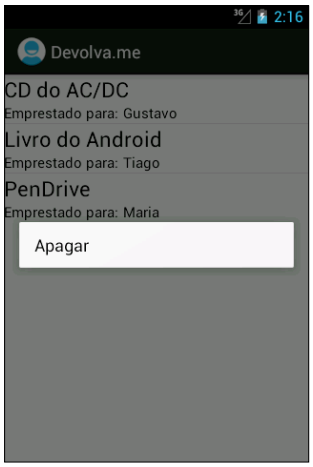


Figura 8.6. Opção *Apagar* da ListView, responsável por remover um determinado item do banco de dados.

8.7 Exercício

Agora que você aprendeu a trabalhar com o banco de dados SQLite no Android, é hora de colocar em prática. Neste exercício, vamos implementar a parte de persistência de dados no projeto **Devolva.me**.

1. Inicialmente devemos definir as duas classes que servirão de modelo para encapsular os dados do objeto a ser emprestado. Para isto, no projeto ***devolva_me*** criado no capítulo anterior, crie um novo pacote chamado ***br.com.hachitecnologia.devolvame.modelo*** e, neste novo pacote, crie duas novas classes, com os nomes ***ObjetoEmprestado*** e ***Contato***.

   Implemente o seguinte conteúdo na classe ***Contato***:

```
public class Contato implements Serializable {

        private static final long serialVersionUID = 1L;

        private String nome;
        private String telefone;

        public String getNome() {
                return nome;
```

```
        }

        public void setNome(String nome) {
            this.nome = nome;
        }

        public String getTelefone() {
            return telefone;
        }

        public void setTelefone(String telefone) {
            this.telefone = telefone;
        }

}
```

Implemente o seguinte conteúdo na classe ***ObjetoEmprestado***:

```
public class ObjetoEmprestado implements Serializable {

        private static final long serialVersionUID = 1L;

        private Long id;
        private String objeto;
        private Calendar dataEmprestimo;
        private Contato contato = new Contato();

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public String getObjeto() {
            return objeto;
        }

        public void setObjeto(String objeto) {
            this.objeto = objeto;
        }

        public Calendar getDataEmprestimo() {
            return dataEmprestimo;
        }

        public void setDataEmprestimo(Calendar dataEmprestimo) {
            this.dataEmprestimo = dataEmprestimo;
        }

        public Contato getContato() {
            return contato;
        }

        public void setContato(Contato contato) {
            this.contato = contato;
        }

}
```

2. Agora precisaremos de uma classe utilitária para definir a estrutura de dados que utilizaremos no aplicativo. Para isto, crie um novo pacote chamado ***br.com.hachitecnologia.devolvame.dao*** e, neste pacote, crie uma nova classe chamada ***DBHelper*** com o seguinte conteúdo:

```
public class DBHelper extends SQLiteOpenHelper {

	// Nome do banco de dados
	private static final String NOME_DO_BANCO = "devolva_me";

	// Versão atual do banco de dados
	private static final int VERSAO_DO_BANCO = 1;

	public DBHelper(Context context) {
		super(context, NOME_DO_BANCO, null, VERSAO_DO_BANCO);
	}

	/**
	 * Cria a tabela "objeto_emprestado" no banco de dados, caso ela não exista.
	 */
	@Override
	public void onCreate(SQLiteDatabase db) {
		String sql = "CREATE TABLE objeto_emprestado (" +
				"_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT" +
				",objeto TEXT NOT NULL" +
				",pessoa TEXT NOT NULL" +
				",telefone TEXT NOT NULL" +
				",data_emprestimo INTEGER NOT NULL" +
				");";
		db.execSQL(sql);
	}

	/**
	 * Atualiza a estrutura da tabela no banco de dados, caso sua versão tenha mudado.
	 */
	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		String sql = "DROP TABLE IF EXISTS objeto_emprestado";
		db.execSQL(sql);
		onCreate(db);
	}
}
```

3. Defina uma classe DAO para centralizar a responsabilidade sobre a tabela "**objeto_emprestado**" do banco de dados. Para isto, dentro do pacote ***br.com.hachitecnologia.devolvame.dao***, crie uma nova classe, chamada ***ObjetoEmprestadoDAO***, com a seguinte implementação:

```
public class ObjetoEmprestadoDAO {

	private DBHelper dbHelper;

	public ObjetoEmprestadoDAO(Context context) {
		dbHelper = new DBHelper(context);
	}

}
```

4. Para adicionar a funcionalidade de inserir dados no banco de dados, crie um método na classe ***ObjetoEmprestadoDAO*** chamado ***adiciona()***, com o seguinte conteúdo:

```
/**
 * Adiciona objeto no banco de dados.
```

```
     */
    public void adiciona(ObjetoEmprestado objeto) {
            // Encapsula no objeto do tipo ContentValues os valores a serem persistidos no banco de dados
            ContentValues values = new ContentValues();
            values.put("objeto", objeto.getObjeto());
            values.put("data_emprestimo", System.currentTimeMillis());
            values.put("pessoa", objeto.getContato().getNome());
            values.put("telefone", objeto.getContato().getTelefone());

            // Instancia uma conexão com o banco de dados, em modo de gravação
            SQLiteDatabase db = dbHelper.getWritableDatabase();

            // Insere o registro no banco de dados
            long id = db.insert("objeto_emprestado", null, values);
            objeto.setId(id);

            // Encerra a conexão com o banco de dados
            db.close();
    }
```

5. Agora você deverá definir uma Activity que será responsável por cadastrar os objetos emprestados no banco de dados. Para isto, dentro do pacote ***br.com.hachitecnologia.devolvame.activity***, crie uma nova Activity, chamada ***CadastraObjetoEmprestadoActivity***, definindo o nome do seu arquivo de Layout (*Layout Name*) para ***activity_cadastra_objeto_emprestado***.

6. Vamos agora definir a tela de cadastro usada pelo usuário para registrar no banco de dados os objetos emprestados. Edite o arquivo de Layout ***/res/layout/activity_cadastra_objeto_emprestado.xml***, da Activity criada anteriormente, deixando-o com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:orientation="vertical"
    android:paddingBottom="15.0dip"
    android:paddingLeft="15.0dip"
    android:paddingRight="15.0dip"
    android:paddingTop="15.0dip" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Objeto:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_objeto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o nome do objeto"
            android:inputType="text" >

            <requestFocus />
        </EditText>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Emprestado para:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_pessoa"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o nome da pessoa"
            android:inputType="textPersonName" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Telefone:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_telefone"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o telefone da pessoa"
            android:inputType="phone" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal"
            android:paddingTop="15.0dip" >

            <Button
                android:id="@+id/cadastra_objeto_botao_salvar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Salvar" />

            <Button
                android:id="@+id/cadastra_objeto_botao_cancelar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Cancelar" />

        </LinearLayout>

    </LinearLayout>
```

7. Faça a implementação para adicionar no banco de dados os dados informados pelo usuário na tela de cadastro. Para isto, edite a Activity ***CadastraObjetoEmprestadoActivity***, deixando-a com a seguinte implementação:

```
public class CadastraObjetoEmprestadoActivity extends Activity {

    private ObjetoEmprestado objetoEmprestado;
    private EditText campoObjeto;
    private EditText campoNomePessoa;
    private EditText campoTelefone;
    private Button botaoSalvar;
    private Button botaoCancelar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Define o Layout Resource da Activity
        setContentView(R.layout.activity_cadastra_objeto_emprestado);

        campoObjeto = (EditText) findViewById(R.id.cadastro_objeto_campo_objeto);
        campoNomePessoa = (EditText) findViewById(R.id.cadastro_objeto_campo_pessoa);
        campoTelefone = (EditText) findViewById(R.id.cadastro_objeto_campo_telefone);
        botaoSalvar = (Button) findViewById(R.id.cadastra_objeto_botao_salvar);
```

```
        botaoCancelar = (Button) findViewById(R.id.cadastra_objeto_botao_cancelar);

        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();

        /**
         * O botao salvar irá salvar o objeto no banco de dados.
         */
        botaoSalvar.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                // Injeta no objeto "objetoEmprestado" os dados informados pelo usuário
                objetoEmprestado.setObjeto(campoObjeto.getText().toString());
                objetoEmprestado.getContato().setNome(campoNomePessoa.getText().toString());
                objetoEmprestado.getContato().setTelefone(campoTelefone.getText().toString());

                // Instancia o DAO para persistir o objeto
                ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());
                // Salva o objeto no banco de dados
                dao.adiciona(objetoEmprestado);

                // Mostra para o usuário uma mensagem de sucesso na operação
                Toast.makeText(getApplicationContext(), "Objeto salvo com sucesso!", Toast.LENGTH_LONG)
                        .show();
                }

            });

            /**
             * O botao cancelar apenas finaliza a Activity.
             */
            botaoCancelar.setOnClickListener(new View.OnClickListener() {

                public void onClick(View v) {
                        finish();
                }
            });

        }
    }
```

8. Para que o sistema possa listar os dados cadastrados no banco de dados, crie um método na classe ***ObjetoEmprestadoDAO***, chamado ***listaTodos()***, com a seguinte implementação:

```
    /**
     * Lista todos os registros da tabela “objeto_emprestado”
     */
    public List<ObjetoEmprestado> listaTodos() {

        // Cria um List para guardar os objetos consultados no banco de dados
        List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

        // Instancia uma nova conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper. getReadableDatabase();

        // Executa a consulta no banco de dados
        Cursor c = db.query("objeto_emprestado", null, null, null, null,
                null, "objeto ASC");

        /**
         *  Percorre o Cursor, injetando os dados consultados em um objeto
         *  do tipo ObjetoEmprestado e adicionando-os na List
         */
```

```
        try {
            while (c.moveToNext()) {
                ObjetoEmprestado objeto = new ObjetoEmprestado();
                objeto.setId(c.getLong(c.getColumnIndex("_id")));
                objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));
                objeto.getContato().setNome(c.getString(c.getColumnIndex("pessoa")));
                objeto.getContato().setTelefone(c.getString(c.getColumnIndex("telefone")));
                objetos.add(objeto);
            }

        } finally {
            // Encerra o Cursor
            c.close();
        }

        // Encerra a conexão com o banco de dados
        db.close();

        // Retorna uma lista com os objetos consultados
        return objetos;
    }
```

9. Agora você deverá criar uma nova Activity que será responsável por listar os registros inseridos no banco de dados e apresentá-los na tela para o usuário. Para isto, no pacote ***br.com.hachitecnologia.devolvame.activity***, crie uma nova Activity, chamada ***ListaObjetosEmprestadosActivity***, definindo o nome do seu arquivo de Layout (*Layout Name*) para ***activity_lista_objetos_emprestados***.

10. Vamos agora definir a tela que conterá a lista dos registros salvos pelo usuário no banco de dados, ou seja, a lista com os objetos emprestados pelo usuário, no projeto ***Devolva.me***. Edite o arquivo de Layout ***/res/layout/activity_lista_objetos_emprestados.xml***, da Activity criada anteriormente, deixando-o com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/lista_objetos_emprestados"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

11. Precisamos agora criar um arquivo de Layout que irá definir a forma com que as informações dos objetos emprestados (cadastrados no banco de dados) sejam apresentados na ListView. Para isto, crie um novo arquivo de Layout, chamado ***item_lista_objetos_emprestados*** com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/item_objeto_emprestado_nome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textSize="22sp" />
```

```
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:textSize="15sp"
            android:text="Emprestado para:"
            android:paddingRight="5.0dip"/>

        <TextView
            android:id="@+id/item_objeto_emprestado_pessoa"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:textSize="15sp" />

    </LinearLayout>

</LinearLayout>
```

12. Agora, devemos definir na Activity o arquivo de Resource Layout que contém a ListView a ser populada dinamicamente via código. Para isto, edite a classe ***ListaObjetosEmprestadosActivity*** inserindo a seguinte implementação no corpo da classe:

```
/**
 * Variável de instância do tipo ListView que fará referência à ListView do
 * nosso Resource Layout e será manipulado via código para
 * ser preenchido com os dados consultados no banco de dados.
 */
private ListView listaObjetosEmprestados;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Definimos o Resource Layout a ser controlado pela Activity
    setContentView(R.layout.activity_lista_objetos_emprestados);
    /**
     * Fazemos a ligação entre o objeto "listaObjetosEmprestados" e a View
     * ListView do nosso Resource Layout.
     */
    listaObjetosEmprestados = (ListView) findViewById(R.id.lista_objetos_emprestados);
}

/**
* Retorna o objeto ListView já preenchido com a lista de
* objetos emprestados.
*/
public ListView getListaObjetosEmprestados() {
        return listaObjetosEmprestados;
}
```

13. Devemos agora definir um Adapter que será responsável por adaptar o conteúdo de cada registro consultado no banco de dados ao arquivo de Layout ***item_lista_objetos_emprestados.xml*** e injetá-los na ListView. Para isto, crie um novo pacote chamado ***br.com.hachitecnologia.devolvame.adapter*** e, dentro dele, crie uma nova classe chamada ***ListaObjetosEmprestadosAdapter***, com a seguinte implementação:

```
public class ListaObjetosEmprestadosAdapter extends ArrayAdapter<ObjetoEmprestado> {

	private final List<ObjetoEmprestado> objetosEmprestados;
	private final Activity activity;

	public ListaObjetosEmprestadosAdapter(Activity activity, int textViewResourceId,
			List<ObjetoEmprestado> objetosEmprestados) {
		super(activity, textViewResourceId, objetosEmprestados);
		this.activity = activity;
		/**
		 *  List de objetos do tipo ObjetoEmprestado recebida como parâmetro
		 *  pelo Construtor.
		 */
		this.objetosEmprestados = objetosEmprestados;
	}

	/**
	 * Aqui que é onde a mágica é feita: as Views TextView do
	 * Resource Layout "item_lista_objetos_emprestados.xml" são preenchidas com os dados
	 * consultados no banco de dados e este Resource Layout será retornado como um objeto
	 * do tipo View para ser adicionado na ListView do Resource
	 * Layout "activity_lista_objetos_emprestados.xml".
	 *
	 * Este método será chamado para cada
	 * registro consultado no banco de dados, fazendo com que a ListView mostre todos
	 * os registros econtrados.
	 */
	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		/**
		 * Obtemos cada objeto do tipo ObjetoEmprestado dentro da List recebida
		 * pelo Construtor, de acordo com a sua posição, para ser mostrado
		 * na mesma posição da ListView, ou seja, o primeiro objeto da List vai
		 * para a primeira posição da ListView e assim sucessivamente.
		 */
		ObjetoEmprestado objetoEmprestado = objetosEmprestados.get(position);

		/**
		 * Referenciamos o Resource Layout "item_lista_objetos_emprestados" em um objeto
		 * do tipo View, que será o objeto de retorno deste método. Esta é a View que será
		 * adaptada e retornada para ser apresentada na ListView.
		 */
		View view = activity.getLayoutInflater().inflate(R.layout.item_lista_objetos_emprestados, null);

		/**
		 *  Injeta o valor do campo referente ao nome do objeto emprestado, do
		 *  registro consultado no banco de dados, na TextView de ID "item_objeto_emprestado_nome".
		 */
		TextView objeto = (TextView) view.findViewById(R.id.item_objeto_emprestado_nome);
		objeto.setText(objetoEmprestado.getObjeto());

		/**
		 *  Injeta o valor do campo referente ao nome da pessoa para quem o
		 *  objeto foi emprestado, do registro consultado no banco de dados, na TextView
		 *  de ID "item_objeto_emprestado_pessoa".
		 */
		TextView pessoa = (TextView) view.findViewById(R.id.item_objeto_emprestado_pessoa);
		pessoa.setText(objetoEmprestado.getContato().getNome());

		// Retorna a View já adaptada para ser apresentada na ListView
		return view;
	}

	/**
```

```
        * Retorna o ID de um determinado item da ListView, de acordo
        * com a sua posição.
        */
        @Override
        public long getItemId(int position) {
                return objetosEmprestados.get(position).getId();
        }

        /**
        * Retorna o número de itens que serão mostrados na ListView.
        */
        @Override
        public int getCount() {
                return super.getCount();
        }

        /**
        * Retorna um determinado item da ListView, de acordo
        * com a sua posição.
        */
        @Override
        public ObjetoEmprestado getItem(int position) {
                return objetosEmprestados.get(position);
        }

}
```

14. Com o Adapter implementado, devemos agora definir o código que irá consultar os dados cadastrados através do DAO e chamar o Adapter para montar a ListView apresentando os dados consultados. Implemente, então, o seguinte código no método ***onResume()*** da Activity ***ListaObjetosEmpresdadosActivity***:

```
@Override
protected void onResume() {
        super.onResume();

        // Consulta os objetos cadastrados no banco de dados através do DAO
        ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

        // Guarda os objetos consultados em uma List
        final List<ObjetoEmprestado> objetosEmprestados = dao.listaTodos();

        // Instancia o Adapter que irá adaptar os dados na ListView
        ArrayAdapter<ObjetoEmprestado> adapter = new ListaObjetosEmprestadosAdapter(this,
                android.R.layout.simple_list_item_1, objetosEmprestados);

        /**
        * Define o Adapter que irá adaptar os dados consultados no banco de dados
        * à nossa ListView do Resource Layout.
        *
        * @param adapter
        */
        listaObjetosEmprestados.setAdapter(adapter);

}
```

15. Devemos agora implementar no DAO o método que irá alterar os dados salvos no banco de dados. Crie um método chamado ***atualiza()*** na classe ***ObjetoEmprestadoDAO*** com a seguinte implementação:

```
/**
* Altera o registro no banco de dados.
*/
```

```
public void atualiza(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem atualizados no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("pessoa", objeto.getContato().getNome());
        values.put("telefone", objeto.getContato().getTelefone());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper. getWritableDatabase();

        // Atualiza o registro no banco de dados
        db.update("objeto_emprestado", values, "_id=?", new String[] { objeto.getId().toString() });

        // Encerra a conexão com o banco de dados
        db.close();
}
```

16. Agora devemos criar um Listener que será responsável por implementar o evento a ser disparado quando o usuário clicar em um dos itens mostrados na ListView. Em nosso caso, o evento deverá abrir a tela para edição do registro selecionado. Para isto, crie um pacote chamado ***br.com.hachitecnologia.devolvame.listener*** e nele adicione uma nova classe chamada ***ListaObjetosEmprestadosListener***, com a seguinte implementação:

```
public class ListaObjetosEmprestadosListener implements AdapterView.OnItemClickListener {

        private final ListaObjetosEmprestadosActivity activity;

        /**
         * Nosso Construtor deverá receber como parâmetro a instância da Activity
         * ListaObjetosEmprestadosActivity para ter acesso ao objeto ListView contendo
         * a lista de objetos emprestados. Isto nos permitirá pegar na lista o item que foi selecionado
         * pelo usuário.
         *
         */
        public ListaObjetosEmprestadosListener(ListaObjetosEmprestadosActivity activity) {
                this.activity = activity;
        }

        /**
        * Método que será disparado quando o usuário selecionar o item na ListView.
        */
        public void onItemClick(AdapterView<?> adapterView, View view, int posicao, long idPosicao) {
                // Declarando a Intent que irá invocar a Activity "CadastraObjetoEmprestadoActivity"
                Intent i = new Intent(activity, CadastraObjetoEmprestadoActivity.class);

                /**
                 *  Passando o objeto selecionado pelo usuário, na ListView, como parâmetro
                 *  para a Activity "CadastraObjetoEmprestadoActivity", para que esta
                 *  possa editar as informações do registro selecionado.
                 */
                i.putExtra("itemSelecionadoParaEdicao",
                        (ObjetoEmprestado)activity.getListaObjetosEmprestados().getItemAtPosition(posicao));

                // Invoca a Activity definida em nossa Intent
                activity.startActivity(i);
        }
}
```

17. Implementado o Listener, devemos agora ativá-lo no objeto ListView da nossa Activity. Para isto, adicione a seguinte linha de código ao final do método ***onResume()*** da Activity ***ListaObjetosEmprestadosActivity***:

```
/**
```

```
* Ativa o Listener que contém o evento a ser disparado ao clicar
* sobre um item da nossa ListView.
*/
listaObjetosEmprestados.setOnItemClickListener(new ListaObjetosEmprestadosListener(this));
```

18. Implemente um método no DAO que execute as tarefas de adicionar e alterar registros no banco de dados, usando o campo ID do objeto para determinar se este deverá ser adicionado ou alterado. Para isto, crie um método na classe ***ObjetoEmprestadoDAO***, chamado ***salva()***, com a seguinte implementação:

```
/**
 * Salva objeto no banco de dados.
 * Caso o objeto não exista no banco de dados, ele o adiciona.
 * Caso o objeto exista no banco de dados, apenas atualiza os valores dos campos modificados.
 *
 * @param objeto
 */
public void salva(ObjetoEmprestado objeto) {
        /**
         *  Se o ID do objeto é nulo é porque ele ainda não existe no banco de dados,
         *  logo subentende-se que queremos adicionar o objeto no banco de dados.
         *  Sendo assim, chamaremos o método adiciona() já definido no DAO.
         */
        if (objeto.getId() == null) {
                adiciona(objeto);
        /**
         * Caso o objeto possua um ID é porque ele já existe no banco de dados, logo
         * subentende-se que queremos alterar seus dados no banco de dados.
         *  Sendo assim, chamaremos o método atualiza() já definido no DAO.
         */
        } else {
                atualiza(objeto);
        }
}
```

19. Altere o código da Activity ***CadastraObjetoEmprestadoActivity*** para que ela possa executar a funcionalidade tanto de cadastrar quanto a de alterar os registros salvos no banco de dados. Para isto, substitua a seguinte linha de código abaixo, presente no método ***onCreate()*** da Activity ***CadastraObjetoEmprestadoActivity***:

```
// Instancia um novo objeto do tipo ObjetoEmprestado
objetoEmprestado = new ObjetoEmprestado();
```

por este trecho de código:

```
/**
* Recebe o objeto recebido como parâmetro da ListView para edição.
 */
objetoEmprestado = (ObjetoEmprestado) getIntent().getSerializableExtra("itemSelecionadoParaEdicao");

if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoTelefone.setText(objetoEmprestado.getContato().getTelefone());
 }
```

20. Ainda na Activity ***CadastraObjetoEmprestadoActivity***, altere o evento ***onClick()*** do botão "**Salvar**" substituindo a linha de código abaixo:

```
// Salva o objeto no banco de dados
dao.adiciona(objetoEmprestado);
```

pela seguinte linha:

```
// Salva o objeto no banco de dados
dao.salva(objetoEmprestado);
```

21. Faça com que a Activity ***ListaObjetosEmprestadosActivity*** seja aberta após o usuário cadastrar ou alterar um registro no banco de dados. Para isto, adicione a seguinte linha de código ao final do evento ***onClick()*** do botão "**Salvar**" da Activity ***CadastraObjetoEmprestadoActivity***:

```
// Depois de salvar, vai para a Lista dos objetos emprestados
startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
```

22. Implemente no DAO, um método que remova um determinado registro no banco de dados. Para isto, adicione na classe ***ObjetoEmprestadoDAO*** um método chamado ***remove()*** com o seguinte conteúdo:

```
/**
 * Remove um registro no banco de dados.
 */
public void remove(ObjetoEmprestado objeto) {
        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Remove o registro no banco de dados
        db.delete("objeto_emprestado", "_id=?", new String[] { objeto.getId().toString() });

        // Encerra a conexão com o banco de dados
        db.close();
}
```

23. Na classe ***ListaObjetosEmprestadosActivity***, crie uma constante que será usada para definir o ID da opção "**Apagar**" no *Context Menu* que iremos costruir para remover do banco de dados um item mostrado na ListView. Crie, então, a seguinte variável na classe ***ListaObjetosEmprestadosActivity***:

```
// ID da opção "Apagar" do menu de contexto
private static final int MENU_APAGAR = Menu.FIRST;
```

24. Na Activity ***ListaObjetosEmprestadosActivity*** defina o método ***onCreateContextMenu()*** que irá criar na ListView um *Context Menu* e adicionar a opção "**Apagar**" para que o usuário possa apagar um determinado item no banco de dados. Para isto, adicione o seguinte método na classe ***ListaObjetosEmprestadosActivity***:

```
/**
 * Cria o menu de contexto.
 */
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);

        // Adiciona a opção "Apagar" ao Context Menu
        menu.add(0, MENU_APAGAR, 0, "Apagar");
```

```
}
```

25. Ative o *Context Menu* criado anteriormente adicionando a seguinte linha de código ao final do método ***onCreate()*** da classe ***ListaObjetosEmprestadosActivity***:

```
// Registra o Context Menu para a ListView da nossa tela
registerForContextMenu(listaObjetosEmprestados);
```

26. Na Activity ***ListaObjetosEmprestadosActivity*** centralize em um único método o código responsável por consultar os dados salvos no banco de dados e adaptar o conteúdo consultado na ListView. Para isto, crie um método chamado ***montaListView()*** na classe ***ListaObjetosEmprestadosActivity*** com a seguinte implementação:

```
/**
 * Monta a ListView com os dados a serem apresentados na tela.
 */
private void montaListView() {

	// Consulta os objetos cadastrados no banco de dados através do DAO
	ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

	// Guarda os objetos consultados em uma List
	final List<ObjetoEmprestado> objetosEmprestados = dao.listaTodos();

	// Instancia o Adapter que irá adaptar os dados na ListView
	ArrayAdapter<ObjetoEmprestado> adapter = new ListaObjetosEmprestadosAdapter(
				this, android.R.layout.simple_list_item_1, objetosEmprestados);

	/**
	 * Define o Adapter que irá adaptar os dados consultados no banco de
	 * dados à nossa ListView do Resource Layout.
	 *
	 * @param adapter
	 */
	listaObjetosEmprestados.setAdapter(adapter);

	/**
	 * Ativa o Listener que contém o evento a ser disparado ao clicar
	 * sobre um item da nossa ListView
	 */
	listaObjetosEmprestados.setOnItemClickListener(new ListaObjetosEmprestadosListener(this));
}
```

27. Refatore o código do método ***onResume()*** da Activity ***ListaObjetosEmprestadosActivity*** para que ele chame o método ***montaListView()*** para executar a tarefa de consultar os dados salvos no banco de dados e adaptar os dados consultados à ListView, evitando código duplicado. Para isto, altere o código do método ***onResume()*** da classe ***ListaObjetosEmprestadosActivity*** para que ele fique com a seguinte implementação:

```
@Override
protected void onResume() {
	super.onResume();

	// Chama o método que irá montar a ListView na tela.
	montaListView();
}
```

28. Na Activity ***ListaObjetosEmprestadosActivity*** crie um método que irá chamar o DAO para remover o registro selecionado pelo usuário e atualizar a ListView para que esta não apresente mais o registro removido. Para isto, crie um método chamado ***remove()*** na classe ***ListaObjetosEmprestadosActivity*** com a seguinte implementação:

```
/**
 * Remove o objeto selecionado.
 */
private void remove(ObjetoEmprestado objeto) {
        // Cria uma nova instância do DAO
        ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());

        // Remove o objeto do banco de dados
        dao.remove(objeto);

        /**
         * Atualiaza a ListView para que o objeto removido não seja mais
         * apresentado na tela.
         */
        montaListView();
}
```

29. Defina na Activity ***ListaObjetosEmprestadosActivity*** o evento que será disparado quando o usuário clicar na opção "**Apagar**" do *Context Menu* de um determinado item mostrado na ListView. Para isto, implemente o método ***onContextItemSelected()*** na classe ***ListaObjetosEmprestadosActivity***:

```
/**
 * Dispara o evento da opção selecionada no menu de contexto.
 */
@Override
public boolean onContextItemSelected(MenuItem item) {
        /**
         * A classe "AdapterView.AdapterContextMenuInfo" irá nos ajudar
         * a extrair algumas informações do nosso Context Menu, como por exemplo
         * a posição da ListView em que a opção do menu foi clicada pelo usuário.
         * Precisaremos exatamente dessa informação para saber qual o item da ListView
         * que o usuário deseja interagir.
         */
        AdapterView.AdapterContextMenuInfo info;
        info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();

        // Trata a ação da opção "Apagar" do menu de contexto
        if (item.getItemId() == MENU_APAGAR) {

                // Obtemos o objeto que o usuário deseja remover através da sua posição na ListView
                ObjetoEmprestado objeto = (ObjetoEmprestado)
                        getListaObjetosEmprestados().getItemAtPosition(info.position);

                // Chama o método que irá remover o objeto do banco de dados
                remove(objeto);

                /**
                 *  Mostra uma mensagem na tela informando ao usuário que a
                 *  operação foi realizada com sucesso
                 */
                Toast.makeText(getApplicationContext(), "Registro removido com sucesso!",
                        Toast.LENGTH_LONG).show();

                /**
                 *  Após tratar com sucesso o evento de uma opção do
                 *  menu de contexto, o retorno deve ser sempre "true"
                 */
                return true;
```

```
            }

            return super.onContextItemSelected(item);
    }
```

8.8 Exercício

Para finalizar este capítulo, precisamos testar as funcionalidades do nosso aplicativo.

1. Configure o projeto *Devolva.me* para que a Activity ***CadastraObjetoEmprestadoActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em ***Run*** > ***Run Configurations***.

2. Execute o projeto no emulador do Android e realize a seguinte operação:

    - Cadastre 5 objetos emprestados no aplicativo;

3. Configure o projeto *Devolva.me* para que a Activity ***ListaObjetosEmprestadosActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em ***Run*** > ***Run Configurations***.

4. Execute o projeto no emulador do Android e realize as seguintes operações:

    - Edite o conteúdo de um dos objetos cadastrados;
    - Remova um dos objetos cadastrados.

9 - Intents e Intent Filters

No **Capítulo 6** nós usamos uma *Intent* para invocar uma Activity através de outra Activity. Agora nós vamos entender melhor o que são as Intents e qual a sua finalidade no Android.

Uma Intent nada mais é do que uma mensagem enviada por um componente de um aplicativo ao núcleo do Android, informando que deseja realizar uma ação, ou seja, uma mensagem informando a intenção de se realizar algo no sistema. Através da Intent podemos pedir para o Android chamar um determinado componente do nosso aplicativo ou um componente de um outro aplicativo instalado no dispositivo. Veja, na **Figura 9.1**, a arquitetura do Android mostrando como as Activities usam as Intents para chamar componentes dos aplicativos.

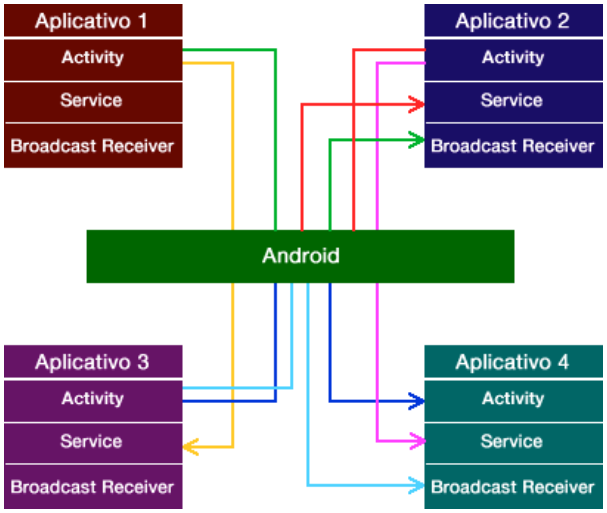


Figura 9.1. Arquitetura do Android no uso das *Intents* para invocar componentes dos aplicativos.

Dica

As *Intents* nos permitem chamar um componente registrado no Android. Este componente pode ser uma *Activity*, um *Service* ou até mesmo o envio de um *Broadcast* para o Android.

As Intents permitem que os aplicativos se comuniquem uns com os outros, fazendo com que um aplicativo aproveite as funcionalidades disponibilizadas por outros aplicativos. Através de uma Intent, em um aplicativo Android, podemos por exemplo:

- Invocar uma determinada *Activity* de um aplicativo;
- Invocar um *Service* para executar um serviço em segundo plano *[Aprenderemos sobre os Services mais à frente neste curso]*;
- Enviar um *Broadcast* para o Android *[Aprenderemos sobre os Broadcasts mais à frente neste curso]*;
- Selecionar um determinado contato no aplicativo de contatos do Android e obter seus dados;
- Abrir o aplicativo de mapas do Android em uma coordenada desejada;

- Abrir uma página da Web;
- Discar para um número de telefone através do aplicativo de telefone do Android;
- Enviar um SMS através do aplicativo de mensagem do Android;
- e várias outras possibilidades.

9.1 Invocando uma Activity através de uma Intent

Como vimos, podemos usar uma Intent para chamar uma determinada Activity em nosso aplicativo. Imagine, por exemplo, que temos uma Activity em nosso aplicativo chamada de *Activity1* e queremos que esta invoque uma segunda Activity, chamada *Activity2*, conforme mostra a **Figura 9.2**. Para que isto seja possível, basta definirmos uma Intent na *Activity1* dizendo ao Android que temos a intenção de abrir uma outra Activity, em nosso caso, a *Activity2*.

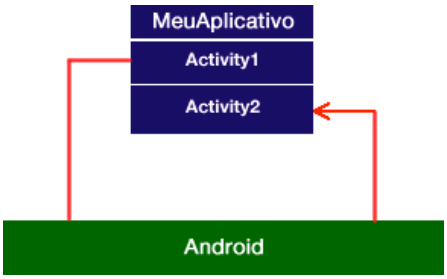


Figura 9.2. *Activity1* enviando uma mensagem ao Android, através de uma *Intent*, solicitando que a *Activity2* seja aberta.

Nota

Lembre-se que no **Capítulo 5** mostramos um exemplo prático de como invocar uma Activity através de outra, onde chamamos uma Activity através do Listener de um Botão.

Invocar uma Activity através de uma Intent no Android é simples e, para isto, usamos a seguinte sintaxe:

```
Intent i = new Intent(getApplicationContext(), Activity2.class);
startActivity(i);
```

A maneira que utilizamos a Intent acima para invocar uma outra Activity é chamada de **explicita** devido ao fato de termos passado como parâmetro o nome da Activity que queremos chamar. Usar uma chamada explicita em uma Intent acaba deixando nosso código altamente acoplado, uma vez que devemos saber o nome da Activity que queremos chamar. Se o nome da Activity a ser invocada for modificado, por exemplo, o código quebraria facilmente e então teríamos que corrigir todas as outras classes que invocam esta Activity.

Dica

As **Intents explícitas** são usadas apenas para chamadas entre componentes do mesmo aplicativo, já que é preciso ter acesso direto e conhecer o componente que será invocado.

9.2 Reduzindo o acoplamento com Intents *implícitas*

No exemplo anterior vimos como usar uma Intent de forma explícita. Iremos agora aprender como reduzir o acoplamento do nosso código ao usar as Intents, através das **Intents implícitas**. As Intents implícitas são mais usadas quando desejamos fazer a interação entre componentes de aplicativos diferentes.

Imagine, por exemplo, que você tem um aplicativo que precise fazer a leitura de um código de barras e executar uma ação com o resultado desta leitura. Ao invés de desenvolver toda a funcionalidade para leitura de código de barras, você pode delegar esta responsabilidade a um aplicativo que já o faça e apenas obter o resultado da leitura realizada e executar a ação desejada. Fazer isto é possível usando as Intents de forma implícita.

Veja a sintaxe básica do uso de uma Intent de forma implícita:

```
Intent i = new Intent("br.com.hachitecnologia.action.ACTIVITY_2");
startActivity(i);
```

Perceba que na forma implícita nós apenas passamos para o construtor da Intent uma String informando o nome que foi definido para o componente que desejamos invocar, neste caso uma Activity. Este nome que passamos como uma String foi definido através de um *Intent Filter*. *[Aprenderemos sobre os Intent Filters mais à frente neste capítulo]*

Veja o quanto ficou desacoplado o nosso código, pois desta forma não precisamos saber o nome da classe do componente e nem mesmo ter acesso direto a ele pois o Android irá fazer o meio de campo para que seu aplicativo consiga chegar até o componente desejado.

Dica

As **Intents implícitas** são mais usadas para realizar a interação entre componentes de aplicativos diferentes, mas você também pode usá-las para chamar componentes de um mesmo aplicativo, deixando seu código mais elegante e desacoplado.

9.3 Objetos de uma Intent

Uma Intent pode armazenar diversos tipos de informações, como: uma *Action* (ação), *Data* (dados), *Category* (categoria) e *Extras* (informações extras). Estas informações armazenadas em uma Intent são recebidas pelos componentes do Android que têm interesse em recebê-las.

Como exemplo, temos uma Intent que solicita ao Android a realização de uma chamada telefônica. Para que isto seja possível, esta Intent deve declarar em seu conteúdo o nome da Action de Telefonia e o número para o qual deseja realizar a chamada. A **Figura 9.3** ilustra como este processo é realizado internamente no Android.

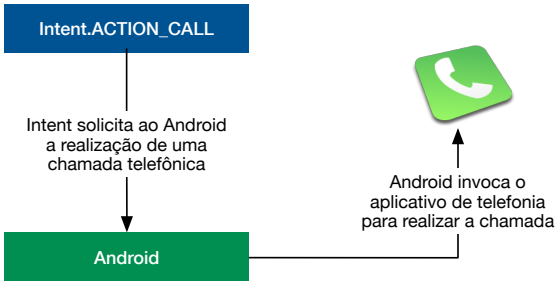


Figura 9.3. Intent disparando a Action *ACTION_CALL* para efetuar uma chamada telefônica.

Veja no trecho de código abaixo como é simples realizar esta tarefa no Android:

```
/**
 * Declara, no construtor da Intent, a Action que será chamada.
 * Em nosso caso, a Action de chamadas telefônicas.
 */
Intent i = new Intent(Intent.ACTION_CALL);

/**
```

```
        * Define o conteúdo a ser passado para a Action
        * de chamadas telefônicas. Em nosso caso, o número
        * de telefone para o qual desejamos ligar.
        */
        i.setData(Uri.parse("tel:8111-1111"));

        // Inicia a Action definida na Intent
        startActivity(i);
```

No trecho de código acima, a Intent declara que deseja chamar a Action que responde pelo nome Intent.***ACTION_CALL***. Nesta Intent definimos também o dado que queremos passar para esta Action e, em nosso caso, passamos o número de telefone para o qual desejamos realizar a chamada. Ao chamar o método ***startActivity()***, passando a Intent como parâmetro, o Android irá receber a solicitação e irá chamar a Action que tem o interesse de receber este tipo de informação, no caso, a Action ***ACTION_CALL***.

Além da *Action* (como exemplo, a que usamos no trecho de código acima para chamar a Action de telefonia do Android) e do *Data* (como exemplo, o número de telefone que passamos para a Action de telefonia através do método ***setData()***), existem outros tipos de objetos que podemos armazenar nas Intents. Veremos sobre cada um destes objetos nos próximos tópicos.

9.3.1 Action

Uma ***Action*** é uma String passada como parâmetro para o construtor de uma Intent informando a ação que queremos executar no Android. Essa String deve ser única em todo o sistema, portanto, para que não haja conflito entre nomes de Actions diferentes, é aconselhável o uso da mesma convenção de **nomes de pacotes do Java** para nomear uma Action.

Nota

Lembre-se que no exempo anterior, onde usamos uma Intent para realizar uma chamada telefônica para um número, usamos uma Action chamada ***ACTION_CALL*** nativa do Android.

Para uma Action de cadastro de pessoas, por exemplo, poderíamos usar o seguinte nome:

br.com.empresa.projeto.action.CADASTRO_PESSOA.

O trecho de código abaixo mostra como seria a chamada para esta Action, no Android:

```
Intent i = new Intent("br.com.empresa.projeto.action.CADASTRO_PESSOA");
startActivity(i);
```

A classe *Intent* do Android define algumas constantes com as Actions nativas da plataforma. Veja, na tabela abaixo, algumas das Actions nativas do Android.

Constante	Tipo de componente	Ação
ACTION_CALL	Activity	Inicia o aplicativo de telefonia
ACTION_EDIT	Activity	Exibe tela para edição de um determinado dado
ACTION_MAIN	Activity	Abre a Activity inicial de um aplicativo
ACTION_SYNC	Activity	Sincroniza os dados do dispositivo com os dados do servidor
ACTION_BATTERY_LOW	Broadcast Receiver	Aviso de bateria fraca
ACTION_HEADSET_PLUG	Broadcast Receiver	Aviso de handset plugado ou desplugado do dispositivo
ACTION_SCREEN_ON	Broadcast Receiver	Aviso de tela acesa

Tabela 9.1. Algumas *Actions* nativas do Android.

A tabela acima mostra apenas algumas das Actions mapeadas na classe Intent. Existem diversas outras Actions nativas e você pode ver a lista completa na documentação da classe Intent, no site do desenvolvedor Android, através do link:

http://developer.android.com/reference/android/content/Intent.html

Nota

Uma Action nativa do Android muito importante que você deve memorizar é a ***ACTION_MAIN***. Esta Action é responsável por definir a Activity inicial de um aplicativo, ou seja, sempre que você definir esta Action no Intent Filter de uma Activity ela será a Activity principal do aplicativo.

9.3.2 Data

O ***Data*** é uma informação a mais que podemos definir na Intent para especificar melhor qual componente será associado à nossa Intent. No Data podemos definir dois tipos de informação: ***URI*** e ***data type***.

Veja novamente o exemplo de uma Intent que utilizamos anteriormente para realizar uma chamada telefônica:

```
Intent i = new Intent(Intent.ACTION_CALL);
i.setData(Uri.parse("tel:8111-1111"));
startActivity(i);
```

No exemplo, ao declarar a *ACTION_CALL* para realizar chamadas telefônicas, usamos justamente uma *URI* para informar à esta Action o número de telefone para o qual desejamos chamar.

A URI é definido na Intent através do método ***setData()***. No trecho de código acima, usamos a seguinte URI:

```
"tel:8111-1111"
```

A URI acima irá pedir para a ACTION_CALL discar para o número 8111-1111.

Cada Action possui um formato específico de URI, utilizado para executar ações de acordo com os dados informados. Um outro exemplo é quando em nosso aplicativo queremos que, quando o usuário clicar em um botão, o Browser do Android seja aberto em uma página específica da Web. Para isto existe uma action chamada ***ACTION_VIEW*** que, quando informamos em sua URI o endereço de um site, o Browser será invocado e direcionado para o endereço do site informado. Veja um exemplo no trecho de código abaixo:

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.hachitecnologia.com.br"));
```

No trecho de código acima informamos para a ACTION_VIEW a URI "http://www.hachitecnologia.com.br" informando que desejamos que o Browser abra a página Web da Hachi Tecnologia.

Perceba que enquanto a Action determina a ação a ser executada, a URI determina como deve ser feito. A ***ACTION_VIEW***, por exemplo, pode ser usada para diversos fins, dependendo da URI recebida, como por exemplo, abrir uma imagem no aplicativo de visualização de imagens do Android, abrir um arquivo de áudio do aplicativo de Sons, mostrar os contatos armazenados o aplicativo de Contatos do Android, etc. Veja o exemplo de uma URI passada para a *ACTION_VIEW* para mostrar os contatos armazenados no Android:

```
Uri uri = Uri.parse("content://contacts/people/");
Intent i = new Intent(Intent.ACTION_VIEW, uri);
startActivity(i);
```

Além do URI, outra informação que podemos passar no *Data* de uma Intent é o ***Data Type***. O Data type especifica o Mime Type do arquivo definido na URI e é definido através do método ***setType()*** da Intent. Imagine, por exemplo, que através de uma Intent você queira chamar um componente do Android para abrir um arquivo de imagem com a extensão .JPG. Para

isto, no URI da Intent você pode especificar o caminho deste arquivo e no Data Type especificar o tipo deste arquivo para que o Android decida qual o melhor componente para abri-lo. Veja o exemplo:

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("/caminho/do/arquivo.jpg"));
i.setType("image/jpeg");
```

9.3.3 Category

Uma *Category* (categoria) é uma String que passamos para a Intent informando qual o componente mais adequado para executar a ação solicitada pela Intent. O uso da Category é aconselhável para especificar mais detalhadamente o componente que melhor atenderia uma determinada solicitação. Veja abaixo o exemplo da definição de uma Category em uma Intent:

```
Intent i = new Intent("br.com.empresa.projeto.action.CADASTRO_PESSOA");
i.addCategory("br.com.empresa.projeto.category.PESSOA_JURIDICA");
```

No exemplo acima, definimos a Action no construtor da Intent e, logo em seguida, definimos a Category através do método ***addCategory()*** da Intent.

Perceba que nossa Intent chama uma Action responsável pelo cadastro de pessoas, chamada ***CADASTRO_PESSOA***. Imagine que esta Action abra uma tela com o formulário para preenchimento dos dados da pessoa que queremos cadastrar. Para tornar as coisas mais específicas, informamos na Category que queremos cadastrar uma pessoa jurídica e, desta forma, podemos ter Activities distintas para cadastro de pessoa física e jurídica, definindo telas distintas com campos específicos de cada caso.

Nota

Ao contrário da Action, é possível especificar mais de uma Category em uma Intent. Para especificar mais de uma Category, basta chamar novamente o método ***addCategory()*** informando as demais Categories desejadas.

9.3.4 Extras

Extras são informações adicionais que desejamos transmitir para o componente que irá executar a ação definida na Intent. Podemos passar qualquer tipo de Objeto do Java como informação Extra. Estas informações são definidas em pares de *CHAVE* / *VALOR*. Veja o exemplo:

```
/**
 * Define o objeto com as informações Extras a serem transmitidas para a Action definida na Intent.
 * Estas informações são definidas no objeto do tipo Bundle em pares de Chave / Valor.
 */
Bundle bundle = new Bundle();
/**
 * Informamos o ID da pessoa cadastrada
 * que queremos editar suas informações.
 * Neste caso, estamos informando o ID 10.
 */
bundle.putLong("id", 10);

/**
 * Define a Action de edição das informações do cadastro
 * de uma pessoa.
 */
Intent i = new Intent("br.com.empresa.projeto.action.EDITAR_CADASTRO_PESSOA");
/**
 * Injeta na Intent as informações Extras
 * definidas no objeto bundle.
 */
i.putExtras(bundle);
```

No exemplo acima nós definimos na Intent que queremos chamar a Action ***EDITAR_CADASTRO_PESSOA*** responsável pela edição das informações do cadastro de uma pessoa. Logo em seguida, através do método ***putExtras()*** da Intent, definimos as informações Extras que queremos passar para a Action. Em nosso caso, nós mapeamos no objeto ***bundle*** o ID da pessoa que queremos editar suas informações de cadastro através do método ***putLong()*** do Bundle.

Na classe Bundle, além do método ***putLong()*** (responsável por guardar um objeto do tipo ***Long***), temos métodos que guardam praticamente qualquer tipo de Objeto do Java. Veja:

• **putInt()**: Método responsável por guardar objetos do tipo ***Intenger*** ou primitivos ***int***;

• **putFloat()**: Método responsável por guardar objetos do tipo ***Float*** ou primitivo ***float***;

• **putDouble()**: Método responsável por guardar objetos do tipo ***Double*** ou primitivo ***double***;

• **putString()**: Método responsável por guardar objetos ***String***;

• e vários outros métodos para guardar outros tipos de Objetos.

A Intent permite também a passagem de informações Extras sem a definição de um objeto do tipo Bundle. Portanto, caso prefira usar um atalho, você também pode definir o mesmo Extra anterior da seguinte forma:

```
/**
 * Define a Action de edição das informações do cadastro
 * de uma pessoa.
 */
Intent i = new Intent("br.com.empresa.projeto.action.EDITAR_CADASTRO_PESSOA");
/**
 *  Informamos o ID da pessoa cadastrada
 *  que queremos editar suas informações.
 *  Neste caso, estamos informando o ID 10.
 */
i.putExtra("id", 10L);
```

Para ler as informações Extras através do componente que as recebeu, basta usar o método ***getExtras()*** da Intent. Veja o exemplo no trecho de código abaixo:

```
Bundle b = getIntent().getExtras();
long id = b.getLong("id");
```

Perceba que usamos o método ***getLong()*** do *Bundle* para recuperar o objeto do tipo *Long* que passamos através da *Intent*. Existem métodos ***getters*** para todos os tipos de dados. Veja alguns deles:

• **getLong()**: lê um objeto do tipo **Long** ou primitivo **long** no Extras de uma Intent;

• **getInt()**: lê um objeto do tipo **Integer** ou primitivo **int** no Extras de uma Intent;

• **getString()**: lê um objeto do tipo **String** no Extras de uma Intent;

• **getDouble()**: lê um objeto do tipo **Double** ou primitivo **double** no Extras de uma Intent;

• **getFloat()**: lê um objeto do tipo **Float** ou primitivo **float** no Extras de uma Intent;

• e vários outros métodos para ler outros tipos de Objetos.

