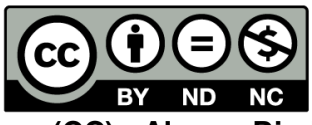
6ª Edição

De : Luciano Alves da Silva (lucianopascal@yahoo.com.br)

www.apostilaandroid.net

ApostilaDeAndroid

Rio de Janeiro - Outubro 2013



Aviso sobre esta apostila

Antes de iniciar a leitura deste material, veja esse aviso:

Este material usa a licensa Creative Commons 

isto significa

que **ELE PODE SER DISTRIBUÍDO LIVREMENTE**, porém, **SOBRE AS SEGUINTES REGRAS** :

Esse material **NÃO PODERÁ SER COMERCIALIZADO**
Essa material **NÃO PODERÁ SER DEVIRADO**
E todos os créditos do autor **DEVERÃO SER MANTIDOS**

O Sucesso da Apostila de Android

A Apostila de Android – Programando Passo a Passo é hoje referência didática de material sobre desenvolvimento de aplicações e sistemas para a plataforma Google Android, conhecido tanto aqui no Brasil quanto em outros países (somando-se mais de 65 países). Hoje a Apostila de Android já chegou a aproximadamente 200.000 acessos (feito pelos mais diversos usuários como estudantes e profissionais da área de programação) Hoje ela é usada por universidades e professores que ministram cursos sobre desenvolvimento Android (Tanto na área de criação de aplicações e jogos).

Sobre o Autor da Apostila

Luciano Alves da Silva é Bacharelado em Ciência da Computação pela UNISUAM (Rio de Janeiro – RJ) e Pós-Graduado em Docência do Ensino Superior pelo Instituto A Vez do Mestre (Universidade Cândido Mendes – UCAM, no Rio de Janeiro). Possui conhecimento e domínio das linguagens de programação Pascal, Java, C/C++, C#, Visual Basic, Delphi, PHP e HTML. Já criou Ambientes de Desenvolvimento Integrado (conhecidos como IDE) como o MakeWare (que trabalha com as linguagens Pascal, C++ e Java) e o AlgoWare (interpretador de algoritmos).

É autor também dos seguintes livros, pela editora AGBOOK

- Aprenda Passo a Passo a Programar em Android – Guia Essencial para Desenvolvedores
- Desenvolvendo Jogos com a Plataforma XNA – Guia para Desenvolvedores.
- Desenvolvendo Jogos com a Ferramenta RPG Maker VX– Guia do Usuário.

Apresentação

O Android é uma plataforma aberta voltada para dispositivos móveis desenvolvida pela Google e atualmente é mantida pela Open Handset Alliance (OHA). Todas as aplicações desenvolvidas para essa plataforma foram criadas com a linguagem Java, o que facilita muitos programadores com conhecimentos em Java (ou de outras linguagens próximas de Java como C++ e C#) a desenvolver aplicações para o Android.

Esta apostila tem por objetivo mostrar de modo fácil e claro como desenvolver aplicações para dispositivos móveis da Google (Android) usando a IDE Android Developer Tools (Eclipse Juno + Android SDK).

Para quem dedico este material?

Este material é dedicado aos usuários experientes ou iniciantes em programação (tanto para Desktop, Móbile e etc.), que já tenha algum contato com a linguagem Java ou com uma de suas derivadas (como C/C++ ou C#).

Índice analítico

Capitulo 1 Visão geral sobre o Google Android

1.1) Introdução

Conforme mencionado na apresentação deste material, o Android é uma plataforma desenvolvida pela Google voltada para dispositivos móveis, totalmente aberta é livre (Open Source), que foi divulgada em 5 de novembro de 2007. Inicialmente o sistema Android foi desenvolvido pelo Google e atualmente essa plataforma é mantida pela OHA (Open Handset Alliance. Visite o link : http://www.openhandsetalliance.com), um grupo constituído por aproximadamente 84 empresas as quais se uniram para inovar e acelerar o desenvolvimento de aplicações e serviços, com o objetivo e trazer aos consumidores uma experiência mais rica em termos de recursos, menos dispendiosa em ternos financeiros para o mercado móvel.