9.4 Intent Filters

Como aprendemos anteriormente, as *Intents* podem ser utilizadas de duas formas: implícita e explícita. Como explicamos, as Intents explícitas são usadas apenas para a interação entre componentes do mesmo aplicativo enquanto que as Intents

implíticas são mais usadas para interação entre componentes de aplicativos distintos. As Intents implícitas são definidas nos *Intent Filters*.

Quando uma Intent implícita é enviada para o Android é ele quem irá decidir qual o componente que irá atender à solicitação desta Intent. Isto nós já havíamos aprendido anteriormente, quando falamos sobre as *Intents implícitas* e as *Actions*, mas o que não aprendemos ainda é como o Android sabe qual o componente que corresponde à Intent implícita chamada.

Os *Intent Filters* são mapeados no arquivo ***AndroidManifest.xml*** de cada aplicativo, dentro da configuração do componente desejado. Veja o exemplo de configuração de um Intent Filter:

```
<intent-filter>
        <action android:name="br.com.empresa.projeto.action.ACTION" />
        <category android:name="br.com.empresa.projeto.category.CATEGORIA" />
</intent-filter>
```

Como exemplo, imagine que desejamos chamar a Activity ***CadastraObjetoEmprestadoActivity***, em nosso projeto *Devolva.me*, através de uma Action definida em uma Intent de forma implícita, conforme mostra o trecho de código abaixo:

```
Intent i = new Intent("br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO");
startActivity(i);
```

O que acontece internamente é que o próprio Android é quem irá decidir qual o componente que corresponde à Action ***br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO***, conforme ilustrado na **Figura 9.4**. Isto nós já havíamos aprendido anteriormente quando falamos sobre as *Intents implícitas* e as *Actions*, mas o que não aprendemos ainda é como o Android sabe qual o componente que corresponde à Action desejada. A explicação para isto é simples: o Android irá procurar internamente um *Intent Filter* que possui uma Action com o nome informado.

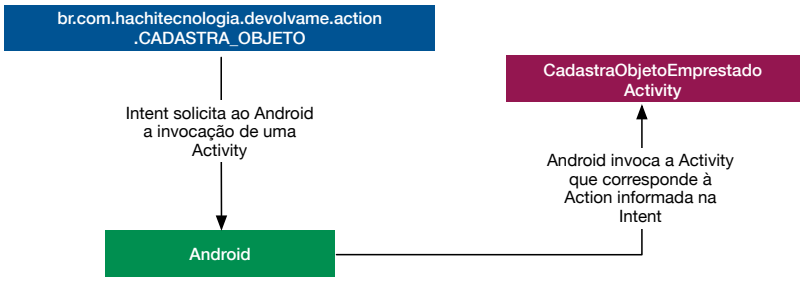


Figura 9.4. Intent disparando a Action *CADASTRA_OBJETO* para chamar a Activity de cadastro de objetos empresdados, do aplicativo *Devolva.me*.

A configuração do exemplo anterior, no arquivo *AndroidManifest.xml* do aplicativo *Devolva.me*, ficaria da seguinte forma:

```
<activity
        android:name=".activity.CadastraObjetoEmprestadoActivity"
        android:label="@string/title_activity_cadastra_objeto_emprestado" >
         <intent-filter>
                <action android:name="br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO" />
                <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
</activity>
```

Com o Intent Filter acima configurado, o Android agora sabe que a Activity ***CadastraObjetoEmprestadoActivity*** responde pela Action ***br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO***, ou seja. qualquer Intent que definir esta Action poderá interagir com a Activity de cadastro de objetos emprestados do aplicativo *Devolva.me*.

Nota

Perceba, no *Intent Filter*, que a Action não precisa ser nomeada utilizando o mesmo nome do pacote em que encontra-se o componente. Como exemplo, nossa Activity ***CadastraObjetoEmprestadoActivity*** está no pacote *br.com.hachitecnologia.devolvame.**activity*** do projeto *Devolva.me*, porém em sua Action (definida no Intent Filter) usamos o nome *br.com.hachitecnologia.devolvame.**action**.CADASTRA_OBJETO*.

No exemplo anterior vimos como definir uma Action em um *Intent Filter* para que o Android possa permitir que outros componentes possam interagir com esta Action. Podemos mapear 3 (três) tipos de componentes nos Intent Filters:

• Action;

• Category;

• Data.

A maneira com que o Android decide qual componente mais indicado para atender à solicitação de uma Intent, usando os Intent Filters, é chamada de *Resolução de Intents*. No tópico a seguir aprenderemos como funciona a Resolução das Intents de cada um dos três tipos de componentes mencionados acima.

9.5 Intent Resolution (Resolução das Intents)

Como vimos, as Intents implícitas são definidas no *Intent Filter* do componente dentro do arquivo *AndroidManifest.xml*. Mas, para que o Android possa decidir qual componente chamar, é preciso que as informações solicitadas por uma Intent casem com as informações contidas em um Intent Filter. Esse processo de Resolução de Intents deve ser estudado com atenção e veremos detalhadamente como ele funciona para cada tipo de componente registrado no Intent Filter.

9.5.1 Teste para resolução do elemento *Action*

No Intent Filter, uma *Action* é definida através do elemento *<action>* da seguinte maneira:

```
<intent-filter>
        <action android:name="br.com.empresa.projeto.action.ACTION_1" />
        <action android:name="br.com.empresa.projeto.action.ACTION_2" />
        <action android:name="br.com.empresa.projeto.action.ACTION_3" />
        . . .
</intent-filter>
```

Nota

Uma Intent pode definir apenas uma Action. Porém, no Intent Filter de um componente, podemos definir mais de uma Action, conforme mostra o exemplo acima.

Em um Intent Filter é obrigatório a definição de pelo menos uma Action para que uma Intent possa alcançá-lo. Como no exemplo acima, é possível definir mais de uma Action no Intent Filter de um componente e, desta maneira, a Intent que definir uma destas Actions conseguirá ter acesso ao componente.

No exemplo dado acima, a associação só ocorrerá se uma Intent definir uma das três Actions definidas no Intent Filter. Veja:

1. `startActivity(new Intent("br.com.empresa.projeto.action.ACTION_1")); // funciona`

    • A Intent acima irá funcionar, pois existe uma Action de mesmo nome definida no Intent Filter

2. `startActivity(new Intent("br.com.empresa.projeto.action.ACTION_2")); // funciona`

    • A Intent acima também irá funcionar, pois existe uma Action de mesmo nome definida no Intent Filter

3. `startActivity(new Intent("br.com.empresa.projeto.action.ACTION_9")); // NÃO funciona`

- A Intent acima **NÃO** irá funcionar, pois **NÃO** existe uma Action de mesmo nome definida no Intent Filter

9.5.1.1 A Action *android.intent.action.MAIN*

A Action ***android.intent.action.MAIN*** é nativa do Android e deve ser adicionada no Intent Filter da Activity principal do aplicativo, ou seja, para que você possa definir a Activity inicial do seu aplicativo você deve definir essa Action no Intent Filter dessa Activity.

Como exemplo, imagine que tenhamos uma Activity chamada ***TelaInicialActivity*** e queremos definí-la como a Activity inicial do nosso aplicativo. Para isto, na configuração desta Activity no arquivo ***AndroidManifest.xml***, basta adicionar a Action *android.intent.action.MAIN* da seguinte forma:

```
<activity android:name=".activity.TelaInicialActivity" . . . >
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                . . .
        </intent-filter>
</activity>
```

Com a configuração acima, a Activity ***TelaInicialActivity*** será executada sempre que o aplicativo for iniciado no Android.

9.5.2 Teste para resolução do elemento *Category*

No Intent Filter, uma *Category* é definida através do elemento <*category*> da seguinte maneira:

```
. . .
<intent-filter>
        <action android:name="br.com.empresa.action.CADASTRA_PESSOA" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="br.com.empresa.category.PESSOA_JURIDICA" />
        <category android:name="br.com.empresa.category.EMPRESA_PRIVADA" />
</intent-filter>
. . .
```

De acordo com o Intent Filter definido acima, para que uma Intent com Categories definidas seja associada a ele, é preciso que todas as Categories desta Intent estejam definidas neste Intent Filter. Veja o exemplo:

- **Exemplo 1**:

```
Intent i = new Intent("br.com.empresa.action.CADASTRA_PESSOA");
i.addCategory("br.com.empresa.category.PESSOA_JURIDICA");
i.addCategory("br.com.empresa.category.EMPRESA_PRIVADA");
startActivity(i);

// A Intent acima irá funcionar, pois todas suas Categories estão definidas no Intent Filter.
```

- **Exemplo 2**:

```
Intent i = new Intent("br.com.empresa.action.CADASTRA_PESSOA");
i.addCategory("br.com.hachitecnologia.category.PESSOA_JURIDICA");
i.addCategory("br.com.hachitecnologia.category.EMPRESA_PUBLICA");
startActivity(i);

// A Intent acima NÃO irá funcionar, pois possui uma Category que NÃO está definida no Intent Filter
```

- **Exemplo 3**:

```
Intent i = new Intent("br.com.empresa.action.CADASTRA_PESSOA");
i.addCategory("br.com.empresa.category.PESSOA_JURIDICA");
startActivity(i);
```

```
// A Intent acima irá funcionar, pois sua Category está definida no Intent Filter.
```

Nota

No Intent Filter utilizado como exemplo, perceba que definimos também uma Category chamada ***android.intent.category.DEFAULT***. Esta Category é nativa do Android e devemos obrigatoriamente adicioná-la ao Intent Filter sempre que disparamos uma Intent implícita através do método ***startActivity()*** ou ***startActivityForResult()***, pois esta será a Category padrão da Intent.

A Category ***android.intent.category.DEFAULT*** não precisa ser adicionada à Intent, pois ela já é adicionada implicitamente à Intent pelo próprio Android.

9.5.2.1 A Category *android.intent.category.LAUNCHER*

Uma Category nativa do Android que devemos conhecer é a *android.intent.category.LAUNCHER*. Ao definir esta Category na configuração de uma Activity, no arquivo ***AndroidManifest.xml***, permite que o Android crie um ícone no Laucher (menu de aplicativos do Android) com um atalho direto para esta Activity.

Imagine que você tenha uma Activity chamada ***TelaInicialActivity*** definida como a Activity inicial do seu aplicativo. Caso você queira colocar um atalho para esta Activity no Laucher (lançador de aplicativos) do Android, basta adicionar a Category *android.intent.category.LAUNCHER* no Intent Filter dessa Activity, deixando sua configuração da seguinte forma:

```
. . .
<activity android:name=".activity.TelaInicialActivity" . . . >
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                . . .
        </intent-filter>
</activity>
. . .
```

9.5.3 Teste para resolução do elemento *Data*

Como vimos anteriormente, podemos especificar dois tipos de filtros em um elemento <*data*> no Intent Filter: ***URI*** e ***data type***.

9.5.3.1 URI

Um *URI* é definido no elemento <*data*> de um Intent Filter, da seguinte forma:

```
scheme://host:port/path
```

Veja o exemplo na **Figura 9.5**:

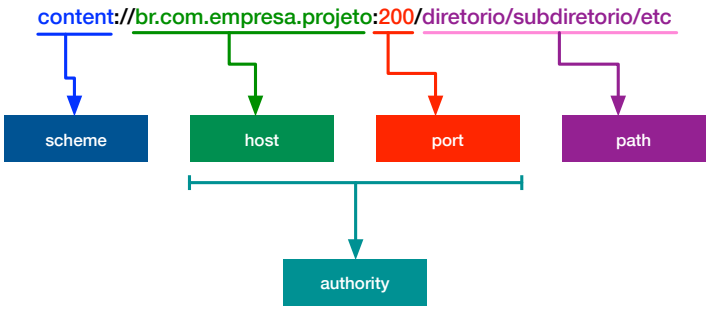


Figura 9.5. Estrutura do *URI* de um elemento <*data*>.

Algumas regras devem ser obedecidas em um URI:

1. O *host* e a *porta* juntos formam o que chamamos de ***authority***;

2. A *porta* deve estar sempre acompanhada do *host*. Caso o *host* não seja informado, a *porta* será ignorada;

3. Cada item de um URI é opcional, porém nunca independente do anterior. Por exemplo: se informarmos o *host* devemos obrigatoriamente informar o *scheme*; se informarmos o *path* devemos obrigatoriamente informar o *authority* e o *scheme*.

Para que um URI de um objeto Intent seja associado a um URI definido em um Intent Filter é preciso que cada item do URI da Intent esteja especificado no URI do Intent Filter. Como exemplo, se o *scheme* "***http://***" estiver especificado no URI do Intent Filter, apenas as Intents com URIs que possuem esse mesmo *scheme* serão associadas. Outro exemplo, se um *scheme* e um *host* estiverem definidos no URI do Intent Filter, apenas as Intents que definirem este mesmo *schema* e *host* no URI serão associadas.

Para entender melhor como funciona a associação entre URIs de Intent e Intent Filters, vejamos alguns exemplos:

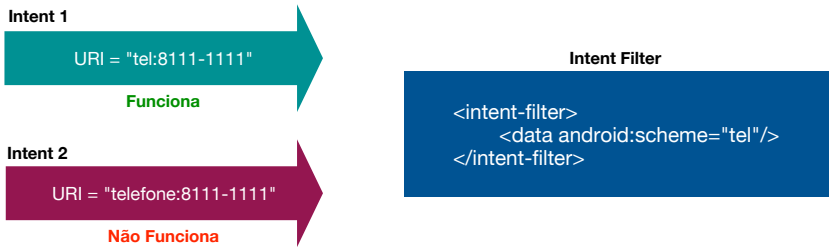


Figura 9.6. Associação de um *Data* através do *URI* - Exemplo 1

No exemplo da **Figura 9.6**, a Intent 1 irá funcionar, pois sua URI define corretamente o *scheme* conforme especificado no Intent Filter. Já a Intent 2 não irá funcionar, pois usa um *scheme* diferente do definido no Intent Filter.

Figura 9.7. Associação de um *Data* através do *URI* - Exemplo 2

No exemplo da **Figura 9.7**, a Intent 1 irá funcionar, pois sua URI define corretamente o *scheme* e o *host* conforme especificado no Intent Filter. Já a Intent 2 não irá funcionar, pois usa um *host* diferente do definido no Intent Filter.

9.5.3.2 Data Type

O *Data Type* de um elemento *Data* é usado para especificar o Mime Type de um arquivo. O *Data Type* é definido pelo atributo ***type*** de um elemento <*data*>.

Imagine, por exemplo, que você queira pedir para o Android abrir um determinado arquivo PDF. Para isto, você pode utilizar uma Intent que irá chamar um componente que faça leitura de arquivos PDF, passando a *URI* com o caminho do arquivo e o *Data Type* com o tipo do arquivo (*Mime Type*) que deseja abrir. Veja o exemplo:

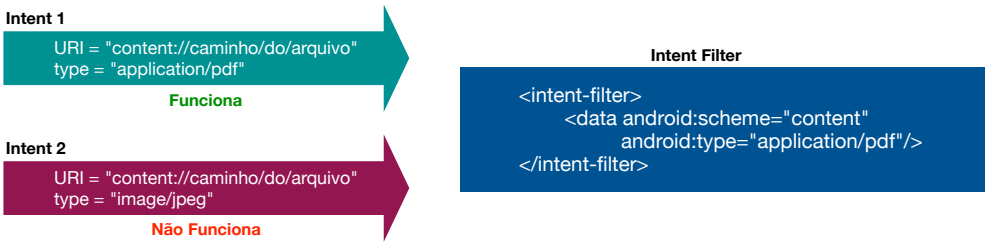


Figura 9.8. Associação de um *Data* através do *URI* e do *Data Type*

No exemplo da **Figura 9.8**, a Intent 1 irá funcionar, pois tanto sua *URI* quanto seu *Data Type* estão definidos conforme especificado no Intent Filter. Já a Intent 2 não irá funcionar, pois seu *Date Type* está diferente do definido no Intent Filter.

9.6 Colocando em prática: usando Intents e Intent Filters no projeto *Devolva.me*

Agora que aprendemos a trabalhar com Intents e Intent Filters, vamos usar este recurso para melhorar o nosso aplicativo *Devolva.me*.

Conforme a especificação do projeto *Devolva.me*, definida no **Capítulo 7**, devemos disponibilizar mais duas funcionalidades na tela que lista os objetos emprestados:

1. Disponibilizar opção para efetuar uma ligação para a pessoa que pegou o objeto emprestado;

2. Disponibilizar uma opção para enviar uma mensagem para lembrar a pessoa de devolver o objeto que pegou emprestado.

Essas duas funcionalidades irão facilitar a vida do usuário permitindo que, do próprio aplicativo, ele faça uma chamada ou envie uma mensagem para a pessoa lembrando-a de devolver o objeto que pegou emprestado. Uma opção mais simples seria implementar essas funcionalidades como itens do ***Context Menu***. *[Lembre-se que aprendemos sobre o Context Menu quando implementamos uma opção para o usuário remover um registro da lista de objetos emprestados, no **Capítulo 8**]*

9.6.1 Usando a *ACTION_CALL* para efetuar chamadas telefônicas

Para possibilitar o usuário a efetuar uma chamada telefônica podemos usar uma Intent invocando a Action ***ACTION_CALL*** passando um *Data* com o número do telefone da pessoa que queremos ligar. Em nosso caso, queremos adicionar um atalho na tela com a lista de objetos emprestados para que nosso usuário possa rapidamente efetuar uma chamada telefônica para o número da pessoa que pegou um de seus objetos emprestado. Para isto, iremos adicionar um item "**Ligar**" ao *Context Menu*.

Para adicionar este item ao *Context Menu*, iremos realizar as seguintes alterações, no projeto *Devolva.me*, na classe ***ListaObjetosEmprestadosActivity***:

1. Adicionaremos uma constante à classe ***ListaObjetosEmprestadosActivity*** que irá representar o ID do item "**Ligar**" do *Context Menu*. Criaremos, então, a seguinte constante:

```
. . .
// ID da opção "Ligar" do menu de contexto
private static final int MENU_LIGAR = Menu.FIRST + 1;
. . .
```

Nota

Observe que incrementamos em 1 o valor do ID da constante ***MENU_LIGAR*** para que não haja colisão com o ID da constante ***MENU_APAGAR*** (criada anteriormente).

2. No método ***onCreateContextMenu()*** adicionaremos o item "**Ligar**" ao *Context Menu*, utilizando o ID que definimos na constante *MENU_LIGAR*. Adicionaremos, então, o seguinte trecho de código ao final deste método:

```
. . .
// Adiciona a opção "Ligar" ao Context Menu
menu.add(0, MENU_LIGAR, 0, "Ligar");
. . .
```

3. Por fim, iremos tratar a ação do item "**Ligar**" do *Context Menu* para que, ao ser clicado, efetue a chamada para a pessoa que pegou emprestado o objeto selecionado na lista. Usaremos uma Intent que irá chamar a Action *ACTION_CALL* para efetuar a chamada telefônica para esta pessoa. Para isto, iremos alterar o método ***onContextItemSelected()*** adicionando o seguinte trecho de código ao final do método (mas antes do ***return*** do método):

```
. . .
// Trata a ação da opção "Ligar" do menu de contexto
if (item.getItemId() == MENU_LIGAR) {

        // Obtemos o objeto selecionado pelo usuário, na ListView
        ObjetoEmprestado objeto = (ObjetoEmprestado) getListaObjetosEmprestados()
                .getItemAtPosition(info.position);

        // Efetua a chamada para o número de telefone cadastrado
        Intent i = new Intent(Intent.ACTION_CALL);
        i.setData(Uri.parse("tel:" + objeto.getContato().getTelefone()));
        startActivity(i);
}
. . .
```

Para que nosso aplicativo possa efetuar chamadas telefônicas, devemos adicionar a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Nossa implementação está pronta. Ao abrir a tela com a lista dos objetos emprestados, podemos efetuar uma chamada para a pessoa que pegou emprestado um dos objetos apresentados na lista. Para isto, basta selecionar um objeto na lista pressionando-o por 3 segundos para que o menu de contexto seja apresentado. Ao aparecer o menu, basta clicar na opção "**Ligar**" que o Android irá chamar a Action *ACTION_CALL*, que definimos em nossa Intent, para efetuar a chamada para o número de telefone da pessoa que pegou emprestado o objeto selecionado. Veja, na **Figura 9.9**, esta implementação em execução no emulador do Android:

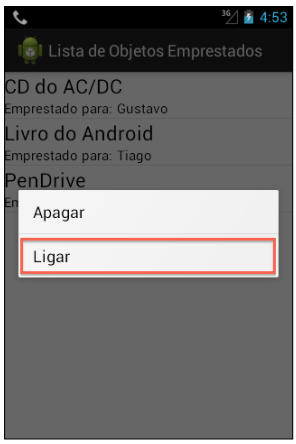


Figura 9.9. Opção "Ligar", no *Context Menu*, para efetuar uma chamada telefônica para a pessoa que pegou emprestado um dos objetos da lista.

9.6.2 Usando a *ACTION_SENDTO* para enviar mensagens SMS

Outra opção que irá facilitar a vida do nosso usuário, ao utilizar nosso aplicativo *Devolva.me*, seria adicionar um outro item, chamado "**Enviar SMS**", ao *Context Menu* da tela com a lista de objetos emprestados para enviar um SMS para a pessoa que pegou emprestado um de seus objetos. No conteúdo do SMS poderíamos adicionar uma mensagem para que esta pessoa possa lembrar de devolver o objeto emprestado.

Para adicionar este item ao *Context Menu*, iremos realizar as seguintes alterações, no projeto *Devolva.me*, na classe ***ListaObjetosEmprestadosActivity***:

1. Adicionaremos uma constante à classe ***ListaObjetosEmprestadosActivity*** que irá representar o ID do item "**Enviar SMS**" do *Context Menu*. Criaremos, então, a seguinte constante:

```
. . .
// ID da opção "Enviar SMS" do menu de contexto
private static final int MENU_ENVIAR_SMS = Menu.FIRST + 2;
. . .
```

Nota

Observe que incrementamos em 2 o valor do ID da constante ***MENU_ENVIAR_SMS*** para que não haja colisão com com os IDs criados anteriormente.

2. No método ***onCreateContextMenu()*** adicionaremos o item "**Enviar SMS**" ao *Context Menu*, utilizando o ID que definimos na constante *MENU_ENVIAR_SMS*. Adicionaremos, então, o seguinte trecho de código ao final deste método:

```
. . .
// Adiciona a opção "Enviar SMS" ao Context Menu
menu.add(0, MENU_ENVIAR_SMS, 0, "Enviar SMS");
. . .
```

3. Por fim, iremos tratar a ação do item "**Eviar SMS**" do *Context Menu* para que, ao ser clicado, abra o aplicativo de envio de mensagens SMS. Usaremos uma Intent que irá chamar a Action *ACTION_SENDTO* passando o scheme "**sms:**" no URI da Intent para que o Android se encarregue de chamar a Action para envio de mensagens SMS, que responde pela URI que iremos passar. Para isto, iremos alterar o método ***onContextItemSelected()*** adicionando o seguinte trecho de código ao final do método (mas antes do ***return*** do método):

```
. . .
// Trata a ação da opção "Enviar SMS" do menu de contexto
if (item.getItemId() == MENU_ENVIAR_SMS) {

        // Obtemos o objeto selecionado pelo usuário na ListView
        ObjetoEmprestado objeto = (ObjetoEmprestado) getListaObjetosEmprestados()
                .getItemAtPosition(info.position);

        // Envia uma mensagem SMS de lembrete para o número de telefone cadastrado
        Intent i = new Intent(Intent.ACTION_SENDTO);
        i.setData(Uri.parse("sms:" + objeto.getContato().getTelefone()));
        i.putExtra("sms_body", "Olá! Você pegou emprestado o meu objeto \"" +
                objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.");
        startActivity(i);
}
. . .
```

Nota

Perceba que na Intent definimos um Extra informando uma String com a mensagem padrão a ser enviada via SMS para facilitar a vida do usuário. O conteúdo de uma mensagem SMS é definido no Extra da Intent através da referência "**sms_body**".

Nossa implementação está pronta. Ao abrir a tela com a lista dos objetos emprestados, podemos enviar uma mensagem SMS para a pessoa que pegou emprestado um dos objetos apresentados na lista. Para isto, basta selecionar um objeto na lista pressionando-o por 3 segundos para que o menu de contexto seja apresentado. Ao aparecer o menu, basta clicar na opção "**Enviar SMS**" que o Android irá chamar a Action *ACTION_SENDTO*, que definimos em nossa Intent, passando os parâmetros informados para o envio do SMS. Veja, na **Figura 9.10**, esta implementação em execução no emulador do Android:

Figura 9.10. Opção "Enviar SMS", no *Context Menu*, para possibilitar o envio de um SMS para a pessoa que pegou emprestado um dos objetos da lista.

9.6.3 Usando a Action *android.intent.action.MAIN* para definir a Activity inicial do aplicativo

Seguindo a especificação do projeto *Devolva.me*, nosso aplicativo já implementa a tela de cadastro e a tela com a lista dos objetos cadastrados. Falta agora definir a tela inicial do aplicativo, que irá listar as opções disponíveis para o usuário. *[Ver tópico **7.2.3** do **Capítulo 7**]*

Para implementar a tela inicial do aplicativo *Devolva.me* criaremos uma nova Activity, chamada ***TelaInicialActivity***, definindo seu arquivo de layout padrão com o nome ***activity_tela_inicial.xml***.

Como a nossa tela inicial deve ter apenas uma lista com as ações disponíveis no aplicativo, usaremos uma ***ListView***, em seu arquivo de layout, para facilitar nosso trabalho. *[Lembre-se que aprendemos a usar uma ListView anteriormente, no **Capítulo 6**, quando falamos sobre os Layouts]*

Editaremos, então, o arquivo de Layout ***activity_tela_inicial.xml*** deixando-o com o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFFFFF"
    android:orientation="vertical" >

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFFFFF" />

</LinearLayout>
```

Com o arquivo de Layout já definido, precisamos agora implementar o código da Activity para definir as opções da ListView e implementar suas ações. Para facilitar nosso trabalho, estenderemos a classe ListActivity do Android em nossa Activity, deixando sua declaração da seguinte forma:

. . .

```
public class TelaInicialActivity extends ListActivity
. . .
```

Para definir as opções que serão apresentadas na ListView do arquivo de Layout, usaremos um ArrayAdapter, passando para ele um Array de String com as opções a serem apresentadas. Devemos, então, definir o nosso Array de String em uma constante da classe, da seguinte forma:

```
. . .
// Opções que serão apresentadas na ListView da tela principal.
private static final String[] OPCOES_DO_MENU = new String[] {
        "Emprestar objeto", "Listar objetos emprestados", "Sair" };
. . .
```

Definidas as opções da tela inicial em um Array de String, devemos agora definir o Adapter que irá popular a ListView com essas opções através do método ***setListAdapter()*** da classe ListActivity. Faremos isto no método ***onCreate()*** da nossa Activity, deixando-o com a seguinte implementação:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Define um ArrayAdapter com as opções definidas
         * no Array de String OPCOES_DO_MENU.
         */
        setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1, OPCOES_DO_MENU));
}
```

Com o conteúdo da nossa tela já definido, falta agora implementar os eventos que serão disparados quando o usuário clicar em um dos itens apresentados na ListView. Estes eventos deverão ser implementados da seguinte forma:

1. Opção "**Emprestar objeto**" da ListView: deverá abrir a Activity ***CadastraObjetoEmprestadoActivity***;

2. Opção "**Listar objetos emprestados**" da ListView: deverá abrir a Activity ***ListaObjetosEmprestadosActivity***;

3. Opção "**Sair**" da ListView: deverá encerrar o aplicativo.

Para definir estes eventos, devemos subscrever o método ***onListItemClick()***, da classe ListActivity, deixando-o com a seguinte implementação:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        // Evento da primeira opção apresentada na ListView: Emprestar objeto
        case 0:
                startActivity(new Intent(getApplicationContext(), CadastraObjetoEmprestadoActivity.class));
                break;
        // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
        case 1:
                startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
                break;
        // Evento da terceira opção apresentada na ListView: Sair
        default:
                finish();
```

```
        }
}
```

A implementação da Activity inicial está completamente definida, porém precisamos dizer ao Android que esta será a Activity principal do aplicativo, ou seja, a Activity que será aberta quando o usuário iniciar o aplicativo. Para isto, devemos definir no Intent Filter dessa Activity a Action ***android.intent.action.MAIN***. Definiremos, então, o Intent Filter da Activity ***TelaInicialActivity***, no arquivo ***AndroidManifest.xml***, da seguinte forma:

```
. . .
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
</intent-filter>
. . .
```

Nota

Lembre-se que apenas uma Activity do aplicativo deverá definir a Action *android.intent.action.MAIN* em seu Intent Filter.

Veja, na **Figura 9.11**, a Activity ***TelaInicialActivity*** em execução no emulador do Android:

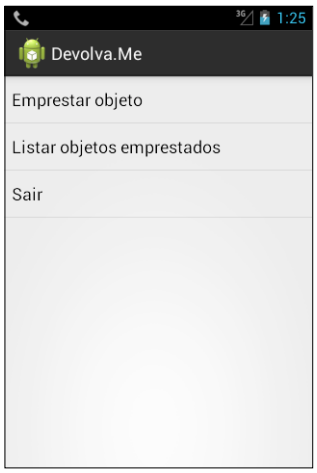


Figura 9.11. Activity *TelaInicialActivity* em execução no emulador do Android.

9.6.4 Usando a Category *android.intent.category.LAUNCHER* para criar um atalho no Launcher

No tópico anterior criamos a Activity inicial do nosso aplicativo e a definimos como principal através da Action *android.intent.action.MAIN*, definida em seu Intent Filter. Agora devemos criar um atalho para esta Activity no menu de aplicativos do Android (*Launcher*) para que o usuário possa abrir o nosso aplicativo através do Launcher.

Para isto, basta adicionar a Category ***android.intent.category.LAUNCHER*** no Intent Filter da Activity ***TelaIncialActivity*** da seguinte forma:

```
. . .
<category android:name="android.intent.category.LAUNCHER" />
. . .
```

Nota

É aconselhável o uso da Category *android.intent.category.LAUNCHER* apenas na Activity principal do aplicativo. Isto fará com que o Android crie um atalho apenas para esta Activity no Launcher.

9.6.5 Definindo Actions para o uso de Intens implícitas

Como vimos anteriormente, é possível definir Actions no Intent Filter de Activities para permitir o uso de Intents implícitas. Isto permite o acesso à estas Activities através de aplicativos externos e reduz o acoplamento entre as classes do mesmo projeto que fazem o uso destas Activities.

Vamos, então, definir Actions no Intent Filter das seguintes Activities:

• ***CadastraObjetoEmprestadoActivity***:

```
. . .
<intent-filter>
        <action android:name="br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO" />
        <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
. . .
```

• ***ListaObjetosEmprestadosActivity***:

```
<intent-filter>
        <action android:name="br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS" />
        <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Nota

Lembre-se que, para chamar uma Action através de uma Intent implícita a partir do método ***startActivity()*** ou ***startActivityForResult()***, é preciso definir a Category ***android.intent.category.DEFAULT*** no Intent Filter da Activity.

Agora que definimos uma Action no Intent Filter das Activities do nosso aplicativo *Devolva.me*, podemos alterar o código da Activity ***TelaInicialActivity*** para que ela use Intents implícitas para chamar estas Activities. Para isto, iremos alterar o método ***onListItemClick()*** deixando-o com a seguinte implementação:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        // Evento da primeira opção apresentada na ListView: Emprestar objeto
        case 0:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO"));
                break;
        // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
        case 1:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS"));
                break;
        // Evento da terceira opção apresentada na ListView: Sair
        default:
                finish();
        }
}
```

Agora qualquer aplicativo externo poderá chamar estas duas Activities através de Intents implícitas, usando as Actions definidas em seus Intent Filters.

9.7 Exercício

Agora que você aprendeu a trabalhar com Intents e Intent Filters, é hora de colocar em prática.

1. No projeto *Devolva.me*, na Activity ***ListaObjetosEmprestadosActivity***, adicione um recurso para que o usuário possa efetuar uma ligação para a pessoa que pegou emprestado um de seus objetos, usando o número de telefone cadastrado. Este recurso deve ser disponibilizado ao usuário através de uma opção no *Context Menu* da ListView. Para implementar este recurso, siga os passos:

a) Adicione a seguinte constante, na classe ***ListaObjetosEmprestadosActivity***, para representar o ID do item do Context Menu responsável por disparar o evento que irá efetuar a chamada telefônica:

```
. . .
// ID da opção "Ligar" do menu de contexto
private static final int MENU_LIGAR = Menu.FIRST + 1;
. . .
```

b) Adicione um item chamado "**Ligar**" ao Context Menu adicionando o seguinte trecho de código ao final do método ***onCreateContextMenu()*** da classe ***ListaObjetosEmprestadosActivity***:

```
. . .
// Adiciona a opção "Ligar" ao Context Menu
menu.add(0, MENU_LIGAR, 0, "Ligar");
. . .
```

c) Implemente o evento que irá efetuar a chamada telefônica quando o usuário clicar no item "**Ligar**" do Context Menu. Para isto, adicione o seguinte trecho de código ao final do método ***onContextItemSelected()*** (mas antes do ***return*** do método) da classe ***ListaObjetosEmprestadosActivity***:

```
. . .
// Trata a ação da opção "Ligar" do menu de contexto
if (item.getItemId() == MENU_LIGAR) {

        // Obtemos o objeto selecionado pelo usuário, na ListView
        ObjetoEmprestado objeto = (ObjetoEmprestado) getListaObjetosEmprestados()
                .getItemAtPosition(info.position);

        // Efetua a chamada para o número de telefone cadastrado
        Intent i = new Intent(Intent.ACTION_CALL);
        i.setData(Uri.parse("tel:" + objeto.getContato().getTelefone()));
        startActivity(i);
}
. . .
```

d) Adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

e) Configure o projeto *Devolva.me* para que a Activity ***ListaObjetosEmprestadosActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em ***Run*** > ***Run Configurations***.

f) Execute o aplicativo no emulador do Android e faça o teste clicando em um item da ListView, pressionando-o durante 3 (três) segundos e, após aparecer o Context Menu, clique no item "**Ligar**" do menu.

2. No projeto *Devolva.me*, na Activity ***ListaObjetosEmprestadosActivity***, adicione um recurso para que o usuário possa enviar uma mensagem SMS para lembrar a pessoa de devolver o objeto que pegou emprestado, usando o número de

telefone cadastrado. Este recurso deve ser disponibilizado ao usuário através de uma opção no *Context Menu* da ListView. Para implementar este recurso, siga os passos:

a) Adicione a seguinte constante, na classe ***ListaObjetosEmprestadosActivity***, para representar o ID do item do Context Menu responsável por disparar o evento que irá enviar uma mensagem SMS para a pessoa que pegou o objeto emprestado:

```
. . .
// ID da opção "Enviar SMS" do menu de contexto
private static final int MENU_ENVIAR_SMS = Menu.FIRST + 2;
. . .
```

b) Adicione um item chamado "**Enviar SMS**" ao Context Menu adicionando o seguinte trecho de código ao final do método ***onCreateContextMenu()*** da classe ***ListaObjetosEmprestadosActivity***:

```
. . .
// Adiciona a opção "Enviar SMS" ao Context Menu
menu.add(0, MENU_ENVIAR_SMS, 0, "Enviar SMS");
. . .
```

c) Implemente o evento que irá chamar o aplicativo para envio de uma mensagem SMS quando o usuário clicar no item "**Enviar SMS**" do Context Menu. Para isto, adicione o seguinte trecho de código ao final do método ***onContextItemSelected()*** (mas antes do ***return*** do método) da classe ***ListaObjetosEmprestadosActivity***:

```
. . .
// Trata a ação da opção "Enviar SMS" do menu de contexto
if (item.getItemId() == MENU_ENVIAR_SMS) {

        // Obtemos o objeto selecionado pelo usuário na ListView
        ObjetoEmprestado objeto = (ObjetoEmprestado) getListaObjetosEmprestados()
                .getItemAtPosition(info.position);

        // Envia uma mensagem SMS de lembrete para o número de telefone cadastrado
        Intent i = new Intent(Intent.ACTION_SENDTO);
        i.setData(Uri.parse("sms:" + objeto.getContato().getTelefone()));
        i.putExtra("sms_body", "Olá! Você pegou emprestado o meu objeto \"" +
                objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.");
        startActivity(i);
}
. . .
```

d) Configure o projeto *Devolva.me* para que a Activity ***ListaObjetosEmprestadosActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em ***Run*** > ***Run Configurations***.

e) Execute o aplicativo no emulador do Android e faça o teste clicando em um item da ListView, pressionando-o durante 3 (três) segundos e, após aparecer o Context Menu, clique no item "**Enviar SMS**" do menu.

9.8 Exercício

Crie uma Activity listando os recursos do aplicativo *Devolva.me* e defina-a como Activity principal. Para isto, siga os passos:

1. Crie uma Activity chamada ***TelaInicialActivity*** definindo seu arquivo de Layout com o nome ***activity_tela_inicial***.

2. Edite o arquivo de Layout ***activity_tela_inicial.xml***, deixando-o com a seguinte implementação:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
```

```
    android:background="#FFFFFF"
    android:orientation="vertical" >

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFFFFF" />

</LinearLayout>
```

3. Faça com que a classe ***TelaInicialActivity*** estenda a classe *ListActivity*, para facilicar o uso da ListView, deixando sua declaração da seguinte forma:

```
. . .
public class TelaInicialActivity extends ListActivity
. . .
```

4. Na classe ***TelaInicialActivity*** defina uma constante com um Array de String contendo as opções: "**Emprestar objeto**", "**Listar objetos emprestados**" e "**Sair**", da seguinte forma:

```
. . .
// Opções que serão apresentadas na ListView da tela principal.
private static final String[] OPCOES_DO_MENU = new String[] {
        "Emprestar objeto", "Listar objetos emprestados", "Sair" };
. . .
```

5. No método ***onCreate()***, da classe ***TelaInicialActivity***, defina um Adapter do tipo *ArrayAdapter* passando o Array de String definido pela constante *OPCOES_DO_MENU* criada anteriormente. Deixe a implementação do método da seguinte forma:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Define um ArrayAdapter com as opções definidas
         * no Array de String OPCOES_DO_MENU.
         */
        setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1, OPCOES_DO_MENU));
}
```

6. Na classe ***TelaInicialActivity*** implemente o método ***onListItemClick()*** definindo um evento para cada item da ListView, disparando suas devidas ações. O método deverá conter a seguinte implementação:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        // Evento da primeira opção apresentada na ListView: Emprestar objeto
        case 0:
                startActivity(new Intent(getApplicationContext(), CadastraObjetoEmprestadoActivity.class));
                break;
        // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
        case 1:
                startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
```

```
                    break;
                // Evento da terceira opção apresentada na ListView: Sair
                default:
                    finish();
                }
        }
```

7. Adicione a Action *android.intent.action.MAIN* ao Intent Filter da Activity ***TelaInicialActivity***, no arquivo *AndroidManifest.xml*, deixando seu Intent Filter da seguinte forma:

```
. . .
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
</intent-filter>
. . .
```

8. Configure o projeto *Devolva.me* para que a Activity ***TelaInicialActivity*** seja executada ao iniciar o aplicativo no emulador do Android, em ***Run*** > ***Run Configurations***.

9. Execute o aplicativo no emulador do Android e faça o teste clicando em um item da ListView, na tela inicial.

9.9 Exercício

Adicione a Category *android.intent.category.LAUNCHER* no Intent Filter da Activity ***TelaInicialActivity***, no arquivo *AndroidManifest.xml*, para que o Android possa criar um atalho para esta Activity no menu de aplicativos (Launcher). Para isto, siga os passos:

1. Adicione a seguinte Category ao Intent Filter da Activity ***TelaInicialActivity***:

```
. . .
<category android:name="android.intent.category.LAUNCHER" />
. . .
```

2. Execute o projeto no emulador do Android.

3. Vá até o menu de aplicativos do Android (Laucher) e acesse o aplicativo através do seu atalho, chamado *Devolva.me*.

9.10 Exercício opcional

Para que as Activities do aplicativo *Devolva.me* possam ser chamadas através de uma Intent implícita, defina uma Action ao Intent Filter das Activities ***CadastraObjetoEmprestadoActivity*** e ***ListaObjetosEmprestadosActivity***. Para isto, siga os passos:

1. No arquivo *AndroidManifest.xml*, defina o seguinte Intent Filter à Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
<intent-filter>
        <action android:name="br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO" />
        <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
. . .
```

2. No arquivo *AndroidManifest.xml*, defina o seguinte Intent Filter à Activity ***ListaObjetosEmprestadosActivity***:

```
. . .
<intent-filter>
        <action android:name="br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS" />
```

```
            <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
. . .
```

3. Na classe ***TelaInicialActivity*** altere o método ***onListItemClick()***, deixando sua implementação da seguinte forma:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        // Evento da primeira opção apresentada na ListView: Emprestar objeto
        case 0:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO"));
                break;
        // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
        case 1:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS"));
                break;
        // Evento da terceira opção apresentada na ListView: Sair
        default:
                finish();
        }
}
```

4. Execute o aplicativo no emulador do Android e faça o teste clicando em um item da ListView, na tela inicial.

10 - Content Providers

No **Capítulo 8**, quando falamos sobre o armazenamento de informações em banco de dados, vimos que, por segurança, o Android não permite que um aplicativo acesse o banco de dados de outro aplicativo, e esta regra serve também para qualquer outro tipo de armazenamento de dados em um aplicativo (como o armazenamento em arquivo, por exemplo). No entanto, o Android disponibiliza uma forma para que um aplicativo possa compartilhar suas informações com outros aplicativos, e isto é feito através dos *Content Providers* (Provedores de Conteúdo).

Content Provider é um recurso do Android que possibilita o compartilhamento de informações entre aplicativos de forma segura e, através dele, podemos permitir outros aplicativos a consultar, inserir, atualizar e deletar as informações compartilhadas.

10.1 Arquitetura do Content Provider

Imagine o Content Provider como uma interface que irá abstrair o acesso às informações armazenadas por um aplicativo, seguindo suas regras definidas. Como exemplo, caso tenhamos um aplicativo de cadastro de pessoas e queiramos disponibilizar suas informações cadastradas para outro aplicativo, podemos criar um Content Provider para fornecer estas informações. A **Figura 10.1** ilustra a arquitetura de um Content Provider.

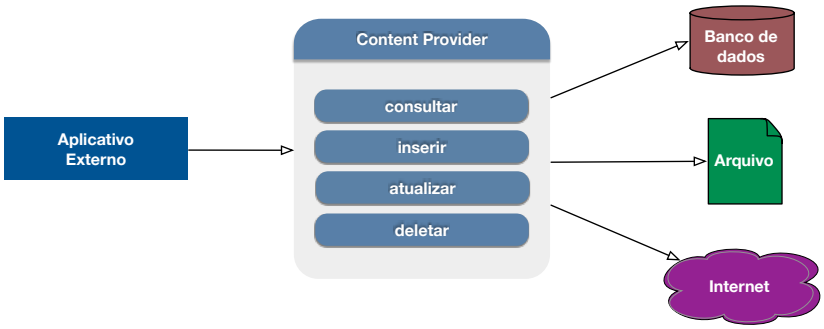


Figura 10.1. Arquitetura de um Content Provider

10.2 Utilizando um Content Provider

Um Content Provider é acessado através de um ***URI*** *[Lembre-se que aprendemos sobre as URIs anteriormente, em outro capítulo]* e, assim como na consulta realizada em um Banco de Dados, um Content Provider retorna o resultado em forma de tabela, em um objeto ***Cursor***, possibilitando a navegação entre os dados consultados.

Para que um aplicativo possa usar o Content Provider, é preciso conhecer algumas informações, como:

• Conhecer seu URI;

• Saber quais os dados que o Content Provider disponibiliza e o tipo desses dados (nome das colunas e os tipos de dados dessas colunas);

• Saber quais as permissões necessárias para acessar o Content Provider.

Como exemplo, o aplicativo nativo de contatos do Android possui um Content Provider que permite outros aplicativos a consultarem os contatos armazenados. Veja, na **Figura 10.2**, o URI usado para acessar o Content Provider do aplicativo de Contatos do Android.

Figura 10.2. URI do Content Provider do aplicativo de contatos do Android

Nota

Lembre-se que o URI é usado para indicar aonde os dados são obtidos e, em um Content Provider, o URI sempre inicia com "***content://***".

O Content Provider disponibiliza uma classe chamada ***ContentResolver*** que provê todos os métodos necessários para que os aplicativos possam manipular os dados compartilhados. Veja os métodos disponíveis nesta classe:

- **query()**: utilizado para realizar uma consulta através de um Content Provider;
- **insert()**: utilizado para inserir dados através de um Content Provider;
- **update()**: utilizado para atualizar dados através de um Content Provider;
- **delete()**: utilizado para remover dados através de um Content Provider.

Nota

Em uma Activity, para obter o objeto ***ContentResolver***, basta usar o método ***getContentResolver()***. Este método estará disponível em qualquer classe que herdar da classe *Context* do Android.

10.2.1 Convenções ao usar um Content Provider

O Android possui algumas convenções para a utilização de Content Provider que devem ser seguidas para que não haja problemas em seu aplicativo.

Uma dessas convenções é a utilização da mesma convenção de nomes de pacotes do Java para o *Authority* da URI, ou seja, a utilização do nome do seu domínio (ao contrário) seguido do nome do projeto e o nome a ser especificado para o Content Provider. Como exemplo, ao criar um Content Provider para compartilhar as informações de cadastro de pessoas, poderíamos usar a seguinte Authority:

br.com.empresa.projeto.pessoas

Outra convenção é a utilização de constantes na classe do Content Provider disponibilizando as URIs e os nomes das colunas disponíveis, para que outros desenvolvedores não precisem decorá-las. *[Veremos na prática como isto funciona, nos próximos tópicos]*

10.3 Realizando uma consulta através de um Content Provider

Como vimos, para consultar dados através de um Content Provider usamos o método ***query()*** da classe *ContentResolver*, parecido com a forma como fazemos para realizar uma consulta no banco de dados.

Veja a sintaxe para realizar uma consulta em um Content Provider através do método ***query()*** da classe ContentResolver:

```
. . .
ContentResolver contentResolver = getContentResolver();
Cursor cursor = contentResolver.query(URI, String[] {COLUNAS}, CLAUSULA_WHERE,
        String[] {ARGUMENTOS_DA_CLAUSULA_WHERE}, ORDEM);
. . .
```

Onde:

- URI = URI do Content Provider a ser consultado;
- String[] {COLUNAS} = Nomes das colunas que queremos consultar no Content Provider;
- CLAUSULA_WHERE = cláusula WHERE que irá especificar os registros a serem consultados. Neste parâmetro, usamos sempre a sintaxe: ***CAMPO = VALOR*** ou ***CAMPO = ?*** (o interrogação é o caractere curinga que será substituído pelos valores contidos no array de String passado em ARGUMENTOS_DA_CLAUSULA_WHERE, ou seja, será substituído pelos argumentos da cláusula WHERE). Como exemplo, podemos usar neste parâmetro o valor: "_id = ?" (para consultar um registro com um determinado ID);
- String[] {ARGUMENTOS_DA_CLAUSULA_WHERE}: array de String com os argumentos a serem injetados na cláusula WHERE, caso tenhamos usado um caractere curinga. Como exemplo, podemos usar neste parâmetro o valor: String[] {"1"} (que irá substituir o primeiro caractere curinga da cláusula WHERE pelo valor "1", e assim sucessivamente);
- ORDEM = ordem do resultado da consulta. Como exemplo: ***nome DESC*** (para ordenar os dados pelo campo "*nome*" de forma descendente); ***nome ASC*** (para ordenar os dados pelo campo "*nome*" de forma ascendente).

Usando um exemplo real, veja como faríamos para consultar um determinado nome cadastrado no aplicativo nativo de Contatos do Android:

```
. . .
ContentResolver contentResolver = getContentResolver();
Cursor cursor = contentResolver.query(Phone.CONTENT_URI,
        new String[]{ Phone._ID, Phone.DISPLAY_NAME, Phone.NUMBER },
        Phone.DISPLAY_NAME + " LIKE ?",
        new String[]{ "%Maria%" },
        Phone.DISPLAY_NAME + " ASC");
. . .
```

No exemplo acima, usamos o Content Provider do aplicativo de contato para localizar um contato que possua o nome "Maria". Veja que, ao invés de especificar o URI e o nome das colunas, usamos as constantes da classe *Phone*, localizada na classe ***ContactsContract*** do Android, para facilitar nosso trabalho.

Nota

A classe ***ContactsContract*** do Android possui constantes com as definições de URIs e nomes de colunas usados no aplicativo de Contatos e em seu Content Provider.

Como mencionamos, além de ter que conhecer o URI e os nomes das colunas para consultar informações em um Content Provider, devemos também conhecer as permissões necessárias para sua utilização. No caso do Content Provider do aplicativo de contatos do Android, para usá-lo é necessário adicionar a seguinte permissão no arquivo *AndroidManifest.xml [Lembre-se que aprendemos sobre as permissões de acesso anteriormente, no* ***Capítulo 5****]*:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

10.4 Colocando em prática: usando um Content Provider no projeto *Devolva.me*

Agora que aprendemos a consultar dados em um Content Provider, vamos usar este recurso para melhorar o nosso aplicativo *Devolva.me*.

De acordo com a definição do projeto *Devolva.me*, no **Capítulo 7**, veja que todos os seus requisitos foram implementados, conforme especificado. Mas, como desenvolvedores, sabemos que o requisito pode mudar. Nosso cliente José, que solicitou o aplicativo *Devolva.me*, está muito satisfeito com o aplicativo que você desenvolveu para ele, porém, após 1 mês

de uso do aplicativo, José percebeu que as pessoas que pegam seus objetos emprestados já estão cadastradas no aplicativo de contatos do seu dispositivo e ele queria uma forma de, ao invés de cadastrar novamente esta pessoa, ele apenas a selecionasse no aplicativo de contatos (que já armazena o nome e o número de telefone da pessoa).

Para deixar nosso cliente José ainda mais satisfeito e facilitar o uso do aplicativo, decidimos atender à sua solicitação. Portanto, iremos fazer alguns ajustes no aplicativo *Devolva.me* para que ele consiga buscar o nome e o número do telefone da pessoa que pegou o objeto emprestado na base de dados do aplicativo de contatos do seu dispositivo com Android. Para implementar esta funcionalidade, consultaremos estas informações através do Content Provider disponibilizado pelo aplicativo nativo de contatos.

Nota

O Content Provider do aplicativo de contatos do Android disponibiliza o acesso às suas informações cadastradas, como: nome do contato, número de telefone, e-mail, endereço, ID do registro, etc.

De acordo com a nova solicitação, especificamos o seguinte requisito:

• Ao cadastrar um novo objeto emprestado, o sistema deverá possibilitar o usuário a selecionar, no aplicativo de contatos do Android, a pessoa que pegou o objeto emprestado.

Em um simples rascunho, junto ao nosso cliente, definimos como ficará a nossa tela de cadastro, após a modificação solicitada, conforme podemos ver na **Figura 10.3**.

Figura 10.3. Nova tela de cadastro do aplicativo *Devolva.me* após solicitação de mudança feita pelo cliente.

10.4.1 Consultando um contato através do Content Provider do aplicativo de Contatos

Na primeira versão do aplicativo *Devolva.me* que desenvolvemos, o cliente tinha que informar o nome da pessoa que pegou o objeto emprestado e o seu número de telefone de contato. Agora iremos pegar estes dados diretamente do aplicativo de Contatos do Android.

Se analisarmos melhor o caso, percebemos que temos duas opções para implementar esta solução:

1. Pegar o nome e o número de telefone no aplicativo de contatos e persistir estas informações no banco de dados do nosso aplicativo;

2. Pegar apenas o ID do contato no aplicativo de contatos e persistí-lo no banco de dados do nosso aplicativo. Mas isso implicaria em alterar a estrutura atual do banco de dados e também a implementação que lista os objetos emprestados já cadastrados, para que ele consulte o nome e o número de telefone no aplicativo de contatos para cada objeto listado.

Se pararmos para pensar, a primeira solução com certeza seria a menos trabalhosa, uma vez que alteraríamos apenas a funcionalidade do cadastro, e consumiria menos recursos do dispositivo. Mas imagine que o nosso cliente precise atualizar o número do telefone de um contato (no aplicativo de contatos) que, por coincidência, está com um de seus objetos emprestados e devidamente cadastrado no aplicativo *Devolva.me*. Se isso acontecesse, o nosso cliente correria o risco de efetuar uma chamada ou enviar uma mensagem SMS através do nosso aplicativo e esta chamada/mensagem cair no número de telefone incorreto. *[Já pensou como nosso cliente ficaria desapontado com o nosso aplicativo, se isso acontecesse?]*

Pensando na felicidade e satisfação do nosso cliente, optaremos por implementar a segunda solução, armazenando apenas o ID do contato. Desta forma teremos a certeza de que a lista dos objetos emprestados, cadastrados em nosso aplicativo, mostrará os dados de contato sempre atualizados, uma vez que estes dados serão consultados em tempo real no aplicativo de contatos do Android, usando seu Content Provider.

Para implementar esta nova funcionalidade, algumas alterações na estrutura do projeto deverão ser feitas:

1. A estrutura do nosso Banco de Dados deverá ser alterada, removendo as colunas "**pessoa**" e "**telefone**" e adicionando a coluna "**contato_id**" (que irá armazenar o ID do contato obtido através do aplicativo de contatos);

2. Na tela de cadastro de um novo objeto emprestado deveremos remover os campos onde informávamos o nome e telefone e, no lugar, adicionar apenas um botão que abrirá o aplicativo de contatos para que o usuário possa selecionar um contato armazenado em seu dispositivo;

3. No método do DAO que consulta os objetos emprestados cadastrados, deveremos realizar algumas modificações para que o mesmo, através do ID de cada contato, consulte o nome e número de telefone do contato, através do Content Provider de Contatos, para mostrar ao usuário.

Alterando a estrutura do Banco de Dados para adequar à nova funcionalidade

Como mencionado, deveremos alterar a estrutura do nosso Banco de Dados, removendo as colunas "**pessoa**" e "**telefone**" e adicionando a coluna "**contato_id**" (que irá armazenar um número inteiro que representará o ID do contato). Nossa nova estrutura de dados ficará da seguinte forma:

objeto_emprestado	
_id	INTEGER
objeto	TEXT
contato_id	INTEGER
data_emprestimo	INTEGER

Figura 10.4. Nova estrutura da tabela "*objeto_emprestado*" do aplicativo *Devolva.me*

Para implementar estas alterações, a primeira alteração a ser feita é alterar a classe modelo ***Contato*** adicionando um atributo para armazenar o ID do contato. Após esta modificação, nossa classe conterá a seguinte implementação:

```
public class Contato implements Serializable {

    private static final long serialVersionUID = 1L;
```

```
        private Integer id;
        private String nome;
        private String telefone;

        // getters e setters
        . . .
    }
```

Nota

Perceba que, apesar da nova estrutura da tabela não conter mais os campos "**pessoa**" e "**telefone**", não os removemos da classe ***Contato***, pois os usaremos para injetar nestes campos os dados consultados através do Content Provider do aplicativo de Contatos.

Criando uma classe utilitária para consultar um contato no aplicativo de Contatos do Android

Para facilitar nosso trabalho, criaremos uma nova classe utilitária, chamada ***Contatos***, que irá consultar os dados de um contato. Nossa classe será criada no pacote ***br.com.hachitecnologia.devolvame.util*** e terá a seguinte implementação:

```
public class Contatos {

    /**
     * Consulta um contato através de seu ID.
     * @param id
     * @param context
     * @return
     */
    public static Contato getContato(int id, Context context) {

        ContentResolver cr = context.getContentResolver();
        /**
         * Consultamos no Content Provider do aplicativo de Contatos do
         * Android o contato que tenha o ID recebido como parâmetro.
         */
        Cursor cursor = cr.query(Phone.CONTENT_URI, null, Phone.CONTACT_ID + " = ?",
                new String[]{String.valueOf(id)}, null);

        Contato contato = null;

        // Iteramos sobre o Cursor para obter os dados desejados
        if (cursor.moveToFirst()) {
            contato = new Contato();
            // Obtém o ID do contato
            contato.setId(id);
            // Obtém o Nome do contato
            contato.setNome(cursor.getString(cursor.getColumnIndex(Phone.DISPLAY_NAME)));
            // Obtém o Telefone do contato
            contato.setTelefone(cursor.getString(cursor.getColumnIndex(Phone.NUMBER)));
        }

        // Fechamos o Cursor
        cursor.close();

        return contato;

    }

}
```

Para que nosso aplicativo possa acessar os contatos do Android, precisamos adicionar a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Agora, deveremos alterar a nossa classe ***DBHelper***. A primeira alteração a ser feita nessa classe é o incremento em uma unidade do valor atribuído à constante **VERSAO_DO_BANCO**, para informarmos ao Android que a estrutura da nossa tabela sofrerá alterações. Para isto, devemos substituir a seguinte linha de código:

```
. . .
// Versão atual do banco de dados
private static final int VERSAO_DO_BANCO = 1;
. . .
```

por esta linha:

```
. . .
// Versão atual do banco de dados
private static final int VERSAO_DO_BANCO = 2;
. . .
```

Com isto, o Android saberá que a estrutura da tabela "**objeto_emprestado**" deverá ser atualizada, pois é esta variável que define a versão atual do banco de dados. Isto fará com que o método ***onUpgrade()*** seja chamado.

Outra alteração de devemos fazer na classe ***DBHelper*** é alterar a SQL que será executada no método ***onCreate()***, para definir a nova estrutura de dados. Deixaremos este método com a seguinte implementação:

```
. . .
@Override
public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                        + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                        + ",objeto TEXT NOT NULL"
                        + ",contato_id INTEGER NOT NULL"
                        + ",data_emprestimo INTEGER NOT NULL"
                        + ");";
        db.execSQL(sql);
}
. . .
```

Nota

Perceba que, na SQL do método ***onCreate()***, removemos os campos "**pessoa**" e "**telefone**" e adicionamos o campo "**contato_id**".

Devemos agora alterar o método ***adiciona()*** do nosso DAO, ***ObjetoEmprestadoDAO***, removendo a referência dos campos que foram removidos e adicionando uma referência para o novo campo da tabela, para que as informações sejam persistidas corretamente. Nosso método ficará com a seguinte implementação:

```
. . .
public void adiciona(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // persistidos no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("data_emprestimo", System.currentTimeMillis());
        values.put("contato_id", objeto.getContato().getId());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Insere o registro no banco de dados
        long id = db.insert("objeto_emprestado", null, values);
```

```
            objeto.setId(id);

            // Encerra a conexão com o banco de dados
            db.close();
    }
    . . .
```

Da mesma forma, deveremos alterar o método ***listaTodos()*** do DAO, removendo os campos que não serão mais utilizados e adicionando o campo "**contato_id**". Após a mudança, nosso método ficará com a seguinte implementação:

```
    . . .
    public List<ObjetoEmprestado> listaTodos() {

            // Cria um List guardar os objetos consultados no banco de dados
            List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

            // Instancia uma nova conexão com o banco de dados em modo leitura
            SQLiteDatabase db = dbHelper. getReadableDatabase();

            // Executa a consulta no banco de dados
            Cursor c = db.query("objeto_emprestado", null, null, null, null, null,
                            "objeto ASC");

            /**
             * Percorre o Cursor, injetando os dados consultados em um objeto do
             * tipo ObjetoEmprestado e adicionando-os na List
             */
            try {
                    while (c.moveToNext()) {
                            ObjetoEmprestado objeto = new ObjetoEmprestado();
                            objeto.setId(c.getLong(c.getColumnIndex("_id")));
                            objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));
                            int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                            Contato contato = Contatos.getContato(contatoID, context);
                            objeto.setContato(contato);

                            objetos.add(objeto);
                    }

            } finally {
                    // Encerra o Cursor
                    c.close();
            }

            // Encerra a conexão com o banco de dados
            db.close();

            // Retorna uma lista com os objetos consultados
            return objetos;
    }
    . . .
```

Por último, devemos alterar também o método ***atualiza()*** do nosso DAO, deixando-o com a seguinte implementação:

```
    . . .
    public void atualiza(ObjetoEmprestado objeto) {
            // Encapsula no objeto do tipo ContentValues os valores a serem
            // atualizados no banco de dados
            ContentValues values = new ContentValues();
            values.put("objeto", objeto.getObjeto());
            values.put("contato_id", objeto.getContato().getId());
```

```
            // Instancia uma conexão com o banco de dados, em modo de gravação
            SQLiteDatabase db = dbHelper.getWritableDatabase();

            // Atualiza o registro no banco de dados
            db.update("objeto_emprestado", values, "_id=?", new String[] { objeto
                        .getId().toString() });

            // Encerra a conexão com o banco de dados
            db.close();
    }
    . . .
```

Alterando a tela e a Activity de cadastro

Outra tarefa que devemos fazer é alterar a tela de cadastro para deixá-la de acordo com o protótipo do rascunho visto na **Figura 10.3**. No arquivo de Layout iremos remover os campos onde o usuário informava o *nome* e o *número de telefone* da pessoa e colocaremos no lugar apenas um botão para o usuário selecionar o contato a partir do aplicativo de contatos do Android. Vamos, então, editar o arquivo de Layout ***activity_cadastra_objeto_emprestado.xml***, substituindo o trecho de código abaixo:

```
. . .
<EditText
   android:id="@+id/cadastro_objeto_campo_pessoa"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Informe o nome da pessoa"
   android:inputType="textPersonName" />

<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:paddingTop="15.0dip"
   android:text="Telefone:" />

<EditText
   android:id="@+id/cadastro_objeto_campo_telefone"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Informe o telefone da pessoa"
   android:inputType="phone" />
. . .
```

pelo seguinte trecho de código:

```
. . .
<Button
   android:id="@+id/botao_selecionar_contato"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center"
   android:text="Selecionar na Lista de Contatos" />

<TextView
   android:id="@+id/cadastro_objeto_campo_pessoa"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center"
   android:textSize="15.0sp"
   android:visibility="gone" />
. . .
```

Dica

No componente *TextView* que inserimos no arquivo de Layout acima, perceba que utilizamos o atributo "***visibility***". O atributo ***android:visibility*** em uma View determina se o componente será ou não apresentado na tela para o usuário. Podemos usar os seguintes valores para este atributo:

- **visible**: o componente é exibido na tela (este é o valor padrão);
- **invisible**: o componente **não** é exibido na tela, porém é deixado um espaço na tela para o componente;
- **gone**: o componente não é exibido na tela e seu espaço não é reservado, podendo ser preenchido por outros componentes.

Após alterações realizadas no arquivo de Layout, precisamos agora alterar o código da nossa Activity de cadastro ***CadastraObjetoEmprestadoActivity***.

A primeira alteração a ser feita na Activity de cadastro é retirar as variáveis que faziam referência para as View que foram removidas do arquivo de Layout e adicionar a referência para as novas Views adicionadas. Devemos, então, substituir o trecho de código abaixo:

```
. . .
private EditText campoNomePessoa;
private EditText campoTelefone;
. . .
```

pelo seguinte trecho de código:

```
. . .
private TextView campoNomePessoa;
private Button botaoSelecionarContato;
. . .
```

No método ***onCreate()*** da Activity devemos corrigir as referências às Views do arquivo de Layout. Para isto, no método ***onCreate()*** vamos substituir o trecho de código abaixo:

```
. . .
campoNomePessoa = (EditText) findViewById(R.id.cadastro_objeto_campo_pessoa);
campoTelefone = (EditText) findViewById(R.id.cadastro_objeto_campo_telefone);
. . .
```

pelo seguinte trecho de código:

```
. . .
campoNomePessoa = (TextView) findViewById(R.id.cadastro_objeto_campo_pessoa);
botaoSelecionarContato = (Button) findViewById(R.id.botao_selecionar_contato);
. . .
```

Adicionada as devidas referências para as novas Views do arquivo de Layout na Activity, precisamos agora definir a ação do botão "**Selecionar contato na lista**". A idéia para este botão é que ele chame uma Action que permita a seleção de um item em uma lista e nos retorne o item selecionado. Para que isto seja possível, devemos passar como parâmetro para esta Action a lista com os contatos armazenados no aplicativo de contatos do Android. Outro ponto importante é que, como iremos esperar um retorno desta Action (no caso, o contato selecionado pelo usuário) iremos usar o método ***startActivityForResult()*** para chamar esta Action. Para isto, devemos definir uma constante na classe que irá identificar o retorno vindo desta Action. Adicionaremos, então, a seguinte constante à Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
```

```
private static final int ID_RETORNO_SELECIONA_CONTATO = 1234;
. . .
```

Com o identificador definido, devemos agora definir a ação para o botão "**Selecionar contato na lista**". Vamos, então, adicionar o seguinte trecho de código ao final do método ***onCreate()*** da Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
/**
 * O botão "Selecionar contato na lista" irá abrir a Activity de
 * Contatos do Android para que o usuário possa selecionar um
 * contato na lista.
 */
botaoSelecionarContato.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                /**
                 * Na Intent, definimos a Action Intent.ACTION_PICK
                 * do Android, usada para retornar um determinado item
                 * selecionado em uma lista.
                 */
                Intent i = new Intent(Intent.ACTION_PICK);
                /**
                 * Passamos a lista de contatos, do aplicativo de Contatos
                 * nativo do Android, para que o usuário possa selecionar
                 * um contato na lista.
                 */
                i.setData(Contacts.CONTENT_URI);
                // Abre a lista com os contatos para seleção
                startActivityForResult(i, ID_RETORNO_SELECIONA_CONTATO);
        }
});
. . .
```

Perceba que, para mostrar uma lista que possibilite a seleção de um item e retorne um resultado, usamos a Action *Intent.**ACTION_PICK*** em uma Intent, passando como parâmetro a lista de contatos amazenados no aplicativo de contatos do Android através da URI *Contacts.**CONTENT_URI*** definida no método ***setData()***.

Nota

A Action *Intent.**ACTION_PICK*** é nativa do Android e serve para retornar um determinado item selecionado em uma lista.

Para permitir também a edição/alteração do contato selecionado, devemos implementar a mesma ação quando o usuário clicar sobre o nome selecionado anteriormente. Para isto, devemos adicionar o seguinte trecho de código ao final do método ***onCreate()*** da Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
/**
 * Ao clicar sobre o nome do Contato selecionado, o usuário
 * será direcionado para a Activity de Contatos do Android,
 * permitindo-o selecionar um outro contato.
 */
campoNomePessoa.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                /**
                 * Na Intent, definimos a Action Intent.ACTION_PICK
                 * do Android, usada para retornar um determinado item
                 * selecionado em uma lista.
                 */
```

```
                Intent i = new Intent(Intent.ACTION_PICK);
                /**
                 * Passamos a lista de contatos, do aplicativo de Contatos
                 * nativo do Android, para que o usuário possa selecionar
                 * um contato na lista.
                 */
                i.setData(Contacts.CONTENT_URI);
                // Abre a lista com os contatos para seleção
                startActivityForResult(i, ID_RETORNO_SELECIONA_CONTATO);
            }
        });
        . . .
```

Como usamos o método ***startActivityForResult()*** para chamar a Action e esperamos um retorno (no caso, o contato selecionado), devemos implementar o método ***onActivityResult()*** para tratar o retorno vindo da Action. Portanto, iremos implementar o seguinte código na Activity ***CadastraObjetoEmprestadoActivity***:

```
/**
 * Trata o resultado vindo de uma Action chamada através
 * do método startActivityForResult().
 */
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    switch (requestCode) {

    /**
     * Trata o retorno vindo da tela de seleção de contatos.
     */
    case (ID_RETORNO_SELECIONA_CONTATO):
        if (resultCode == Activity.RESULT_OK) {
            /**
             *  Após selecionado o contato pelo usuário, recebemos uma
             *  URI que irá apontar para o contato selecionado.
             */
            Uri contactData = data.getData();
            /**
             *  Usaremos um ContentResolver para consultar os dados
             *  do contato selecionado no ContentProvider do aplicativo
             *  de Contatos do Android.
             */
            ContentResolver contentResolver = getContentResolver();
            /**
             * Executamos a query e atribuímos o resultado em um Cursor
             * para navegarmos sobre os dados retornados pelo ContentProvider.
             * Na query passamos apenas a URI, sem definir nenhum parâmetro adicional,
             * já que a URI retornada pela Action aponta diretamente para o contato
             * selecionado.
             */
            Cursor cursor = contentResolver.query(contactData, null, null, null, null);
            // Iteramos sobre o Cursor para obter os dados desejados
            if (cursor.moveToFirst()) {
                // Obtém o ID do contato
                objetoEmprestado.getContato().setId(cursor
                        .getInt(cursor.getColumnIndex(Phone._ID)));
                // Obtém o Nome do contato
                objetoEmprestado.getContato().setNome(cursor.getString(cursor
                        .getColumnIndex(Phone.DISPLAY_NAME)));

                /**
                 * Após selecionado o contato, não há mais necessidade de
                 * mostrar o botão "Selecionar contato na lista". Portanto,
                 * iremos esconder o botão definindo sua visibilidade para GONE.
```

```
                                        */
                                        botaoSelecionarContato.setVisibility(View.GONE);

                                        /**
                                         * Alteramos a TextView "cadastro_objeto_campo_pessoa" definindo
                                         * em seu texto o nome do contato selecionado e definimos a
                                         * visibilidade desta View para VISIBLE, para que ela fique
                                         * visível para o usuário.
                                         */
                                        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
                                        campoNomePessoa.setVisibility(View.VISIBLE);
                                }

                                // Fechamos o Cursor
                                cursor.close();

                        }
                        break;
                }
        }
```

Perceba que, no método ***onActivityResult()*** nós tratamos o retorno vindo da Action de seleção de contato. O retorno que recebemos nada mais é do que uma URI que aponta para o contato selecionado na lista, pelo usuário. De posse desta URI, usamos um *ContentResolver* para consultar no Content Provider do aplicativo de contatos do Android os dados do contato selecionado.

Como não iremos mais persistir o nome e nem o número de telefone da pessoa que pegou um objeto emprestado, em nosso banco de dados, podemos remover as seguintes linhas do código que implementa a ação do botão salvar, da Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
objetoEmprestado.getContato().setNome(campoNomePessoa.getText().toString());
objetoEmprestado.getContato().setTelefone(campoTelefone.getText().toString());
. . .
```

Nosso cadastro de objetos emprestados está preparado para usar o Content Provider do aplicativo de contatos do Android para definir a pessoa que pegou um objeto emprestado. Porém, se executarmos o projeto no emulador, percebemos que a ação de editar um registro salvo no aplicativo está quebrada, ou seja, parou de funcionar. Isto aconteceu porque essa funcionalidade esperava vir do banco de dados o nome e o número de telefone da pessoa que pegou o objeto emprestado para mostrar na tela, mas agora estes dados não são mais salvos no banco de dados, pois guardamos apenas o ID do contato para que estas informações sejam recuperadas do aplicativo de contatos.

Devemos, então, corrigir o código para edição dos registros cadastrados em nosso aplicativo. Para isto, devemos alterar o código da nossa classe ***CadastraObjetoEmprestadoActivity***. Nesta classe, vamos apenas alterar o método ***onCreate()*** substituindo o trecho de código abaixo:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoTelefone.setText(objetoEmprestado.getContato().getTelefone());
}
. . .
```

pelo seguinte trecho de código:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());

        // Remove da tela o botão "Selecionar contato na lista"
        botaoSelecionarContato.setVisibility(View.GONE);

        /**
         * Alteramos a TextView "cadastro_objeto_campo_pessoa" definindo
         * em seu texto o nome do contato que consultamos no Content Provider
         * e definimos a visibilidade desta View para VISIBLE, para que ela
         * fique visível para o usuário.
         */
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoNomePessoa.setVisibility(View.VISIBLE);
}
. . .
```

Pronto! Nossa implementação está completa e atendendo perfeitamente à solicitação do nosso cliente. Veja, na **Figura 10.5**, como ficou a nossa tela de cadastro ao executar o aplicativo no emulador do Android.

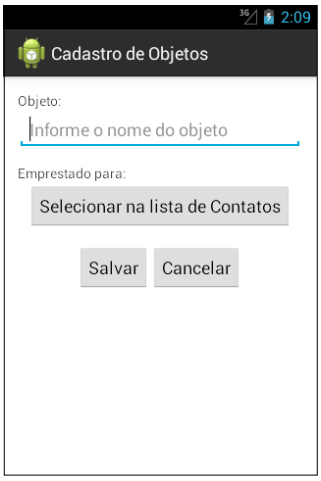


Figura 10.5. Nova tela de cadastro de objetos emprestados do aplicativo *Devolva.me*.

10.5 Criando um Content Provider

Até aqui nós aprendemos o conteito de um Content Provider e como realizar uma consulta através deste recurso. Iremos agora aprender a criar nosso próprio Content Provider, permitindo que outros aplicativos possam ter acesso às informações armazenadas pelo nosso aplicativo.

Nota

É importante lembrar que um Content Provider expõe os dados do seu aplicativo para que outros aplicativos possam ter acesso, portanto crie um Content Provider para seu aplicativo apenas se realmente desejar compartilhar informações com outros aplicativos. Fique também atento à sua segurança, expondo os dados de forma segura de acordo com a sua necessidade: se deverá apenas prover acesso às informações (modo leitura) ou se permitirá também a inserção, alteração ou remoção de dados através do Content Provider.

Para criar um Content Provider precisamos criar uma classe que estenda a classe ***ContentProvider*** do Android e configurá-lo no arquivo *AndroidManifest.xml*.

1. Criando um Content Provider

Um classe que servirá de Content Provider deve estender a classe *ContentProvider* do Android. Veja abaixo a estrutura de um Content Provider:

```
public class PessoaProvider extends ContentProvider {

    @Override
    public boolean onCreate() {
        . . .
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        . . .
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        . . .
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        . . .
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        . . .
    }

    @Override
    public String getType(Uri uri) {
        . . .
    }

}
```

O exemplo acima poderia ser um Content Provider para compartilhar informações de cadastro de pessoas. Perceba que subscrevemos alguns métodos da classe *ContentProvider*. A implementação desses métodos é necessária. Veja abaixo a finalidade de cada um destes métodos:

- **query()**: utilizado para realizar uma consulta;
- **insert()**: utilizado para inserir dados;
- **update()**: utilizado para atualizar dados;
- **delete()**: utilizado para remover dados;
- **getType()**: retorna o tipo dos registros de um Content Provider.

10.6 Colocando em prática: criando um Content Provider no projeto *Devolva.me*

Agora que aprendemos a criar um em um Content Provider, vamos usar este recurso para melhorar o nosso aplicativo *Devolva.me*.

Nosso projeto trata-se de um aplicativo simples, sem muita necessidade de um recurso de compartilhamento de informações. Mas imagine que, por algum motivo, você deseja compartilhar as informações armazenadas no banco de dados com outro aplicativo, para que este possa ter acesso aos dados dos objetos emprestados. Podemos fazer isto de maneira simples usando um Content Provider.

A idéia para o nosso Content Provider é simples: iremos permitir que outros aplicativos possam ter acesso a um determinado registro do banco de dados através de seu ID ou permitir que seja feita uma consulta mais abrangente, em cima de todos os registros. Além disso, iremos disponibilizar a funcionalidade de inserir dados em nosso banco de dados através de nosso Content Provider.

No projeto *Devolva.me* vamos criar uma classe, chamada ***ObjetosEmprestadosProvider***, dentro do pacote ***br.com.hachitecnologia.devolvame.provider***, que será o nosso Content Provider. O primeiro passo para esta classe é fazer com que ela estenda a classe *ContentProvider* do Android, deixando sua declaração da seguinte forma:

```
public class ObjetosEmprestadosProvider extends ContentProvider {

        . . .

}
```

O segundo passo é definir a assinatura dos métodos que devem ser implementados obrigatoriamente. Vamos então definir a assinatura destes métodos, conforme segue:

```
. . .

@Override
public boolean onCreate() {
        . . .
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
        . . .
}

@Override
public Uri insert(Uri uri, ContentValues values) {
        . . .
}

@Override
public int update(Uri uri, ContentValues values, String selection,
                String[] selectionArgs) {
        . . .
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
        . . .
}

@Override
public String getType(Uri uri) {
        . . .
}
. . .
```

Como iremos trabalhar diretamente com o banco de dados, para ter acesso aos dados armazenados pelo aplicativo, iremos usar a classe ***DBHelper*** que criamos para nos auxiliar nesta tarefa. Criaremos, então, uma variável de instância na classe ***ObjetosEmprestadosProvider*** para referenciar um objeto do tipo *DBHelper*, conforme segue:

```
. . .
private DBHelper dbHelper;
. . .
```

No método ***onCreate()***, que será chamado no momento em que nosso Content Provider for iniciado, iremos apenas inicializar o nosso *DBHelper*, deixando sua implementação da seguinte forma:

```
. . .
@Override
public boolean onCreate() {
        dbHelper = new DBHelper(getContext());
        return true;
}
. . .
```

O próximo passo é definir o URI para o nosso Content Provider. Para deixar nosso código mais flexível, vamos quebrar este passo em 3 etapas:

1. Através de uma constante, em nossa classe *ObjetosEmprestadosProvider*, vamos definir a Authority que iremos usar em nosso URI. Definiremos, então, a seguinte constante em nossa classe:

```
. . .
// Authority que iremos usar no URI do nosso Content Provider
private static final String AUTHORITY = "br.com.hachitecnologia.devolvame.provider";
. . .
```

Nota

Lembre-se que, por convenção, devemos usar a mesma convenção de nome de pacotes do Java para a *Authority*.

2. Usando outra constante, vamos definir o Path que iremos usar em nosso URI. Definiremos, então, a seguinte constante em nossa classe:

```
. . .
// Path padrão que iremos usar no URI do nosso Content Provider
public static final String PATH = "objeto_emprestado";
. . .
```

Nota

Lembre-se que, por convenção, é importante que o *Path* tenha o mesmo nome da tabela que será acessada.

3. Com o Authority e o Path definidos, aproveitaremos suas constantes para criar nosso *URI*. Definiremos, então, a seguinte constante em nossa classe para definir nosso URI:

```
. . .
// URI
public static final Uri URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
. . .
```

Dica

É importante usar o modificador de acesso ***public*** para a constante que define o URI para que outras classes do nosso projeto tenham acesso a esta constante.

Definido nosso URI padrão, precisamos agora definir quais os URIs que iremos aceitar em nosso Content Provider para a consulta de dados. Como iremos permitir tanto a busca pelo ID como uma busca mais abrangente (ou seja, em cima de todos os dados) iremos aceitar dois tipos de URIs. Para facilitar nosso trabalho, vamos definir duas constantes em nossa classe que irá identificar cada um desses dois tipos de URIs que iremos aceitar para realizar a busca de dados. Criaremos, então, as seguintes constantes em nossa classe *ObjetosEmprestadosProvider*:

```
. . .
// Identificador de busca para todos os registros
private static final int BUSCA_TODOS = 1;

// Identificador de busca para um registro de ID específico
private static final int BUSCA_POR_ID = 2;
. . .
```

Agora que definimos os identificadores para os dois modos de consulta que aceitaremos em nosso Content Provider, devemos definir o padrão de URI para estas duas modalidades de busca. Para isto utilizamos a classe *UriMatcher* do Android.

Nota

A classe ***UriMatcher*** do Android é uma classe utilitária que facilita a definição e identificação de URIs que serão aceitas por um Content Provider.

Para facilitar a definição dos dois tipos de URIs que iremos aceitar para realizar a busca de dados em nosso Content Provider, vamos quebrar esta tarefa em duas etapas:

1. Iremos declarar um UriMatcher para nos auxiliar nesta tarefa, criando a seguinte váriável de instância em nossa classe *ObjetosEmprestadosProvider*:

```
/**
 * Declaramos o UriMatcher, que será usado
 * para definir as URIs que iremos aceitar
 * em nosso Content Provider.
 */
private static UriMatcher uriMatcher;
```

2. Criaremos um bloco estático em nossa classe para adicionar os dois possíveis URIs que iremos aceitar para a busca:

```
/**
 * Definimos os possíveis URIs que serão aceitos para realizar uma busca através do nosso Content Provider.
 */
static {
        // Inicializamos nosso UriMatcher
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        // Definimos o URI para a busca abrangente
        uriMatcher.addURI(AUTHORITY, PATH, BUSCA_TODOS);
        // Definimos o URI para a busca por ID
        uriMatcher.addURI(AUTHORITY, PATH + "/#", BUSCA_POR_ID);
}
```

Nota

O método ***addURI()*** da classe *UriMatcher* adiciona um padrão de URI que iremos utilizar para identificar os possíveis URIs que nosso Content Provider poderá receber. No código acima, a seguinte linha:

```
uriMatcher.addURI(AUTHORITY, PATH, BUSCA_TODOS);
```

adiciona o URI **content://br.com.hachitecnologia.devolvame.provider/objeto_emprestado** como um dos URIs que iremos aceitar em nosso Content Provider e o identifica com o nosso identificador **BUSCA_TODOS** que definimos em uma constante da classe. Isto quer dizer que, quando o nosso Content Provider receber este URI, devemos realizar a busca mais abrangente em nosso banco de dados.

Já a linha:

```
uriMatcher.addURI(AUTHORITY, PATH + "/#", BUSCA_POR_ID);
```

adiciona o URI **content://br.com.hachitecnologia.devolvame.provider/objeto_emprestado/#** como o URI que será usado para realizar a busca por ID em nosso banco de dados.

Dica

O caractere **#** é um curinga usado em um URI e será substituído por um valor especificado no URI passado para o Content Provider. Isso irá permitir ao nosso Content Provider receber, por exemplo, o seguinte URI:

content://br.com.hachitecnologia.devolvame.provider/objeto_emprestado/10

onde o valor “10”, em nosso caso, será o ID do registro a ser consultado em nosso banco de dados ao realizar a busca de um registro pelo seu ID.

Definidos os padrões de URIs que iremos aceitar para a busca de dados realizada através de nosso Content Provider, podemos agora implementar o método ***query()***, que irá realizar esta tarefa de busca. Implementaremos, então, o seguinte código no método ***query()*** do nosso Content Provider:

```
. . .
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
        // Inicia a conexão com o banco de dados, em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        /**
         * Identifica a URI recebida e executa a ação solicitada,
         * de acordo com os padrões de URI que definimos anteriormente.
         */
        switch (uriMatcher.match(uri)) {
                // Realiza a busca por ID
                case BUSCA_POR_ID:
                        // Recupera o ID enviado na URI
                        String id = uri.getPathSegments().get(1);
                        return db.query(PATH, projection, "_id = ?",
                                        new String[] {id}, null, null, sortOrder);
                /**
                 * Realiza uma busca geral, abrangendo todos os
                 * registros (de acordo com os parâmetros recebidos)
                 */
                case BUSCA_TODOS:
                        return db.query(PATH, projection, selection,
                                        selectionArgs, null, null, sortOrder);
                // Lança uma Exception, caso tenha recebido um URI não esperado
                default:
                        throw new IllegalArgumentException("URI inválido!");
        }
```

```
}
. . .
```

Nota

O método ***getPathSegments()*** da classe Uri do Android, usado no trecho de código acima, serve para recuperar um determinado trecho presente no Path do URI transmitido ao Content Provider. Caso queira recuperar a String passada no primeiro segmento do Path, por exemplo, basta passar o valor 0 (zero) para este método.

Em nosso caso, se nosso Content Provider recebesse, por exemplo, o seguinte URI:

content://br.com.hachitecnologia.devolvame.provider/objeto_emprestado/10

poderíamos recuperar o ID "10" passando o valor 1 como parâmetro para o método ***getPathSegments()***, por estar na segunda posição do Path, ficando sua chamada da seguinte forma:

```
uri.getPathSegments().get(1);
```

A chamada acima retornará uma String com o valor usado na segunda posição do Path do URI e poderá ser atribuída a uma variável do tipo String para ser usada em algum momento, no código da classe.

O método ***query()*** do Content Provider retorna um objeto do tipo Cursor, possibilitando que a classe que realizou a busca navegue sobre os dados encontrados.

O próximo passo é implementar o método ***insert()*** que será utilizado para inserir dados através do Content Provider. Nosso método conterá a seguinte implementação:

```
. . .
@Override
public Uri insert(Uri uri, ContentValues values) {
        // Inicia a conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        long id = db.insert(PATH, null, values);

        /**
         *  Retorna ao solicitante a URI recebida junto
         *  com o ID do registro inserido.
         */
        return ContentUris.withAppendedId(URI, id);
}
. . .
```

Nota

O método ***withAppendedId()*** da classe *ContentUris* do Android adiciona um determinado ID ao URI passado como parâmetro. Em nosso caso, adicionamos o ID do registro inserido no banco de dados ao URI recebido pelo método ***insert()*** do nosso Content Provider, para que a classe chamadora possa utilizar este ID, caso necessário.

O método ***insert()*** do Content Provider retorna um objeto do tipo Uri, nos possibilitando retornar para a classe chamadora o URI que dará acesso ao registro que foi inserido no banco de dados.

Por último, devemos implementar o método ***getType()***, que serve para retornar o tipo de dado retornado pelo Content Provider. O retorno deste método é utilizado pelo Android para detectar o tipo de dado: se é um registro único ou um registro com várias linhas, se trata de uma imagem, etc. Este método deve obrigatoriamente retornar uma String no seguinte padrão:

1. **vnd.android.cursor.dir/vnd.um_nome_qualquer**

    Onde:

um_nome_qualquer = um nome de sua escolha. Como exemplo, podemos usar o nome do pacote ou o URI padrão.

Este padrão acima é usado para dizer ao Android que estamos retornando um cursor com vários registros, identificado pela palavra "**dir**" na String acima.

2. **vnd.android.cursor.item/vnd.um_nome_qualquer**

Este padrão é usado para dizer ao Android que estamos retornando um cursor com apenas um registro, identificado pela palavra "**item**" na String acima.

Seguindo esta definição, iremos implementar nosso método ***getType()*** da seguinte forma:

```
. . .
@Override
public String getType(Uri uri) {
        /**
         * Identifica o tipo do dado que estamos retornando
         */
        switch (uriMatcher.match(uri)) {
                // Tipo de retorno da busca geral
                case BUSCA_TODOS:
                        return "vnd.android.cursor.dir/vnd." +
                                "br.com.hachitecnologia.devolvame.provider/objeto_emprestado";
                // Tipo de retorno da busca por ID
                case BUSCA_POR_ID:
                        return "vnd.android.cursor.item/vnd." +
                                "br.com.hachitecnologia.devolvame.provider/objeto_emprestado";
                // Lança uma Exception caso receba um URI não esperado
                default:
                        throw new IllegalArgumentException("URI inválido");
        }
}
. . .
```

Em nosso Content Provider, como não iremos disponibilizar a atualização e nem a remoção de dados, podemos deixar os métodos ***update()*** e ***delete()*** sem implementação, deixando-os da seguinte forma:

```
. . .
@Override
public int update(Uri uri, ContentValues values, String selection,
                String[] selectionArgs) {
        // TODO A implementar, caso necessário
        return 0;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
        // TODO A implementar, caso necessário
        return 0;
}
. . .
```

Finalizada a implementação do nosso Content Provider, precisamos agora configurá-lo no arquivo ***AndroidManifest.xml*** para que o Android possa registrá-lo e disponibilizá-lo dentro da plataforma, permitindo que outros aplicativos possam usá-lo. Para isto, basta inserir as seguintes linhas no arquivo *AndroidManifest.xml*:

```
. . .
<provider
    android:name=".provider.ObjetosEmprestadosProvider"
```

```
        android:authorities="br.com.hachitecnologia.devolvame.provider" >
    </provider>
    . . .
```

Nota

A tag **<provider>** deve ser colocada entre as tags **<application>** e **</application>** do arquivo *AndroidManifest.xml* com os seguintes atributos:

android:name: identifica a classe do projeto que será o Content Provider;

android:authorities: define o Authority que o Content Provider responderá.

Com o Content Provider implementado e devidamente declarado no arquivo *AndroidManifest.xml*, nosso aplicativo já poderá compartilhar informações com outros aplicativos instalados no dispositivo.

10.7 Exercício

Agora que você aprendeu a trabalhar com Content Providers, é hora de colocar em prática.

Neste exercício, iremos modificar o nosso projeto ***Devolva.me*** para que ele consulte um contato, através do Content Provider do aplicativo de Contatos nativo do Android, e associe este contato a um objeto emprestado.

1. Altere o código da classe modelo ***Contato***, deixando-a com a seguinte implementação:

```
public class Contato implements Serializable {

    private static final long serialVersionUID = 1L;

    private Integer id;
    private String nome;
    private String telefone;

    // getters e setters
    . . .
}
```

2. Crie uma nova classe, chamada ***Contatos***, no pacote ***br.com.hachitecnologia.devolvame.util***, com a seguinte implementação:

```
public class Contatos {

    /**
     * Consulta um contato através de seu ID.
     * @param id
     * @param context
     * @return
     */
    public static Contato getContato(int id, Context context) {

        ContentResolver cr = context.getContentResolver();
        /**
         * Consultamos no Content Provider do aplicativo de Contatos do
         * Android o contato que tenha o ID recebido como parâmetro.
         */
        Cursor cursor = cr.query(Phone.CONTENT_URI, null, Phone.CONTACT_ID + " = ?",
                new String[]{String.valueOf(id)}, null);

        Contato contato = null;

        // Iteramos sobre o Cursor para obter os dados desejados
```

```
                    if (cursor.moveToFirst()) {
                        contato = new Contato();
                        // Obtém o ID do contato
                        contato.setId(id);
                        // Obtém o Nome do contato
                        contato.setNome(cursor.getString(cursor.getColumnIndex(Phone.DISPLAY_NAME)));
                        // Obtém o Telefone do contato
                        contato.setTelefone(cursor.getString(cursor.getColumnIndex(Phone.NUMBER)));
                    }

                    // Fechamos o Cursor
                    cursor.close();

                    return contato;
                }
            }
```

3. Adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

4. Na classe ***DBHelper***, para que a estrutura do banco de dados seja atualizada, incremente a variável que define a versão do banco de dados em uma unidade, deixando-a da seguinte forma:

```
. . .
// Versão atual do banco de dados
private static final int VERSAO_DO_BANCO = 2;
. . .
```

5. Ainda na classe ***DBHelper***, altere o método ***onCreate()*** definindo a nova estrutura do banco de dados que irá armazenar o ID do contato ao invés de guardar o nome e número de telefone da pessoa que pegou um objeto emprestado, deixando sua implementação da seguinte forma:

```
. . .
@Override
public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                    + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                    + ",objeto TEXT NOT NULL"
                    + ",contato_id INTEGER NOT NULL"
                    + ",data_emprestimo INTEGER NOT NULL"
                    + ");";
        db.execSQL(sql);
}
. . .
```

6. Modifique o método ***adiciona()*** da classe ***ObjetoEmprestadoDAO*** removendo a referência dos campos que foram removidos e adicionando uma referência para o novo campo da tabela (*contato_id*), para que as informações sejam persistidas corretamente. Deixe este método com a seguinte implementação:

```
. . .
public void adiciona(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // persistidos no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("data_emprestimo", System.currentTimeMillis());
        values.put("contato_id", objeto.getContato().getId());

        // Instancia uma conexão com o banco de dados, em modo de gravação
```

```
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Insere o registro no banco de dados
        long id = db.insert("objeto_emprestado", null, values);
        objeto.setId(id);

        // Encerra a conexão com o banco de dados
        db.close();
    }
    . . .
```

7. Na classe ***ObjetoEmprestadoDAO***, adicione a seguinte variável de instância:

```
private Context context;
```

8. Ao final do construtor da classe ***ObjetoEmprestadoDAO***, adicione a seguinte linha de código:

```
this.context = context;
```

9. Altere o método ***listaTodos()*** do DAO, removendo os campos que não serão mais utilizados e adicionando o campo "**contato_id**", deixando-o com a seguinte implementação:

```
. . .
public List<ObjetoEmprestado> listaTodos() {

        // Cria um List guardar os objetos consultados no banco de dados
        List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

        // Instancia uma nova conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        // Executa a consulta no banco de dados
        Cursor c = db.query("objeto_emprestado", null, null, null, null, null,
                        "objeto ASC");

        /**
         * Percorre o Cursor, injetando os dados consultados em um objeto do
         * tipo ObjetoEmprestado e adicionando-os na List
         */
        try {
                while (c.moveToNext()) {
                        ObjetoEmprestado objeto = new ObjetoEmprestado();
                        objeto.setId(c.getLong(c.getColumnIndex("_id")));
                        objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));

                        int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                        Contato contato = Contatos.getContato(contatoID, context);
                        objeto.setContato(contato);

                        objetos.add(objeto);
                }

        } finally {
                // Encerra o Cursor
                c.close();
        }

        // Encerra a conexão com o banco de dados
        db.close();

        // Retorna uma lista com os objetos consultados
        return objetos;
```

```
    }
    . . .
```

10. Altere também o método ***atualiza()*** do DAO, deixando-o com a seguinte implementação:

```
    . . .
    public void atualiza(ObjetoEmprestado objeto) {
            // Encapsula no objeto do tipo ContentValues os valores a serem
            // atualizados no banco de dados
            ContentValues values = new ContentValues();
            values.put("objeto", objeto.getObjeto());
            values.put("contato_id", objeto.getContato().getId());

            // Instancia uma conexão com o banco de dados, em modo de gravação
            SQLiteDatabase db = dbHelper.getWritableDatabase();

            // Atualiza o registro no banco de dados
            db.update("objeto_emprestado", values, "_id=?", new String[] { objeto
                    .getId().toString() });

            // Encerra a conexão com o banco de dados
            db.close();
    }
    . . .
```

11. Edite o arquivo de Layout ***activity_cadastra_objeto_emprestado.xml***, substituindo o trecho de código abaixo:

```
    . . .
    <EditText
        android:id="@+id/cadastro_objeto_campo_pessoa"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Informe o nome da pessoa"
        android:inputType="textPersonName" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="15.0dip"
        android:text="Telefone:" />

    <EditText
        android:id="@+id/cadastro_objeto_campo_telefone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Informe o telefone da pessoa"
        android:inputType="phone" />
    . . .
```

pelo seguinte trecho de código:

```
    . . .
    <Button
        android:id="@+id/botao_selecionar_contato"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Selecionar na Lista de Contatos" />

    <TextView
        android:id="@+id/cadastro_objeto_campo_pessoa"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="15.0sp"
        android:visibility="gone" />
    . . .
```

12. Altere o código da Activity de cadastro ***CadastraObjetoEmprestadoActivity***, adicionando as variáveis que farão referência às novas Views adicionadas ao arquivo de Layout. Para isto, substitua o trecho de código abaixo:

```
. . .
private EditText campoNomePessoa;
private EditText campoTelefone;
. . .
```

pelo seguinte trecho de código:

```
. . .
private TextView campoNomePessoa;
private Button botaoSelecionarContato;
. . .
```

13. Ainda na Activity de cadastro ***CadastraObjetoEmprestadoActivity***, no método ***onCreate()***, corrija as referências às Views do arquivo de Layout, substituindo o trecho de código abaixo:

```
. . .
campoNomePessoa = (EditText) findViewById(R.id.cadastro_objeto_campo_pessoa);
campoTelefone = (EditText) findViewById(R.id.cadastro_objeto_campo_telefone);
. . .
```

pelo seguinte trecho de código:

```
. . .
campoNomePessoa = (TextView) findViewById(R.id.cadastro_objeto_campo_pessoa);
botaoSelecionarContato = (Button) findViewById(R.id.botao_selecionar_contato);
. . .
```

14. Novamente na Activity de cadastro ***CadastraObjetoEmprestadoActivity***, adicione a seguinte constante nesta classe para identificar o retorno para a Action de seleção de contato:

```
. . .
private static final int ID_RETORNO_SELECIONA_CONTATO = 1234;
. . .
```

15. Na Activity de cadastro ***CadastraObjetoEmprestadoActivity***, no método ***onCreate()***, defina a ação para o botão "**Selecionar contato na lista**", adicionando o seguinte trecho de código ao final deste método:

```
. . .
/**
 * O botão "Selecionar contato na lista" irá abrir a Activity de
 * Contatos do Android para que o usuário possa selecionar um
 * contato na lista.
 */
botaoSelecionarContato.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
                    /**
                     * Na Intent, definimos a Action Intent.ACTION_PICK
                     * do Android, usada para retornar um determinado item
                     * selecionado em uma lista.
                     */
                    Intent i = new Intent(Intent.ACTION_PICK);
                    /**
                     * Passamos a lista de contatos, do aplicativo de Contatos
                     * nativo do Android, para que o usuário possa selecionar
                     * um contato na lista.
                     */
                    i.setData(Contacts.CONTENT_URI);
                    // Abre a lista com os contatos para seleção
                    startActivityForResult(i, ID_RETORNO_SELECIONA_CONTATO);
                }
        });
        . . .
```

16. Na Activity de cadastro ***CadastraObjetoEmprestadoActivity***, no método ***onCreate()***, permita que o usuário edite/altere um contato selecionado anteriormente. Para isto, devemos adicionar o seguinte trecho de código ao final do método ***onCreate()*** da Activity ***CadastraObjetoEmprestadoActivity***:

```
. . .
        /**
         * Ao clicar sobre o nome do Contato selecionado, o usuário
         * será direcionado para a Activity de Contatos do Android,
         * permitindo-o selecionar um outro contato.
         */
        campoNomePessoa.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    /**
                     * Na Intent, definimos a Action Intent.ACTION_PICK
                     * do Android, usada para retornar um determinado item
                     * selecionado em uma lista.
                     */
                    Intent i = new Intent(Intent.ACTION_PICK);
                    /**
                     * Passamos a lista de contatos, do aplicativo de Contatos
                     * nativo do Android, para que o usuário possa selecionar
                     * um contato na lista.
                     */
                    i.setData(Contacts.CONTENT_URI);
                    // Abre a lista com os contatos para seleção
                    startActivityForResult(i, ID_RETORNO_SELECIONA_CONTATO);
                }
        });
        . . .
```

17. Ainda na Activity de cadastro, Implemente o método ***onActivityResult()*** para tratar o retorno vindo da Action de seleção de contato disparada pelo método ***startActivityForResult()***. Para isto, adicione o seguinte método à classe ***CadastraObjetoEmprestadoActivity***:

```
        . . .
        /**
         * Trata o resultado vindo de uma Action chamada através
         * do método startActivityForResult().
         */
        @Override
        public void onActivityResult(int requestCode, int resultCode, Intent data) {
                super.onActivityResult(requestCode, resultCode, data);
```

```
		switch (requestCode) {

		/**
		 * Trata o retorno vindo da tela de seleção de contatos.
		 */
		case (ID_RETORNO_SELECIONA_CONTATO):
			if (resultCode == Activity.RESULT_OK) {
				/**
				 *  Após selecionado o contato pelo usuário, recebemos uma
				 *  URI que irá apontar para o contato selecionado.
				 */
				Uri contactData = data.getData();
				/**
				 *  Usaremos um ContentResolver para consultar os dados
				 *  do contato selecionado no ContentProvider do aplicativo
				 *  de Contatos do Android.
				 */
				ContentResolver contentResolver = getContentResolver();
				/**
				 * Executamos a query e atribuímos o resultado em um Cursor
				 * para navegarmos sobre os dados retornados pelo ContentProvider.
				 * Na query passamos apenas a URI, sem definir nenhum parâmetro adicional,
				 * já que a URI retornada pela Action aponta diretamente para o contato
				 * selecionado.
				 */
				Cursor cursor = contentResolver.query(contactData, null, null, null, null);
				// Iteramos sobre o Cursor para obter os dados desejados
				if (cursor.moveToFirst()) {
					// Obtém o ID do contato
					objetoEmprestado.getContato().setId(cursor.getInt(cursor
							.getColumnIndex(Phone._ID)));
					// Obtém o Nome do contato
					objetoEmprestado.getContato().setNome(cursor.getString(cursor
							.getColumnIndex(Phone.DISPLAY_NAME)));

					/**
					 * Após selecionado o contato, não há mais necessidade de
					 * mostrar o botão "Selecionar contato na lista". Portanto,
					 * iremos esconder o botão definindo sua visibilidade para GONE.
					 */
					botaoSelecionarContato.setVisibility(View.GONE);

					/**
					 * Alteramos a TextView "cadastro_objeto_campo_pessoa" definindo
					 * em seu texto o nome do contato selecionado e definimos a
					 * visibilidade desta View para VISIBLE, para que ela fique
					 * visível para o usuário.
					 */
					campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
					campoNomePessoa.setVisibility(View.VISIBLE);
				}

				// Fechamos o Cursor
				cursor.close();
			}
			break;
		}
	}
	. . .
```

18. Na classe ***CadastraObjetoEmprestadoActivity***, na ação do botão “Salvar”, remova as seguintes linhas de código:

```
	. . .
```

```
objetoEmprestado.getContato().setNome(campoNomePessoa.getText().toString());
objetoEmprestado.getContato().setTelefone(campoTelefone.getText().toString());
. . .
```

19. Corrija a ação de edição de dados para que ela siga a nova implementação. Para isto, altere o método ***onCreate()***, da classe ***CadastraObjetoEmprestadoActivity***, substituindo o trecho de código abaixo:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoTelefone.setText(objetoEmprestado.getContato().getTelefone());
}
. . .
```

pelo seguinte trecho de código:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());

        // Remove da tela o botão "Selecionar contato na lista"
        botaoSelecionarContato.setVisibility(View.GONE);

        /**
         * Alteramos a TextView "cadastro_objeto_campo_pessoa" definindo
         * em seu texto o nome do contato que consultamos no Content Provider
         * e definimos a visibilidade desta View para VISIBLE, para que ela
         * fique visível para o usuário.
         */
        campoNomePessoa.setText(objetoEmprestado.getContato().getNome());
        campoNomePessoa.setVisibility(View.VISIBLE);
}
. . .
```

20. Execute o projeto no emulador do Android e cadastre um novo objeto emprestado para verificar as novas mudanças realizadas no aplicativo.