Um dos primeiros SmartPhones que ofereceu suporte a esse sistema operacional foi o G1 da empresa T-Mobile. Confira na imagem seguinte:

G1 - T-Mobile

Atualmente o sistema Android se encontra hoje disponível tanto em SmartPhones quanto nos Tablets. Confira abaixo alguns dos dispositivos encontramos hoje no mercado com o sistema operacional Android:

SmartPhone Samsung Galaxy S3

Tablet Samgung Galaxy Note

1.2) Estrutura Geral da plataforma Google Android

O Android SDK é uma ferramenta de desenvolvimento que disponibiliza um conjunto de APIs necessárias para desenvolver aplicações para a plataforma Android, utilizando a linguagem Java.

Vamos conhecer os recursos encontrados nessa plataforma:

- **Application framework:** Permite a reutilização e substituição de componentes ;

- **Dalvik virtual machine:** É uma Máquina Virtual Java (JVM) voltada para dispositivos móveis ;

- **Browser Integrado** baseado no webkit engine ;

- **Gráficos Otimizados** O Android é constituído por bibliotecas 2D e 3D baseada na especificação OpenGL ES 1.0 ;

- **SQLite:** Sistema Gerenciador de Banco de Dados (SGBD) já embutido no Android para guardar dados ;

- **Suporte multimídia:** A plataforma já oferece para áudio, vídeo e formatos de imagem (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF) ;

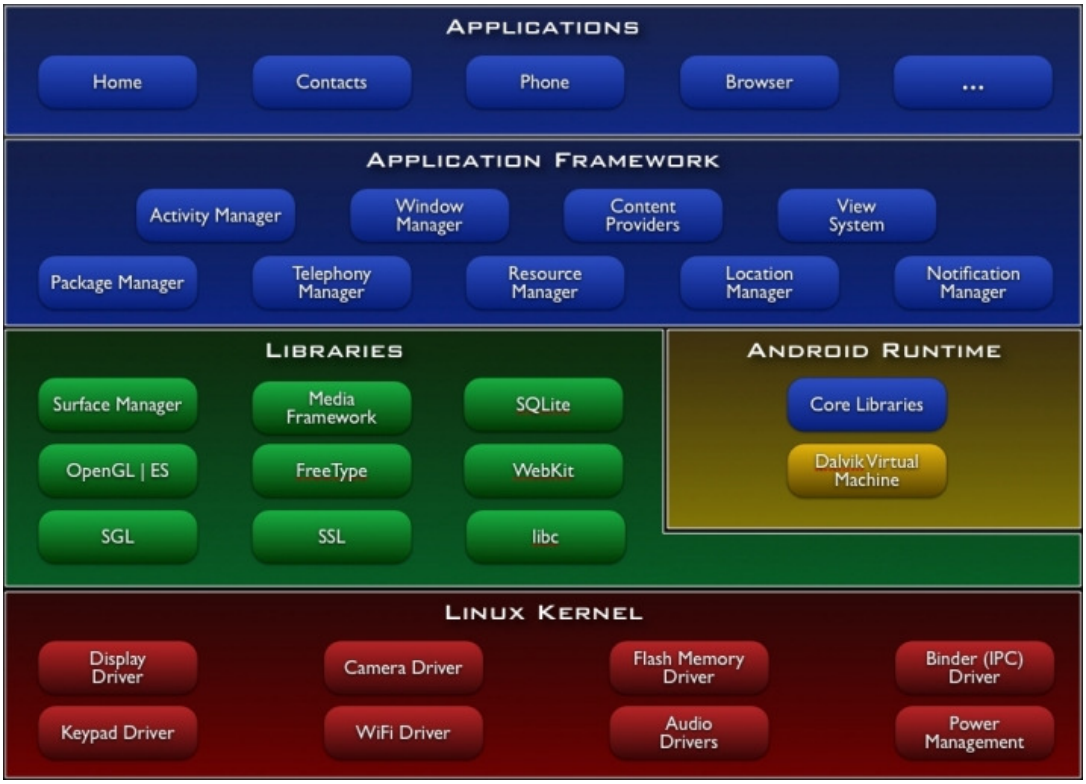
- **Telefonia GSM** (dependente de hardware) ;