10.8 Exercício opcional

Neste exercício iremos criar um Content Provider em nosso projeto *Devolva.me* para que outros aplicativos possam consultar os dados armazenados em nosso banco de dados e inserir de dados.

1. No projeto *Devolva.me*, crie uma nova classe no pacote ***br.com.hachitecnologia.devolvame.provider***, chamada ***ObjetosEmprestadosProvider***, com a seguinte implementação:

```
public class ObjetosEmprestadosProvider extends ContentProvider {

        private DBHelper dbHelper;

        // Authority que iremos usar no URI do nosso Content Provider
        private static final String AUTHORITY = "br.com.hachitecnologia.devolvame.provider";
        // Path padrão que iremos usar no URI do nosso Content Provider
```

```
public static final String PATH = "objeto_emprestado";
// URI
public static final Uri URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);

// Identificador de busca por todos os registros
private static final int BUSCA_TODOS = 1;
// Identificador de busca por registro de ID específico
private static final int BUSCA_POR_ID = 2;

/**
 * Declaramos o UriMatcher, que será usado
 * para definir as URIs que iremos aceitar
 * em nosso Content Provider.
 */
private static UriMatcher uriMatcher;

/**
 * Definimos os possíveis URIs que serão aceitos para
 * realizar uma busca através do nosso Content Provider.
 */
static {
        // Inicializamos nosso UriMatcher
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        // Definimos o URI para a busca abrangente
        uriMatcher.addURI(AUTHORITY, PATH, BUSCA_TODOS);
        // Definimos o URI para a busca por ID
        uriMatcher.addURI(AUTHORITY, PATH + "/#", BUSCA_POR_ID);
}

@Override
public boolean onCreate() {
        dbHelper = new DBHelper(getContext());
        return true;
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
        // Iniciamos a conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        /**
         * Identifica a URI recebida e executa a ação solicitada.
         */
        switch (uriMatcher.match(uri)) {
                // Realiza a busca por ID
                case BUSCA_POR_ID:
                        // Recupera o ID enviado na URI
                        String id = uri.getPathSegments().get(1);

                        return db.query(PATH, projection, "_id = ?",
                                        new String[] {id}, null, null, sortOrder);
                /**
                 * Realiza uma busca geral, abrangendo todos os
                 * registros (de acordo com os parâmetros recebidos)
                 */
                case BUSCA_TODOS:
                        return db.query(PATH, projection, selection,
                                        selectionArgs, null, null, sortOrder);
                // Lança uma Exception caso tenha recebido uma URI não esperada
                default:
                        throw new IllegalArgumentException("URI inválido!");
        }
}
```

```
	@Override
	public Uri insert(Uri uri, ContentValues values) {
		// Inicia a conexão com o banco de dados, em modo de gravação
		SQLiteDatabase db = dbHelper.getWritableDatabase();
		long id = db.insert(PATH, null, values);

		/**
		 *  Retorna ao solicitante a URI recebida junto
		 *  com o ID do registro inserido.
		 */
		return ContentUris.withAppendedId(URI, id);
	}

	@Override
	public int update(Uri uri, ContentValues values, String selection,
			String[] selectionArgs) {
		// TODO A immplementar, caso necessário
		return 0;
	}

	@Override
	public int delete(Uri uri, String selection, String[] selectionArgs) {
		// TODO A implementar, caso necessário
		return 0;
	}

	@Override
	public String getType(Uri uri) {
		/**
		 * Identifica o tipo do dado que estamos retornando
		 */
		switch (uriMatcher.match(uri)) {
			// Tipo de retorno da busca geral
			case BUSCA_TODOS:
				return "vnd.android.cursor.dir/vnd." +
						"br.com.hachitecnologia.devolvame.provider/
objeto_emprestado";
			// Tipo de retorno da busca por ID
			case BUSCA_POR_ID:
				return "vnd.android.cursor.item/vnd." +
						"br.com.hachitecnologia.devolvame.provider/
objeto_emprestado";
			// Lança uma Exception caso receba um URI não esperado
			default:
				throw new IllegalArgumentException("URI inválido");
		}
	}
}
```

11 - Interagindo com a câmera do dispositivo

O Android nos permite interagir com vários recursos de hardware do dispositivo, como a câmera, o GPS, o bluetooth, etc. Isto nos permite desenvolver aplicativos muito poderosos e com diversos recursos. Neste capítulo vamos aprender a integrar um aplicativo com a câmera do dispositivo móvel.

11.1 Utilizando o aplicativo nativo de fotos para capturar uma imagem

Muitas vezes, em nossos aplicativos, precisamos obter uma imagem através da câmera do dispositivo. Realizar esta tarefa é simples, já que o Android nos permite utilizar o próprio aplicativo nativo de fotos para obter uma imagem através da câmera. Desta forma não precisamos desenvolver um aplicativo que tire fotos, bastando pedir para o aplicativo nativo tirar a foto e nos entregar a imagem capturada.

A chamada do aplicativo nativo de fotos do Android é feita através da Action ***android.media.action.IMAGE_CAPTURE***, invocada por uma Intent. Como resultado, o aplicativo de fotos nos devolve um objeto do tipo ***Bitmap*** com a imagem capturada pela câmera do dispositivo. Este objeto pode ser apresentado em um componente *ImageView* na tela ou até mesmo armazenado em um diretório do dispositivo ou em um banco de dados.

Veja, no trecho de código abaixo, o exemplo de uma chamada ao aplicativo de fotos do Android para obter uma imagem através da câmera do dispositivo:

```
. . .
Intent i = new Intent("android.media.action.IMAGE_CAPTURE");
startActivityForResult(i, 5678);
. . .
```

Nota

A Intent que define a action ***android.media.action.IMAGE_CAPTURE***, usada para chamar o aplicativo de fotos do Android, deve ser invocada usando o método ***startActivityForResult()***, já que esperamos um retorno desta Activity que, em nosso caso, retornará um objeto *Bitmap* com a imagem capturada pela câmera.

Além da chamada da Action, devemos implementar o método ***onActivityResult()*** para implementar o código que irá receber a imagem capturada pela câmera do dispositivo. O trecho de código abaixo mostra o exemplo de como implementar este método para obter a imagem capturada pela câmera:

```
. . .
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        switch (requestCode) {
        case 5678:
                if (data != null) {
                        Bundle bundle = data.getExtras();
                        if (bundle != null) {
                                Bitmap foto = (Bitmap) bundle.get("data");
                                . . .
                        }
                }
        }
}
. . .
```

11.2 Colocando em prática: usando a câmera no projeto *Devolva.me*

Agora que aprendemos a usar a câmera do dispositivo, vamos usar este recurso para melhorar o nosso aplicativo *Devolva.me*.

Nosso cliente José, mesmo satisfeito com o resultado do aplicativo *Devolva.me*, faz uma nova solicitação para que seja implementado um novo recurso no aplicativo. Desta vez, José solicita um recurso para possibilitá-lo a tirar uma foto do objeto emprestado a partir da tela de cadastro, para que ele possa armazenar uma foto do objeto, o que o ajudará a identificá-lo melhor.

De acordo com a nova solicitação, especificamos o seguinte requisito:

• Ao cadastrar um novo objeto emprestado, o sistema deverá possibilitar o usuário a tirar uma foto do objeto e esta foto deve ser armazenada pelo aplicativo.

Em um simples rascunho, junto ao nosso cliente, definimos como ficará a nossa tela de cadastro após a modificação solicitada, conforme ilustra a **Figura 11.1**.

Figura 11.1. Nova tela de cadastro do aplicativo *Devolva.me*, contemplando o recurso para tirar uma foto do objeto emprestado.

Alterando a tela de cadastro

A primeira alteração que devemos fazer em nosso aplicativo, para contemplar a nova implementação, é alterar a tela de cadastro de objetos. Nesta tela iremos adicionar um ícone com imagem de uma câmera que, ao ser clicado, abrirá o aplicativo de fotos do Android.

Adicionaremos, então, o seguinte trecho de código dentro, e no início, do *Element Root* (View principal) do arquivo de layout ***activity_cadastra_objeto_emprestado.xml***:

```
. . .
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center" >
```

```
    <ImageView
        android:id="@+id/foto_objeto"
        android:layout_width="86.0dip"
        android:layout_height="86.0dip"
        android:padding="0dip"
        android:src="@android:drawable/ic_menu_camera" />

</LinearLayout>
. . .
```

Nota

O Android possui diversos ícones e imagens disponíveis para utilizarmos em nosso aplicativo. Estes recursos podem ser acessados, em um arquivo de layout, através da referência: ***@android:drawable/NOME_DA_IMAGEM***.

Como exemplo, a referência ***@android:drawable/ic_menu_camera*** poderá ser usada para mostrar o ícone de uma câmera em um componente *ImageView*, em um arquivo de layout.

Alterando a classe ObjetoEmprestado

Como nossa intenção é persistir no banco de dados a foto capturada, devemos adicionar em nossa classe modelo um campo que irá guardar esta foto. A foto será armazenada em **array de bytes** no banco de dados, portanto usaremos um atributo do tipo ***byte[ ]*** em nossa classe modelo para armazená-la.

Sendo assim, em nossa classe ***ObjetoEmprestado***, adicionaremos o seguinte atributo:

```
. . .
private byte[ ] foto;
. . .
```

Criaremos também os *setters* e *getters* para este atributo:

```
. . .
public byte[ ] getFoto() {
        return foto;
}

public void setFoto(byte[ ] foto) {
        this.foto = foto;
}
. . .
```

Alterando a estrutura de dados

Para implementar a nova especificação, devemos alterar a estrutura de dados da tabela "**objeto_emprestado**" do banco de dados, adicionando 1 campo do tipo BLOB para armazenar a foto.

A **Figura 11.2** mostra o diagrama com a nova estrutura de dados da tabela ***objeto_emprestado***.

objeto_emprestado	
_id	INTEGER
objeto	TEXT
contato_id	INTEGER
data_emprestimo	INTEGER
foto	BLOB

Figura 11.2. Nova estrutura de dados da tabela *objeto_emprestado* do projeto *Devolva.me*.

Para implementar nossa nova estrutura de dados, devemos alterar o método ***onCreate()*** da classe ***DBHelper***, deixando-o com a seguinte implementação:

```
/**
 * Cria a tabela no banco de dados, caso ela nao exista.
 */
@Override
public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                        + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                        + ",objeto TEXT NOT NULL"
                        + ",contato_id INTEGER NOT NULL"
                        + ",data_emprestimo INTEGER NOT NULL"
                        + ",foto BLOB"
                        + ");";
        db.execSQL(sql);
}
```

Além dessa alteração, para que o Android atualize a nossa estrutura de dados, na classe ***DBHelper*** devemos incrementar o valor da variável ***VERSAO_DO_BANCO*** em 1 unidade, deixando-a com o seguinte valor:

```
. . .
private static final int VERSAO_DO_BANCO = 3;
. . .
```

Alterando o DAO

Devemos também alterar nosso DAO para que ele possa persistir no banco de dados a foto capturada e consultá-la quando necessário. Portanto, realizaremos as seguintes alterações na classe ***ObjetoEmprestadoDAO***:

1. A primeira coisa a ser feita é alterar o método ***adiciona()*** do DAO, adicionando o seguinte trecho:

```
. . .
values.put("foto", objeto.getFoto());
. . .
```

Nota

O trecho acima deverá ser colocado logo abaixo da seguinte linha:

```
values.put("contato_id", objeto.getContato().getId());
```

2. O mesmo deve ser feito no método ***atualiza()***. Portanto, adicione neste método o seguinte trecho:

```
. . .
values.put("foto", objeto.getFoto());
. . .
```

Nota

O trecho acima deverá ser colocado logo abaixo da seguinte linha:

```
values.put("contato_id", objeto.getContato().getId());
```

3. A última alteração a ser feita no DAO é no método ***listaTodos()***, adicionando em seu bloco *while* a seguinte linha:

```
. . .
objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));
. . .
```

Nota

O trecho acima deverá ser colocado logo abaixo da seguinte linha (dentro do bloco *while*):

```
objeto.setContato(contato);
```

Criando uma classe utilitária para converter um objeto *Bitmap* em *byte[ ]* e vice-versa

Como a foto capturada pela câmera do dispositivo, através do aplicativo de fotos, é retornada em um objeto do tipo *Bitmap*, devemos criar um método capaz de converter este objeto em um objeto do tipo *byte[ ]* (array de bytes) para podermos gravar a foto no banco de dados. Além disso devemos criar um método que faz o inverso também, já que ao consultar um objeto salvo no banco de dados precisaremos converter o *byte[ ]* em *Bitmap*, para apresentar a foto em um componente *ImageView* na tela do dispositivo. Portanto, criaremos uma classe utilitária, chamada ***Util***, dentro do pacote ***br.com.hachitecnologia.devolvame.util***, com a seguinte implementação:

```
public class Util {

	/**
	 * Converte um objeto do tipo Bitmap para byte[] (array de bytes)
	 * @param imagem
	 * @param qualidadeDaImagem
	 * @return
	 */
	public static byte[] converteBitmapPraByteArray(Bitmap imagem, int qualidadeDaImagem) {
		ByteArrayOutputStream stream = new ByteArrayOutputStream();
		imagem.compress(CompressFormat.PNG, qualidadeDaImagem, stream);
		return stream.toByteArray();
	}

	/**
	 * Converte um objeto do tipo byte[] (array de bytes) para Bitmap
	 * @param imagem
	 * @return
	 */
	public static Bitmap converteByteArrayPraBitmap(byte[] imagem) {
		return BitmapFactory.decodeByteArray(imagem, 0, imagem.length);
	}

}
```

Alterando a Activity de cadastro

Por fim, precisamos realizar as seguintes alterações na Activity ***CadastraObjetoEmprestadoActivity***:

1. Adicionar na classe um atributo que irá referenciar o componente ImageView da tela de cadastro. Adicionaremos, então, o seguinte atributo na classe ***CadastraObjetoEmprestadoActivity***:

```
. . .
private ImageView campoFotoObjeto;
. . .
```

2. No método ***onCreate()*** devemos ligar o atributo adicionado ao componente ImageView da tela de cadastro. Para isto, adicionaremos a seguinte linha no método ***onCreate()***:

```
. . .
campoFotoObjeto = (ImageView) findViewById(R.id.foto_objeto);
. . .
```

Nota

O trecho acima deverá ser colocado logo abaixo da seguinte linha (dentro do método *onCreate()*):

```
botaoSelecionarContato = (Button) findViewById(R.id.botao_selecionar_contato);
```

3. Adicionar uma constante que será usada como identificador, na chamada da Activity do aplicativo de foto, para identificar o retorno desta Activity. Adicionaremos, então, a seguinte constante à nossa Activity:

```
. . .
private static final int ID_RETORNO_TIRA_FOTO_OBJETO = 5678;
. . .
```

4. Implementar o evento que irá invocar o aplicativo de fotos do Android, quando o usuário clicar no ícone da câmera na tela de cadastro. Para isto, adicionaremos o seguinte trecho de código ao final do método ***onCreate()***:

```
. . .
campoFotoObjeto.setOnClickListener(new View.OnClickListener() {
	@Override
	public void onClick(View v) {
		Intent i = new Intent("android.media.action.IMAGE_CAPTURE");
		startActivityForResult(i, ID_RETORNO_TIRA_FOTO_OBJETO);
	}
});
. . .
```

5. Implementar, no método ***onActivityResult()***, o código que irá receber a imagem capturada pela câmera e mostrá-la no componente *ImageView* da tela de cadastro. Esta imagem deverá também ser injetada no atributo "**foto**" do objeto ***ObjetoEmprestado***. Para isto, adicionaremos o seguinte trecho de código ao método ***onActivityResult()***, dentro (e ao final) do bloco *switch*:

```
. . .
/**
 * Trata o retorno vindo da captura de imagem através da câmera.
 */
case ID_RETORNO_TIRA_FOTO_OBJETO:
	if (data != null) {
		// Recupera o objeto Bundle recebido do aplicativo de fotos.
		Bundle bundle = data.getExtras();
		if (bundle != null) {
			/*
			 *  Recupera o objeto Bitmap com a foto capturada, recebido através
			 *  do objeto Bundle.
			 */
			Bitmap foto = (Bitmap) bundle.get("data");
			/*
			 * Define a imagem no componente ImageView da tela, para ser
			 * apresentada ao usuário.
			 */
			campoFotoObjeto.setImageBitmap(foto);
			/*
			 * Converte o objeto Bitmap com a foto em um array de bytes (byte[])
			 * e o injeta no objeto ObjetoEmprestado para ser persistido no banco
			 * de dados.
			 */
			objetoEmprestado.setFoto(Util.converteBitmapPraByteArray(foto, 70));
		}
	}
	break;
. . .
```

6. No método ***onCreate()***, implementar o código que irá apresentar a foto, de um objeto salvo, no componente ImageView da tela de cadastro. Para isto, localize o seguinte trecho de código:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();
} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
. . .
```

e insira o código abaixo dentro do bloco *else*, abaixo da última linha do trecho acima:

```
. . .
if (objetoEmprestado.getFoto() != null)
        campoFotoObjeto.setImageBitmap(Util.converteByteArrayPraBitmap(objetoEmprestado.getFoto()));
. . .
```

Pronto! Nossas alterações estão finalizadas e nosso aplicativo *Devolva.me* está pronto para que nosso usuário possa tirar e armazenar as fotos de seus objetos emprestados, assim como solicitado pelo nosso cliente. A **Figura 11.3** ilustra a nossa nova tela de cadastro em execução em um dispositivo real com Android.

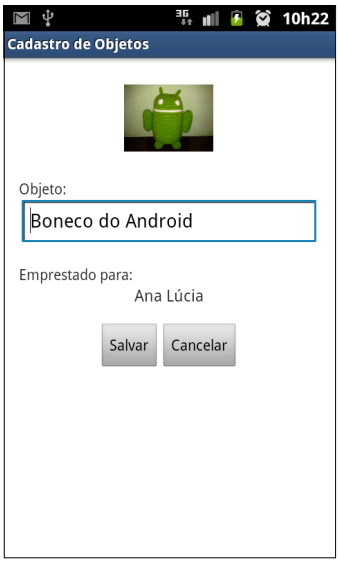


Figura 11.3. Nova tela de cadastro do aplicativo *Devolva.me* executando em um dispositivo com Android.

11.3 Exercício

Agora que você aprendeu a capturar fotos com a câmera do dispositivo, usando o aplicativo nativo de fotos, é hora de colocar em prática.

Neste exercício iremos implementar uma funcionalidade no aplicativo *Devolva.me* para capturar uma foto do objeto emprestado e armazená-la no banco de dados

1. Na tela de cadastro, crie um ícone com a imagem ***@android:drawable/ic_menu_camera***, definindo o ID ***foto_objeto***. Para isto, altere o arquivo ***activity_cadastra_objeto_emprestado.xml***, adicionando o seguinte trecho de código dentro, e no início, do *Element Root* (View principal):

```
. . .
<LinearLayout
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center" >

    <ImageView
        android:id="@+id/foto_objeto"
        android:layout_width="86.0dip"
        android:layout_height="86.0dip"
        android:padding="0dip"
        android:src="@android:drawable/ic_menu_camera" />

</LinearLayout>
. . .
```

2. Crie um atributo chamado ***foto***, do tipo ***byte[ ]***, na classe ***ObjetoEmprestado***. Para isto, adicione a seguinte linha de código na classe ***ObjetoEmprestado***:

```
. . .
private byte[ ] foto;
. . .
```

3. Implemente os seguintes *setters* e *getters* para o atributo ***foto***, na classe ***ObjetoEmprestado***:

```
. . .
public byte[ ] getFoto() {
        return foto;
}

public void setFoto(byte[ ] foto) {
        this.foto = foto;
}
. . .
```

4. Modifique o método ***onCreate()*** da classe ***DBHelper***, deixando-o com a seguinte implementação:

```
/**
 * Cria a tabela no banco de dados, caso ela nao exista.
 */
@Override
public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                        + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                        + ",objeto TEXT NOT NULL"
                        + ",contato_id INTEGER NOT NULL"
                        + ",data_emprestimo INTEGER NOT NULL"
                        + ",foto BLOB"
                        + ");";
        db.execSQL(sql);
}
```

5. Na classe ***DBHelper*** incremente o valor da variável ***VERSAO_DO_BANCO*** em 1 unidade, deixando-a com o seguinte valor:

```
. . .
private static final int VERSAO_DO_BANCO = 3;
. . .
```

6. No método ***adiciona()*** da classe ***ObjetoEmprestadoDAO***, adicione o seguinte trecho de código:

```
. . .
values.put("foto", objeto.getFoto());
. . .
```

abaixo da seguinte linha:

```
. . .
values.put("contato_id", objeto.getContato().getId());
. . .
```

7. No método ***atualiza()*** da classe ***ObjetoEmprestadoDAO***, adicione o seguinte trecho de código:

```
. . .
values.put("foto", objeto.getFoto());
. . .
```

abaixo da seguinte linha:

```
. . .
values.put("contato_id", objeto.getContato().getId());
. . .
```

8. No método ***listaTodos()*** da classe ***ObjetoEmprestadoDAO***, dentro do bloco *while*, adicione o seguinte trecho de código:

```
. . .
objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));
. . .
```

abaixo da seguinte linha:

```
. . .
objeto.setContato(contato);
. . .
```

9. No pacote ***br.com.hachitecnologia.devolvame.util*** crie uma nova classe, chamada ***Util***, com a seguinte implementação:

```
public class Util {

	/**
	 * Converte um objeto do tipo Bitmap para byte[] (array de bytes)
	 * @param imagem
	 * @param qualidadeDaImagem
	 * @return
	 */
	public static byte[] converteBitmapPraByteArray(Bitmap imagem, int qualidadeDaImagem) {
		ByteArrayOutputStream stream = new ByteArrayOutputStream();
		imagem.compress(CompressFormat.PNG, qualidadeDaImagem, stream);
		return stream.toByteArray();
	}

	/**
	 * Converte um objeto do tipo byte[] (array de bytes) para Bitmap
	 * @param imagem
	 * @return
	 */
	public static Bitmap converteByteArrayPraBitmap(byte[] imagem) {
		return BitmapFactory.decodeByteArray(imagem, 0, imagem.length);
	}

}
```

10. Na classe ***CadastraObjetoEmprestadoActivity***, adicione o seguinte atributo:

```
. . .
private ImageView campoFotoObjeto;
. . .
```

11. No método ***onCreate()*** da classe ***CadastraObjetoEmprestadoActivity***, adicione a seguinte linha de código:

```
. . .
campoFotoObjeto = (ImageView) findViewById(R.id.foto_objeto);
. . .
```

abaixo da seguinte linha:

```
. . .
botaoSelecionarContato = (Button) findViewById(R.id.botao_selecionar_contato);
. . .
```

12. Na classe ***CadastraObjetoEmprestadoActivity***, adicione a seguinte constante:

```
. . .
private static final int ID_RETORNO_TIRA_FOTO_OBJETO = 5678;
. . .
```

13. Ao final do método ***onCreate()*** da classe ***CadastraObjetoEmprestadoActivity***, adicione o seguinte trecho de código:

```
. . .
campoFotoObjeto.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                Intent i = new Intent("android.media.action.IMAGE_CAPTURE");
                startActivityForResult(i, ID_RETORNO_TIRA_FOTO_OBJETO);
        }
});
. . .
```

14. No método ***onActivityResult()*** da classe ***CadastraObjetoEmprestadoActivity***, dentro do bloco *switch*, implemente o seguinte *case*:

```
. . .
/**
 * Trata o retorno vindo da captura de imagem através da câmera.
 */
case ID_RETORNO_TIRA_FOTO_OBJETO:
        if (data != null) {
                // Recupera o objeto Bundle recebido do aplicativo de fotos.
                Bundle bundle = data.getExtras();
                if (bundle != null) {
                        /*
                         *  Recupera o objeto Bitmap com a foto capturada, recebido através
                         *  do objeto Bundle.
                         */
                        Bitmap foto = (Bitmap) bundle.get("data");
                        /*
                         * Define a imagem no componente ImageView da tela, para ser
                         * apresentada ao usuário.
                         */
                        campoFotoObjeto.setImageBitmap(foto);
                        /*
                         * Converte o objeto Bitmap com a foto em um array de bytes (byte[])
                         * e o injeta no objeto ObjetoEmprestado para ser persistido no banco
                         * de dados.
```

```
                                        */
                                        objetoEmprestado.setFoto(Util.converteBitmapPraByteArray(foto, 70));
                                }
                        }
                        break;
        . . .
```

15. No método ***onCreate()*** da classe ***CadastraObjetoEmprestadoActivity***, localize o seguinte trecho de código:

```
        . . .
        if (objetoEmprestado == null) {
                // Instancia um novo objeto do tipo ObjetoEmprestado
                objetoEmprestado = new ObjetoEmprestado();
        } else {
                campoObjeto.setText(objetoEmprestado.getObjeto());
        . . .
```

e insira o código abaixo dentro do bloco *else*, abaixo da última linha do trecho acima:

```
        . . .
        if (objetoEmprestado.getFoto() != null)
                campoFotoObjeto.setImageBitmap(Util.converteByteArrayPraBitmap(objetoEmprestado.getFoto()));
        . . .
```

16. Execute o projeto no emulador do Android e faça o teste, cadastrando um novo objeto com sua devida foto.

12 - Entendendo as threads no Android

Como vimos no início deste curso, o Android roda em Linux, se beneficiando dos recursos que este Sistema Operacional oferece. Um desses recursos que o Android aproveita no Linux é a capacidade de executar múltiplas tarefas simultaneamente.

Cada aplicativo no Android roda em um processo exclusivo e este processo pode conter várias *threads* em execução. A **Figura 12.1** mostra o funcionamento das *threads* no Android.

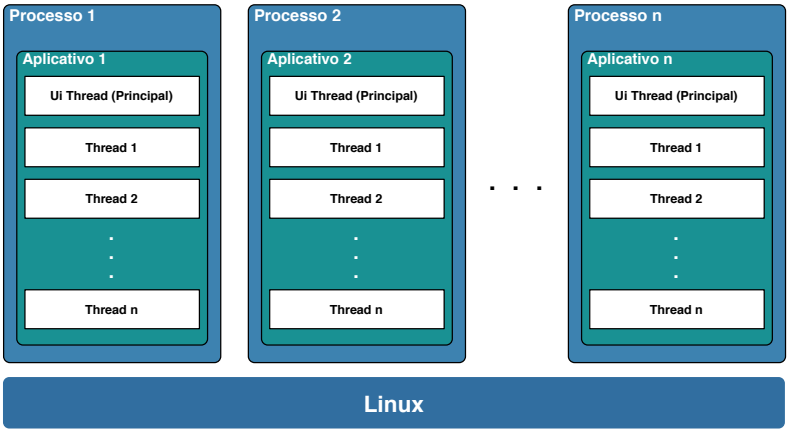


Figura 12.1. Esquema de funcionamento das threads no Android.

12.1 UI Thread (a Thread principal)

Sempre que um aplicativo é iniciado no Android, um novo processo é criado exclusivamente para este aplicativo e, neste processo, uma nova *thread* é iniciada, a thread principal. Esta thread principal é chamada de ***UI Thread***, gerenciada pelo próprio Android.

A UI Thread é responsável por controlar os componentes do aplicativo (activities, services e broadcast receivers), desenhar a tela e tratar os eventos do aplicativo, ou seja, esta é a thread responsável por toda a execução de código de um aplicativo no Android.

Por ser a thread principal, a UI Thread merece uma atenção especial, pois qualquer código implementado nela que demore um certo tempo para ser executado pode deixar todo o aplicativo lento, prejudicando a sua usabilidade.

Imagine, por exemplo, que você tenha desenvolvido um simples programa para ser instalado em um Tablet, que ficará na porta de saída de uma loja, onde o cliente pode responder a uma pesquisa de satisfação. O fluxo deste aplicativo seria o seguinte:

1. O cliente informa seu nome e endereço de e-mail, responde a pesquisa e clica em “Enviar”;

2. O aplicativo envia a pesquisa respondida pelo cliente para o e-mail da loja;

3. O aplicativo mostra uma tela agradecendo a visita e perguntando se o cliente gostaria de receber um e-mail com as ofertas da semana (disponibilizando os botões “Sim” e “Não”).

Perceba que o passo 2 trata-se do envio de um e-mail, e sabemos que esta é uma tarefa demorada, pois depende da velocidade da rede/Internet para que o e-mail seja enviado. Se colocássemos todas estas tarefas juntas na thread principal (UI Thread) a tela de agradecimento com o botão para que o cliente possa clicar para receber as ofertas da semana (passo 3) demoraria para aparecer, pois teria que aguardar o passo 2 ser concluído, e correríamos o risco do cliente não esperar por esta tela, perdendo a chance de enviar as ofertas a ele por e-mail.

O caso acima mostra como devemos tomar cuidado ao implementar certas tarefas na thread principal, para não corrermos o risco de deixar o aplicativo lento e consequentemente deixar nosso cliente impaciente. Para este caso, a solução seria criar uma thread secundária para o envio do e-mail, liberando a thread principal para executar as demais tarefas e deixando o cliente satisfeito com a velocidade do aplicativo.

Quando uma tarefa está demorando muito para ser executada pela UI Thread (cerca de 5 segundos), o Android lança uma mensagem para o usuário informando que o aplicativo está muito lento e pergunda se o usuário deseja aguardar a tarefa ser concluída ou se ele deseja forçar o encerramento do aplicativo. No Android esta tela é chamada de **ANR** (*Application not Responding*) e serve como medida para impedir que a plataforma fique inoperante ou trave. A **Figura 12.2** mostra o exemplo de uma tela ANR.

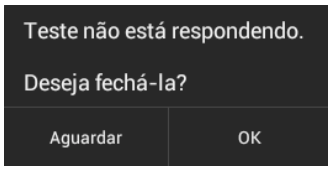


Figura 12.2. Exemplo de tela ANR apresentada quando um aplicativo está demorando para executar uma determinada tarefa.

Nota

Por motivos de segurança e concorrência, a UI Thread é a única thread no Android que pode manipular diretamente os componentes da tela de um aplicativo.

12.2 Threads secundárias

Como mostra a **Figura 12.1** o Android é multithread, ou seja, permite a execução de várias threads ao mesmo tempo em um único processo. Além da UI Thread, vista no tópico anterior, é possível criar threads secundárias em um aplicativo e estas podem ser usadas para executar tarefas com processamento mais demorado, deixando a interface gráfica mais rápida para interação com o usuário, já que a UI Thread (responsável por desenhar a tela) ficará livre para processar apenas as tarefas mais rápidas.

Trabalhar com Threads no Android é uma tarefa simples e fazemos isso da mesma forma com que trabalhamos com Threads no Java SE: usando a classe *Thread* e a interface *Runnable*.

Como exemplo, no caso anterior, onde o aplicativo precisava enviar um e-mail e logo em seguida executar outras tarefas, poderíamos criar uma thread secundária responsável apenas pelo envio do e-mail, deixando a thread principal (UI Thread) livre para executar as demais tarefas. A **Figura 12.3** ilustra este processo.

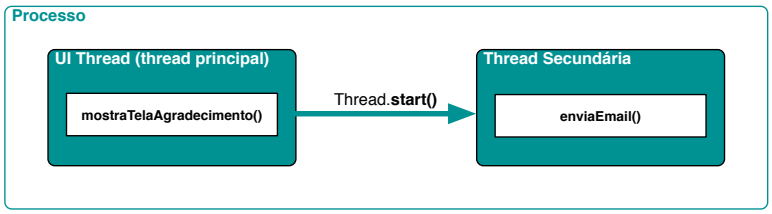


Figura 12.3. Delegando a responsabilidade do envio de e-mail para uma thread secundária.

O código para implementação desta funcionalidade teria uma estrutura parecida com esta:

```
public class PesquisaSatisfacaoActivity extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_pesquisa_satisfacao);
		. . .

		Button botaoEnviar = (Button) findViewById(R.id.botao_enviar);
		// Ação do botão "Enviar"
		botaoEnviar.setOnClickListener(new View.OnClickListener() {

			@Override
			public void onClick(View v) {
				/**
				 *  Definimos um Runnable para disparar o envio
				 *  do e-mail por outra Thread.
				 */
				Runnable runnable = new Runnable() {

					@Override
					public void run() {
						enviaEmail();
					}

				};

				/**
				 *  Enviamos o Runnable com o código para envio do e-mail
				 *  para ser executado por uma nova Thread, diferente da UI Thread.
				 */
				Thread thread = new Thread(runnable);
				thread.start();

				// Mostra ao cliente a tela de agradecimento
				mostraTelaAgradecimento();

			}
		});

	}

	/**
	 * Envia a resposta da pesquisa para o e-mail da loja.
	 * Retorna “true” caso o e-mail tenha sido enviado com sucesso.
	 */
	private boolean enviaEmail() {
		// TODO Implementação responsável pelo envio de e-mail...
		return true;
	}

	/**
	 * Abre a tela agradecendo o cliente por ter respondido à pesquisa.
	 */
	private void mostraTelaAgradecimento() {
		Intent i = new Intent(getApplicationContext(), TelaAgradecimentoActivity.class);
		startActivity(i);
	}

}
```

No código da Activity acima perceba que criamos uma *Thread* secundária passando para ela a implementação de um *Runnable* com o código a ser executado, em nosso caso o envio de e-mail. Desta forma, esta thread secundária irá executar a tarefa de envio de e-mail deixando a thread principal (UI Thread) livre para executar a tarefa de mostrar a tela de agradecimento ao cliente, ou seja, a tela não terá que esperar a conclusão do envio do e-mail para ser apresentada.

Nota

Para que um código seja executado por uma nova *thread* é preciso implementá-lo no método ***run()*** do *Runnable* que será enviado como parâmetro para o construtor desta *Thread*.

12.3 Handlers

Handlers são componentes que gerenciam uma fila de mensagens a serem processadas por uma thread. Os handlers permitem a comunicação entre threads diferentes através do envio de mensagens de uma thread para a outra.

Estas mensagens são códigos a serem executados por uma thread. O Handler simplesmente enfileira todas essas mensagens para que a thread possa executá-las em um determinado momento, após realizar as suas demais tarefas.

O maior motivo para a utilização de um Handler no Android é quando uma thread secundária precisa modificar algum componente na tela do aplicativo e, como vimos anteriormente, a UI Thread é a única capaz de realizar esta tarefa. Para que isto seja possível, basta utilizar um Handler na thread secundária para enviar uma mensagem à thread principal (UI Thread) solicitando a modificação em um determinado componente na tela.

Nota

Lembre-se sempre que a UI Thread é a única thread que pode manipular diretamente os componentes da tela de um aplicativo, mas a boa notícia é que o Android nos permite utilizar um *Handler* para que uma thread secundária possa pedir para a UI Thread modificar um determinado componente.

Continuando com nosso exemplo, na Activity ***PesquisaSatisfacaoActivity***, imagine que precisamos mostrar uma mensagem na tela avisando que o e-mail foi enviado com sucesso. Como a ação de enviar e-mail é executada por uma thread secundária, precisaremos utilizar um *Handler* para solicitar à UI Thread a apresentação dessa mensagem na tela do aplicativo. A **Figura 12.4** ilustra este processo.

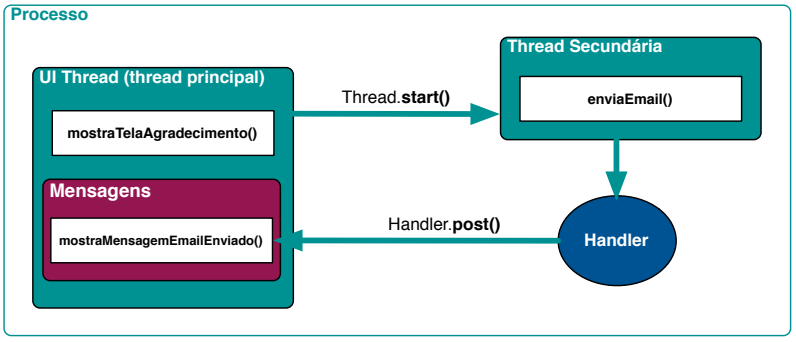


Figura 12.4. Utilizando um Handler para enviar mensagens para a UI Thread.

Para implementar esta funcionalidade devemos realizar uma pequena alteração em nossa Activity, deixando-a com a seguinte implementação:

```
public class PesquisaSatisfacaoActivity extends Activity {

    /*
     * Handler da UI Thread para enfileirar mensagens
     * a serem executadas por esta thread.
     */
```

```
private Handler handler = new Handler();

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pesquisa_satisfacao);

        . . .

        Button botaoEnviar = (Button) findViewById(R.id.botao_enviar);
        // Ação do botão "Enviar"
        botaoEnviar.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View v) {
                        /**
                         * Definimos um Runnable para disparar o envio
                         * do e-mail por outra Thread.
                         */
                        Runnable runnable = new Runnable() {

                                @Override
                                public void run() {
                                        if (enviaEmail()) {
                                           /*
                                            * Adiciona uma mensagem através do Handler
                                            * para ser enviada para a UI Thread.
                                            */
                                           handler.post(new Runnable() {
                                                   @Override
                                                   public void run() {
                                                   /*
                                                    * Usa o Toast para apresentar uma simples
                                                    * mensagem na tela
                                                    * informando que o e-mail foi enviado.
                                                    */
                                                   Toast.makeText(getApplicationContext(),
                                                      "Pesquisa enviada com sucesso!",
                                                      Toast.LENGTH_LONG).show();
                                                   }

                                           });

                                        }
                                }

                        };

                        /**
                         * Enviamos o Runnable com o código para envio do e-mail
                         * para ser executado por uma nova Thread, diferente da UI Thread.
                         */
                        Thread thread = new Thread(runnable);
                        thread.start();

                        // Mostra ao cliente a tela de agradecimento
                        mostraTelaAgradecimento();

                }
        });

}

/**
```

```
        * Envia a resposta da pesquisa para o e-mail da loja.
        */
        private boolean enviaEmail() {
                // TODO Implementação responsável pelo envio de e-mail...
                return true;
        }

        /**
        * Abre a tela agradecendo o cliente por ter respondido à pesquisa.
        */
        private void mostraTelaAgradecimento() {
                Intent i = new Intent(getApplicationContext(), TelaAgradecimentoActivity.class);
                startActivity(i);
        }

}
```

No código acima perceba que, assim como a classe *Thread*, um *Handler* também recebe um objeto do tipo *Runnable* como parâmetro, com a implementação do código a ser executado, no método ***run()***. O *Runnable* é definido em um *Handler* através do método ***post()***.

Além do método ***post()***, um *Handler* disponibiliza outros métodos que podem ser utilizados para agendar uma mensagem ou definir um atraso para que ela seja enfileirada, conforme mostra a **Tabela 12.1**.

Método	Finalidade
post(Runnable)	A mensagem é enfileirada de imediato
postAtTime(Runnable, long)	A mensagem é enfileirada em um horário determinado, definido através de um parâmetro do tipo long
postDelayed(Runnable, long)	A mensagem é enfileirada com um atraso em milisegundos, definido através de um parâmetro do tipo long

Tabela 12.1. Métodos do Handler para enfileiramento de mensagens em uma thread.

Nota

O Android disponibiliza um atalho para enviar uma mensagem para a UI Thread, o método ***runOnUIThread()***. Desta forma não precisamos declarar um *Handler*, pois este método faz isto de forma transparente. Na Activity usada como exemplo, poderíamos simplesmente enviar uma mensagem para a UI Thread da seguinte forma:

```
. . .
runOnUiThread(new Runnable() {
        @Override
        public void run() {
                /*
                * Usa o Toast para apresentar uma simples mensagem na tela
                * informando que o e-mail foi enviado.
                */
                Toast.makeText(getApplicationContext(), "Pesquisa enviada com sucesso!",
                        Toast.LENGTH_LONG).show();
        }
});
. . .
```

13 - Broadcast Receivers

Broadcast Receiver é um componente do Android que responde a determinados anúncios enviados pelo sistema. O Android envia uma mensagem para todo o sistema quando determinados eventos acontecem, e esta mensagem pode ser respondida por quem tiver interesse em recebê-las. Esta mensagem é chamada de *broadcast* e o componente responsável por respondê-la é chamado de *Broadcast Receiver*.

Como exemplo, quando a bateria do dispositivo está fraca o Android envia uma mensagem (*broadcast*) para todo o sistema informando que a carga da bateria está acabando, para que os aplicativos que têm interesse em receber essa mensagem possam executar uma determinada ação a partir desta informação. O próprio Android possui um componente para responder a esta mensagem, exibindo uma caixa de diálogo na tela informando ao usuário que é preciso conectar o dispositivo ao carregador. O componente que responde à esta mensagem é um *Broadcast Receiver*.

Além do exemplo citado acima, existem vários outros eventos no Android que enviam um broadcast para o sistema, como por exemplo: quando a tela do dispositivo é desligada, quando o carregador é plugado ao dispositivo, quando um fone de ouvido é conectado ao dispositivo, quando uma chamada telefônica é recebida, quando o sistema é iniciado e vários outros.

Um *broadcast* pode ser enviado tanto por aplicativos nativos do Android quanto pelos aplicativos que nós desenvolvemos. Estes broadcasts são enviados através de Intents, da mesma forma como fazemos para invocar uma Activity. Sendo assim, é o *Intent Resolution* (Resolução de Intents) quem vai dizer quais os *Broadcast Receivers* que estão interessados em responder ao *broadcast*. A **Figura 13.1** ilustra o exemplo de um broadcast enviado pelo Android, através de uma Intent, informando que a carga da bateria está baixa.

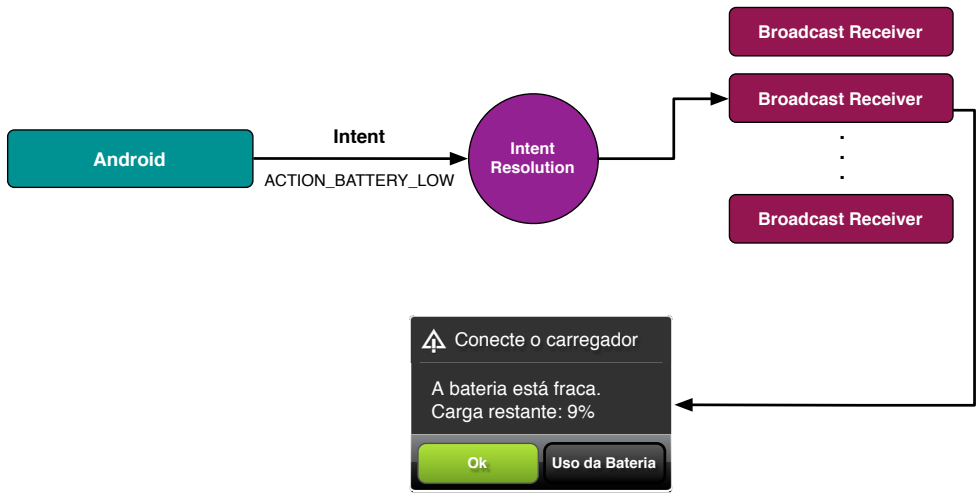


Figura 13.1. Caixa de diálogo disparada por um Broadcast Receiver, após receber um broadcast do Android, informando que a bateria do dispositivo está com carga baixa.

Através dos Broadcast Receivers podemos disparar determinadas ações quando um evento acontece. Imagine, por exemplo, que você resolveu acoplar um tablet ao painel do seu carro para usá-lo como central multimídia e computador de bordo, conectando seu cabo de força à uma fonte de energia do carro e a saída de áudio no aparelho de som do carro. Claro que você correria o risco de alguém abrir o seu carro para furtar este tablet. Pensando nessa possibilidade, um simples Broadcast Receiver poderia ajudá-lo neste caso, onde uma mensagem SMS (ou e-mail) poderia ser enviada para o

seu celular caso alguém desconectasse o cabo de força ou o cabo da saída de áudio deste tablet (o que aconteceria caso alguém o retirasse do painel com a intenção de furtá-lo). Além de enviar um SMS avisando que o dispositivo foi desconectado do painel do carro, o Broadcast Receiver também poderia ativar um serviço de localização que, de tempos em tempos, lhe enviasse um SMS com a localização geográfica do seu tablet, obtida através do GPS, ajudando-o na recuperação do seu tablet furtado.

Veja que são inúmeras as possibilidades que um Broadcast Receiver pode nos oferecer e neste capítulo aprenderemos a trabalhar com este componente.

Nota

O *Broadcast Receiver* é um componente do Android que executa em segundo plano, não depende de uma interface gráfica e não interage com o usuário.

13.1 Criando um Broadcast Receiver

Criar um *Broadcast Receiver* no Android é uma tarefa simples, bastando criar uma classe que herde a classe ***BroadcastReceiver*** e implementar um único método, o método ***onReceive()***. Veja abaixo o exemplo da estrutura de um Broadcast Receiver:

```
public class ExemploDeBroadcastReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // TODO Implementação do Broadcast Receiver . . .
        }

}
```

O método ***onReceive()*** é chamado pelo próprio Android e recebe como argumento um objeto do tipo *Context* e um *Intent*, possibilitando ao Broadcast Receiver o uso dos métodos dessas duas classes, além de permitir o recebimento de parâmetros através da Intent. É neste método que implementamos todo o código que desejamos que o Broadcast Receiver execute em segundo plano.

13.2 Configurando um Broadcast Receiver

Existem duas formas de configurar um *Broadcast Receiver* no Android:

1. **Estática** (configurado no arquivo *AndroidManifest.xml*);

2. **Dinâmica** (configurado no código de uma classe do aplicativo).

As duas formas são válidas, porém existe uma grande diferença: apenas a forma estática permite que um Broadcast Receiver seja invocado mesmo que o aplicativo esteja fechado.

13.2.1 Configurando um Broadcast Receiver de forma estática

Configurar um Broadcast Receiver de forma estática é a mais recomendada, já que desta forma ele poderá ser chamado mesmo se o aplicativo não estiver em execução. Sua configuração é feita no arquivo ***AndroidManifest.xml***, muito semelhante à forma com que configuramos uma Activity. Veja o exemplo abaixo:

```
. . .
<receiver android:name=".broadcastreceiver.ExemploDeBroadcastReceiver">
   <intent-filter>
      <action android:name="br.com.empresa.projeto.action.EXEMPLO_DE_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>
. . .
```

Perceba que configuramos uma Action no Intent Filter do nosso Broadcast Receiver. Desta forma, basta que uma Intent defina esta Action para que o Broadcast Receiver seja invocado pelo Android, através do Intent Resolution. *[Lembre-se que estudamos sobre as Intents e Intent Resolution no **Capítulo 9**]*

Nota

Além de definir uma Action, podemos também definir uma Category no Intent Filter de um Broadcast Receiver, seguindo as definições de Category que aprendemos no **Capítulo 9**.

13.2.2 Configurando um Broadcast Receiver de forma dinâmica

Configurar um Broadcast Receiver de forma dinâmica não é muito recomendado, pois desta forma ele ficará disponível apenas se o seu aplicativo estiver em execução. Podemos usar uma Activity para configurar um Broadcast Receiver de forma dinâmica, conforme o seguinte exemplo:

```
public class RegistraBroadcastReceiverActivity extends Activity {

        private ExemploDeBroadcastReceiver broadcastReceiver;
        private IntentFilter intentFilter;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);

                // Instancia o Broadcast Receiver
                broadcastReceiver = new ExemploDeBroadcastReceiver();
                // Instancia o Intent Filter
                intentFilter = new IntentFilter();
                /**
                 * Define a Action do Intent Filter que será usada
                 * para invocar o Broadcast Receiver
                 */
                intentFilter.addAction("br.com.empresa.projeto.action.EXEMPLO_DE_BROADCAST_RECEIVER");
        }

        @Override
        protected void onResume() {
                super.onResume();
                /**
                 * Registra o Broadcast Receiver no Android
                 * para que os aplicativos possam utilizá-lo.
                 */
                registerReceiver(broadcastReceiver, intentFilter);
        }

        @Override
        protected void onPause() {
                super.onPause();
                // Encerra o Broadcast Receiver
                unregisterReceiver(broadcastReceiver);
        }

}
```

Perceba que, mesmo na forma dinâmica, é preciso definir uma Action no Intent Filter para permitir que o Broadcast Receiver seja invocado pelos aplicativos através de uma Intent. Mas desta forma o Intent Filter é definido no próprio código da classe que o configura e não no arquivo *AndroidManifest.xml*.

13.3 Invocando um Broadcast Receiver

Agora que aprendemos a criar e configurar um Broadcast Receiver, precisamos entender como é feita a sua chamada no Android. Como foi dito, um Broadcast Receiver é invocado através de uma Intent, semelhante à forma com que invocamos uma Activity, porém usamos o método ***sendBroadcast()*** para iniciá-lo. O trecho de código abaixo mostra um exemplo de como chamar um Broadcast Receiver:

```
. . .
Intent intent = new Intent("br.com.empresa.projeto.action.EXEMPLO_DE_BROADCAST_RECEIVER");
sendBroadcast(intent);
. . .
```

O trecho de código acima irá invocar um Broadcast Receiver que tenha declarado em seu Intent Filter a Action definida nesta Intent e este será executado em segundo plano, de forma assíncrona, permitindo que qualquer código que venha depois da sua chamada seja executado normalmente, sem que este tenha que aguardar o Broadcast Receiver ser completamente executado.

Nota

O método ***sendBroadcast()*** é usado apenas para invocar Actions de componentes Broadcast Receivers, ou seja, se você usá-lo para chamar a Action de um outro tipo de componente (uma Activity, por exemplo) ocorrerá um erro.

13.4 Definindo a ordem de execução dos Broadcast Receivers

Como os Broadcast Receivers são invocados pela Action definida em seu Intent Filter, não é difícil de imaginar que possa existir mais de um Broadcast Receiver que declare a mesma Action. Neste caso, quando um broadcast com esta Action for disparado, todos estes Broacast Receivers receberão o broadcast ao mesmo tempo. Desta forma não é possível garantir em qual ordem estes Broadcast Receivers serão executados.

Dentro deste cenário, o que podemos fazer para garantir que um determinado Broadcast Receiver tenha prioridade sobre outro, ou seja, seja executado antes dos demais, é definir sua ordem no próprio Intent Filter. Esta ordem é definida por um valor inteiro, onde o maior valor tem prioridade sobre o menor. Este valor é definido através do atributo **android:priority**. Veja abaixo o exemplo, no trecho de um arquivo *AndroidManifest.xml*:

```
. . .
<receiver android:name="BroadcastReceiverA" >
    <intent-filter android:priority="20">
        <action android:name="MEU_BROADCAST_RECEIVER" />
    </intent-filter>
</receiver>

<receiver android:name="BroadcastReceiverB" >
    <intent-filter android:priority="30">
        <action android:name="MEU_BROADCAST_RECEIVER" />
    </intent-filter>
</receiver>

<receiver android:name="BroadcastReceiverC" >
    <intent-filter android:priority="10">
        <action android:name="MEU_BROADCAST_RECEIVER" />
    </intent-filter>
</receiver>
. . .
```

No exemplo dado temos a configuração de três Broadcast Receivers, que serão executados na seguinte ordem:

1. BroadcastReceiverB: será o primeiro por ter o maior valor de prioridade (30);

2. BroadcastReceiverA: será o segundo, por ter o segundo maior valor de prioridade (20);

3. BroadcastReceiverC: será o último, por ter o menor valor de prioridade (10).

Para chamar um Broadcast Receiver de forma ordenada usamos o método ***sendOrderedBroadcast()***. Veja o exemplo abaixo:

```
. . .
Intent intent = new Intent("MEU_BROADCAST_RECEIVER");
sendOrderedBroadcast(intent, null);
. . .
```

13.5 Impedindo a propagação de um Broadcast Receiver

Além de poder definir uma ordem de prioridade para execução dos Broadcast Receivers, é possível interferir na sua propagação, onde um determinado Broadcast Receiver pode impedir que os demais sejam executados. Para isto, no método ***onReceive()*** (que contém a implementação do Broadcast Receiver) basta chamar o método ***abortBroadcast()***.

No exemplo do tópico anterior, se chamássemos o método ***abortBroadcast()*** no Broadcast Receiver *BroadcastReceiverB*, os demais (*BroadcastReceiverA* e *BroadcastReceiverC*) não seriam executados.

Nota

O método ***abortBroadcast()*** só terá eficiência em Broadcast Receivers ordenados, já que para os **não** ordenados não dá para garantir a ordem em que eles serão executados.

13.6 Propagando informações entre Broadcast Receivers

O Android permite o compartilhamento de dados entre Broadcast Receivers ordenados, disponibilizando os seguintes métodos da classe *BroadcastReceiver*:

• **setResultExtras()**: usado para passar um objeto do tipo *Bundle* para os próximos Broadcast Receivers a serem executados;

• **setResultData()**: usado para passar uma String para os próximos Broadcast Receivers a serem executados;

E seus respectivos métodos para leitura dos dados propagados:

• **getResultExtras()**: usado para ler um objeto do tipo Bundle recebido por um Broadcast Receiver executado anteriormente;

• **getResultData()**: usado para ler uma String recebida por um Broadcast Receiver executado anteriormente.

Nota

Como é possível compartilhar um objeto do tipo *Bundle* entre os Broadcast Receivers, podemos compartilhar praticamente qualquer tipo de objeto do Java, já que o *Bundle* aceita vários tipos de objetos.

13.7 Alterando os dados na propagação de um Broadcast Receiver

As informações propagadas entre Broadcast Receivers podem ser alteradas a qualquer momento e estas alterações serão propagadas para os demais Broadcast Receivers a serem executados. Para realizar estas alterações, basta obter os dados recebidos de um Broadcast Receiver executado anteriormente, através dos métodos ***getResultExtras()*** ou ***getResultData()***, e alterá-los através dos métodos ***setResultExtras()*** ou ***setResultData()*** para que as alterações sejam propagadas para os demais Broadcast Receivers.

13.8 Regras e recomendações dos Broadcast Receivers

O Android determina algumas regras que devemos seguir para a implementação dos Broadcast Receivers. São elas:

1. O objeto que implementa o Broadcast Receiver existirá somente até o método ***onReceive()*** completar a sua execução. Devido a isto, não é permitido a implementação de tarefas assíncronas ou a criação de novas threads no método ***onReceive()***, pois estas correm o risco do Broadcast Receiver deixar de existir antes de completarem suas tarefas;

2. O Android impõe o limite máximo de 10 segundos para a execução de um Broadcast Receiver. Sendo assim, não devemos jamais implementar um código mais pesado/demorado nestes componentes;

3. Não é aconselhável que um Broadcast Receiver interfira nas tarefas que o usuário possa estar fazendo, ou seja, não deve-se mostrar mensagens ou abrir uma tela através deste componente. Imagine, por exemplo, que o usuário esteja no meio de um jogo ou escrevendo uma mensagem SMS: obviamente ele não iria gostar de ser incomodado com uma mensagem ou uma tela aparecendo no meio de sua atividade.

Nota

Caso queira executar tarefas pesadas em segundo plano (como baixar um arquivo através da Internet, por exemplo) ou enviar uma mensagem para o usuário, o Android disponibiliza outros recursos para este tipo de tarefa. Aprenderemos sobre estes recursos nos próximos capítulos, onde estudaremos sobre os ***Services*** e os ***Notificadores***.

13.9 Broadcast Receivers nativos do Android

Além de poder criar e lançar nossos próprios *broadcasts*, existem também *broadcasts* nativos que são lançados pelo próprio Android, nos permitindo criar Broadcast Receivers que respondam a eles. Veja, na tabela abaixo, alguns exemplos de *broadcasts* nativos do Android:

Broadcast	Descrição
ACTION_BATTERY_LOW	Enviado quando a bateria do dispositivo está com carga baixa.
ACTION_HEADSET_PLUG	Enviado quando um fone de ouvido é conectado ou desconectado do dispositivo.
ACTION_SCREEN_ON	Enviado quando a tela do dispositivo é ligada.
NEW_OUTGOING_CALL	Enviado quando uma nova chamada telefônica é efetuada.
BOOT_COMPLETED	Enviado quando o sistema completa o boot inicial.
ACTION_POWER_CONNECTED	Enviado quando o dispositivo é conectado à energia.
ACTION_POWER_DISCONNECTED	Enviado quando o dispositivo é desconectado da energia.
CAMERA_BUTTON	Enviado quando o botão da câmera é disparado.

Tabela 13.1. Alguns *broadcasts* nativos do Android.

Além dos citados acima, existem vários outros *broadcasts* nativos do Android para diversas funcionalidades. Consulte a documentação no site do desenvolvedor Android para conhecer todos os *broadcasts* nativos disponíveis, através do link: http://developer.android.com.

Nota

O *broadcast* nativo **BOOT_COMPLETED** exige a seguinte permissão especial para ser utilizado:

RECEIVE_BOOT_COMPLETED

O *broadcast* nativo **NEW_OUTGOING_CALL** exige a seguinte permissão especial para ser utilizado:

PROCESS_OUTGOING_CALLS

13.10 Colocando em prática: trabalhando com Broadcast Receivers

13.10.1 Criando um novo projeto para explorar os Broadcast Receivers

Para demonstrar o funcionamento dos Broadcast Receivers do Android iremos criar um novo projeto, chamado ***TrabalhandoComReceivers***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.receivers***.

13.10.2 Criando uma Activity para disparar um *broadcast*

Em nosso novo projeto vamos criar uma nova Activity, chamada ***DisparaBroadcastActivity***, definindo o seu arquivo de layout, chamado ***activity_dispara_broadcast.xml***. Para deixar nosso projeto mais organizado, colocaremos nossa Activity dentro do pacote ***br.com.hachitecnologia.receivers.activity***.

O arquivo de layout ***activity_dispara_broadcast.xml*** da nossa Activity terá o seguinte conteúdo:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/botao_dispara_broadcast"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Dispara broadcast" />

</RelativeLayout>
```

Nosso arquivo de layout possui apenas um botão, conforme ilustra a **Figura 13.2**. Ao ser clicado, este botão deverá disparar um *broadcast* e, para isto, devemos implementar nossa Activity que irá implementar a ação deste botão.

Figura 13.2. Tela com o botão que irá disparar um broadcast.

Nossa Activity ***DisparaBroadcastActivity*** conterá a seguinte implementação:

```
public class DisparaBroadcastActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dispara_broadcast);
        // Botão que irá disparar o broadcast
        Button botao = (Button) findViewById(R.id.botao_dispara_broadcast);
```

```
                // Definindo o evento do botão
                botao.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {
                                // Criamos uma Intent com a Action "MEU_BROADCAST_RECEIVER"
                                Intent intent = new Intent("MEU_BROADCAST_RECEIVER");
                                /**
                                 *  Inserimos uma String na Intent com uma mensagem
                                 *  que será enviada ao Broadcast Receiver.
                                 */
                                intent.putExtra("mensagem", "Minha mensagem");
                                // Envia um broadcast
                                sendBroadcast(intent);
                        }
                });

        }

}
```

Nota

Para que nossa Activity seja iniciada ao executar o aplicativo e para criar seu atalho no launcher do Android, é preciso definir em seu Intent Filter a Action ***MAIN*** e a Category ***LAUNCHER***. *[Lembre-se que aprendemos sobre Intent Filters no **Capítulo 9**]*

13.10.3 Implementando Broadcast Receivers no novo projeto

Implementada a Activity que irá disparar o *broadcast*, precisamos agora implementar um Broadcast Receiver que será executado quando um broadcast com a Action "**MEU_BROADCAST_RECEIVER**" for disparado. Para demonstrar melhor seus recursos, iremos criar três Broadcast Receivers, no pacote ***br.com.hachitecnologia.receivers.broadcastreceiver***: ***BroadcastReceiverA***, ***BroadcastReceiverB*** e ***BroadcastReceiverC***.

Implementação do ***BroadcastReceiverA***:

```
public class BroadcastReceiverA extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // Mensagem recebida pela Intent que disparou o broadcast
                String msg = intent.getStringExtra("mensagem");
                // Apresenta a mensagem na tela, através de um Toast
                Toast.makeText(context, "BroadcastReceiverA: " + msg, Toast.LENGTH_LONG)
                                .show();
        }

}
```

Implementação do ***BroadcastReceiverB***:

```
public class BroadcastReceiverB extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // Mensagem recebida pela Intent que disparou o broadcast
                String msg = intent.getStringExtra("mensagem");
                // Apresenta a mensagem na tela, através de um Toast
                Toast.makeText(context, "BroadcastReceiverB: " + msg, Toast.LENGTH_LONG)
                                .show();
        }
```

```
}
```

Implementação do ***BroadcastReceiverC***:

```
public class BroadcastReceiverC extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // Mensagem recebida pela Intent que disparou o broadcast
        String msg = intent.getStringExtra("mensagem");
        // Apresenta a mensagem na tela, através de um Toast
        Toast.makeText(context, "BroadcastReceiverC: " + msg, Toast.LENGTH_LONG)
                .show();
    }

}
```

Vamos agora configurar nossos Broadcast Receivers de forma estática, adicionando a seguinte configuração no arquivo *AndroidManifest.xml*:

```
<receiver android:name=".broadcastreceiver.BroadcastReceiverA">
    <intent-filter>
        <action android:name="MEU_BROADCAST_RECEIVER"/>
    </intent-filter>
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverB">
    <intent-filter>
        <action android:name="MEU_BROADCAST_RECEIVER"/>
    </intent-filter>
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverC">
    <intent-filter>
        <action android:name="MEU_BROADCAST_RECEIVER"/>
    </intent-filter>
</receiver>
```

Com nossos Broadcast Receivers implementados e devidamente configurados, ao executar o aplicativo no emulador do Android e clicar no botão "**Dispara broadcast**" da nossa Activity, um Toast exibirá as seguintes mensagens:

1. **BroadcastReceiverA**: Minha mensagem;

2. **BroadcastReceiverB**: Minha mensagem;

3. **BroadcastReceiverC**: Minha mensagem.

Apesar de nossos Broadcast Receivers não estarem ordenados, eles serão disparados na ordem acima (A, B e C). Isto acontece porque os três Broadcast Receivers estão no mesmo aplicativo, sendo assim serão executados na ordem em que foram configurados. Mas imagine se cada um estivesse em um aplicativo diferente: não daria para garantir em qual ordem eles seriam executados.

13.10.4 Definindo a ordem de execução dos Broadcast Receivers

Vamos agora alterar a ordem de execução dos nossos Broadcast Receivers, deixando-os na seguinte ordem: B, A e C. Para isto, devemos definir o atributo **android:priority** no Intent Filter de cada um, deixando-os da seguinte forma:

```
<receiver android:name=".broadcastreceiver.BroadcastReceiverA">
    <intent-filter android:priority="20">
```

```
            <action android:name="MEU_BROADCAST_RECEIVER"/>
        </intent-filter>
    </receiver>

    <receiver android:name=".broadcastreceiver.BroadcastReceiverB">
        <intent-filter android:priority="30">
            <action android:name="MEU_BROADCAST_RECEIVER"/>
        </intent-filter>
    </receiver>

    <receiver android:name=".broadcastreceiver.BroadcastReceiverC">
        <intent-filter android:priority="10">
            <action android:name="MEU_BROADCAST_RECEIVER"/>
        </intent-filter>
    </receiver>
```

Além de definir a prioridade, devemos alterar nossa Activity para que ela envie o *broadcast* de forma ordenada. Para isto, no código da nossa Activity, substituiremos o seguinte trecho de código:

```
. . .
// Envia um broadcast
sendBroadcast(intent);
. . .
```

por este trecho de código:

```
. . .
// Envia um broadcast de forma ordenada
sendOrderedBroadcast(intent, null);
. . .
```

Ao executar o aplicativo no emulador do Android e clicar no botão "**Dispara broadcast**", as mensagens serão exibidas na seguinte ordem:

1. **BroadcastReceiverB**: Minha mensagem;

2. **BroadcastReceiverA**: Minha mensagem;

3. **BroadcastReceiverC**: Minha mensagem.

13.10.5 Compartilhando dados entre os Broadcast Receivers

Como aprendemos, é possível compartilhar informações entre Broadcast Receivers. Para demonstrar este recurso, iremos enviar uma mensagem do ***BroadcastReceiverB*** para os demais Broadcast Receivers e estes deverão apresentar na tela esta nova mensagem, ao invés da mensagem original da Intent que disparou o *broadcast*. Para isto, iremos alterar o código da classe ***BroadcastReceiverB***, adicionando o seguinte trecho de código ao final do método ***onReceive()***:

```
. . .
// Define um novo objeto Bundle
Bundle b = new Bundle();
// Adiciona ao Bundle uma String com uma mensagem
b.putString("mensagem", "Minha nova mensagem");
// Disponibiliza o Bundle para os demais Broadcast Receivers a serem executados.
setResultExtras(b);
. . .
```

O trecho de código acima, no ***BroadcastReceiverB***, irá disponibilizar uma String com a mensagem "**Minha nova mensagem**" para os demais Broadcast Receivers a serem executados, em nosso caso o ***BroadcastReceiverA*** e o ***BroadcastReceiverC***. Precisamos agora receber esta mensagem nos Broadcast Receivers A e C e apresentá-la na tela.

Para isto, no código das classes ***BroadcastReceiverA*** e ***BroadcastReceiverC***, iremos substituir o trecho de código abaixo:

```
. . .
// Mensagem recebida pela Intent que disparou o broadcast
String msg = intent.getStringExtra("mensagem");
. . .
```

pelo seguinte trecho de código:

```
. . .
// Recebe o objeto Bundle disponibilizado pelo Broadcast Receiver executado anteriormente.
Bundle b = getResultExtras(true);
// Recupera a mensagem recebida no Bundle
String msg = b.getString("mensagem");
. . .
```

Após realizar as devidas alterações, executamos o aplicativo no emulador do Android e, ao clicar no botão "**Dispara broadcast**", as seguintes mensagens serão exibidas nesta ordem:

1. **BroadcastReceiverB**: Minha mensagem;

2. **BroadcastReceiverA**: Minha nova mensagem;

3. **BroadcastReceiverC**: Minha nova mensagem.

13.10.6 Alterando os dados propagados entre os Broadcast Receivers

Como aprendemos, além de compartilhar dados entre Broascast Receivers ordenados, podemos também alterar estes dados propagados em um determinado Broadcast Receiver. Para demonstrar este recurso, iremos alterar o código da nossa classe ***BroadcastReceiverA***, modificando o conteúdo da mensagem recebida pelo ***BroadcastReceiverB***. Para isto, adicionaremos o seguinte trecho de código ao final do método ***onReceive()*** da classe ***BroadcastReceiverA***:

```
. . .
// Altera o conteúdo da mensagem do Bundle
b.putString("mensagem", "Minha mensagem modificada");
// Disponibiliza o Bundle alterado para os próximos Broadcast Receivers
setResultExtras(b);
. . .
```

Ao executar o aplicativo no emulador do Android e clicar no botão "**Dispara broadcast**", as mensagens serão exibidas na seguinte ordem:

1. **BroadcastReceiverB**: Minha mensagem;

2. **BroadcastReceiverA**: Minha nova mensagem;

3. **BroadcastReceiverC**: Minha mensagem modificada.

13.11 Exercício

Agora que você aprendeu a trabalhar com Broadcast Receivers, é hora de colocar em prática.

Neste exercício iremos criar três Broadcast Receivers e explorar seus recursos.

1. Crie um novo projeto do Android, chamado ***TrabalhandoComReceivers***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.receivers***.

2. Neste novo projeto crie uma nova Activity, chamada ***DisparaBroadcastActivity***, definindo o seu arquivo de layout com o nome ***activity_dispara_broadcast.xml***. Para deixar o projeto mais organizado, coloque esta nova Activity no pacote ***br.com.hachitecnologia.receivers.activity***.

3. Em seu arquivo de Layout, defina apenas um botão com o texto "Dispara broadcast". Para isto, edite o arquivo ***activity_dispara_broadcast.xml***, deixando-o com o seguinte conteúdo:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <Button
      android:id="@+id/botao_dispara_broadcast"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="Dispara broadcast" />

</RelativeLayout>
```

4. O botão definido no arquivo de layout, ao ser clicado pelo usuário, deverá disparar um *broadcast* com a Action "MEU_BROADCAST_RECEIVER". Na Intent que irá disparar o *broadcast*, deverá ser enviada uma String de identificador "mensagem" com o conteúdo "Minha mensagem". Para isto, edite a classe ***DisparaBroadcastActivity***, deixando-a com o seguinte conteúdo:

```
public class DisparaBroadcastActivity extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_dispara_broadcast);
                // Botão que irá disparar o broadcast
                Button botao = (Button) findViewById(R.id.botao_dispara_broadcast);
                // Definindo o evento do botão
                botao.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {
                                // Criamos uma Intent com a Action "MEU_BROADCAST_RECEIVER"
                                Intent intent = new Intent("MEU_BROADCAST_RECEIVER");
                                /**
                                 *  Inserimos uma String na Intent com uma mensagem
                                 *  que será enviada ao Broadcast Receiver.
                                 */
                                intent.putExtra("mensagem", "Minha mensagem");
                                // Envia um broadcast
                                sendBroadcast(intent);
                        }
                });

        }

}
```

5. No Intent Filter da nova Activity, defina a Action ***MAIN*** e a Category ***LAUNCHER***, para que ela seja iniciada ao executar o aplicativo e para criar seu atalho no launcher do Android.

6. Crie três Broascast Receivers, no pacote ***br.com.hachitecnologia.receivers.broadcastreceiver***, com os respectivos nomes: ***BroadcastReceiverA***, ***BroadcastReceiverB*** e ***BroadcastReceiverC***. Estes Broadcast Receivers deverão implementar o seguinte código:

Implementação do ***BroadcastReceiverA***:

```
public class BroadcastReceiverA extends BroadcastReceiver {

	@Override
	public void onReceive(Context context, Intent intent) {
		// Mensagem recebida pela Intent que disparou o broadcast
		String msg = intent.getStringExtra("mensagem");
		// Apresenta a mensagem na tela, através de um Toast
		Toast.makeText(context, "BroadcastReceiverA: " + msg, Toast.LENGTH_LONG)
				.show();
	}

}
```

Implementação do ***BroadcastReceiverB***:

```
public class BroadcastReceiverB extends BroadcastReceiver {

	@Override
	public void onReceive(Context context, Intent intent) {
		// Mensagem recebida pela Intent que disparou o broadcast
		String msg = intent.getStringExtra("mensagem");
		// Apresenta a mensagem na tela, através de um Toast
		Toast.makeText(context, "BroadcastReceiverB: " + msg, Toast.LENGTH_LONG)
				.show();
	}

}
```

Implementação do ***BroadcastReceiverC***:

```
public class BroadcastReceiverC extends BroadcastReceiver {

	@Override
	public void onReceive(Context context, Intent intent) {
		// Mensagem recebida pela Intent que disparou o broadcast
		String msg = intent.getStringExtra("mensagem");
		// Apresenta a mensagem na tela, através de um Toast
		Toast.makeText(context, "BroadcastReceiverC: " + msg, Toast.LENGTH_LONG)
				.show();
	}

}
```

7. Configure os três Broadcast Receivers de forma estática e sem ordenação. Para isto, adicione a seguinte configuração no arquivo ***AndroidManifest.xml***:

```
<receiver android:name=".broadcastreceiver.BroadcastReceiverA">
   <intent-filter>
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverB">
   <intent-filter>
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
```

```
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverC">
   <intent-filter>
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>
```

8. Execute o projeto no emulador do Android e, ao iniciar o aplicativo, clique no botão “Dispara broadcast” e veja o resultado.

13.12 Exercício

Neste exercício iremos definir uma ordem de execução dos três Broadcast Receivers implementados no projeto ***TrabalhandoComReceivers***.

1. Defina uma ordem de execução dos Broadcast Receivers, usando o atributo android:priority do Intent Filter, deixando-os na seguinte ordem: B, A e C. Para isto, altere sua configuração, deixando-a da seguinte forma:

```
<receiver android:name=".broadcastreceiver.BroadcastReceiverA">
   <intent-filter android:priority="20">
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverB">
   <intent-filter android:priority="30">
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>

<receiver android:name=".broadcastreceiver.BroadcastReceiverC">
   <intent-filter android:priority="10">
      <action android:name="MEU_BROADCAST_RECEIVER"/>
   </intent-filter>
</receiver>
```

Altere também o código da Activity ***DisparaBroadcastActivity***, substituindo o trecho de código abaixo:

```
. . .
// Envia um broadcast
sendBroadcast(intent);
. . .
```

pelo seguinte trecho de código:

```
. . .
// Envia um broadcast de forma ordenada
sendOrderedBroadcast(intent, null);
. . .
```

2. Execute o projeto no emulador do Android e, ao iniciar o aplicativo, clique no botão “Dispara broadcast” e veja o resultado.

13.13 Exercício opcional

Neste exercício iremos implementar o compartilhamento de dados entre os três Broadcast Receivers implementados no projeto ***TrabalhandoComReceivers***. Iremos compartilhar uma String, com a mensagem "Minha nova mensagem", usando o identificador "mensagem".

1. Altere o código da classe ***BroadcastReceiverB***, adicionando o seguinte trecho de código ao final do método ***onReceive()***:

```
. . .
// Define um novo objeto Bundle
Bundle b = new Bundle();
// Adiciona ao Bundle uma String com uma mensagem
b.putString("mensagem", "Minha nova mensagem");
// Disponibiliza o Bundle para os demais Broadcast Receivers a serem executados.
setResultExtras(b);
. . .
```

2. No código das classes ***BroadcastReceiverA*** e ***BroadcastReceiverC***, substitua o trecho de código abaixo:

```
. . .
// Mensagem recebida pela Intent que disparou o broadcast
String msg = intent.getStringExtra("mensagem");
. . .
```

pelo seguinte trecho de código:

```
. . .
// Recebe o objeto Bundle disponibilizado pelo Broadcast Receiver executado anteriormente.
Bundle b = getResultExtras(true);
// Recupera a mensagem recebida no Bundle
String msg = b.getString("mensagem");
```

3. Execute o projeto no emulador do Android e, ao iniciar o aplicativo, clique no botão "Dispara broadcast" e veja o resultado.

13.14 Exercício opcional

Neste exercício iremos alterar os dados propagados entre os três Broadcast Receivers implementados no projeto ***TrabalhandoComReceivers***.

1. Adicione o seguinte trecho de código ao final do método ***onReceive()*** da classe ***BroadcastReceiverA***:

```
. . .
// Altera o conteúdo da mensagem do Bundle
b.putString("mensagem", "Minha mensagem modificada");
// Disponibiliza o Bundle alterado para os próximos Broadcast Receivers
setResultExtras(b);
. . .
```

2. Execute o projeto no emulador do Android e, ao iniciar o aplicativo, clique no botão "Dispara broadcast" e veja o resultado.

14 - Services

Service é um componente do Android que executa tarefas em segundo plano. Ao contrário dos Broadcast Receivers, os Services podem executar tarefas demoradas, como o envio de um e-mail, o download de um arquivo da Internet, etc.

Os Services são prioridade no Android, ou seja, se em um determinado momento o Android precisar liberar espaço na Memória Ram do dispositivo, os Services serão os últimos componentes a serem eliminados. Em termos de prioridade, os Services perdem apenas para a Activity em execução, que tem a mais alta prioridade no Android.

Um exemplo de Service são os famosos aplicativos que tocam música, permitindo que o usuário possa executar outras tarefas enquanto uma música está sendo tocada e a qualquer momento ele pode voltar ao aplicativo de música e parar a música.

Um *Service* pode ser iniciado através de qualquer componente do Android, ou seja, tanto uma Activity quanto um Broadcast Receiver ou um próprio Service pode iniciá-lo.

14.1 Criando um Service

Para criar um *Service* no Android, devemos estender a classe ***Service***. Veja abaixo a estrutura básica de um Service:

```
public class ExemploDeService extends Service {

	@Override
	public void onCreate() {
		// Tarefas a serem executadas quando o service é criado
	}

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {
		// Tarefas a serem executadas quando o service é iniciado
	}

	@Override
	public void onDestroy() {
		// Tarefas a serem executadas quando o service é destruído
	}

	@Override
	public IBinder onBind(Intent intent) {
		return null;
	}


}
```

• O método ***onCreate()*** é executado apenas uma vez, quando o Service é criado.

• O método ***onStartCommand()*** é executado toda vez que o Service é chamado, através do método ***startService()***, e recebe como parâmetro a Intent utilizada em sua chamada; *[Aprenderemos sobre o método startService() nos próximos tópicos]*

• O método ***onBind()*** é o único que precisa ser obrigatoriamente implementado em um Service. Este método nos permite conectar a um Service em execução e controlá-lo; *[Aprenderemos sobre este recurso mais adiante ainda neste capítulo]*

• O método ***onDestroy()*** é executado apenas uma vez, quando o Service é destruído.

14.2 Configurando um Service

Assim como as Activities e os Broadcast Receivers, é preciso configurar o *Service* no arquivo ***AndroidManifest.xml***. Veja abaixo o exemplo da configuração de um *Service* no Android:

```
. . .
<service android:name=".service.ExemploDeService">
   <intent-filter>
      <action android:name="br.com.hachitecnologia.action.INICIAR_SERVICE_DE_EXEMPLO" />
   </intent-filter>
</service>
. . .
```

Nota

Como os *Services* são invocados a partir de uma Intent, é preciso definir uma Action em seu Intent Filter, assim como mostra o exemplo acima.

14.3 Iniciando um Service

Para iniciar um *Service* usamos o método ***startService()***, passando como parâmetro uma Intent com a sua Action definida. Veja abaixo o exemplo da chamada de um Service:

```
. . .
Intent intent = new Intent("br.com.hachitecnologia.action.INICIAR_SERVICE_DE_EXEMPLO");
startService(intent);
. . .
```

Nota

No construtor da Intent passamos a Action definida no Intent Filter do *Service*, conforme mostra o exemplo acima.

14.4 Parando um Service

Existem duas formas de parar um *Service* em execução, veja:

1. A primeira forma de parar um Service é chamando o método ***stopService()*** na classe que o chamou, passando como parâmetro a Intent usada para iniciá-lo;

2. Outra forma de parar um Service é chamando o método ***stopSelf()*** dentro do próprio Service.

14.5 Considerações importantes sobre os Services

É preciso levar em conta as seguintes considerações importantes sobre os *Services*, no Android:

1. Um *Service* executa em segundo plano de forma independente do componente que o chamou, ou seja, se uma Activity iniciar um Service, ele continuará em execução mesmo que a Activity seja destruída;

2. Um *Service* não executa em um processo ou thread separados, ou seja, ele executa no mesmo processo responsável pela execução do aplicativo, na UI Thread. Isso significa que se o processo do aplicativo for finalizado, o Service também será finalizado;

3. Como os *Services* executam na UI Thread e são usados para processar tarefas mais demoradas em segundo plano, é recomendado que seja criada uma nova thread pra executar suas tarefas. Desta forma a UI Thread ficará livre para executar as demais tarefas do aplicativo, fazendo com que o aplicativo responda mais rápido aos eventos solicitados pelo usuário.

14.6 Conectando-se a um Service em execução

O Android nos permite conectar a um Service em execução para permitir o controle de suas tarefas. Imagine, por exemplo, um aplicativo desenvolvido para tocar músicas onde as músicas são tocadas em segundo plano através de um Service. Obviamente em um determinado momento precisaremos nos conectar a este Service para parar ou pausar a música. Isto é possível através de um processo chamado ***bind*** e ***unbind***.

14.6.1 A classe *Binder*

Binder é um recurso do Android que permite a comunicação entre um componente do Android e um Service em execução, ou seja, é ele que fará o meio de campo quando precisamos nos conectar a um Service para controlá-lo.

No Android, para criar um *Binder* é preciso seguir as seguintes regras:

1. Criar uma classe que estenda a classe ***Binder*** do Android;

2. Disponibilizar no Binder um método que permita o acesso ao Service, retornando a sua referência.

Veja abaixo o exemplo de um *Binder*, usado para controlar com um Service em execução:

```
public class ExemploDeBinder extends Binder {

	private ExemploDeService exemploDeService;

	public ExemploDeBinder(ExemploDeService exemploDeService) {
		this.exemploDeService = exemploDeService;
	}

	/**
	 * Método responsável por permitir o acesso ao Service.
	 * @return
	 */
	public ExemploDeService getExemploDeService() {
		return exemploDeService;
	}

}
```

14.6.2 O método *onBind()*

Mencionamos anteriormente sobre o método ***onBind()*** de implementação obrigatória em um Service. É justamente este método que usamos para nos conectar a um Service em execução.

Este método deve simplesmente retornar o ***Bind*** implementado para se comunicar com o Service. Veja abaixo o exemplo da implementação do método ***onBind()*** em um Service:

```
public class ExemploDeService extends Service {

	private IBinder binder = new ExemploDeBinder(this);

	. . .

	@Override
	public IBinder onBind(Intent intent) {
		return binder;
	}

}
```

No exemplo dado, perceba que apenas inicializamos o ***Binder*** em uma variável de instância no Service e, no método ***onBind()***, retornamos a referência a este *Binder*.

14.6.3 A interface *ServiceConnection*

Além do *Binder*, é necessário criar um componente chamado *Service Connection* para permitir a comunicação com um Service. É justamente este componente que irá se conectar ao Service e usar o *Binder* para controlá-lo.

No Android, para criar um *Service Connection* é preciso seguir as seguintes regras:

1. Criar uma classe que implemente a Interface ***ServiceConnection*** do Android;

2. Implementar os métodos ***onServiceConnected()*** e ***onServiceDisconnected()*** da Interface ***ServiceConnection***. Estes métodos serão executados quando algum componente se conectar e desconectar do Service, respectivamente;

3. No método ***onServiceConnected()***, definir o *Binder* que será utilizado para se comunicar com o Service;

4. Disponibilizar um método que permita o acesso ao Service através do *Binder*, retornando a sua referência.

Veja abaixo o exemplo de um *Service Connection*, usado para se conectar a um Service em execução:

```
public class ExemploDeServiceConnection implements ServiceConnection {

	private ExemploDeService exemploDeService;

	@Override
	public void onServiceConnected(ComponentName component, IBinder binder) {
		ExemploDeBinder exemploDeBinder = (ExemploDeBinder) binder;
		this.exemploDeService = exemploDeBinder.getExemploDeService();
	}

	@Override
	public void onServiceDisconnected(ComponentName name) {
		// Tarefas a serem executadas ao se desconectar do Service
	}

	public ExemploDeService getExemploDeService() {
		return exemploDeService;
	}

}
```

No exemplo dado, perceba que no método ***onServiceConnected()*** nós apenas definimos o *Binder* que será utilizado para controlar o Service e disponibilizamos um método, chamado ***getExemploDeService()***, que apenas fará a ponte para o método de mesmo nome que criamos no *Binder*, o que nos permitirá usar este *Service Connection* para conectar e controlar o Service.

14.6.4 O método *bindService()*

Com um *Binder* e um *Service Connection* implementados, podemos nos conectar ao Service através de um componente do Android. Para isto, basta usar o método ***bindService()***, da classe Context do Android, passando como parâmetro a mesma Intent usada para iniciar o Service juntamente com o *Service Connection* definido. Veja o exemplo:

```
. . .
Intent intent = new Intent("br.com.hachitecnologia.action.INICIAR_SERVICE_DE_EXEMPLO");
ExemploDeServiceConnection connection = new ExemploDeServiceConnection();
bindService(intent, connection, 0);
. . .
```

No exemplo acima, após chamar o método bindService(), nosso componente já estará conectado ao Service em execução. Desta forma, podemos usar o objeto do *Service Connection* para obter a referência do Service e controlá-lo, acessando qualquer um de seus métodos disponíveis.

Nota

Ao chamar o ***bindService()***, o método ***onServiceConnected()*** do *Service Connection* é invocado.

14.6.5 O método *unbindService()*

O método ***unbindService()*** do Android é usado para se desconectar de um Service. Este método recebe como parâmetro a referência ao objeto *Service Connection* usado para se conectar ao Service. Veja o exemplo:

```
. . .
unbindService(connection);
. . .
```

Nota

Ao chamar o ***unbindService()***, o método ***onServiceDisconnected()*** do *Service Connection* é invocado.

14.7 Colocando em prática: criando um tocador de música

Para demonstrar o funcionamento dos Services, iremos criar um aplicativo que irá tocar uma música MP3 em segundo plano, disponibilizando as seguintes funcionalidades: Tocar, Pausar e Parar. Estas funcionalidades deverão ser disponibilizadas por uma Activity, que apresentará três botões, permitindo o usuário a tocar, pausar e parar a música.

Em nosso aplicativo, usaremos a Activity para iniciar o Service e se conectar a ele para controlar a execução do arquivo de áudio através do Service.

Nota

Para permitir que nosso aplicativo toque uma música, usaremos a classe ***MediaPlayer*** nativa do Android, usada para controlar arquivos de áudio e vídeo. Usaremos apenas as funcionalidades básicas desta classe, como *start*, *stop* e *pause*, usados para tocar, parar e pausar um arquivo de áudio/vídeo, respectivamente. Porém esta classe é muito poderosa e disponibiliza diversos outros recursos. Caso tenha interesse em se aprofundar no assunto, o link abaixo disponibiliza a documentação completa da classe ***MediaPlayer***:

http://developer.android.com/reference/android/media/MediaPlayer.html

14.7.1 Criando um novo projeto para explorar os Services

Para implementar nosso aplicativo tocador de música criaremos um novo projeto, chamado ***TocadorDeMusica***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.tocadordemusica***.

14.7.2 Implementando o Service

Como nosso Service será responsável por tocar, pausar e parar um arquivo de áudio, implementaremos um método para cada uma dessas funcionalidades. Estes métodos deverão ser expostos (usando o modificador de acesso ***public***) para que uma Activity possa ter acesso a eles para controlar a execução da música, ao se conectar a este Service. Criaremos então, no pacote ***br.com.hachitecnologia.tocadordemusica.service***, uma nova classe, chamada ***TocaMusicaService***, com a seguinte implementação:

```
public class TocaMusicaService extends Service {

    private MediaPlayer mediaPlayer;
    // Posição atual da música em milissegundos
    private int posicao = 0;
    // Música a ser tocada, localizada no diretório "assets"
    private static final String MUSICA = "born_to_be_wild.mp3";

    /**
     * Método responsável por tocar a Música
```

```
		 */
		public void tocaMusica() {
			/*
			 * Se a música foi pausada, dizemos ao MediaPlayer que a música deve ser
			 * iniciada a partir da posição em que ela parou de tocar.
			 */
			if (posicao > 0)
				mediaPlayer.seekTo(posicao);
			/*
			 * Se a música NÃO foi pausada, dizemos o MediaPlayer para tocar a música
			 * a partir do início.
			 */
			else {
				mediaPlayer = new MediaPlayer();
				try {
					AssetFileDescriptor afd = getApplicationContext().
						getAssets().openFd(MUSICA);
					mediaPlayer.setDataSource(afd.getFileDescriptor(),
						afd.getStartOffset(), afd.getLength());
					mediaPlayer.prepare();
				} catch (Exception e) {
					// TODO
				}
			}
			// Pede para o MediaPlayer tocar a música
			mediaPlayer.start();
		}

		/**
		 * Método responsável por parar a música
		 */
		public void paraMusica() {
			if (mediaPlayer.isPlaying()) {
				mediaPlayer.stop();
				// Ao parar a música, retorna para a sua posição inicial (0)
				posicao = 0;
			}
		}

		/**
		 * Método responsável por pausar a música
		 */
		public void pausaMusica() {
			if (mediaPlayer.isPlaying()) {
				mediaPlayer.pause();
				// Ao pausar a música, guarda a posição em que ela parou de tocar
				posicao = mediaPlayer.getCurrentPosition();
			}
		}

		@Override
		public IBinder onBind(Intent intent) {
			return null;
		}

	}
```

Mesmo sendo pequeno, o código do nosso *Service* implementa perfeitamente as funcionalidades que queremos, disponibilizando os métodos: ***tocaMusica()***, ***paraMusica()*** e ***pausaMusica()*** responsáveis por tocar, parar e pausar uma música, respectivamente.

Na classe criamos uma constante, chamada **MUSICA**, definindo o nome do arquivo de áudio a ser tocado, em nosso caso o arquivo **born_to_be_wild.mp3**. Em nosso código estamos utilizando o método ***getAssets()*** da classe *Context* para localizar este arquivo dentro do diretório ***assets*** do nosso aplicativo, no caso o arquivo **born_to_be_wild.mp3**, que adicionamos a este diretório. Para tornar nosso exemplo mais simples, o código irá tocar apenas esta música, mas você poderia facilmente criar um algoritimo que localize todos os arquivos de áudio do dispositivo e apresente-os em uma *ListView* para o usuário selecionar qual música deseja ouvir.

Nota

Veja como a classe ***MediaPlayer***, nativa do Android, facilitou nosso trabalho. Desta forma não precisamos nos preocupar em como funciona um tocador de arquivos de áudio/vídeo pois o Android nos disponibiliza uma API completa para realizarmos esta tarefa. Além desta, o Android disponibiliza diversas outras APIs para realizar diversas tarefas. Tenha sempre o costume de consultar a documentação oficial do Android antes de iniciar um aplicativo, para se beneficiar das APIs já existentes, o que irá facilitar o seu trabalho. Consulte a documentação completa no site: **http://developer.android.com**.

14.7.3 Configurando o Service

Com o nosso Service implementado, precisamos agora configurá-lo e definir a sua Action. Para isto, adicionaremos o seguinte trecho no arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.TocaMusicaService" >
   <intent-filter>
      <action android:name="br.com.hachitecnologia.action.TOCAR_MUSICA" />
   </intent-filter>
</service>
. . .
```

14.7.4 Implementando o Binder

Com o nosso Service já implementado e devidamente configurado, precisamos agora criar o *Binder* que irá permitir a comunicação entre uma Activity e o Service. No pacote ***br.com.hachitecnologia.tocadordemusica.service*** criaremos, então, uma nova classe chamada ***TocaMusicaBinder***, com a seguinte implementação:

```
public class TocaMusicaBinder extends Binder {

        private TocaMusicaService tocaMusicaService;

        public TocaMusicaBinder(TocaMusicaService tocaMusicaService) {
                this.tocaMusicaService = tocaMusicaService;
        }

        /**
         * Método responsável por permitir o acesso ao Service.
         * @return
         */
        public TocaMusicaService getTocaMusicaService() {
                return tocaMusicaService;
        }

}
```

Perceba que em nosso Binder disponibilizamos o método ***getTocaMusicaService()*** para permitir o acesso ao Service.

Com o nosso Binder definido, precisamos agora definí-lo no método ***onBind()*** do nosso Service. Para isto, adicionaremos a seguinte variável de instância à nossa classe ***TocaMusicaService***:

```
. . .
```

```
private IBinder binder = new TocaMusicaBinder(this);
. . .
```

Precisamos também modificar o retorno do método ***onBind()***, deixando-o da seguinte forma:

```
. . .
@Override
public IBinder onBind(Intent intent) {
        return binder;
}
. . .
```

14.7.5 Implementando o Service Connection

Agora que temos o Service e o Binder já implementados, precisamos criar o *Service Connection* que fará a conexão com o Service. No pacote ***br.com.hachitecnologia.tocadordemusica.service*** criaremos, então, uma nova classe chamada ***TocaMusicaServiceConnection*** com a seguinte implementação:

```
public class TocaMusicaServiceConnection implements ServiceConnection {

        private TocaMusicaService tocaMusicaService;

        @Override
        public void onServiceConnected(ComponentName component, IBinder binder) {
                TocaMusicaBinder tocaMusicaBinder = (TocaMusicaBinder) binder;
                this.tocaMusicaService = tocaMusicaBinder.getTocaMusicaService();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
                // TODO
        }

        public TocaMusicaService getTocaMusicaService() {
                return tocaMusicaService;
        }

}
```

Perceba que em nosso Service Connection disponibilizamos também um método ***getTocaMusicaService()***, que chamará o método de mesmo nome do Binder para permitir o acesso ao Service.

14.7.6 Implementando a Activity

Precisamos agora implementar a Activity e a tela que irá permitir ao usuário controlar a música através dos botões: Tocar, Pausar e Parar. Para isto, no pacote ***br.com.hachitecnologia.tocadordemusica.activity***, criaremos uma nova Activity, chamada ***TocadorDeMusicaActivity***, definindo seu arquivo de layout de nome ***activity_tocador_de_musica.xml***.

Como nossa tela conterá apenas três botões, implementaremos o arquivo ***activity_tocador_de_musica.xml*** da seguinte forma:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:gravity="center" >

   <Button
      android:id="@+id/botao_toca_musica"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
            android:text="Tocar" />

        <Button
            android:id="@+id/botao_pausa_musica"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pausar"
            android:enabled="false" />

        <Button
            android:id="@+id/botao_para_musica"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Parar"
            android:enabled="false" />

    </LinearLayout>
```

Na classe ***TocadorDeMusicaActivity*** implementaremos o seguinte código:

```
public class TocadorDeMusicaActivity extends Activity {

    private Button botaoTocaMusica;
    private Button botaoParaMusica;
    private Button botaoPausaMusica;

    private Intent intent;
    private TocaMusicaServiceConnection connection;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tocador_de_musica);

        // Inicia o Service
        iniciaService();
        // Conecta ao Service
        conectaAoService();

        botaoTocaMusica = (Button) findViewById(R.id.botao_toca_musica);
        botaoParaMusica = (Button) findViewById(R.id.botao_para_musica);
        botaoPausaMusica = (Button) findViewById(R.id.botao_pausa_musica);

        // Evento do botão "Tocar"
        botaoTocaMusica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Pede para o Service tocar a música
                connection.getTocaMusicaService().tocaMusica();
                // Desativa o botão "Tocar" e ativa os demais
                ativaEDesativaBotoesDoTocador(true);
            }
        });

        // Evento do botão "Parar"
        botaoParaMusica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Pede para o Service parar a música
                connection.getTocaMusicaService().paraMusica();
                // Ativa o botão "Tocar" e desativa os demais
                ativaEDesativaBotoesDoTocador(false);
            }
```

```
                        });

                        // Evento do botão "Pause"
                        botaoPausaMusica.setOnClickListener(new View.OnClickListener() {
                                @Override
                                public void onClick(View v) {
                                        // Pede para o Service pausar a música
                                        connection.getTocaMusicaService().pausaMusica();
                                        // Ativa o botão "Tocar" e desativa os demais
                                        ativaEDesativaBotoesDoTocador(false);
                                }
                        });

                }

                /**
                 * Método responsável por iniciar o Service
                 */
                private void iniciaService() {
                        intent = new Intent("br.com.hachitecnologia.action.TOCAR_MUSICA");
                        startService(intent);
                }

                /**
                 * Método responsável por conectar ao Service
                 */
                private void conectaAoService() {
                        connection = new TocaMusicaServiceConnection();
                        bindService(intent, connection, 0);
                }

                /**
                 * Método que irá ativar/desativar os botões do Tocador de música.
                 * @param musicaEstaTocando
                 */
                private void ativaEDesativaBotoesDoTocador(boolean musicaEstaTocando) {
                        botaoTocaMusica.setEnabled(!musicaEstaTocando);
                        botaoParaMusica.setEnabled(musicaEstaTocando);
                        botaoPausaMusica.setEnabled(musicaEstaTocando);
                }

                @Override
                protected void onDestroy() {
                        super.onDestroy();
                        // Desconecta do Service quando a Activity é destruída
                        unbindService(connection);
                }

        }
```

Por fim, copiamos o arquivo de áudio chamado born_to_be_wild.mp3 para o diretório ***assets*** do nosso projeto, configuramos nossa Activity para ser iniciada ao abrir o aplicativo no Android e adicionamos seu ícone no Launcher.

Nosso aplicativo está pronto para tocar o nosso arquivo de áudio. A **Figura 14.1** mostra o aplicativo em execução no emulador do Android.

Figura 14.1. Tela do aplicativo *TocadorDeMusica*.

14.8 Exercício

Agora que você aprendeu a trabalhar com Services, é hora de colocar em prática.

Neste exercício iremos criar um aplicativo para tocar música no Android.

1. Crie um novo projeto do Android, chamado ***TocadorDeMusica***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.tocadordemusica***.

2. No pacote ***br.com.hachitecnologia.tocadordemusica.service***, crie uma nova classe, chamada ***TocaMusicaService***, com a seguinte implementação:

```
public class TocaMusicaService extends Service {

	private MediaPlayer mediaPlayer;
	// Posição atual da música em milissegundos
	private int posicao = 0;
	// Música a ser tocada, localizada no diretório "assets"
	private static final String MUSICA = "born_to_be_wild.mp3";

	/**
	 * Método responsável por tocar a Música
	 */
	public void tocaMusica() {
		/*
		 * Se a música foi pausada, dizemos ao MediaPlayer que a música deve ser
		 * iniciada a partir da posição em que ela parou de tocar.
		 */
		if (posicao > 0)
			mediaPlayer.seekTo(posicao);
		/*
		 * Se a música NÃO foi pausada, dizemos o MediaPlayer para tocar a música
		 * a partir do início.
		 */
		else {
			mediaPlayer = new MediaPlayer();
			try {
				AssetFileDescriptor afd = getApplicationContext().
					getAssets().openFd(MUSICA);
				mediaPlayer.setDataSource(afd.getFileDescriptor(),
					afd.getStartOffset(), afd.getLength());
				mediaPlayer.prepare();
			} catch (Exception e) {
				// TODO
```

```
                        }
                    }
                    // Pede para o MediaPlayer tocar a música
                    mediaPlayer.start();
                }

                /**
                 * Método responsável por parar a música
                 */
                public void paraMusica() {
                    if (mediaPlayer.isPlaying()) {
                        mediaPlayer.stop();
                        // Ao parar a música, retorna para a sua posição inicial (0)
                        posicao = 0;
                    }
                }

                /**
                 * Método responsável por pausar a música
                 */
                public void pausaMusica() {
                    if (mediaPlayer.isPlaying()) {
                        mediaPlayer.pause();
                        // Ao pausar a música, guarda a posição em que ela parou de tocar
                        posicao = mediaPlayer.getCurrentPosition();
                    }
                }

                @Override
                public IBinder onBind(Intent intent) {
                    return null;
                }

    }
```

3. Configure o Service no aplicativo, adicionando o seguinte trecho ao arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.TocaMusicaService" >
    <intent-filter>
        <action android:name="br.com.hachitecnologia.action.TOCAR_MUSICA" />
    </intent-filter>
</service>
. . .
```

4. No pacote ***br.com.hachitecnologia.tocadordemusica.service***, crie uma nova classe, chamada ***TocaMusicaBinder***, com a seguinte implementação:

```
public class TocaMusicaBinder extends Binder {

    private TocaMusicaService tocaMusicaService;

    public TocaMusicaBinder(TocaMusicaService tocaMusicaService) {
        this.tocaMusicaService = tocaMusicaService;
    }

    /**
     * Método responsável por permitir o acesso ao Service.
     * @return
     */
    public TocaMusicaService getTocaMusicaService() {
        return tocaMusicaService;
    }
```

```
}
```

5. Adicione a seguinte variável de instância à classe ***TocaMusicaService***:

```
. . .
private IBinder binder = new TocaMusicaBinder(this);
. . .
```

6. Modifique o método ***onBind()*** da classe ***TocaMusicaService***, deixando-o com a seguinte implementação:

```
. . .
@Override
public IBinder onBind(Intent intent) {
        return binder;
}
. . .
```

7. Dentro do pacote ***br.com.hachitecnologia.tocadordemusica.service***, crie uma nova classe, chamada ***TocaMusicaServiceConnection***, com a seguinte implementação:

```
public class TocaMusicaServiceConnection implements ServiceConnection {

        private TocaMusicaService tocaMusicaService;

        @Override
        public void onServiceConnected(ComponentName component, IBinder binder) {
                TocaMusicaBinder tocaMusicaBinder = (TocaMusicaBinder) binder;
                this.tocaMusicaService = tocaMusicaBinder.getTocaMusicaService();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
                // TODO
        }

        public TocaMusicaService getTocaMusicaService() {
                return tocaMusicaService;
        }

}
```

8. Dentro do pacote ***br.com.hachitecnologia.tocadordemusica.activity***, crie uma nova Activity, chamada ***TocadorDeMusicaActivity***, definindo seu arquivo de layout com o nome ***activity_tocador_de_musica.xml***.

9. Modifique o arquivo de Layout ***activity_tocador_de_musica.xml***, deixando-o da seguinte forma:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:gravity="center" >

   <Button
      android:id="@+id/botao_toca_musica"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tocar" />

   <Button
      android:id="@+id/botao_pausa_musica"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Pausar"
        android:enabled="false" />

    <Button
        android:id="@+id/botao_para_musica"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Parar"
        android:enabled="false" />

</LinearLayout>
```

10. Modifique a classe ***TocadorDeMusicaActivity***, deixando-a com a seguinte implementação:

```
public class TocadorDeMusicaActivity extends Activity {

    private Button botaoTocaMusica;
    private Button botaoParaMusica;
    private Button botaoPausaMusica;

    private Intent intent;
    private TocaMusicaServiceConnection connection;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tocador_de_musica);

        // Inicia o Service
        iniciaService();
        // Conecta ao Service
        conectaAoService();

        botaoTocaMusica = (Button) findViewById(R.id.botao_toca_musica);
        botaoParaMusica = (Button) findViewById(R.id.botao_para_musica);
        botaoPausaMusica = (Button) findViewById(R.id.botao_pausa_musica);

        // Evento do botão "Tocar"
        botaoTocaMusica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Pede para o Service tocar a música
                connection.getTocaMusicaService().tocaMusica();
                // Desativa o botão "Tocar" e ativa os demais
                ativaEDesativaBotoesDoTocador(true);
            }
        });

        // Evento do botão "Parar"
        botaoParaMusica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Pede para o Service parar a música
                connection.getTocaMusicaService().paraMusica();
                // Ativa o botão "Tocar" e desativa os demais
                ativaEDesativaBotoesDoTocador(false);
            }
        });

        // Evento do botão "Pause"
        botaoPausaMusica.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
                                public void onClick(View v) {
                                        // Pede para o Service pausar a música
                                        connection.getTocaMusicaService().pausaMusica();
                                        // Ativa o botão "Tocar" e desativa os demais
                                        ativaEDesativaBotoesDoTocador(false);
                                }
                        });

        }

        /**
         * Método responsável por iniciar o Service
         */
        private void iniciaService() {
                intent = new Intent("br.com.hachitecnologia.action.TOCAR_MUSICA");
                startService(intent);
        }

        /**
         * Método responsável por conectar ao Service
         */
        private void conectaAoService() {
                connection = new TocaMusicaServiceConnection();
                bindService(intent, connection, 0);
        }

        /**
         * Método que irá ativar/desativar os botões do Tocador de música.
         * @param musicaEstaTocando
         */
        private void ativaEDesativaBotoesDoTocador(boolean musicaEstaTocando) {
                botaoTocaMusica.setEnabled(!musicaEstaTocando);
                botaoParaMusica.setEnabled(musicaEstaTocando);
                botaoPausaMusica.setEnabled(musicaEstaTocando);
        }

        @Override
        protected void onDestroy() {
                super.onDestroy();
                // Desconecta do Service quando a Activity é destruída
                unbindService(connection);
        }

}
```

11. No arquivo ***AndroidManifest.xml***, configure a Activity ***TocadorDeMusicaActivity*** para que ela seja iniciada ao abrir o aplicativo no Android. Adicione também a Activity no Launcher do Android.

12. Copie um arquivo de áudio .MP3, de nome born_to_be_wild.mp3 (ou com o nome que desejar, lembrando-se de modificar seu nome também na constante da classe *TocaMusicaService*) para o diretório ***assets*** do projeto. *[Lembre-se que o nome do arquivo não pode conter espaços, acentos ou símbolos]*

13. Execute e teste o aplicativo no emulador do Android.

15 - Usando a API de SMS

No **Capítulo 9** nós usamos a Action ***ACTION_SENDTO*** para enviar uma mensagem SMS, através de uma Intent, usando o aplicativo nativo de mensagens do Android. Este mecanismo funciona perfeitamente, mas desta forma o envio do SMS não fica totalmente automatizado, pois ao usar esta Action o programa de mensagens do Android solicita que o usuário confirme o envio do SMS clicando no botão "Enviar".

O Android disponibiliza uma forma para enviar um SMS de forma transparente, sem a necessidade de chamar o aplicativo de envio de SMS, tornando esta tarefa totalmente automatizada em seu aplicativo. Isto é possível através da classe ***SmsManager*** do Android.

15.1 Usando a classe SmsManager para o envio automatizado de SMS

A classe ***SmsManager*** do Android é utilizada para enviar um SMS de forma automatizada, sem a necessidade da intervenção do usuário. Veja abaixo o exemplo do envio de um SMS através do *SmsManager*:

```
. . .
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage("8765-4321", null, "Mensagem de teste", null, null);
. . .
```

No exemplo acima, o código irá enviar uma mensagem SMS para o número **8765-4321**, com a seguinte mensagem: **Mensagem de teste**.

Para utilizar a classe SmsManager no Android é preciso definir a permissão ***android.permission.SEND_SMS***. Para isto, basta adicionar a seguinte linha ao arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.SEND_SMS" />
. . .
```

O método ***sendTextMessage()*** da classe *SmsManager* é usado apenas para enviar mensagens curtas, que não ultrapassam o tamanho limite de uma mensagem SMS. Para enviar mensagens de texto maiores, deve ser usado o método ***sendMultipartTextMessage()*** em conjunto com o método ***divideMessage()***, que irá dividir a mensagem e enviá-la em partes cujo tamanho é suportado. Veja abaixo o exemplo do envio de uma mensagem longa através da classe *SmsManager*:

```
. . .
String msg = "Esta mensagem é grande demais para ser enviada em apenas " +
                    "um SMS e deve ser dividida em mais de uma parte para podermos " +
                    "usar a classe SmsManager para seu envio. Por isso, devemos usar o " +
                    "método sendMultipartTextMessage() para que esta mensagem seja " +
                    "enviada.";
SmsManager sms = SmsManager.getDefault();
// Dividimos a mensagem através do método divideMessage()
ArrayList<String> mensagemParticionada = sms.divideMessage(msg);
// Usamos o método sendMultipartTextMessage() para enviar cada parte da mensagem em um SMS
sms.sendMultipartTextMessage(numeroDestino, null, mensagemParticionada, null, null);
. . .
```

15.2 Colocando em prática: usando o SmsManager no projeto *Devolva.me*

Agora que aprendemos a trabalhar com a classe *SmsManager* do Android para o envio de SMS de forma automática, vamos usar este recurso em nosso projeto ***Devolva.me***, substituindo o código que usava a Action ***ACTION_SENDTO*** para o envio do SMS de lembrete.

Para realizar esta tarefa, no projeto ***Devolva.me*** iremos criar uma nova classe, chamada ***Telefonia***, dentro do pacote ***br.com.hachitecnologia.devolvame.util***, com a seguinte implementação:

```
public class Telefonia {

	/**
	 * Método responsável por enviar mensagens SMS.
	 * @param numeroTelefone
	 * @param mensagem
	 */
	public static void enviaSMS(String numeroTelefone, String mensagem) {
		SmsManager sms = SmsManager.getDefault();
		// Divide a mensagem em partes
		ArrayList<String> mensagemArray = sms.divideMessage(mensagem);
		// Envia cada parte da mensagem em um SMS
		sms.sendMultipartTextMessage(numeroTelefone, null, mensagemArray, null, null);
	}

}
```

Agora precisamos apenas modificar a classe ***ListaObjetosEmprestadosActivity***, no método ***onContextItemSelected()***, substituindo o trecho de código abaixo:

```
. . .
// Envia uma mensagem SMS de lembrete para o número de telefone cadastrado
Intent i = new Intent(Intent.ACTION_SENDTO);
i.setData(Uri.parse("sms:" + objeto.getContato().getTelefone()));
i.putExtra("sms_body", "Olá, você pegou emprestado o meu objeto \"" +
		objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.");
startActivity(i);
. . .
```

pelo seguinte trecho de código:

```
. . .
String mensagem = "Olá, você pegou emprestado o meu objeto \"" +
		objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.";
Telefonia.enviaSMS(objeto.getContato().getTelefone(), mensagem);
Toast.makeText(getApplicationContext(), "Lembrete enviado.", Toast.LENGTH_LONG).show();
. . .
```

Pronto! Agora nosso aplicativo enviará um SMS com um lembrete de forma mais automatizada, lembrando a pessoa de devolver o objeto emprestado. Para testar esta funcionalidade, basta abrir a tela com a lista dos objetos emprestados, selecionar um objeto na lista, pressionando-o por 3 segundos, e clicar na opção "**Enviar SMS**".

15.3 Exercício

Agora que você aprendeu a trabalhar com a API de SMS do Android, é hora de colocar em prática.

Neste exercício iremos automatizar o envio do lembrete SMS no projeto ***Devolva.me***.

1. No projeto ***Devolva.me***, crie uma nova classe no pacote ***br.com.hachitecnologia.devolvame.util***, chamada ***Telefonia***, com a seguinte implementação:

```
public class Telefonia {

	/**
	 * Método responsável por enviar mensagens SMS.
	 * @param numeroTelefone
	 * @param mensagem
	 */
	public static void enviaSMS(String numeroTelefone, String mensagem) {
		SmsManager sms = SmsManager.getDefault();
		// Divide a mensagem em partes
		ArrayList<String> mensagemArray = sms.divideMessage(mensagem);
		// Envia cada parte da mensagem em um SMS
		sms.sendMultipartTextMessage(numeroTelefone, null, mensagemArray, null, null);
	}

}
```

2. Na classe ***ListaObjetosEmprestadosActivity***, no método ***onContextItemSelected()***, substitua o trecho de código abaixo:

```
. . .
// Envia uma mensagem SMS de lembrete para o número de telefone cadastrado
Intent i = new Intent(Intent.ACTION_SENDTO);
i.setData(Uri.parse("sms:" + objeto.getContato().getTelefone()));
i.putExtra("sms_body", "Olá, você pegou emprestado o meu objeto \"" +
	objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.");
startActivity(i);
. . .
```

pelo seguinte trecho de código:

```
. . .
String mensagem = "Olá, você pegou emprestado o meu objeto \"" +
	objeto.getObjeto() + "\" e ainda não o devolveu. Por favor, devolva-me o quanto antes.";
Telefonia.enviaSMS(objeto.getContato().getTelefone(), mensagem);
Toast.makeText(getApplicationContext(), "Lembrete enviado.", Toast.LENGTH_LONG).show();
```

3. Adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.SEND_SMS" />
. . .
```

4. Execute o aplicativo no emulador do Android e faça o teste da nova funcionalidade.

16 - Alarmes e Notificação

16.1 Alarmes

Alarme é um recurso do Android que permite agendar uma ação para ser executada em um determinado momento no futuro, ou em intervalos de tempo pré-definidos.

Os Alarmes são mantidos mesmo que o dispositivo esteja em estado de espera, podendo acordar o dispositivo, se necessário, quando for disparado.

Nota

Os Alarmes não são mantidos se o dispositivo for desligado ou reiniciado. Portanto, caso necessite que um Alarme seja mantido mesmo que o dispositivo tenha sido reiniciado, é preciso reagendá-lo após o boot do sistema, através de um Broadcast Receiver.

16.1.1 Tipos de Alarme

O Alarme possui quatro tipos definidos. São eles:

Tipo de Alarme	Descrição
RTC	Dispara uma Intent em um horário definido.
RTC_WAKEUP	Acorda o dispositivo e dispara uma Intent em um horário definido.
ELAPSED_REALTIME	Dispara uma Intent depois de um determinado tempo que o dispositivo foi ligado.
ELAPSED_REALTIME_WAKEUP	Acorda o dispositivo e dispara uma Intent depois de um determinado tempo que o dispositivo foi ligado.

Tabela 16.1. Tipos de Alarme.

16.1.2 Agendando um Alarme

Para criar um Alarme no Android usamos a classe ***AlarmManager***. Esta classe é responsável por agendar um Alarme e sua instância é obtida através da seguinte sintaxe:

```
AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

16.1.2.1 O método *set()*

O método ***set()*** da classe ***AlarmManager*** é usado para definir um Alarme em um horário específico. Veja sua sintaxe:

```
set(TIPO_DO_ALARME, TEMPO_DEFINIDO, PendingIntent);
```

Onde:

- TIPO_DO_ALARME: é o tipo que queremos definir para o Alarme, conforme visto na **Tabela 16.1**;
- TEMPO_DEFINIDO: é um long, com o horário definido (em milissegundos) que queremos que o Alarme seja disparado;
- PendingIntent: objeto *PendingIntent* contendo a Intent que queremos executar quando o Alarme for disparado.

Nota

O objeto ***PendingIntent*** é usado para definir uma Intent que ficará pendente, ou seja, que será disparada em algum determinado momento no futuro.

16.1.2.2 O método *setRepeating()*

O método ***setRepeating()*** é usado para definir um Alarme que será repetido em um determinado momento. Veja sua sintaxe:

```
setRepeating(TIPO_DO_ALARME, TEMPO_DEFINIDO, INTERVALO, PendingIntent);
```

Onde:

• INTERVALO: é um long com o intervalo definido (em milissegundos) que queremos que o Alarme seja repetido.

16.1.2.3 O método *setInexactRepeating()*

O método ***setInexactingRepeating()*** define a mesma funcionalidade do método ***setRepeating()***, porém em intervalos de tempo inexatos. Sua sintaxe é idêntica à do método ***setRepeating()***, veja:

```
setInexactRepeating(TIPO_DO_ALARME, TEMPO_DEFINIDO, INTERVALO, PendingIntent);
```

Onde:

No parâmetro INTERVALO podemos passar como argumento as seguintes constantes da classe ***AlarmManager***:

• **INTERVAL_FIFTEEN_MINUTES**: repete o Alarme de 15 em 15 minutos;

• **INTERVAL_HALF_HOUR**: repete o Alarme de meia em meia hora;

• **INTERVAL_HOUR**: repete o alarme de hora em hora;

• **INTERVAL_HALF_DAY**: repete o Alarme de 12 em 12 horas;

• **INTERVAL_DAY**: repete o Alarme a cada 24 horas.

16.1.2.4 O método *cancel()*

O método ***cancel()*** é usado para cancelar o agendamento de um Alarme. Veja sua sintaxe:

```
cancel(PendingIntent);
```

Onde:

• PendingIntent: é a mesma ***PendingIntent*** que foi usada para agendar o Alarme.

16.2 Notificação

O Android possui um recurso que permite um aplicativo a notificar o usuário quando algo aconteceu em segundo plano, e este recurso é chamado de ***Notificação***. O serviço de Notificação pode ser tanto visível (uma mensagem na barra de status ou o piscar da tela) quanto sonoro (um som de toque ou vibração).

Nota

O serviço de Notificação do Android é o mais aconselhável quando um Service ou um Broadcast Receiver precisa mostrar uma mensagem de notificação ao usuário, pois este recurso não é intrusivo e não atrapalha a tarefa que o usuário possa estar executando no momento em que queremos notificá-lo.

16.2.1 Gerando uma Notificação

Para gerar uma Notificação, no Android, devemos seguir os seguintes passos:

1. Definir um objeto ***Notification***;

2. Definir uma ***PendingIntent***;

3. Definir os dados da Notificação através do método ***setLatestEventInfo()***;

4. Obter o ***NotificafionManager***;

5. Chamar o método ***notify()*** da classe ***NotificationManager***, passando como parâmetro o objeto ***Notification*** e a ***PendingIntent*** definidos.

16.2.1.1 Definindo o objeto *Notification*

O primeiro passo para criar uma Notificação no Android é definir um objeto ***Notification***. Veja sua sintaxe:

```
Notification notification = new Notification(ICONE, TITULO, TEMPO_DEFINIDO);
```

Onde:

• ICONE: é um inteiro referente ao valor definido na classe ***R*** que faz referência para a imagem que desejamos utilizar como ícone, ou seja, é a referência a um Drawable Resource. Este é o ícone que será apresentado na barra de notificação;

• TITULO: é uma String com o título que será apresentado na barra de status;

• TEMPO_DEFINIDO: é um inteiro, definindo o tempo (em milissegundos) que queremos que a notificação seja apresentada.

Veja o exemplo:

```
. . .
long tempoDefinido = System.currentTimeMillis();
String titulo = "Exemplo de notificação";
Notification notification = new Notification(R.drawable.ic_launcher, titulo, tempoDefinido);
. . .
```

16.2.1.2 Definindo uma *PendingIntent*

Devemos, também, definir uma ***PendingIntent*** para gerar uma Notificação. Uma ***PendingIntent*** é usada para definir a Intent que será disparada quando o usuário clicar sobre a notificação, na barra de status. *[Lembre-se que usamos uma PendingIntent quando falamos sobre os Alarmes]*

Veja o exemplo:

```
. . .
Intent intent = new Intent(context, TelaInicialActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
. . .
```

No exemplo dado, estamos criando uma *PendingIntent* que irá disparar uma **Activity** em um determinado momento. Além de disparar uma Activity, podemos também disparar um Broadcast Receiver ou um Service, usando os métodos ***getBroadcast()*** e ***getService()*** da classe *PendingIntent*, respectivamente

16.2.1.3 Definindo os dados da Notificação

Para definir os dados da Notificação, usamos o método ***getLatestEventInfo()***. Veja sua sintaxe:

```
notification.setLatestEventInfo(CONTEXT, TITULO, MENSAGEM, PendingIntent);
```

Onde:

• CONTEXT: é o objeto Context;

• TITULO: é uma String com o título que será apresentado ao expandir a notificação;

• MENSAGEM: é uma String com a mensagem da notificação;

• PendingIntent: é um objeto *PendingIntent* com a Intent que será disparada quando o usuário clicar sobre a notificação.

Veja o exemplo:

```
. . .
Intent intent = new Intent(context,TelaInicialActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
notification.setLatestEventInfo(context, "Título da notificação", "Mensagem da notificação", pendingIntent);
. . .
```

16.2.1.4 Obtendo o *NotificationManager*

Uma Notificação é criada através da classe ***NotificationManager***. Sua instância é obtida através da seguinte sintaxe:

```
NotificationManager notificationManager =
        (NotificationManager).getSystemService(Context.NOTIFICATION_SERVICE);
```

16.2.1.5 Chamando o método *notify()*

Após obter o ***NotificationManager***, para gerar a notificação basta chamar seu método ***notify()***. Este método possui a seguinte sintaxe:

```
notify(ID, NOTIFICATION);
```

Onde:

• ID: é um inteiro com um ID a ser definido para a Notificação. É através deste ID que podemos cancelar uma Notificação agendada;

• NOTIFICATION: é o objeto ***Notification*** definido.

16.2.2 Definindo alertas para a Notificação

Além da mensagem, podemos também utilizar alertas sonoros, vibração e fazer piscar o LED do dispositivo quando uma notificação é disparada.

Existem duas formas de definir estes recursos: especificando cada recurso separadamente ou definindo todos usando o padrão de toque, vibração e LED do dispositivo no atributo ***defaults*** do objeto ***Notification***. Veja o exemplo:

```
notification.defaults = Notification.DEFAULT_ALL;
```

O exemplo dado fará com que o dispositivo use como som de toque, alerta vibratório e LED os mesmos padrões definidos pelo usuário na configuração do dispositivo.

Além desta forma, podemos definir cada recurso separadamente, como veremos nos próximos tópicos.

16.2.2.1 Definindo um som de toque para a Notificação

Para definir um som de toque para uma Notificação, usamos o atributo ***sound*** do objeto ***Notification*** definido. Este atributo recebe como valor um URI. Desta forma podemos usar qualquer som de toque que esteja disponível no dispositivo. Veja o exemplo:

```
notification.sound = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
```

No exemplo dado, definimos como som de alerta o toque padrão de notificações configurado no dispositivo.

16.2.2.2 Definindo um alerta vibratório para a Notificação

Para definir padrão de vibração para uma Notificação, usamos o atributo ***vibrate*** do objeto ***Notification*** definido. Este atributo recebe como parâmetro um array de long, que deve ser preenchido com o padrão Pausa/Vibração em milissegundos. Veja o exemplo:

```
notification.vibrate = new long[]{500, 1000, 500, 1000, 500, 1000, 500, 1000};
```

No exemplo dado, ao ser disparada a Notificação, o dispositivo irá vibrar 4 vezes, com vibrações de 1000 milissegundos e em intervalos de 500 milissegundos.

Nota

Para utilizar o recurso de vibração em um aplicativo, é preciso declarar a permissão ***android.permission.VIBRATE***. Para isto, basta adicionar a seguinte linha no arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.VIBRATE" />
```

16.2.2.3 Fazendo o LED piscar ao disparar uma Notificação

O Android permite também a definição de um padrão de cor para o LED piscar quando uma Notificação é disparada. Veja o exemplo:

```
notification.ledARGB = Color.GREEN;
notification.ledOnMS = 500;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

No exemplo dado, definimos que o LED irá piscar na cor verde e cada piscada irá durar 500 milissegundos, em intervalos de 1000 milissegundos.

Nota

Para que este recurso funcione é preciso que o dispositivo tenha um LED (mini lâmpada) de notificação.

16.2.3 Cancelando uma Notificação

Após o usuário visualizar a Notificação, para cancelá-la, ou seja, para não deixá-la mais visível na barra de status, usamos o método ***cancel()*** da classe ***NotificationManager***. Este método recebe como parâmetro o **ID** que definimos no método ***notify()***, quando agendamos a Notificação. Veja o exemplo:

```
NotificationManager notificationManager =
        (NotificationManager).getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.cancel(0);
```

No exemplo dado, estamos cancelando uma Notificação de ID 0 (zero).

Além desta maneira, podemos configurar a Notificação para que ela seja cancelada automaticamente após o usuário clicar sobre ela. Para isto, no momento da criação da Notificação, basta definir a flag ***Notification.FLAG_AUTO_CANCEL***. Veja o exemplo:

```
notification.flags |= Notification.FLAG_AUTO_CANCEL;
```

16.2.4 Exemplo de Notificação

Veja abaixo o exemplo completo do agendamento de uma Notificação:

```
. . .
```

```
// Título da Notificação
String titulo = "Título da Notificação";
// Mensagem da Notificação
String mensagem = "Mensagem da Notificação";
// Tempo em que a Notificação será disparada
long tempoDefinido = System.currentTimeMillis();

// Objeto Notification
Notification notification = new Notification(R.drawable.ic_launcher, titulo, tempoDefinido);

// Intent que será disparada quando o usuário clicar sobre a Notificação
Intent intent = new Intent(context, TelaInicialActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);

// Configurando os dados da Notificação
notification.setLatestEventInfo(context, titulo, mensagem, pendingIntent);

// Cancela (oculta) a notificação após o usuário clicar sobre ela
notification.flags |= Notification.FLAG_AUTO_CANCEL;
// Define o som de toque, alerta vibratório e LED padrões do dispositivo
notification.defaults = Notification.DEFAULT_ALL;

// Agendando a Notificação
NotificationManager notificationManager = (NotificationManager) context
                .getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(0, notification);
. . .
```

16.2.5 Colocando em prática: trabalhando com Alarmes e Notificação

Para demonstrar o funcionamento dos Alarmes e das Notificações do Android, iremos criar um novo projeto. Neste projeto iremos apenas disponibilizar uma tela com um botão que, ao ser pressionado, irá definir um Alarme para ser disparado em 30 segundos. Ao ser disparado, o Alarme deverá mostrar uma Notificação avisando que o mesmo foi disparado.

16.2.5.1 Criando um novo projeto para explorar os Alarmes e a Notificação

Criaremos, então, nosso novo projeto, chamado ***AlarmesENotificacao***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.alarmesenotificacao***.

16.2.5.2 Criando um Service para mostrar a Notificação

O primeiro passo em nosso novo projeto é implementar o Service que irá mostrar a Notificação. Para isto, criaremos uma nova classe, chamada ***MostraNotificacaoService***, no pacote ***br.com.hachitecnologia.alarmesenotificacao.service***, com a seguinte implementação:

```
public class MostraNotificacaoService extends Service {

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        // Título do Alarme
        String titulo = "Alarme disparado";
        // Mensagem do Alarme
        String mensagem = "O alarme foi disparado.";

        // Definindo o objeto Notification
        Notification notification = new Notification(R.drawable.ic_launcher, titulo,
                System.currentTimeMillis());

        /*
         * Criando a PendingIntent definindo a Intent que será executada
         * quando o usuário clicar sobre a Notificação. Em nosso caso,
         * será invocada a Activity AgendaAlarmeActivity.
         */
        Intent i = new Intent(getApplicationContext(), AgendaAlarmeActivity.class);
```

```
                PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), 0, i, 0);

                // Definimos os dados da Notificação
                notification.setLatestEventInfo(getApplicationContext(), titulo, mensagem, pendingIntent);
                // Cancela automaticamente a Notificação quando o usuário clicar sobre ela
                notification.flags |= Notification.FLAG_AUTO_CANCEL;
                // Define som de toque, alerta vibratório e LED para os padrões do dispositivo
                notification.defaults = Notification.DEFAULT_ALL;

                // Agenda a Notificação
                NotificationManager notificationManager = (NotificationManager)
                        getSystemService(Context.NOTIFICATION_SERVICE);
                notificationManager.notify(0, notification);

                return super.onStartCommand(intent, flags, startId);
        }

        @Override
        public IBinder onBind(Intent intent) {
                return null;
        }

}
```

Vamos definir a Action ***br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO*** para o nosso Service, adicionando a seguinte configuração no arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.MostraNotificacaoService" >
   <intent-filter>
      <action android:name="br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO" />
   </intent-filter>
</service>
. . .
```

Nota

Como a nossa Notificação irá usar o recurso de vibração, devemos adicionar a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

16.2.5.3 Criando uma Activity para agendar o Alarme

Como queremos que um Alarme seja agendado ao clicar no botão de uma tela, iremos criar uma Activity, chamada ***AgendaAlarmeActivity***, dentro do pacote ***br.com.hachitecnologia.alarmesenotificacao.activity***, definindo seu arquivo de Layout com o nome ***activity_agenda_alarme.xml***.

Nosso arquivo de Layout ***activity_agenda_alarme.xml*** terá o seguinte conteúdo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:gravity="center" >

   <Button
      android:id="@+id/botao_agendar_alarme"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
```

```
            android:text="Tocar alarme em 30 segundos" />

    </LinearLayout>
```

Em nossa Activity ***AgendaAlarmeActivity*** definiremos a seguinte implementação:

```
public class AgendaAlarmeActivity extends Activity {

    private Button botao;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_agenda_alarme);

        // Faz referência ao botão do arquivo de Layout
        botao = (Button) findViewById(R.id.botao_agendar_alarme);
        // Define o evento do botão
        botao.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {

                /*
                 *  Horário em que o Alarme será disparado. Em mosso caso, em 30
                 *  segundo após o clique no botão.
                 */
                long horaDoDisparo = System.currentTimeMillis() + 30000;

                /*
                 * Define a Intent que será executada quando o Alarme for disparado. Em
                 * nosso caso, chamaremos o Service MostraNotificacaoService para mostrar
                 * uma Notificação informando que o Alarme foi disparado.
                 */
                Intent i = new
                        Intent("br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO");
                PendingIntent pendingIntent = PendingIntent
                        .getService(getApplicationContext(), 0, i, 0);

                // Define o Alarme
                AlarmManager alarmManager = (AlarmManager)
                        getSystemService(Context.ALARM_SERVICE);
                alarmManager.set(AlarmManager.RTC_WAKEUP, horaDoDisparo,
                        pendingIntent);

                // Desabilita o botão para impedir cliques repetidos.
                botao.setEnabled(false);

                // Mostra um Toast informando que o Alarme foi agendado.
                Toast.makeText(getApplicationContext(), "Alarme agendado!",
                        Toast.LENGTH_LONG).show();

            }
        });

    }

}
```

Nossa implementação está pronta! Ao executar o aplicativo no emulador do Android e clicar no botão da tela, o Alarme será agendado e após 30 segundos ele será disparado, mostrando uma Notificação.

16.3 Exercício

Agora que você aprendeu a trabalhar com Alarmes e Notificação, é hora de colocar em prática.

Neste exercício iremos criar um aplicativo que deverá agendar um Alarme para ser disparado após 30 segundos, quando o usuário clicar no botão apresentado em uma tela.

1. Crie um novo projeto, chamado ***AlarmesENotificacao***, definindo o nome do pacote padrão para ***br.com.hachitecnologia.alarmesenotificacao***.

2. No pacote ***br.com.hachitecnologia.alarmesenotificacao.service***, crie uma nova classe Service, chamada ***MostraNotificacaoService***, com a seguinte implementação:

```
public class MostraNotificacaoService extends Service {

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {

		// Título do Alarme
		String titulo = "Alarme disparado";
		// Mensagem do Alarme
		String mensagem = "O alarme foi disparado.";

		// Definindo o objeto Notification
		Notification notification = new Notification(R.drawable.ic_launcher, titulo,
			System.currentTimeMillis());

		/*
		 * Criando a PendingIntent definindo a Intent que será executada
		 * quando o usuário clicar sobre a Notificação. Em nosso caso,
		 * será invocada a Activity AgendaAlarmeActivity.
		 */
		Intent i = new Intent(getApplicationContext(), AgendaAlarmeActivity.class);
		PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), 0, i, 0);

		// Definimos os dados da Notificação
		notification.setLatestEventInfo(getApplicationContext(), titulo, mensagem, pendingIntent);
		// Cancela automaticamente a Notificação quando o usuário clicar sobre ela
		notification.flags |= Notification.FLAG_AUTO_CANCEL;
		// Define som de toque, alerta vibratório e LED para os padrões do dispositivo
		notification.defaults = Notification.DEFAULT_ALL;

		// Agenda a Notificação
		NotificationManager notificationManager = (NotificationManager)
			getSystemService(Context.NOTIFICATION_SERVICE);
		notificationManager.notify(0, notification);

		return super.onStartCommand(intent, flags, startId);
	}

	@Override
	public IBinder onBind(Intent intent) {
		return null;
	}

}
```

3. Defina a Action ***br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO*** para o novo Service, adicionando a seguinte configuração no arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.MostraNotificacaoService" >
```

```
        <intent-filter>
            <action android:name="br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO" />
        </intent-filter>
    </service>
    . . .
```

4. Para permitir a vibração do dispositivo quando a Notificação for apresentada, adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

5. No pacote ***br.com.hachitecnologia.alarmesenotificacao.activity***, crie uma nova classe Activity, chamada ***AgendaAlarmeActivity***, definindo seu arquivo de Layout com o nome ***activity_agenda_alarme.xml***.

6. Defina o seguinte conteúdo ao arquivo ***activity_agenda_alarme.xml***:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center" >

    <Button
        android:id="@+id/botao_agendar_alarme"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Tocar alarme em 30 segundos" />

</LinearLayout>
```

7. Defina a seguinte implementação na Activity ***AgendaAlarmeActivity***:

```
public class AgendaAlarmeActivity extends Activity {

        private Button botao;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_agenda_alarme);

                // Faz referência ao botão do arquivo de Layout
                botao = (Button) findViewById(R.id.botao_agendar_alarme);
                // Define o evento do botão
                botao.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {

                                /*
                                 *  Horário em que o Alarme será disparado. Em mosso caso, em 30
                                 *  segundo após o clique no botão.
                                 */
                                long horaDoDisparo = System.currentTimeMillis() + 30000;

                                /*
                                 * Define a Intent que será executada quando o Alarme for disparado. Em
                                 * nosso caso, chamaremos o Service MostraNotificacaoService para mostrar
```

```
                                        * uma Notificação informando que o Alarme foi disparado.
                                        */
                                       Intent i = new
                                               Intent("br.com.hachitecnologia.action.MOSTRA_NOTIFICACAO");
                                       PendingIntent pendingIntent = PendingIntent
                                               .getService(getApplicationContext(), 0, i, 0);

                                       // Define o Alarme
                                       AlarmManager alarmManager = (AlarmManager)
                                               getSystemService(Context.ALARM_SERVICE);
                                       alarmManager.set(AlarmManager.RTC_WAKEUP, horaDoDisparo,
                                               pendingIntent);

                                       // Desabilita o botão para impedir cliques repetidos.
                                       botao.setEnabled(false);

                                       // Mostra um Toast informando que o Alarme foi agendado.
                                       Toast.makeText(getApplicationContext(), "Alarme agendado!",
                                               Toast.LENGTH_LONG).show();

                                }
                        });

                }

        }
```

8. Configure a Activity ***AgendaAlarmeActivity*** para que seja iniciada ao abrir o aplicativo no Android e para que seja mostrada no Launcher.

9. Execute o aplicativo no emulador do Android e faça o teste.

17 Apêndice - Aperfeiçoando o projeto "Devolva.me"

Como profissionais de TI sabemos que em todo projeto podem surgir novas especificações. No projeto *Devolva.me*, nosso cliente José decide solicitar uma nova funcionalidade para facilitar ainda mais a sua vida.

No desenvolvimento destes novos recursos do projeto iremos explorar os Services, Broadcast Receivers, Alarmes e Notificações do Android.

17.1 A história

Em nova conversa, José solicita a implementação de um recurso que permitirá definir um alarme para lembrá-lo, na data/hora definida, de pedir de volta um objeto que ele emprestou.

José solicita, se possível, uma forma mais automatizada para o lembrete, de forma que a própria pessoa que pegou o objeto emprestado possa receber via SMS um aviso para devolver o objeto. Além disso, José quer receber uma notificação de que este lembrete foi enviado conforme a data/hora definida.

17.2 Definição da nova solicitação

De acordo com a necessidade do cliente, deverão ser feitas as seguintes considerações:

1. O sistema deverá enviar automaticamente um SMS notificando a pessoa para devolver o objeto emprestado;
2. A tela de cadastro de objetos deve disponibilizar uma opção para que o usuário possa definir uma data/hora para que o lembrete seja enviado;
3. Ao enviar a notificação via SMS, o sistema deverá informar o usuário de que o lembrete foi enviado com sucesso.

17.2.1 A nova tela de cadastro

De acordo com a nova especificação solicitada pelo cliente, a nova tela de cadastro do aplicativo deverá conter também os seguintes recursos:

- **Uma opção para o usuário ativar/desativar o envio do lembrete;**
- **Um campo para o usuário definir a data para envio do lembrete;**
- **Um campo para o usuário definir a hora de envio do lembrete.**

Em um simples rascunho definimos a nova tela de cadastro do aplicativo, conforme mostra a **Figura 17.1**.

Figura 17.1. Rascunho da nova tela de cadastro do aplicativo *Devolva.me*.

17.3 Implementando as novas funcionalidades

Agora que temos todas as alterações do projeto já especificadas, vamos implementá-las.

17.3.1 Alterando a classe *ObjetoEmprestado*

Como primeiro passo, iremos alterar a classe ***ObjetoEmprestado***, adicionando a ela dois atributos:

1. Um atributo do tipo ***boolean***, que será definido para *true* se o lembrete for ativado pelo usuário e *false* se o mesmo for desativado;

2. Um atributo do tipo ***Calendar***, responsáveis por armazenar a data/hora de disparo do Alarme com o lembrete.

Seguindo estes requisitos, adicionaremos os seguintes atributos na classe ***ObjetoEmprestado***:

```
. . .
private boolean lembreteAtivo;
private Calendar dataLembrete;
. . .
```

Devemos também adicionar seus *setters* e *getters*:

```
. . .
public boolean isLembreteAtivo() {
        return lembreteAtivo;
}

public void setLembreteAtivo(boolean lembreteAtivo) {
        this.lembreteAtivo = lembreteAtivo;
}

public Calendar getDataLembrete() {
```

```
        return dataLembrete;
    }

    public void setDataLembrete(Calendar dataLembrete) {
        this.dataLembrete = dataLembrete;
    }
    . . .
```

17.3.2 Alterando a estrutura do banco de dados

A próxima alteração a ser feita é definir a nova estrutura do banco de dados, adicionando dois novos campos:

1. O campo ***lembrete_ativo***, que irá armazenar o valor 0 (zero) caso o lembrete esteja desativado, e o valor 1 caso o lembrete seja ativado;

2. O campo ***data_lembrete***, que irá armazenar a data e a hora que o lembrete (Alarme) deverá ser disparado.

O diagrama da **Figura 17.2** mostra como ficará a nossa nova estrutura de dados.

objeto_emprestado	
_id	INTEGER
objeto	TEXT
contato_id	INTEGER
data_emprestimo	INTEGER
foto	BLOB
lembrete_ativo	INTEGER
data_lembrete	INTEGER

Figura 17.2. Nova estrutura de dados da tabela "*objeto_emprestado*" do projeto *Devolva.me*.

Nota

No banco de dados SQLite as Datas podem ser armazenadas em um campo **INTEGER** ou **TEXT**. Seguindo a documentação, o mais aconselhável é utilizarmos um campo INTEGER para o armazenamento de datas, armazenando-as em milissegundos.

Para definir a nova estrutura do banco de dados, devemos alterar o método ***onCreate()*** da classe ***DBHelper***, deixando-o com a seguinte implementação:

```
    /**
     * Cria a tabela no banco de dados, caso ela nao exista.
     */
    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                + ",objeto TEXT NOT NULL"
                + ",contato_id INTEGER NOT NULL"
                + ",data_emprestimo INTEGER NOT NULL"
                + ",foto BLOB"
                + ",lembrete_ativo INTEGER NOT NULL DEFAULT 0"
                + ",data_lembrete INTEGER"
                + ");";
        db.execSQL(sql);
    }
```

Além disso, para que o Android execute o método que irá atualizar a estrutura de dados, devemos incrementar em 1 unidade o valor da variável ***VERSAO_DO_BANCO***, deixando-a com o seguinte valor:

```
. . .
private static final int VERSAO_DO_BANCO = 4;
. . .
```

17.3.3 Alterando o DAO

O próximo passo é alterar o DAO para suportar as novas informações a serem persistidas e consultadas no banco de dados.

Devemos então alterar o método ***adiciona()***, da classe ***ObjetoEmprestadoDAO***, deixando-o com a seguinte implementação:

```
. . .
/**
 * Adiciona objeto no banco de dados.
 */
public void adiciona(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // persistidos no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("data_emprestimo", System.currentTimeMillis());
        values.put("contato_id", objeto.getContato().getId());
        values.put("foto", objeto.getFoto());
        values.put("lembrete_ativo", objeto.isLembreteAtivo());
        values.put("data_lembrete", objeto.getDataLembrete().getTimeInMillis());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Insere o registro no banco de dados
        long id = db.insert("objeto_emprestado", null, values);
        objeto.setId(id);

        // Encerra a conexão com o banco de dados
        db.close();
}
. . .
```

O método ***atualiza()*** também deve ser alterado, ficando com a seguinte implementação:

```
. . .
/**
 * Altera o registro no banco de dados.
 */
public void atualiza(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // atualizados no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("contato_id", objeto.getContato().getId());
        values.put("foto", objeto.getFoto());
        values.put("lembrete_ativo", objeto.isLembreteAtivo());
        values.put("data_lembrete", objeto.getDataLembrete().getTimeInMillis());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Atualiza o registro no banco de dados
        db.update("objeto_emprestado", values, "_id=?", new String[] { objeto
                        .getId().toString() });

        // Encerra a conexão com o banco de dados
```

```
            db.close();
    }
    . . .
```

Para finalizar as alterações no DAO, modificaremos também o método ***listaTodos()***, deixando-o com a seguinte implementação:

```
    . . .
    /**
     * Lista todos os registros da tabela “objeto_emprestado”
     */
    public List<ObjetoEmprestado> listaTodos() {

            // Cria um List guardar os objetos consultados no banco de dados
            List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

            // Instancia uma nova conexão com o banco de dados em modo leitura
            SQLiteDatabase db = dbHelper.getReadableDatabase();

            // Executa a consulta no banco de dados
            Cursor c = db.query("objeto_emprestado", null, null, null, null, null,
                            "objeto ASC");

            /**
             * Percorre o Cursor, injetando os dados consultados em um objeto do
             * tipo ObjetoEmprestado e adicionando-os na List
             */
            try {
                    while (c.moveToNext()) {
                            ObjetoEmprestado objeto = new ObjetoEmprestado();
                            objeto.setId(c.getLong(c.getColumnIndex("_id")));
                            objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));

                            int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                            Contato contato = Contatos.getContato(contatoID, context);
                            objeto.setContato(contato);

                            objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));

                            boolean lembreteAtivo = c.getInt(c.getColumnIndex("lembrete_ativo")) == 1
                                    ? true : false;
                            objeto.setLembreteAtivo(lembreteAtivo);
                            long dataLembrete = c.getLong((c.getColumnIndex("data_lembrete")));
                            Calendar cal = Calendar.getInstance();
                            cal.setTimeInMillis(dataLembrete);
                            objeto.setDataLembrete(cal);

                            objetos.add(objeto);
                    }

            } finally {
                    // Encerra o Cursor
                    c.close();
            }

            // Encerra a conexão com o banco de dados
            db.close();

            // Retorna uma lista com os objetos consultados
            return objetos;
    }
    . . .
```

17.3.4 Alterando a tela de cadastro

Devemos também alterar a tela de cadastro, seguindo a especificação. De acordo com a especificação, devemos adicionar 3 novos componentes na tela: um CheckBox, um DatePicker e um TimePicker, responsáveis por ativar/desativar e definir a data e hora do alarme.

Como iremos colocar mais componentes e nossa tela irá crescer verticalmente, precisamos de um *View Group* que suporte a rolagem vertical através de uma barra de rolagem. Para isto, devemos alterar o *Element Root* (View principal) para um *ScrollView*.

Vamos então deixar o arquivo de Layout da tela de cadastro, chamado ***activity_cadastra_objeto_emprestado.xml***, com o seguinte conteúdo:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#ffffff"
        android:orientation="vertical"
        android:paddingBottom="15.0dip"
        android:paddingLeft="15.0dip"
        android:paddingRight="15.0dip"
        android:paddingTop="15.0dip" >

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:gravity="center" >

            <ImageView
                android:id="@+id/foto_objeto"
                android:layout_width="86.0dip"
                android:layout_height="86.0dip"
                android:padding="0dip"
                android:src="@android:drawable/ic_menu_camera" />
        </LinearLayout>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="15.0dip"
            android:text="Objeto:" />

        <EditText
            android:id="@+id/cadastro_objeto_campo_objeto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Informe o nome do objeto"
            android:inputType="textPersonName" >

            <requestFocus />
        </EditText>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Emprestado para:" />
```

```
<Button
    android:id="@+id/botao_selecionar_contato"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Selecionar na lista de Contatos" />

<TextView
    android:id="@+id/cadastro_objeto_campo_pessoa"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:textSize="15.0sp"
    android:visibility="gone" />

<CheckBox
    android:id="@+id/cadastro_objeto_campo_lembrete_ativo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ativar lembrete?"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/cadastro_objeto_label_lembrete_data_hora"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:paddingTop="15.0dip"
    android:text="Data/Hora da notificação"
    android:visibility="gone" />

<TextView
    android:id="@+id/cadastro_objeto_label_lembrete_data"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="15.0dip"
    android:text="Data:"
    android:visibility="gone" />

<DatePicker
    android:id="@+id/cadastra_objeto_campo_lembrete_data"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:calendarViewShown="false"
    android:visibility="gone" />

<TextView
    android:id="@+id/cadastro_objeto_label_lembrete_hora"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="15.0dip"
    android:text="Hora:"
    android:visibility="gone" />

<TimePicker
    android:id="@+id/cadastra_objeto_campo_lembrete_hora"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:visibility="gone" />

<LinearLayout
    android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal"
            android:paddingTop="15.0dip" >

            <Button
               android:id="@+id/cadastra_objeto_botao_salvar"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="Salvar" />

            <Button
               android:id="@+id/cadastra_objeto_botao_cancelar"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="Cancelar" />
         </LinearLayout>
      </LinearLayout>

   </ScrollView>
```

17.3.5 Alterando a Activity de cadastro

O próximo passo é alterar a Activity ***CadastraObjetoEmprestadoActivity*** para referenciar os novos campos e salvar suas informações.

Para referenciar os novos campos, vamos criar na classe ***CadastraObjetoEmprestadoActivity*** as seguintes variáveis de instância:

```
. . .
private CheckBox campoLembreteAtivo;
private DatePicker campoLembreteData;
private TimePicker campoLembreteHora;
private TextView labelLembreteDataHora;
private TextView labelLembreteData;
private TextView labelLembreteHora;
. . .
```

Devemos agora ligar estas variáveis aos componentes definidos no arquivo de Layout. Para isto, no método ***onCreate()***, abaixo da seguinte linha:

```
campoFotoObjeto = (ImageView) findViewById(R.id.foto_objeto);
```

adicionaremos o seguinte trecho de código:

```
. . .
campoLembreteAtivo = (CheckBox) findViewById(R.id.cadastro_objeto_campo_lembrete_ativo);
campoLembreteData = (DatePicker) findViewById(R.id.cadastra_objeto_campo_lembrete_data);
campoLembreteHora = (TimePicker) findViewById(R.id.cadastra_objeto_campo_lembrete_hora);
labelLembreteDataHora = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_data_hora);
labelLembreteData = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_data);
labelLembreteHora = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_hora);
. . .
```

De acordo com a especificação, devemos ativar/desativar os componentes da tela responsáveis por definir as informações do lembrete caso o usuário tenha ativado/desativado este recurso. Para facilitar esta ação, criaremos um novo método, chamado ***habilitaDesabilitaCamposLembrete()***, na classe ***CadastraObjetoEmprestadoActivity***, com a seguinte implementação:

```
/**
 * Método responsável por habilitar/desabilitar os componentes da tela
 * para definição do lembrete, caso o usuário os tenha ativado/desativado.
 * @param lembreteAtivo
 */
private void habilitaDesabilitaCamposLembrete(boolean lembreteAtivo) {
        // Caso o usuário tenha ativado o lembrete, habilita os campos
        if (lembreteAtivo) {
                campoLembreteData.setVisibility(View.VISIBLE);
                campoLembreteHora.setVisibility(View.VISIBLE);
                labelLembreteDataHora.setVisibility(View.VISIBLE);
                labelLembreteData.setVisibility(View.VISIBLE);
                labelLembreteHora.setVisibility(View.VISIBLE);
        // Caso o usuário tenha desativado o lembrete, desabilita os campos
        } else {
                campoLembreteData.setVisibility(View.GONE);
                campoLembreteHora.setVisibility(View.GONE);
                labelLembreteDataHora.setVisibility(View.GONE);
                labelLembreteData.setVisibility(View.GONE);
                labelLembreteHora.setVisibility(View.GONE);
        }
}
```

Devemos também preparar a edição do registro para que suporte os novos componentes adicionados. Para isto, no método ***onCreate()*** da classe ***CadastraObjetoEmprestadoActivity***, devemos localizar o seguinte bloco ***if/else***:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();

} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
. . .
```

e inserir, dentro do bloco ***else*** acima, o seguinte trecho de código:

```
. . .
/*
 *  Habilita/desabilita o lembrete de acordo com a informação do
 *  registro salvo no banco de dados.
 */
campoLembreteAtivo.setChecked(objetoEmprestado.isLembreteAtivo());
/*
 *  Preenche o componente de data do lembrete com a data do registro
 *  salvo no banco de dados.
 */
Calendar cal = objetoEmprestado.getDataLembrete();
campoLembreteData.updateDate(cal.get(Calendar.YEAR),
                cal.get(Calendar.MONTH), cal.get(Calendar.DAY_OF_MONTH));

/*
 *  Preenche o componente de hora do lembrete com a hora do registro
 *  salvo no banco de dados.
 */
campoLembreteHora.setCurrentHour(cal.get(Calendar.HOUR_OF_DAY));
campoLembreteHora.setCurrentMinute(cal.get(Calendar.MINUTE));

/*
 *  Habilita/desabilita os componentes de definição do lembrete de
 *  acordo com as informações do registro salvo no banco de dados.
 */
```

```
habilitaDesabilitaCamposLembrete(objetoEmprestado.isLembreteAtivo());
. . .
```

Quando o usuário ativar o lembrete, no campo CheckBox da tela de cadastro, devemos habilitar os componentes responsáveis por definir as informações do lembrete. Para isto, ao final do método ***onCreate()***, da classe ***CadastraObjetoEmprestadoActivity***, devemos adicionar o seguinte trecho de código:

```
. . .
campoLembreteAtivo.setOnCheckedChangeListener(new OnCheckedChangeListener() {

        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                habilitaDesabilitaCamposLembrete(isChecked);
                objetoEmprestado.setLembreteAtivo(isChecked);
        }
});
. . .
```

Nota

O método ***setOnCheckedChangeListener()*** é usado em uma View CheckBox para definir o Listener que será executado quando este componente é marcado/desmarcado. Usamos neste método o Listener ***OnCheckedChangeListener***, implementando em seu método ***onCheckedChanged()*** a ação a ser executada.

Devemos também guardar no objeto do tipo "***ObjetoEmprestado***" a data do lembrete a ser disparado, para que esta informação possa ser armazenada no banco de dados. Para isto, ainda na classe ***CadastraObjetoEmprestadoActivity***, no evento do botão ***Salvar*** (em seu método ***onClick()***), abaixo da seguinte linha:

```
objetoEmprestado.setObjeto(campoObjeto.getText().toString());
```

devemos adicionar o seguinte trecho de código:

```
// Define no objeto "ObjetoEmprestado" a data/horário do lembrete
Calendar cal = Calendar.getInstance();
cal.set(campoLembreteData.getYear(), campoLembreteData.getMonth(), campoLembreteData.getDayOfMonth(),
        campoLembreteHora.getCurrentHour(), campoLembreteHora.getCurrentMinute());
objetoEmprestado.setDataLembrete(cal);
```

17.3.6 Implementando uma classe utilitária para disparar uma Notificação

Para facilitar nosso trabalho, vamos desenvolver uma classe que será utilizada para mostrar uma Notificação ao usuário. Usaremos esta classe para mostrar uma Notificação quando a mensagem SMS com o lembrete tiver sido enviada pelo aplicativo.

No pacote ***br.com.hachitecnologia.devolvame.util*** vamos criar uma nova classe, chamada ***Notificacao***, com a seguinte implementação:

```
public class Notificacao {

        public static void mostraNotificacao(String titulo, String mensagem, Context context) {

                // Tempo em que a Notificação será disparada
                long tempoDefinido = System.currentTimeMillis();

                // Objeto Notification
                Notification notification = new Notification(R.drawable.ic_launcher,
                                titulo, tempoDefinido);
```

```
                    // Intent que será disparada quando o usuário clicar sobre a Notificação
                    Intent intent = new Intent(context,
                            TelaInicialActivity.class);
                    PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
                            intent, 0);

                    // Configurando os dados da Notificação
                    notification.setLatestEventInfo(context, titulo, mensagem, pendingIntent);

                    // Oculta a notificação após o usuário clicar sobre ela
                    notification.flags |= Notification.FLAG_AUTO_CANCEL;
                    // Define o som de toque, alerta vibratório e LED padrões do dispositivo
                    notification.defaults = Notification.DEFAULT_ALL;

                    // Agendando a Notificação
                    NotificationManager notificationManager = (NotificationManager) context
                            .getSystemService(Context.NOTIFICATION_SERVICE);
                    notificationManager.notify(0, notification);
            }

    }
```

Em nossa classe implementamos o método ***mostraNotificacao()*** de forma estática, para não precisarmos criar uma instância da classe sempre que precisarmos mostrar uma Notificação. Este método recebe como parâmetro o título e a mensagem a serem mostrados na Notificação, além do objeto *Context*.

Como usaremos o recurso de Vibração do dispositivo quando a Notificação for disparada, devemos adicionar a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.VIBRATE" />
. . .
```

17.3.7 Implementando uma classe utilitária para o envio de SMS

De acordo com a especificação, precisaremos de um recurso para enviar um SMS automaticamente, que usaremos para enviar o lembrete para as pessoas que pegaram um objeto emprestado. Para isto, criaremos uma classe utilitária, responsável pelo envio de SMS.

No pacote ***br.com.hachitecnologia.devolvame.util*** vamos criar uma nova classe, chamada ***Telefonia***, com a seguinte implementação:

```
public class Telefonia {

        /**
         * Método responsável por enviar mensagens SMS.
         * @param numeroTelefone
         * @param mensagem
         */
        public static void enviaSMS(String numeroTelefone, String mensagem) {
                SmsManager sms = SmsManager.getDefault();
                // Divide a mensagem em partes
                ArrayList<String> mensagemArray = sms.divideMessage(mensagem);
                // Envia cada parte da mensagem em um SMS
                sms.sendMultipartTextMessage(numeroTelefone, null, mensagemArray, null, null);
        }

}
```

Nesta classe implementamos o método ***enviaSMS()***, que recebe como parâmetro o número do telefone para o qual o SMS será enviado e a mensagem que será enviada.

Para permitir o envio do SMS, devemos adicionar ao arquivo ***AndroidManifest.xml*** a seguinte permissão:

```
. . .
<uses-permission android:name="android.permission.SEND_SMS" />
. . .
```

17.3.8 Implementando o Service responsável pelo envio do SMS e disparo da Notificação

O envio do SMS e disparo da Notificação serão executados por um Service, que realizará estas tarefas em segundo plano. Para isto, no pacote ***br.com.hachitecnologia.devolvame.service*** criaremos uma nova classe, chamada ***EnviaSMSLembreteService***, com a seguinte implementação:

```
public class EnviaSMSLembreteService extends Service {

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {
		/*
		 * Recebe como Extra da Intent os parâmetros necessários
		 * para o envio do SMS e disparo da Notificação.
		 */
		String textoMensagemSMS = intent.getStringExtra("textoMensagemSMS");
		String numeroTelefone = intent.getStringExtra("numeroTelefone");
		String textoNotificacao = intent.getStringExtra("textoNotificacao");
		String tituloNotificacao = intent.getStringExtra("tituloNotificacao");

		// Envia o SMS
		Telefonia.enviaSMS(numeroTelefone, textoMensagemSMS);
		// Dispara a Notificação
		Notificacao.mostraNotificacao(tituloNotificacao, textoNotificacao, getApplicationContext());

		return super.onStartCommand(intent, flags, startId);
	}

	@Override
	public IBinder onBind(Intent arg0) {
		return null;
	}

}
```

Além de implementar o Service, devemos configurá-lo e definir sua Action. Para isto, adicionaremos o seguinte trecho ao arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.EnviaSMSLembreteService">
    <intent-filter>
        <action android:name="br.com.hachitecnologia.devolvame.action.ENVIA_SMS_LEMBRETE" />
    </intent-filter>
</service>
. . .
```

17.3.9 Implementando uma classe utilitária para agendamento do Alarme

Vamos agora definir a classe que irá agendar o Alarme. Esta classe deverá chamar o Service que implementamos anteriormente para enviar o SMS e disparar a Notificação.

Criaremos então, no pacote ***br.com.hachitecnologia.devolvame.util***, uma nova classe, chamada ***Alarme***, com a seguinte implementação:

```
public class Alarme {

	/**
	 * Método responsável por agendar um Alarme.
	 * @param objeto
	 * @param context
	 */
	public static void defineAlarme(ObjetoEmprestado objeto, Context context) {
		AlarmManager alarmManager = (AlarmManager)
			context.getSystemService(Context.ALARM_SERVICE);
		alarmManager.set(AlarmManager.RTC_WAKEUP, objeto.getDataLembrete().getTimeInMillis(),
			getPendingIntent(objeto, context));
	}

	/**
	 * Método responsável por definir a PendingIntent que será usada
	 * para criar e cancelar um Alarme. Criamos um método com essa
	 * responsabilidade para reaproveitar o código na hora de executar
	 * estas duas tarefas.
	 * @param objeto
	 * @param context
	 * @return
	 */
	private static PendingIntent getPendingIntent(ObjetoEmprestado objeto, Context context) {
		/*
		 *  Texto da mensagem que será enviada como lembrete, via SMS. Esta mensagem será
		 *  enviada como Extra para o Service encarregado de enviar o SMS.
		 */
		String textoMensagemSMS = "Você pegou emprestado o meu objeto \"%s\" e ainda não " +
				"o devolveu. Por favor, devolva-me o quanto antes. Obrigado. ";
		textoMensagemSMS = String.format(textoMensagemSMS, objeto.getObjeto());

		// Texto que será mostrado na notificação
		String textoNotificacao = "Lembrete do objeto \"%s\" enviado para %s.";
		textoNotificacao = String.format(textoNotificacao, objeto.getObjeto(),
			objeto.getContato().getNome());
		// Título da Notificação
		String tituloNotificacao = "Lembrete enviado para %s";
		tituloNotificacao = String.format(tituloNotificacao, objeto.getContato().getNome());

		/*
		 *  Define a Intent que irá invocar o Service responsável pelo envio do
		 *  SMS e disparo da Notificação.
		 */
		Intent intent = new Intent("br.com.hachitecnologia.devolvame.action.ENVIA_SMS_LEMBRETE");
		/*
		 *  Injeta na Intent os parâmetros necessários para o envio do SMS e disparo
		 *  da Notificação.
		 */
		intent.putExtra("textoMensagemSMS", textoMensagemSMS);
		intent.putExtra("numeroTelefone", objeto.getContato().getTelefone());
		intent.putExtra("textoNotificacao", textoNotificacao);
		intent.putExtra("tituloNotificacao", tituloNotificacao);

		PendingIntent pendingIntent = PendingIntent.getService(context,
			objeto.getId().intValue(), intent, 0);

		return pendingIntent;
	}

	/**
```

```
		 * Método responsável por cancelar um Alarme, se necessário.
		 * @param objeto
		 * @param context
		 */
		public static void cancelaAlarme(ObjetoEmprestado objeto, Context context) {
			AlarmManager alarmManager = (AlarmManager)
				context.getSystemService(Context.ALARM_SERVICE);
			alarmManager.cancel(getPendingIntent(objeto, context));
		}

	}
```

17.3.10 Alterando a Activity de cadastro para ativar o agendamento do Alarme

Por fim, para que o alarme seja agendado ao cadastrar um novo objeto emprestado, devemos ativá-lo no momento do cadastro. Para isto, na Activity ***CadastraObjetoEmprestadoActivity***, dentro do método ***onClick()*** do botão Salvar *(referenciado pela variável **botaoSalvar**)*, antes da seguinte linha:

```
. . .
// Depois de salvar, vai para a Lista dos objetos emprestados
startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
. . .
```

adicionaremos o seguinte trecho de código:

```
. . .
// Define o Alarme
if (objetoEmprestado.isLembreteAtivo())
		// Agenda o Alarme, caso o usuário tenha habilitado o lembrete
		Alarme.defineAlarme(objetoEmprestado, getApplicationContext());
else
		// Cancela o Alarme, caso o usuário tenha desabilitado o lembrete
		Alarme.cancelaAlarme(objetoEmprestado, getApplicationContext());
. . .
```

17.3.11 Reativando os Alarmes após o boot do dispositivo

Até aqui, nosso código já está agendando o Alarme que irá disparar o envio do lembrete. Porém se desligarmos ou reiniciarmos o dispositivo, o agendamento do Alarme é perdido. Para resolver este problema, podemos usar um *Broadcast Receiver*, que será invocado após o boot do dispositivo, e este poderá chamar um *Service* que consultará no banco de dados todos os registros que precisam ter o Alarme agendado novamente.

Para implementar esta funcionalidade, devemos primeiro criar um método em nosso DAO que irá consultar apenas os registros que precisam ter os Alarmes agendados, ou seja, os registros que têm o valor do campo "**lembrete_ativo**" definido para 1 (true). Iremos então implementar um novo método na classe ***ObjetoEmprestadoDAO***, chamado ***consultaObjetosComAlarmeASerDisparado()***, com a seguinte implementação:

```
. . .
public List<ObjetoEmprestado> consultaObjetosComAlarmeASerDisparado() {
		// Cria um List guardar os objetos consultados no banco de dados
		List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

		// Instancia uma nova conexão com o banco de dados em modo leitura
		SQLiteDatabase db = dbHelper.getReadableDatabase();

		// Executa a consulta no banco de dados
		String sql = "select * from objeto_emprestado " +
				"where lembrete_ativo = 1 and data_lembrete > ?";
		Long dataAtualEmMilissegundos = Calendar.getInstance().getTimeInMillis();
```

```
        Cursor c = db.rawQuery(sql, new String[]{dataAtualEmMilissegundos.toString()});

        /**
         * Percorre o Cursor, injetando os dados consultados em um objeto do
         * tipo ObjetoEmprestado e adicionando-os na List
         */
        try {
            while (c.moveToNext()) {
                ObjetoEmprestado objeto = new ObjetoEmprestado();
                objeto.setId(c.getLong(c.getColumnIndex("_id")));
                objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));

                int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                Contato contato = Contatos.getContato(contatoID, context);
                contato.setId(contatoID);
                objeto.setContato(contato);

                objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));
                boolean lembreteAtivo = c.getInt(c.getColumnIndex("lembrete_ativo")) == 1
                        ? true : false;
                objeto.setLembreteAtivo(lembreteAtivo);
                long dataLembrete = c.getLong((c.getColumnIndex("data_lembrete")));
                Calendar cal = Calendar.getInstance();
                cal.setTimeInMillis(dataLembrete);
                objeto.setDataLembrete(cal);

                objetos.add(objeto);
            }

        } finally {
            // Encerra o Cursor
            c.close();
        }

        // Encerra a conexão com o banco de dados
        db.close();

        // Retorna uma lista com os objetos consultados
        return objetos;
    }
    . . .
```

Nota

No código acima, ao invés de utilizar o método ***query()***, da classe ***SQLiteDatabase***, para consulta, utilizamos o método ***rawQuery()***. O método ***rawquery()*** nos permite escrever uma SQL manualmente e executá-la.

O segundo passo é implementar o *Service* que realizará a consulta no banco de dados, procurando pelos registros que precisam ter o Alarme reagendado, e ativar seus Alarmes. Para isto, criaremos um novo service, chamado ***AtivaAlarmesService***, no pacote ***br.com.hachitecnologia.devolvame.service***, com a seguinte implementação:

```
public class AtivaAlarmesService extends Service {

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());
        // Consulta todos os registros que precisam ter o Alarme ativado
        List<ObjetoEmprestado> objetos = dao.consultaObjetosComAlarmeASerDisparado();

        // Ativa o Alarme para cada registro retornado na consulta
        for (ObjetoEmprestado objeto: objetos) {
```

```
                    Alarme.defineAlarme(objeto, getApplicationContext());
                }

                return super.onStartCommand(intent, flags, startId);
            }

            @Override
            public IBinder onBind(Intent arg0) {
                // TODO Auto-generated method stub
                return null;
            }

        }
```

O próximo passo é implementar um *Broadcast Receiver* que será chamado após o boot completo do sistema. Este *Broadcast Receiver* irá invocar o *Service* implementado anteriormente para reagendar os Alarmes após o boot. Sendo assim, no pacote ***br.com.hachitecnologia.devolvame.receiver***, criaremos uma nova classe, chamada ***AtivaAlarmesNoBoot***, com a seguinte implementação:

```
public class AtivaAlarmesNoBoot extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // Define uma nova Intent, que irá chamar o Service "AtivaAlarmesService"
        Intent i = new Intent("br.com.hachitecnologia.devolvame.action.ATIVA_ALARMES");
        // Executa o Service
        context.startService(i);
    }
}
```

Precisamos também configurar o *Broadcast Receiver*, adicionando o seguinte trecho ao arquivo ***AndroidManifest.xml***:

```
. . .
<receiver android:name=".receiver.AtivaAlarmesNoBoot">
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
. . .
```

Por fim, para usar a Action ***android.intent.action.BOOT_COMPLETED***, precisamos definir a seguinte permissão no arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
. . .
```

Pronto! Nossa implementação está completa e seguindo a definição das novas especificações do projeto *Devolva.me*.

17.4 Exercício

Neste exercício iremos implementar as novas especificações do projeto *Devolva.me*, de acordo com a solicitação do nosso cliente José, definidas o início deste capítulo.

1. No projeto *Devolva.me*, altere a classe modelo ***ObjetoEmprestado***, adicionando os seguintes atributos:

```
. . .
private boolean lembreteAtivo;
```

```
private Calendar dataLembrete;
. . .
```

Adicione também seus *setters* e *getters*:

```
. . .
public boolean isLembreteAtivo() {
        return lembreteAtivo;
}

public void setLembreteAtivo(boolean lembreteAtivo) {
        this.lembreteAtivo = lembreteAtivo;
}

public Calendar getDataLembrete() {
        return dataLembrete;
}

public void setDataLembrete(Calendar dataLembrete) {
        this.dataLembrete = dataLembrete;
}
. . .
```

2. Na classe ***DBHelper***, altere o método ***onCreate()*** deixando-o com a seguinte implementação:

```
/**
 * Cria a tabela no banco de dados, caso ela nao exista.
 */
@Override
public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE objeto_emprestado ("
                        + "_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT"
                        + ",objeto TEXT NOT NULL"
                        + ",contato_id INTEGER NOT NULL"
                        + ",data_emprestimo INTEGER NOT NULL"
                        + ",foto BLOB"
                        + ",lembrete_ativo INTEGER NOT NULL DEFAULT 0"
                        + ",data_lembrete INTEGER"
                        + ");";
        db.execSQL(sql);
}
```

3. Ainda na classe ***DBHelper***, incremente em 1 unidade o valor da variável ***VERSAO_DO_BANCO***, deixando-a com o seguinte valor:

```
. . .
private static final int VERSAO_DO_BANCO = 4;
. . .
```

4. Na classe ***ObjetoEmprestadoDAO***, altere o método ***adiciona()***, deixando-o com a seguinte implementação:

```
/**
 * Adiciona objeto no banco de dados.
 */
public void adiciona(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // persistidos no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
```

```
        values.put("data_emprestimo", System.currentTimeMillis());
        values.put("contato_id", objeto.getContato().getId());
        values.put("foto", objeto.getFoto());
        values.put("lembrete_ativo", objeto.isLembreteAtivo());
        values.put("data_lembrete", objeto.getDataLembrete().getTimeInMillis());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Insere o registro no banco de dados
        long id = db.insert("objeto_emprestado", null, values);
        objeto.setId(id);

        // Encerra a conexão com o banco de dados
        db.close();
}
```

5. Ainda na classe ***ObjetoEmprestadoDAO***, altere o método ***atualiza()***, deixando-o com a seguinte implementação:

```
/**
 * Altera o registro no banco de dados.
 */
public void atualiza(ObjetoEmprestado objeto) {
        // Encapsula no objeto do tipo ContentValues os valores a serem
        // atualizados no banco de dados
        ContentValues values = new ContentValues();
        values.put("objeto", objeto.getObjeto());
        values.put("contato_id", objeto.getContato().getId());
        values.put("foto", objeto.getFoto());
        values.put("lembrete_ativo", objeto.isLembreteAtivo());
        values.put("data_lembrete", objeto.getDataLembrete().getTimeInMillis());

        // Instancia uma conexão com o banco de dados, em modo de gravação
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // Atualiza o registro no banco de dados
        db.update("objeto_emprestado", values, "_id=?", new String[] { objeto
                        .getId().toString() });

        // Encerra a conexão com o banco de dados
        db.close();
}
```

6. Novamente na classe ***ObjetoEmprestadoDAO***, altere o método ***listaTodos()***, deixando-o com a seguinte implementação:

```
/**
 * Lista todos os registros da tabela “objeto_emprestado”
 */
public List<ObjetoEmprestado> listaTodos() {

        // Cria um List guardar os objetos consultados no banco de dados
        List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

        // Instancia uma nova conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        // Executa a consulta no banco de dados
        Cursor c = db.query("objeto_emprestado", null, null, null, null, null,
                        "objeto ASC");

        /**
```

```
                * Percorre o Cursor, injetando os dados consultados em um objeto do
                * tipo ObjetoEmprestado e adicionando-os na List
                */
                try {
                        while (c.moveToNext()) {
                                ObjetoEmprestado objeto = new ObjetoEmprestado();
                                objeto.setId(c.getLong(c.getColumnIndex("_id")));
                                objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));

                                int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                                Contato contato = Contatos.getContato(contatoID, context);
                                objeto.setContato(contato);

                                objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));

                                boolean lembreteAtivo = c.getInt(c.getColumnIndex("lembrete_ativo")) == 1
                                                ? true : false;
                                objeto.setLembreteAtivo(lembreteAtivo);
                                long dataLembrete = c.getLong((c.getColumnIndex("data_lembrete")));
                                Calendar cal = Calendar.getInstance();
                                cal.setTimeInMillis(dataLembrete);
                                objeto.setDataLembrete(cal);

                                objetos.add(objeto);
                        }

                } finally {
                        // Encerra o Cursor
                        c.close();
                }

                // Encerra a conexão com o banco de dados
                db.close();

                // Retorna uma lista com os objetos consultados
                return objetos;
        }
```

7. Altere o arquivo de Layout ***activity_cadastra_objeto_emprestado.xml***, deixando-o com o seguinte conteúdo:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#ffffff"
        android:orientation="vertical"
        android:paddingBottom="15.0dip"
        android:paddingLeft="15.0dip"
        android:paddingRight="15.0dip"
        android:paddingTop="15.0dip" >

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:gravity="center" >

            <ImageView
                android:id="@+id/foto_objeto"
                android:layout_width="86.0dip"
```

```
            android:layout_height="86.0dip"
            android:padding="0dip"
            android:src="@android:drawable/ic_menu_camera" />
    </LinearLayout>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15.0dip"
        android:text="Objeto:" />

    <EditText
        android:id="@+id/cadastro_objeto_campo_objeto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Informe o nome do objeto"
        android:inputType="textPersonName" >

        <requestFocus />
    </EditText>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="15.0dip"
        android:text="Emprestado para:" />

    <Button
        android:id="@+id/botao_selecionar_contato"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Selecionar na lista de Contatos" />

    <TextView
        android:id="@+id/cadastro_objeto_campo_pessoa"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="15.0sp"
        android:visibility="gone" />

    <CheckBox
        android:id="@+id/cadastro_objeto_campo_lembrete_ativo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ativar lembrete?"
        android:textAppearance="?android:attr/textAppearanceSmall" />

    <TextView
        android:id="@+id/cadastro_objeto_label_lembrete_data_hora"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:paddingTop="15.0dip"
        android:text="Data/Hora da notificação"
        android:visibility="gone" />

    <TextView
        android:id="@+id/cadastro_objeto_label_lembrete_data"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="15.0dip"
        android:text="Data:"
```

```
            android:visibility="gone" />

        <DatePicker
            android:id="@+id/cadastra_objeto_campo_lembrete_data"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:calendarViewShown="false"
            android:visibility="gone" />

        <TextView
            android:id="@+id/cadastro_objeto_label_lembrete_hora"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingTop="15.0dip"
            android:text="Hora:"
            android:visibility="gone" />

        <TimePicker
            android:id="@+id/cadastra_objeto_campo_lembrete_hora"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:visibility="gone" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal"
            android:paddingTop="15.0dip" >

            <Button
                android:id="@+id/cadastra_objeto_botao_salvar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Salvar" />

            <Button
                android:id="@+id/cadastra_objeto_botao_cancelar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Cancelar" />
        </LinearLayout>
    </LinearLayout>

</ScrollView>
```

8. Na classe ***CadastraObjetoEmprestadoActivity*** adicione as seguintes variáveis de instância:

```
. . .
private CheckBox campoLembreteAtivo;
private DatePicker campoLembreteData;
private TimePicker campoLembreteHora;
private TextView labelLembreteDataHora;
private TextView labelLembreteData;
private TextView labelLembreteHora;
. . .
```

9. No método ***onCrate()*** da classe ***CadastraObjetoEmprestadoActivity***, abaixo da seguinte linha:

```
campoFotoObjeto = (ImageView) findViewById(R.id.foto_objeto);
```

adicione o seguinte trecho de código:

```
. . .
campoLembreteAtivo = (CheckBox) findViewById(R.id.cadastro_objeto_campo_lembrete_ativo);
campoLembreteData = (DatePicker) findViewById(R.id.cadastra_objeto_campo_lembrete_data);
campoLembreteHora = (TimePicker) findViewById(R.id.cadastra_objeto_campo_lembrete_hora);
labelLembreteDataHora = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_data_hora);
labelLembreteData = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_data);
labelLembreteHora = (TextView) findViewById(R.id.cadastro_objeto_label_lembrete_hora);
. . .
```

10. Na classe ***CadastraObjetoEmprestadoActivity*** adicione o seguinte método:

```
/**
 * Método responsável por habilitar/desabilitar os componentes da tela
 * para definição do lembrete, caso o usuário os tenha ativado/desativado.
 * @param lembreteAtivo
 */
private void habilitaDesabilitaCamposLembrete(boolean lembreteAtivo) {
        // Caso o usuário tenha ativado o lembrete, habilita os campos
        if (lembreteAtivo) {
                campoLembreteData.setVisibility(View.VISIBLE);
                campoLembreteHora.setVisibility(View.VISIBLE);
                labelLembreteDataHora.setVisibility(View.VISIBLE);
                labelLembreteData.setVisibility(View.VISIBLE);
                labelLembreteHora.setVisibility(View.VISIBLE);
        // Caso o usuário tenha desativado o lembrete, desabilita os campos
        } else {
                campoLembreteData.setVisibility(View.GONE);
                campoLembreteHora.setVisibility(View.GONE);
                labelLembreteDataHora.setVisibility(View.GONE);
                labelLembreteData.setVisibility(View.GONE);
                labelLembreteHora.setVisibility(View.GONE);
        }
}
```

No método ***onCreate()*** da classe ***CadastraObjetoEmprestadoActivity***, localize o seguinte bloco ***if/else***:

```
. . .
if (objetoEmprestado == null) {
        // Instancia um novo objeto do tipo ObjetoEmprestado
        objetoEmprestado = new ObjetoEmprestado();

} else {
        campoObjeto.setText(objetoEmprestado.getObjeto());
. . .
```

e adicione, dentro do bloco ***else*** acima, o seguinte trecho de código:

```
. . .
/*
 *  Habilita/desabilita o lembrete de acordo com a informação do
 *  registro salvo no banco de dados.
 */
campoLembreteAtivo.setChecked(objetoEmprestado.isLembreteAtivo());
/*
 *  Preenche o componente de data do lembrete com a data do registro
 *  salvo no banco de dados.
 */
Calendar cal = objetoEmprestado.getDataLembrete();
campoLembreteData.updateDate(cal.get(Calendar.YEAR),
```

```
                cal.get(Calendar.MONTH), cal.get(Calendar.DAY_OF_MONTH));

        /*
         * Preenche o componente de hora do lembrete com a hora do registro
         * salvo no banco de dados.
         */
        campoLembreteHora.setCurrentHour(cal.get(Calendar.HOUR_OF_DAY));
        campoLembreteHora.setCurrentMinute(cal.get(Calendar.MINUTE));

        /*
         * Habilita/desabilita os componentes de definição do lembrete de
         * acordo com as informações do registro salvo no banco de dados.
         */
        habilitaDesabilitaCamposLembrete(objetoEmprestado.isLembreteAtivo());
        . . .
```

11. Na classe ***CadastraObjetoEmprestadoActivity***, ao final do método ***onCreate()***, adicione o seguinte trecho de código:

```
        . . .
        campoLembreteAtivo.setOnCheckedChangeListener(new OnCheckedChangeListener() {

                @Override
                public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                        habilitaDesabilitaCamposLembrete(isChecked);
                        objetoEmprestado.setLembreteAtivo(isChecked);
                }
        });
        . . .
```

12. Ainda na classe ***CadastraObjetoEmprestadoActivity***, no evento do botão ***Salvar*** (em seu método ***onClick()***), abaixo da seguinte linha:

```
objetoEmprestado.setObjeto(campoObjeto.getText().toString());
```

adicione o seguinte trecho de código:

```
// Define no objeto "ObjetoEmprestado" a data/horário do lembrete
Calendar cal = Calendar.getInstance();
cal.set(campoLembreteData.getYear(), campoLembreteData.getMonth(), campoLembreteData.getDayOfMonth(),
        campoLembreteHora.getCurrentHour(), campoLembreteHora.getCurrentMinute());
objetoEmprestado.setDataLembrete(cal);
```

13. No pacote ***br.com.hachitecnologia.devolvame.util***, crie uma nova classe, chamada ***Notificacao***, com a seguinte implementação:

```
public class Notificacao {

        public static void mostraNotificacao(String titulo, String mensagem, Context context) {

                // Tempo em que a Notificação será disparada
                long tempoDefinido = System.currentTimeMillis();

                // Objeto Notification
                Notification notification = new Notification(R.drawable.ic_launcher,
                                titulo, tempoDefinido);

                // Intent que será disparada quando o usuário clicar sobre a Notificação
                Intent intent = new Intent(context,
                                TelaInicialActivity.class);
                PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
```

```
                                        intent, 0);

                // Configurando os dados da Notificação
                notification.setLatestEventInfo(context, titulo, mensagem, pendingIntent);

                // Oculta a notificação após o usuário clicar sobre ela
                notification.flags |= Notification.FLAG_AUTO_CANCEL;
                // Define o som de toque, alerta vibratório e LED padrões do dispositivo
                notification.defaults = Notification.DEFAULT_ALL;

                // Agendando a Notificação
                NotificationManager notificationManager = (NotificationManager) context
                                .getSystemService(Context.NOTIFICATION_SERVICE);
                notificationManager.notify(0, notification);
        }

}
```

14. Adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.VIBRATE" />
. . .
```

15. No pacote ***br.com.hachitecnologia.devolvame.service*** crie uma classe, chamada ***EnviaSMSLembreteService***, com a seguinte implementação:

```
public class EnviaSMSLembreteService extends Service {

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
                /*
                 * Recebe como Extra da Intent os parâmetros necessários
                 * para o envio do SMS e disparo da Notificação.
                 */
                String textoMensagemSMS = intent.getStringExtra("textoMensagemSMS");
                String numeroTelefone = intent.getStringExtra("numeroTelefone");
                String textoNotificacao = intent.getStringExtra("textoNotificacao");
                String tituloNotificacao = intent.getStringExtra("tituloNotificacao");

                // Envia o SMS
                Telefonia.enviaSMS(numeroTelefone, textoMensagemSMS);
                // Dispara a Notificação
                Notificacao.mostraNotificacao(tituloNotificacao, textoNotificacao, getApplicationContext());

                return super.onStartCommand(intent, flags, startId);
        }

        @Override
        public IBinder onBind(Intent arg0) {
                return null;
        }

}
```

16. Configure o Service implementado anteriormente, adicionando o seguinte trecho ao arquivo ***AndroidManifest.xml***:

```
. . .
<service android:name=".service.EnviaSMSLembreteService">
    <intent-filter>
```

```
            <action android:name="br.com.hachitecnologia.devolvame.action.ENVIA_SMS_LEMBRETE" />
        </intent-filter>
    </service>
    . . .
```

17. No pacote ***br.com.hachitecnologia.devolvame.util***, crie uma nova classe, chamada ***Alarme***, com a seguinte implementação:

```
public class Alarme {

    /**
     * Método responsável por agendar um Alarme.
     * @param objeto
     * @param context
     */
    public static void defineAlarme(ObjetoEmprestado objeto, Context context) {
        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP, objeto.getDataLembrete().getTimeInMillis(),
                getPendingIntent(objeto, context));
    }

    /**
     * Método responsável por definir a PendingIntent que será usada
     * para criar e cancelar um Alarme. Criamos um método com essa
     * responsabilidade para reaproveitar o código na hora de executar
     * estas duas tarefas.
     * @param objeto
     * @param context
     * @return
     */
    private static PendingIntent getPendingIntent(ObjetoEmprestado objeto, Context context) {
        /*
         *  Texto da mensagem que será enviada como lembrete, via SMS. Esta mensagem será
         *  enviada como Extra para o Service encarregado de enviar o SMS.
         */
        String textoMensagemSMS = "Você pegou emprestado o meu objeto \"%s\" e ainda não " +
                "o devolveu. Por favor, devolva-me o quanto antes. Obrigado. ";
        textoMensagemSMS = String.format(textoMensagemSMS, objeto.getObjeto());

        // Texto que será mostrado na notificação
        String textoNotificacao = "Lembrete do objeto \"%s\" enviado para %s.";
        textoNotificacao = String.format(textoNotificacao, objeto.getObjeto(),
                objeto.getContato().getNome());
        // Título da Notificação
        String tituloNotificacao = "Lembrete enviado para %s";
        tituloNotificacao = String.format(tituloNotificacao, objeto.getContato().getNome());

        /*
         *  Define a Intent que irá invocar o Service responsável pelo envio do
         *  SMS e disparo da Notificação.
         */
        Intent intent = new Intent("br.com.hachitecnologia.devolvame.action.ENVIA_SMS_LEMBRETE");
        /*
         *  Injeta na Intent os parâmetros necessários para o envio do SMS e disparo
         *  da Notificação.
         */
        intent.putExtra("textoMensagemSMS", textoMensagemSMS);
        intent.putExtra("numeroTelefone", objeto.getContato().getTelefone());
        intent.putExtra("textoNotificacao", textoNotificacao);
        intent.putExtra("tituloNotificacao", tituloNotificacao);

        PendingIntent pendingIntent = PendingIntent.getService(context,
```

```
                                        objeto.getId().intValue(), intent, 0);

                        return pendingIntent;
                }

                /**
                 * Método responsável por cancelar um Alarme, se necessário.
                 * @param objeto
                 * @param context
                 */
                public static void cancelaAlarme(ObjetoEmprestado objeto, Context context) {
                        AlarmManager alarmManager = (AlarmManager)
                                context.getSystemService(Context.ALARM_SERVICE);
                        alarmManager.cancel(getPendingIntent(objeto, context));
                }

        }
```

18. Na Activity ***CadastraObjetoEmprestadoActivity***, dentro do método ***onClick()*** do botão Salvar *(referenciado pela variável **botaoSalvar**)*, antes da seguinte linha:

```
. . .
// Depois de salvar, vai para a Lista dos objetos emprestados
startActivity(new Intent(getApplicationContext(), ListaObjetosEmprestadosActivity.class));
. . .
```

adicione o seguinte trecho de código:

```
. . .
// Define o Alarme
if (objetoEmprestado.isLembreteAtivo())
        // Agenda o Alarme, caso o usuário tenha habilitado o lembrete
        Alarme.defineAlarme(objetoEmprestado, getApplicationContext());
else
        // Cancela o Alarme, caso o usuário tenha desabilitado o lembrete
        Alarme.cancelaAlarme(objetoEmprestado, getApplicationContext());
. . .
```

19. Na classe ***ObjetoEmprestadoDAO***, crie um novo método, chamado ***consultaObjetosComAlarmeASerDisparado()***, com a seguinte implementação:

```
public List<ObjetoEmprestado> consultaObjetosComAlarmeASerDisparado() {
        // Cria um List guardar os objetos consultados no banco de dados
        List<ObjetoEmprestado> objetos = new ArrayList<ObjetoEmprestado>();

        // Instancia uma nova conexão com o banco de dados em modo leitura
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        // Executa a consulta no banco de dados
        String sql = "select * from objeto_emprestado " +
                        "where lembrete_ativo = 1 and data_lembrete > ?";
        Long dataAtualEmMilissegundos = Calendar.getInstance().getTimeInMillis();
        Cursor c = db.rawQuery(sql, new String[]{dataAtualEmMilissegundos.toString()});

        /**
         * Percorre o Cursor, injetando os dados consultados em um objeto do
         * tipo ObjetoEmprestado e adicionando-os na List
         */
        try {
                while (c.moveToNext()) {
                        ObjetoEmprestado objeto = new ObjetoEmprestado();
                        objeto.setId(c.getLong(c.getColumnIndex("_id")));
```

```
                        objeto.setObjeto(c.getString(c.getColumnIndex("objeto")));

                        int contatoID = c.getInt(c.getColumnIndex("contato_id"));
                        Contato contato = Contatos.getContato(contatoID, context);
                        contato.setId(contatoID);
                        objeto.setContato(contato);

                        objeto.setFoto(c.getBlob(c.getColumnIndex("foto")));
                        boolean lembreteAtivo = c.getInt(c.getColumnIndex("lembrete_ativo")) == 1
                                ? true : false;
                        objeto.setLembreteAtivo(lembreteAtivo);
                        long dataLembrete = c.getLong((c.getColumnIndex("data_lembrete")));
                        Calendar cal = Calendar.getInstance();
                        cal.setTimeInMillis(dataLembrete);
                        objeto.setDataLembrete(cal);

                        objetos.add(objeto);
                }

        } finally {
                // Encerra o Cursor
                c.close();
        }

        // Encerra a conexão com o banco de dados
        db.close();

        // Retorna uma lista com os objetos consultados
        return objetos;
}
```

20. No pacote ***br.com.hachitecnologia.devolvame.service*** crie um novo service, chamado ***AtivaAlarmesService***, com a seguinte implementação:

```
public class AtivaAlarmesService extends Service {

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {

                ObjetoEmprestadoDAO dao = new ObjetoEmprestadoDAO(getApplicationContext());
                // Consulta todos os registros que precisam ter o Alarme ativado
                List<ObjetoEmprestado> objetos = dao.consultaObjetosComAlarmeASerDisparado();

                // Ativa o Alarme para cada registro retornado na consulta
                for (ObjetoEmprestado objeto: objetos) {
                        Alarme.defineAlarme(objeto, getApplicationContext());
                }

                return super.onStartCommand(intent, flags, startId);
        }

        @Override
        public IBinder onBind(Intent arg0) {
                // TODO Auto-generated method stub
                return null;
        }

}
```

21. No pacote ***br.com.hachitecnologia.devolvame.receiver***, crie uma nova classe, chamada ***AtivaAlarmesNoBoot***, com a seguinte implementação:

```
public class AtivaAlarmesNoBoot extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                // Define uma nova Intent, que irá chamar o Service "AtivaAlarmesService"
                Intent i = new Intent("br.com.hachitecnologia.devolvame.action.ATIVA_ALARMES");
                // Executa o Service
                context.startService(i);
        }
}
```

22. Configure o *Broadcast Receiver* implementado anteriormente, adicionando o seguinte trecho ao arquivo ***AndroidManifest.xml***:

```
. . .
<receiver android:name=".receiver.AtivaAlarmesNoBoot">
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
. . .
```

23. Adicione a seguinte permissão ao arquivo ***AndroidManifest.xml***:

```
. . .
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
. . .
```

24. Execute o aplicativo no emulador do Android e faça o teste.

18 Apêndice - Preferências

O Android possui um sistema de armazenamento de preferências do usuário e este mecanismo é chamado de Preferências. O recurso de Preferências basicamente é utilizado para guardar as configurações de um aplicativo e é exatamente o mesmo mecanismo que o próprio Android utiliza para guardar as configurações do dispositivo.

O mecanismo de Preferências guarda as informações em arquivo e é gerenciado pelo próprio Android, sendo assim não precisamos nos preocupar com o armazenamento destes dados, uma vez que o Android fará todo este trabalho automaticamente.

18.1 Criando uma tela de Preferências

Assim como as demais telas de um aplicativo do Android, a tela de Preferências também é definida através de um arquivo de layout XML, porém com algumas diferenças:

• O arquivo de layout de uma tela de Preferências deve estar dentro do diretório ***/res/xml***;

• O ElementRoot (View principal) do arquivo de layout de uma tela de Preferências deve ser o ***PreferenceScreen***;

• Os componentes (Views) utilizados em uma tela de Preferências são: *CheckBoxPreference*, *EditTextPreference*, *ListPreference*, *RingtonePreference*, *MultiSelectListPreference* e *SwitchPreference*.

Veja o exemplo de um arquivo de layout de uma tela de Preferências:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

  <PreferenceCategory android:title="Configurações básicas" >
    <CheckBoxPreference
      android:key="ativar_notificacao"
      android:summary="Marque esta opção para receber uma notificação quando receber novos e-mails"
      android:title="Ativar" />

    <EditTextPreference
      android:key="email"
      android:summary="Informe seu endereço de e-mail"
      android:title="E-mail" />

    <ListPreference
      android:entries="@array/protocolos_email"
      android:entryValues="@array/protocolos_email"
      android:key="protocolo"
      android:summary="Selecione o tipo do protocolo de recebimento de e-mail"
      android:title="Protocolo" />

    <RingtonePreference
      android:key="toque_notificacao"
      android:summary="Informe o som de notificação ao receber um novo e-mail"
      android:title="Som" />
  </PreferenceCategory>

</PreferenceScreen>
```

A **Figura 18.1** mostra, no emulador do Android, a tela de Preferências referente ao exemplo dado.

Figura 18.1. Exemplo de tela de Preferências do Android.

18.1.1 CheckBoxPreference

O componente *CheckBoxPreference* de uma tela de Preferências é usado para marcar/desmarcar uma opção. Veja o exemplo:

```
<CheckBoxPreference
        android:key="ativar_notificacao"
        android:summary="Marque esta opção para receber uma notificação quando receber novos e-mails"
        android:title="Ativar" />
```

Nota

Em um componente da tela de Preferências, devemos preencher os seguintes atributos básicos:

- android:key: define um nome que será usado como identificador do componente;
- android:summary: define um texto explicativo para o usuário;
- android:title: define o título do componente.

A **Figura 18.2** mostra o exemplo de um CheckBoxPreference em uma tela de Preferências:

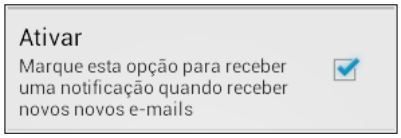


Figura 18.2. Exemplo de um CheckBoxPreference.

18.1.2 EditTextPreference

O componente *EditTextPreference* de uma tela de Preferências é basicamente um campo de texto onde o usuário poderá digitar alguma informação. Veja o exemplo:

```
<EditTextPreference
    android:key="email"
    android:summary="Informe seu endereço de e-mail"
    android:title="E-mail" />
```

A **Figura 18.3** mostra o exemplo de um EditTextPreference em uma tela de Preferências:

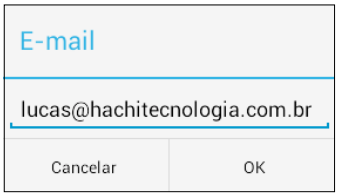


Figura 18.3. Exemplo de um EditTextPreference.

18.1.3 ListPreference

O componente *ListPreference* de uma tela de Preferências é basicamente uma lista onde o usuário poderá selecionar uma das opções apresentadas. Veja o exemplo:

```
<ListPreference
    android:entries="@array/protocolos_email"
    android:entryValues="@array/protocolos_email"
    android:key="protocolo"
    android:summary="Selecione o tipo do protocolo de recebimento de e-mail"
    android:title="Protocolo" />
```

A **Figura 18.4** mostra o exemplo de um ListPreference em uma tela de Preferências:

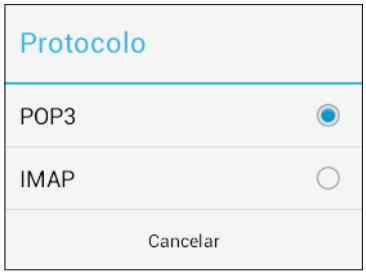


Figura 18.4. Exemplo de um ListPreference.

Em um componente ListPreference devemos preencher os seguintes atributos adicionais:

• android:entries: array de String com o nome dos itens a serem apresentados na lista, apenas para deixar mais legível para o usuário;

• android:entryValues: array de String com o valor dos itens listados.

O array de String passado como parâmetro para estes dois atributos pode ser definido no arquivo de String Resource e será acessado pelo arquivo XML de layout da tela de Preferências da seguinte forma:

```
@array/NOME_DO_ARRAY
```

Veja o exemplo de um Array de String definido em um arquivo de String Resource:

```
<string-array name="protocolos_email">
   <item>POP3</item>
   <item>IMAP</item>
</string-array>
```

18.1.4 RingtonePreference

O componente *RingtonePreference* de uma tela de Preferências é basicamente uma lista onde o usuário poderá selecionar um ringtone (som de toque) armazenado no dispositivo. Veja o exemplo:

```
<RingtonePreference
    android:key="toque_notificacao"
    android:summary="Informe o som de notificação ao receber um novo e-mail"
    android:title="Som" />
```

A **Figura 18.5** mostra o exemplo de um RingtonePreference em uma tela de Preferências:

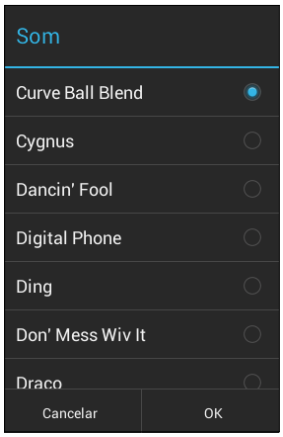


Figura 18.5. Exemplo de um RingtonePreference.

18.2 Definindo uma Activity para a tela de Preferências

Uma tela de Preferências precisa de uma Activity especial para definir seus eventos e iniciá-la. Esta Activity deve herdar a classe ***PreferenceActivity*** do Android. Veja o exemplo de uma Activity de Preferências:

```
public class ConfiguracaoEmailActivity extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.tela_configuracao_email);
    }
}
```

Nota

O método ***addPreferencesFromResource()*** deve ser usado para definir o arquivo de layout da tela de Preferências na Activity. Como parâmetro, este método recebe a referência mapeada na classe ***R*** para o arquivo XML de layout da tela de Preferências, localizado no diretório ***/res/xml***.

Assim como uma Activity normal, uma PreferenceActivity também deve ser configurada no arquivo ***AndroidManifest.xml*** para que o Android possa executá-la. Veja, no trecho abaixo, o exemplo de configuração de uma PreferenceActivity no arquivo *AndroidManifest.xml*:

```
. . .
<activity
    android:name=".preferencia.ConfiguracaoEmailActivity"
    android:label="Preferências">
</activity>
. . .
```

Nota

Na PreferenceActivity não é necessário definir um evento para salvar os valores definidos pelo usuário na tela de Preferências, pois o próprio Android se encarregará desta tarefa. Os valores definidos pelo usuário serão salvos e ficarão armazenados mesmo que o usuário saia do aplicativo.

18.3 Lendo as Preferências salvas em um aplicativo

Até agora nós aprendemos a criar uma tela de Preferências e definir uma Activity para esta tela. Agora iremos aprender a ler os valores definidos pelo usuário em uma tela de Preferências.

Para ler estes valores usamos a classe ***SharedPreferences*** do próprio Android. Em um componente do Android, obtemos um objeto ***SharedPreferences*** através do método ***getDefaultSharedPreferences()*** da classe ***PreferenceManager***. Veja o exemplo no trecho de código abaixo:

```
. . .
// Obtemos o acesso às Preferências definidas pelo usuário
SharedPreferences preferencias = PreferenceManager.getDefaultSharedPreferences(context);
/*
 *  Lemos o valor definido pelo usuário em um componente CheckBoxPreference
 *  da tela de Preferências.
 */
boolean ativo = preferencias.getBoolean("ativar_notificacao", false);
/*
 *  Lemos o valor definido pelo usuário em um componente EditTextPreference
 *  da tela de Preferências.
 */
String codigoLongaDistancia = preferencias.getString("email", "");
. . .
```

Os métodos ***getBoolean()*** e ***getString()*** da classe ***SharedPreferences*** são usados para ler um valor definido pelo usuário em um componente da tela de Preferências e devem receber dois parâmetros:

1. Primeiro parâmetro: o nome (identificador) definido para o componente através do atributo android:key;

2. Segundo parâmetro: um valor padrão, caso o usuário não tenha definido nenhum valor para o componente.

A classe ***SharedPreferences*** disponibiliza outros métodos para obter o valor determinado para cada tipo de componente de uma tela de Preferências.

18.4 Colocando em prática: usando Preferências no projeto *Devolva.me*

Para explorar o recurso de Preferências do Android, vamos criar em nosso projeto *Devolva.me* uma simples tela de Preferência onde o usuário poderá configurar o som de toque da Notificação disparada quando um lembrete é enviado.

18.4.1 Definindo a tela de Preferência

O primeiro passo é definir a tela de Preferência. Para isto, devemos clicar com o botão direito sobre o projeto *Devolva.me* na View *Package Explorer* do Eclipse e selecionar a opção "**New**" > "**Other**", conforme mostra a **Figura 18.6**.

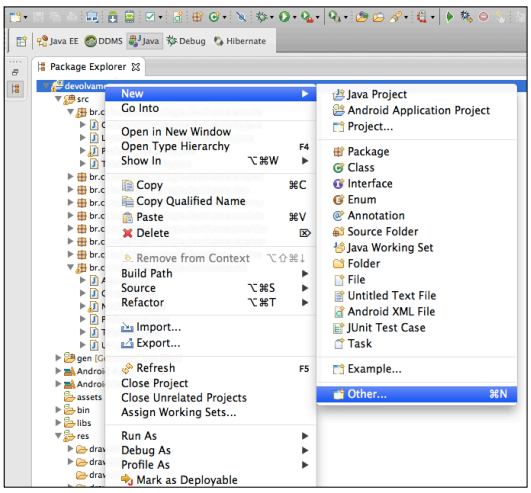


Figura 18.6. Criando um novo arquivo XML de Preferência no projeto *Devolva.me*.

Na próxima tela, selecionamos a opção "**Android**" > "**Android XML File**" e clicamos em **Next**, conforme mostra a **Figura 18.7**.

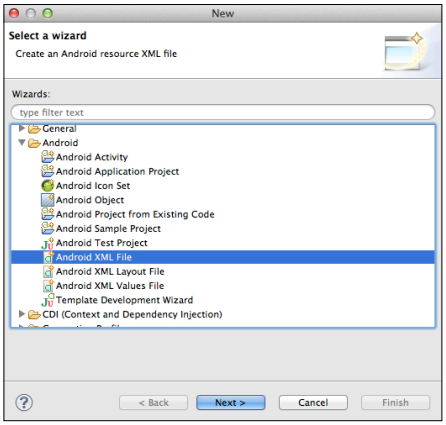


Figura 18.7. Criando um novo arquivo XML de Preferência no projeto *Devolva.me*.

Na próxima tela, em "**Resource Type**" selecione a opção "**Preference**"; em "**File**" informe o nome "**preferencia**" para o arquivo; em "**Root Element**" selecione a opção "**PreferenceScreen**" e clique em "**Finish**", conforme mostra a **Figura 18.8**.

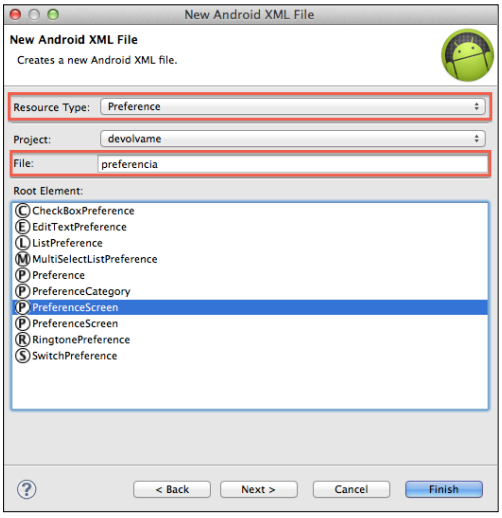


Figura 18.8. Criando um novo arquivo XML de Preferência no projeto *Devolva.me*.

Os passos acima irão criar um arquivo XML chamado ***preferencia.xml*** no diretório "***/res/xml***" do projeto *Devolva.me*. É neste diretório que ficam os arquivos de Layout de Preferências do Android.

Após ter criado nosso arquivo de Layout de Preferências, vamos editá-lo, deixando-o com o seguinte conteúdo:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

   <PreferenceCategory android:title="Preferências" >
      <RingtonePreference
         android:key="toque_notificacao"
         android:summary="Selecione o som de toque da Notificação quando um lembrete é enviado"
         android:title="Som de toque da Notificação" />
   </PreferenceCategory>

</PreferenceScreen>
```

Veja que em nosso arquivo de Preferências usamos apenas o componente ***RingtonePreference***, que usaremos para definir o som de toque da Notificação.

Como próximo passo, devemos definir a Activity responsável pela tela de preferências que criamos. Para isto, no pacote ***br.com.hachitecnologia.devolvame.activity*** vamos criar uma nova classe, chamada ***PreferenciaActivity***, com a seguinte implementação:

```
public class PreferenciaActivity extends PreferenceActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                // Define o arquivo de layout da tela de Preferências
                addPreferencesFromResource(R.xml.preferencia);
        }
}
```

Devemos também configurar a nova Activity. Para isto, adicionaremos o seguinte trecho no arquivo ***AndroidManifest.xml***:

```
. . .
<activity
   android:name=".activity.PreferenciaActivity"
   android:label="Preferências" >
   <intent-filter>
      <action android:name="br.com.hachitecnologia.devolvame.action.PREFERENCIAS" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
</activity>
. . .
```

18.4.2 Criando uma classe utilitária para ler o valor definido nas Preferências

Para facilitar nosso trabalho, vamos criar uma classe utilitária que irá identificar o RingTone definido pelo usuário na tela de Preferências. Para isto, no pacote ***br.com.hachitecnologia.devolvame.util*** vamos criar uma nova classe, chamada ***Preferencias***, com a seguinte implementação:

```
public class Preferencias {

        /**
         * Retorna o RingTone definido na tela de Preferências.
         * @param context
         * @return
         */
        public static String getSomDeToqueDaNotificacao(Context context) {
                SharedPreferences preferencias = PreferenceManager.getDefaultSharedPreferences(context);
                return preferencias.getString("toque_notificacao", "default ringtone");
```

```
        }

    }
```

18.4.3 Definindo o som de toque da Notificação

O próximo passo é modificar o som de toque da Notificação para que ela use o RingTone que o usuário definiu na tela de Preferências. Para isto, vamos alterar a classe ***Notificacao***, substituindo a linha abaixo:

```
notification.defaults = Notification.DEFAULT_ALL;
```

pelo seguinte trecho de código:

```
. . .
notification.defaults = Notification.DEFAULT_VIBRATE;

// Som de toque da Notificação
String somDeToque = Preferencias.getSomDeToqueDaNotificacao(context);
notification.sound = Uri.parse(somDeToque);
. . .
```

18.4.4 Criando um atalho para a Activity de Preferências na tela inicial

Por fim, devemos criar um atalho para a Activity de Preferências na tela inicial do nosso aplicativo. Para isto, na classe ***TelaInicialActivity*** vamos alterar o valor da constante ***OPCOES_DO_MENU***, deixando-a com o seguinte conteúdo:

```
. . .
// Opções que serão apresentadas na ListView da tela principal.
private static final String[] OPCOES_DO_MENU = new String[] {
        "Emprestar objeto", "Listar objetos emprestados", "Preferências", "Sair" };
. . .
```

Ainda na classe ***TelaInicialActivity***, vamos alterar o método ***onListItemClick()***, deixando-o com a seguinte implementação:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
        switch (position) {
        // Evento da primeira opção apresentada na ListView: Emprestar objeto
        case 0:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO"));
                break;
        // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
        case 1:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS"));
                break;
        // Evento da terceira opção apresentada na ListView: Preferências
        case 2:
                startActivity(new Intent("br.com.hachitecnologia.devolvame.action.PREFERENCIAS"));
                break;
        // Evento da terceira opção apresentada na ListView: Sair
        default:
                finish();
        }
}
```

Pronto! Nossa implementação está completa. Agora, ao abrir a tela de Preferências na Activity inicial, podemos definir o som de toque da Notificação disparada quando um novo lembrete é enviado pelo aplicativo *Devolva.me*. A **Figura 18.9** mostra a nossa tela de Preferências em execução no emulador do Android.

Figura 18.9. Tela de Preferências do aplicativo *Devolva.me*.

18.5 Exercício

Agora que você aprendeu a trabalhar com o recurso de Preferências do Android, é hora de colocar em prática.

Neste exercício vamos implementar uma tela de Preferências, no aplicativo ***Devolva.me***, para que o usuário possa configurar o som de toque da Notificação que é disparada quando um novo lembrete é enviado.

1. No projeto ***Devolva.me***, crie um novo arquivo XML de Layout de Preferência do Android, chamado "**preferencia**", com o seguinte conteúdo:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

   <PreferenceCategory android:title="Preferências" >
      <RingtonePreference
         android:key="toque_notificacao"
         android:summary="Selecione o som de toque da Notificação quando um lembrete é enviado"
         android:title="Som de toque da Notificação" />
   </PreferenceCategory>

</PreferenceScreen>
```

2. No pacote ***br.com.hachitecnologia.devolvame.activity***, crie uma nova Activity, chamada ***PreferenciaActivity***, com a seguinte implementação:

```
public class PreferenciaActivity extends PreferenceActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                // Define o arquivo de layout da tela de Preferências
                addPreferencesFromResource(R.xml.preferencia);
        }
}
```

3. Configure a nova Activity, adicionando o seguinte trecho no arquivo ***AndroidManifest.xml***:

```
. . .
```

```
<activity
   android:name=".activity.PreferenciaActivity"
   android:label="Preferências" >
   <intent-filter>
      <action android:name="br.com.hachitecnologia.devolvame.action.PREFERENCIAS" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
</activity>
. . .
```

4. No pacote ***br.com.hachitecnologia.devolvame.util***, crie uma nova classe, chamada ***Preferencias***, com a seguinte implementação:

```
public class Preferencias {

	/**
	 * Retorna o RingTone definido na tela de Preferências.
	 * @param context
	 * @return
	 */
	public static String getSomDeToqueDaNotificacao(Context context) {
		SharedPreferences preferencias = PreferenceManager.getDefaultSharedPreferences(context);
		return preferencias.getString("toque_notificacao", "default ringtone");
	}

}
```

5. Modifique a classe ***Notificacao***, substituindo a linha abaixo:

```
notification.defaults = Notification.DEFAULT_ALL;
```

pelo seguinte trecho de código:

```
. . .
notification.defaults = Notification.DEFAULT_VIBRATE;

// Som de toque da Notificação
String somDeToque = Preferencias.getSomDeToqueDaNotificacao(context);
notification.sound = Uri.parse(somDeToque);
. . .
```

6. Na classe ***TelaInicialActivity*** modifique o valor da constante ***OPCOES_DO_MENU***, deixando-a com o seguinte conteúdo:

```
. . .
// Opções que serão apresentadas na ListView da tela principal.
private static final String[] OPCOES_DO_MENU = new String[] {
		"Emprestar objeto", "Listar objetos emprestados", "Preferências", "Sair" };
. . .
```

7. Ainda na classe ***TelaInicialActivity***, modifique o método ***onListItemClick()***, deixando-o com a seguinte implementação:

```
/**
 * Inicia os eventos de acordo com a opção selecionada pelo usuário na ListView.
 */
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
	switch (position) {
	// Evento da primeira opção apresentada na ListView: Emprestar objeto
	case 0:
		startActivity(new Intent("br.com.hachitecnologia.devolvame.action.CADASTRA_OBJETO"));
```

```
                    break;
                // Evento da segunda opção apresentada na ListView: Listar objetos emprestados
                case 1:
                    startActivity(new Intent("br.com.hachitecnologia.devolvame.action.LISTA_OBJETOS"));
                    break;
                // Evento da terceira opção apresentada na ListView: Preferências
                case 2:
                    startActivity(new Intent("br.com.hachitecnologia.devolvame.action.PREFERENCIAS"));
                    break;
                // Evento da terceira opção apresentada na ListView: Sair
                default:
                    finish();
                }
        }
```

8. Execute o aplicativo no emulador do Android e faça o teste.

19 Apêndice - Publicando aplicativos na Google Play

A Google Play Store é um portal que a Google oferece para disponibilizar seus aplicativos para que usuários com dispositivos Android possam baixá-los.

Existem inúmeras vantagens em optar por usar a Google Play para vender/distribuir seus aplicativos, veja algumas delas:

• Maior público alvo, já que todo dispositivo com Android possui por padrão o aplicativo que dá acesso à Google Play Store;

• Maior credibilidade por parte do usuário, que tem maior confiança em baixar aplicativos da Google Play ao invés de sites desconhecidos da Internet;

• Maior segurança para o usuário, uma vez que todos os aplicativos cadastrados na Google Play são analisados contra códigos maliciosos;

• Compatibilidade garantida, já que a Google Play mostra para um usuário apenas os aplicativos que seu dispositivo com Android tenha suporte para executar;

• Facilidade de atualização, já que a Goole Play oferece um mecanismo para avisar o usuário de que existe uma nova versão do seu aplicativo para ser instalada;

• e diversas outras.

19.1 Criando um perfil de desenvolvedor na Google Play Store

Para publicar aplicativos na Google Play Store é preciso ter uma conta cadastrada no Google. Além disso, é preciso seguir os seguintes passos:

1. Criar um perfil de desenvolvedor, através do link: **https://play.google.com/apps/publish/signup** (a **Figura 19.1** mostra a página inicial de cadastro do perfil de desenvolvedor no site da Google Play Store);

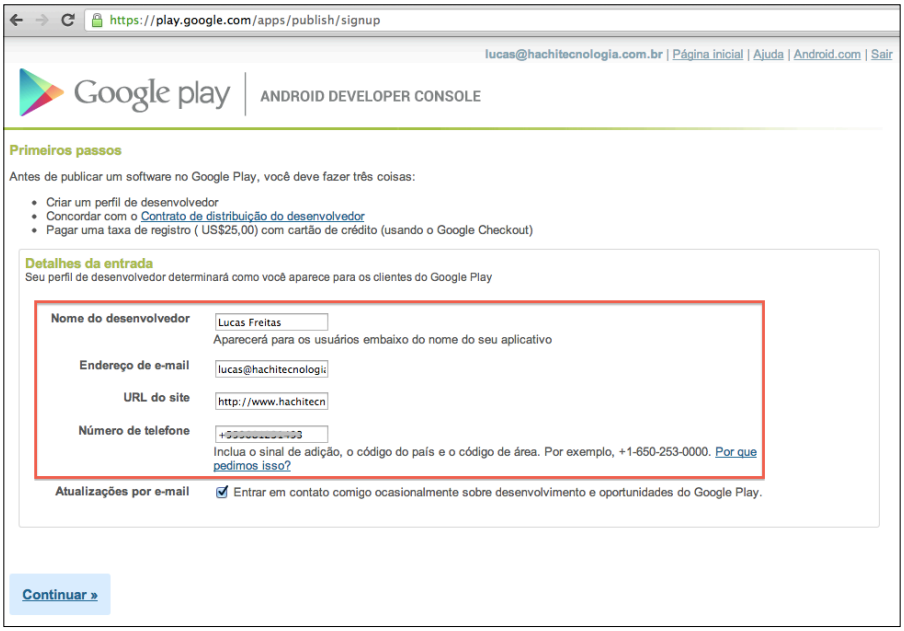


Figura 19.1. Tela inicial de cadastro do perfil de desenvolvedor na Google Play Store.

2. Concordar com o Contrato de distribuição do desenvolvedor (conforme mostra a **Figura 19.2**);

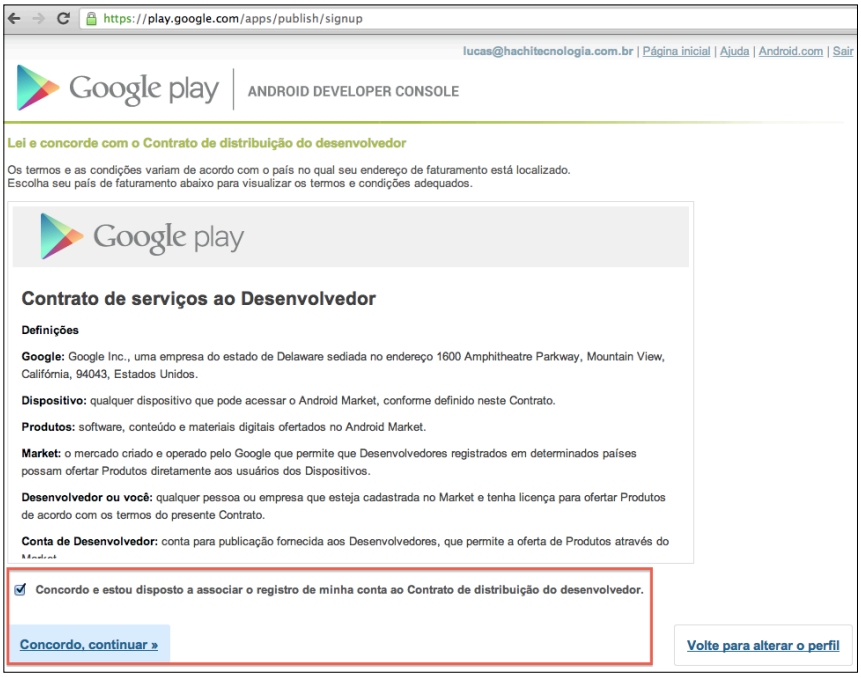


Figura 19.2. Tela com o termo de Contrato de distribuição do desenvolvedor, na Google Play Store.

3. Pagar uma taxa de registro (US$25,00) com cartão de crédito (usando o Google Checkout) - esta taxa é única, ou seja, você só a pagará no momento do cadastro (a **Figura 19.3** mostra a tela com o botão para pagamento da taxa).

Figura 19.3. Tela para efetuar o pagamento da taxa única do perfil de desenvolvedor, na Google Play Store.

Após seguir os passos acima, você já poderá acessar o site da Google Play e publicar seus aplicativos.

19.2 Dicas para a publicação de novos aplicativos

1. Quando se publica novos aplicativos na Google Play é preciso ter certeza que seu aplicativo irá funcionar corretamente nos mais diversos dispositivos encontrados no mercado. Para isto, teste seu aplicativo em diversos dispositivos diferentes ou em emuladores do Android com diferentes configurações;

2. Escolha um bom nome para seu aplicativo;

3. Defina um ícone que tenha um bom visual;

4. Desabilite o atributo ***debuggable*** do seu aplicativo no arquivo ***AndroidManifest.xml***, definindo-o para ***false***. Veja o exemplo abaixo:

```
. . .
<application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme"
      android:debuggable="false">

      . . .

</application>
. . .
```

Nota

Por padrão o atributo ***debuggable*** é definido para ***false***, caso ele não seja declarado no arquivo ***AndroidManifest.xml***.

5. Defina a versão do seu aplicativo no arquivo ***AndroidManifest.xml***. Veja o exemplo:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.com.hachitecnologia.devolvame"
   android:versionCode="1"
   android:versionName="1.0" >

   . . .

</manifest>
```

Nota

O atributo ***versionCode*** do arquivo ***AndroidManifest.xml*** deve ser único e decimal. Este valor deve ser incrementado em 1 unidade a cada nova versão publicada do seu aplicativo. A Google Play Store usará este valor para identificar se o usuário que baixou seu aplicativo precisa atualizá-lo ou não.

Já o atributo ***versionName*** é uma String com um nome amigável da versão, que será visível ao usuário que irá baixá-lo na Google Play Store.

19.3 Definindo as Features que o seu aplicativo usa

Para que a Google Play Store possa detectar quais recursos você usa em seu aplicativo, é importante que você declare suas Features. Se o seu aplicativo faz uso da câmera, por exemplo, é excencial que você declare essa informação nas Features. Desta forma, a Google Play Store não permitirá que um usuário que tenha um dispositivo sem câmera possa baixar e instalar este aplicativo. Isso é importante para garantir a boa imagem do seu aplicativo, pois um usuário não iria gostar nada de baixar um aplicativo que não funcione 100% em seu dispositivo.

As Features são declaradas no próprio aplicativo, no arquivo ***AndroidManifest.xml***. Como exemplo, para informar que seu aplicativo irá usar o recurso de câmera do dispositivo, basta inserir dentro da tag ***<manifest>*** a seguinte Feature:

```
<uses-feature android:name="android.hardware.camera"/>
```

Para ver a lista completa de Features disponíveis, basta acessar o link:

http://developer.android.com/guide/topics/manifest/uses-feature-element.html

19.4 Assinatura Digital

Para que seu aplicativo seja publicado na Google Play Store é preciso assiná-lo digitalmente com um certificado válido, para garantir sua autoria. Desta forma, quando você publicar uma nova atualização do seu aplicativo o Google irá certificar se realmente é você quem está publicando a atualização. Isto garante uma maior segurança para você e para os usuários que irão baixar seus aplicativos.

Para que a Google Play reconheça que realmente é você quem está fazendo a publicação, é preciso que você utilize sempre o mesmo certificado tanto para o aplicativo quanto para as suas atualizações, além de manter o mesmo nome de pacote no arquivo ***AndroidManifest.xml***.

Este certificado digital pode ser gerado por você mesmo, em seu computador, não necessitando ser emitido por uma Autoridade Certificadora. Ao gerar o certificado, é recomendado que você o crie com prazo de validade acima de 25 anos, para que você não corra o risco do certificado expirar a curto prazo.

19.5 Assinando seu aplicativo para publicá-lo na Google Play

O próprio plugin ADT do Eclipse provê uma ferramenta que o auxilia na exportação do pacote .APK do seu aplicativo já assinado digitalmente, através de um wizard.

Para exportar seu aplicativo (.APK) já assinado digitalmente, na View *Package Explorer* clique com o botão direito sobre o seu projeto Android no Eclipse e vá até a opção "**Android Tools**" > "**Export Signed Application Package**", conforme mostra a **Figura 19.4**.

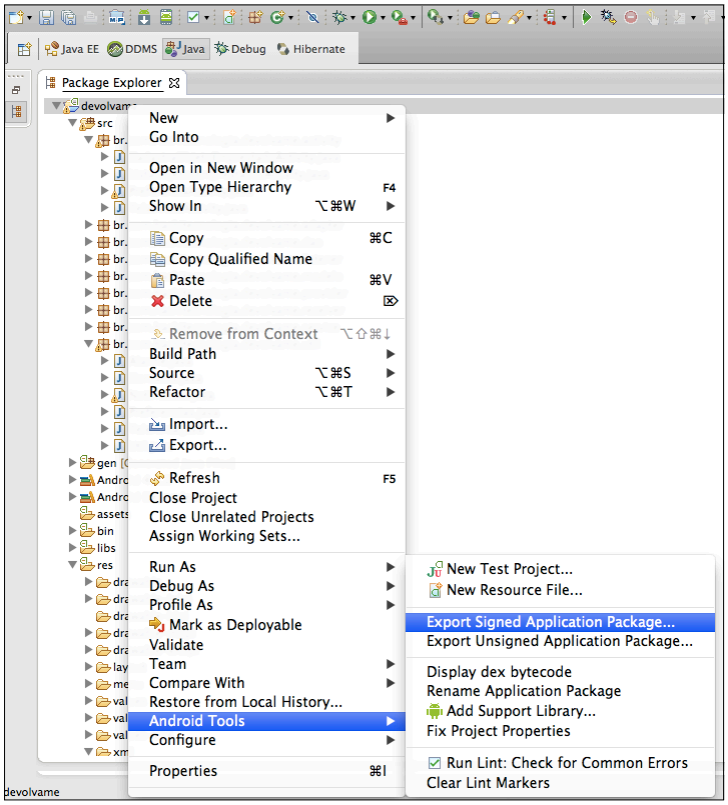


Figura 19.4. Exportando um aplicativo do Android já assinando-o digitalmente, através do plugin ADT no Eclipse.

Na próxima tela é preciso selecionar o projeto a ser exportado. Por padrão o ADT já trás o projeto que você selecionou com o botão direito, antes de abrir o assistente. A **Figura 19.5**.

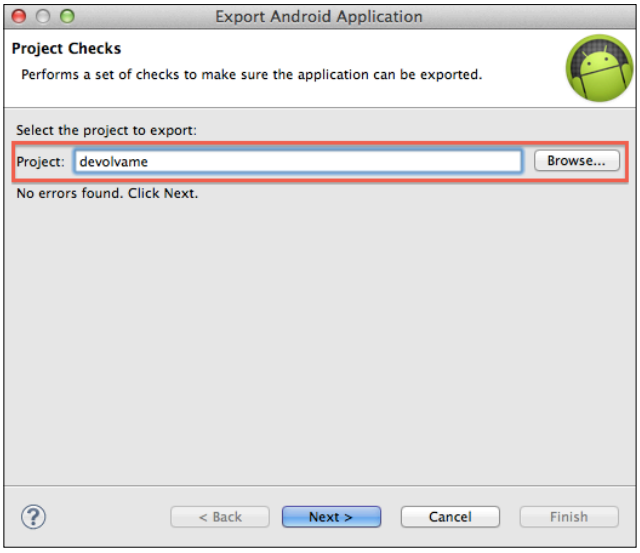


Figura 19.5. Exportando um aplicativo do Android já assinando-o digitalmente, através do plugin ADT no Eclipse.

No próximo passo é preciso selecionar a opção "**Create new keystore**" para dizer ao ADT que você quer gerar um novo certificado e, em "**Location**" informar o caminho (diretório) aonde o certificado deve ser gravado. Deve ser informado também uma senha para o novo certificado.

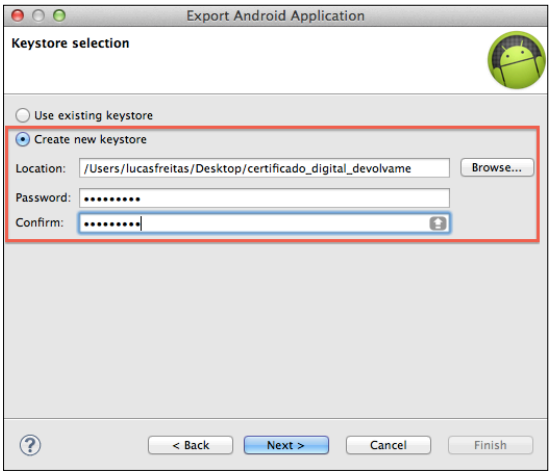


Figura 19.6. Exportando um aplicativo do Android já assinando-o digitalmente, através do plugin ADT no Eclipse.

Nota

É preciso que seu certificado seja guardado em um local seguro, pois é este mesmo certificado que você deverá usar para assinar as versões de atualização do seu aplicativo.

Nota

No passo acima, caso esteja exportando uma versão de atualização do seu aplicativo, você deve marcar a opção "**Use existing keystore**" e informar o caminho do seu certificado digital, já gerado anteriormente.

Nota

Para deixar seu certificado seguro, é preciso que você defina uma senha para ele, preenchendo o campo "**Password**" na hora de gerar o seu certificado.

No próximo passo, você precisa preencher as informações do seu certificado, conforme mostra o exemplo na **Figura 19.7**.

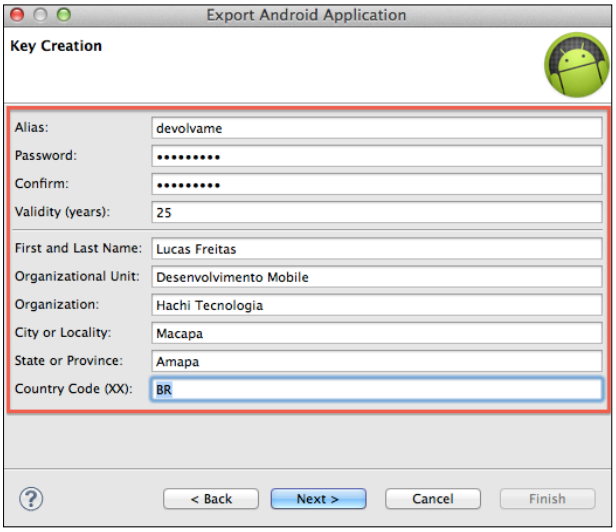


Figura 19.7. Exportando um aplicativo do Android já assinando-o digitalmente, através do plugin ADT no Eclipse.

Por fim, no próximo e último passo você deve informar o caminho (diretório) aonde o aplicativo (pacote .APK) deve ser gerado em seu computador, preenchendo o campo "**Destination APD file**". Após isso, seu pacote .APK já estará pronto para ser publicado na Google Play Store. A **Figura 19.8** mostra este último passo.

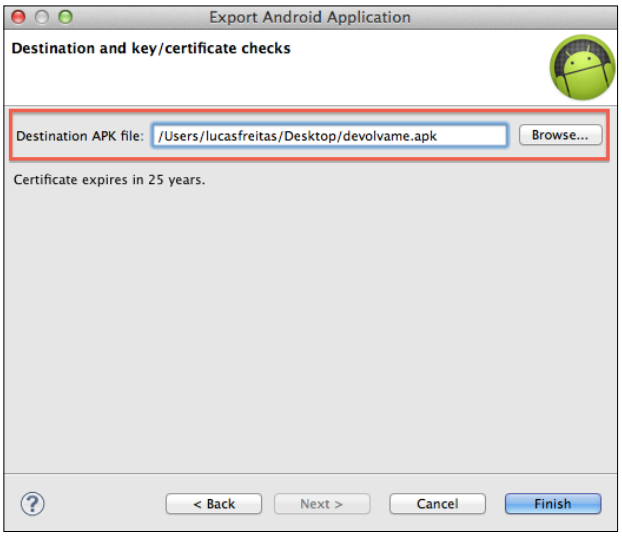


Figura 19.8. Exportando um aplicativo do Android já assinando-o digitalmente, através do plugin ADT no Eclipse.

19.6 Publicando seu aplicativo assinado na Google Play Store

Após ter assinado digitalmente seu aplicativo, basta acessar o site da Google Play, através do link:

https://play.google.com/apps/publish

Na primeira página, basta clicar no botão "**+ Adicionar novo aplicativo**". Ao abrir a janela, basta preencher o nome do seu aplicativo e clicar no botão "**Enviar APK**", conforme mostra a **Figura 19.9**.

Figura 19.9. Adicionando um novo aplicativo na Google Play Store.

No próximo passo, você deve fazer upload do seu APK. Após fazer o upload, aparecerá uma tela parecida com a da **Figura 19.10**.

Figura 19.10. Adicionando um novo aplicativo na Google Play Store.

Após realizar estes passos, é excencial que você defina as informações do seu aplicativo, como: título, descrição, texto promocional, imagens das telas do aplicativo, categoria do aplicativo e suas informações de contato. Estas informações são definidas no link "**Detalhes do aplicativo**".

Nota

Para que seu aplicativo seja bem visto pelos usuários e para que ele faça sucesso na Google Play, além de ser bem desenvolvido (claro) é excencial que suas informações sejam bem definidas no momento de sua publicação. Isto dá maior credibilidade para seu aplicativo e para você, como desenvolvedor. Você também pode adicionar descrições em outros idiomas para seu aplicativo, desta forma você conseguirá atingir um público alvo ainda maior.

Outra informação que você pode preencher é o preço (valor) de venda do seu aplicativo. Esta informação é definida no link "**Preço e distribuição**".

20 Apêndice - Internacionalização

O Android possui suporte nativo a internacionalização, nos permitindo desenvolver aplicativos que suportem vários idiomas. Esta é uma grande vantagem quando vamos distribuir um aplicativo pela Google Play Store, nos possibilitando alcançar usuários de diversos países, com idiomas diferentes.

Neste capítulo vamos aprender a deixar nossos aplicativos prontos para serem traduzidos para diversos idiomas.

20.1 Seguindo as boas práticas

Quando desenvolvemos um aplicativo, devemos sempre prezar por centralizar certos dados para facilitar sua reutilização e manutenção. O Android facilita o trabalho de centralizar recursos como imagens, Strings, layouts, cores, etc. Desta forma, se tivermos a preocupação de centralizar estes recursos, quando precisarmos mudar uma frase que é utilizada em mais de uma tela, por exemplo, basta modificá-la em um só lugar e isto irá refletir em todo o aplicativo.

Ao construir um aplicativo para o Android, é essencial seguir a boa prática de centralizar seus recursos. Além de facilitar a manutenção e reutilização, isto nos permite internacionalizar nosso aplicativo de maneira fácil.

20.1.1 Centralizando String Resources

Para centralizar textos e frases de um aplicativo para Android basta colocar seu conteúdo dentro de um arquivo XML no diretório ***res/values***. Por padrão, o Android sugere o arquivo ***res/values/strings.xml*** para a centralização de Strings. Cada String deve ser adicionada a este arquivo, dentro da tag ***<resource>*** (tag padrão dos arquivos de Resource), da seguinte forma:

```
<string name="nome_da_string">Conteúdo da String</string>
```

No exemplo dado, a String "***Conteúdo da String***" pode facilmente ser utilizada em qualquer classe ou tela do aplicativo, bastando usar sua referência "***nome_da_string***" da seguinte forma:

1. Em um arquivo de Layout:

```
@string/nome_da_string
```

2. Em uma classe:

```
R.string.nome_da_string
```

Nota

Como sabemos, a classe ***R*** disponibiliza apenas as constantes com os IDs dos recursos que definimos em um aplicativo do Android. Para acessar o conteúdo de uma String de um String Resource, em uma classe, usamos o método ***getString()*** da classe *Context* do Android. Como exemplo, para obter o conteúdo da String do nosso exemplo em uma classe Java, fazemos da seguinte forma:

```
getString(R.string.nome_da_string);
```

20.2 Usando Resources internacionalizáveis

Como mencionamos, para deixar seu aplicativo internacionalizável é preciso centralizar todos os seus textos e frases no String Resource ***res/values/strings.xml***. Este é o arquivo que deverá conter o idioma padrão, que será apresentado caso seu aplicativo não suporte o idioma configurado no Android do dispositivo em que foi instalado.

Nota

O diretório ***res/values*** é o diretório padrão de String Resource, que deverá conter as Strings com o idioma padrão do seu aplicativo. Assim como o String Resource, podemos também internacionalizar os demais recursos do Android, como imagens, arquivos de áudio, arquivos de Layout, etc, e cada recurso possui seu diretório padrão, como exemplo:

res/drawable: diretório padrão do Drawable Resource;

res/layout: diretório padrão do Layout Resource.

Para adicionar suporte a vários idiomas para os Resources do Android, usamos o seguinte padrão:

res/<qualificadores>

Onde:

<qualificadores>: é a linguagem específica ou a combinação linguagem/região.

Veja o exemplo:

res/values-pt/strings.xml: String Resource para idioma em **português**;

res/values-fr/strings.xml: String Resource para idioma em **francês**;

res/values-en: String Resource para idioma em **inglês britânico**;

res/drawable-ja: Drawable Resources para idioma em **japonês**.

No exemplo dado, veja que definimos apenas o idioma, não importando a sua região. Desta forma, o String Resource **res/values-pt/strings.xml**, por exemplo, representa o idioma português padrão, ou seja, de Portugal. Para usar um idioma mais específico, podemos usar a combinação idioma/região da seguinte forma:

res/values-pt-rBR/strings.xml: String Resource para idioma em **português do Brasil**;

res/values-en-rUS/strings.xml: String Resource para idioma em **Inglês Americano**.

Nota

Ao usar a combinação idioma/região a letra "**r**" deve sempre ser usada antes da sigla da região, indicando que o que vem a seguir é o código da região, assim como no exemplo dado acima.

Neste momento você deve estar se perguntando: **mas como o Android irá decidir qual idioma do aplicativo ele deverá usar?** A resposta para esta pergunta é simples: o Android irá procurar no aplicativo o mesmo idioma configurado no dispositivo do usuário e, caso este idioma não seja suportado pelo aplicativo, ele irá usar o idioma padrão deste aplicativo.

20.3 Por que o Resource padrão é tão importante

Como vimos, mesmo que seu aplicativo tenha suporte a vários idiomas, é essencial definir os Resources padrão para definir um idioma padrão para o seu aplicativo, que será usado caso o idioma do usuário não seja suportado.

Nota

Quando se disponibiliza um aplicativo em vários idiomas, uma boa prática é definir o idioma padrão para Inglês Americano, já que este é o idioma mais utilizado no mundo. Desta forma o aplicativo conseguirá alcançar um público alvo consideravelmente maior.

O Resource padrão deve conter todos os recursos presentes em seu aplicativo, sem exceção. Caso contrário, seu aplicativo apresentará uma tela de erro ao usuário, forçando o fechamento do aplicativo. Como exemplo, imagine o seguinte cenário:

Imagine que você tenha um aplicativo com o idioma padrão definido para **português brasileiro** mas disponibiliza também o idioma **inglês**, e seu String Resource contenha uma String de nome "**minha_string**". Seu String Resource ficaria com a seguinte estrutura:

res/values/strings.xml: String Resource padrão em portugês brasileiro;

res/values-en: String Resource em inglês.

Seguindo o exemplo dado, se a String de nome "**minha_string**" fosse definida apenas no arquivo "**res/values-en**" (inglês), caso o dispositivo do usuário estivesse configurado para outro idioma diferente do inglês uma mensagem de erro seria apresentada e o aplicativo seria encerrado. Portanto, para evitar que ocorra erro em seu aplicativo, tenha sempre em mente que todos os Resources utilizados em seu aplicativo devem também estar no Resource padrão.

20.4 Exercício

Agora que você aprendeu a deixar seu aplicativo Android internacionalizável, suportando diversos idiomas, é hora de colocar em prática.

1. No aplicativo ***Devolva.me*** que você desenvolveu no decorrer do curso, centralize todos os textos de suas telas no String Resource padrão e adicione suporte ao idioma inglês. (Caso não domine o idioma inglês, use o Google Tradutor - **http://translate.google.com.br** - para ajudá-lo nesta tarefa)

2. Altere a configuração do seu Android entre português e inglês e veja o aplicativo suportando estes dois idiomas.