- **Bluetooth, EDGE, 3G, e WiFi** (dependente de hardware) ;

- **Câmera, GPS, compasso, e acelerômetro** (dependente de hardware) ;

- **Rico ambiente de desenvolvimento** , incluindo um emulador de dispositivo, ferramentas de depuração, memória, performance e um plugin para o Eclipse (ADT) ;

1.2.1) A arquitetura do Android



Arquitetura geral da plataforma

1.2.2) Aplicações

O Android nos fornece um conjunto de aplicações fundamentais, são elas:

- um cliente de e-mail;
- programa de SMS;
- agenda;
- mapas;
- navegador;
- contatos entre outros.

Todos os aplicativos acima presentes no Android foram desenvolvidos na linguagem de programação Java.

O Android nos fornece um conjunto de bibliotecas C/C++ utilizadas por vários componentes do sistema. Veja algumas das bibliotecas abaixo:

- **System C library:** Consiste em uma implementação derivada da biblioteca C padrão baseado no sistema (libc) do BSD sintonizada para dispositivos rodando Linux.

- **Media Libraries:** Baseado no PacketVideo's OpenCORE; são as bibliotecas que suportam os mais diversos formatos de áudio e vídeo, incluindo também imagens.

- **Surface Manager:** Responsável pelo acesso ao subsistema de exibição bem como as múltiplas camadas de aplicações 2D e 3D;

- **LibWebCore**: Consiste em um web browser engine utilizado tanto no Android Browser quanto para exibições web.

- **SGL – o engine de gráficos 2D**
- **3D libraries**: Uma implementação baseada no OpenGL ES 1.0 APIs; As bibliotecas utilizam aceleração 3D via hardware (quando disponível) ou o software de renderização 3D altamente otimizado incluído no Android.
- **FreeType** – Biblioteca responsável pela renderização de fontes bitmap e vector;

- **SQLite** – Conforme já mencionado, consiste no sistema gerenciador de banco de dados (SGBD) relacional disponível para todas as aplicações.

1.2.3) Android Runtime

O Android é constituído por um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem Java.

Toda aplicação Android roda em seu próprio processo, com sua própria instância da máquina virtual Dalvik. O Dalvik foi escrito de forma a executar várias VMs eficientemente. Ele executa arquivos .dex, que é otimizado para consumo mínimo de memória. A VM é baseada em registros e roda classes compiladas pela linguagem Java que foram transformadas em arquivos .dex, através da ferramenta "dx" incluída no SDK.

O Dalvik VM foi baseado no kernel do Linux para funcionalidades subjacentes como o encadeamento e a gestão de baixo nível de memória.

1.2.4) Linux Kernel

O Android foi projetado em cima da versão 2.6 do kernel do Linux para os serviços centrais do sistema, tais como segurança, gestão de memória, gestão de processos, etc. O kernel também atua como uma camada de abstração entre o hardware e o resto do software.

1.3) Para qual versão do Android devemos desenvolver as aplicações ?

Quando desenvolvemos uma aplicação para um determinado sistema operacional, normalmente, precisamos fazer a seguinte pergunta : Para qual versão do S.O devemos desenvolver ?

Considero esse um dos pontos mais importantes que devemos refletir antes de desenvolvermos uma aplicação. Como neste material iremos desenvolver aplicações voltados para a plataforma Android, devemos pensar para qual versão da plataforma precisamos desenvolver.

1.3.1) Qual versão da plataforma Android é a mais utilizada no momento ?

Para falar a respeito dessa situação, irei mostrar aqui um gráfico que mostra quais versões do Android são as mais usadas no mundo todo :

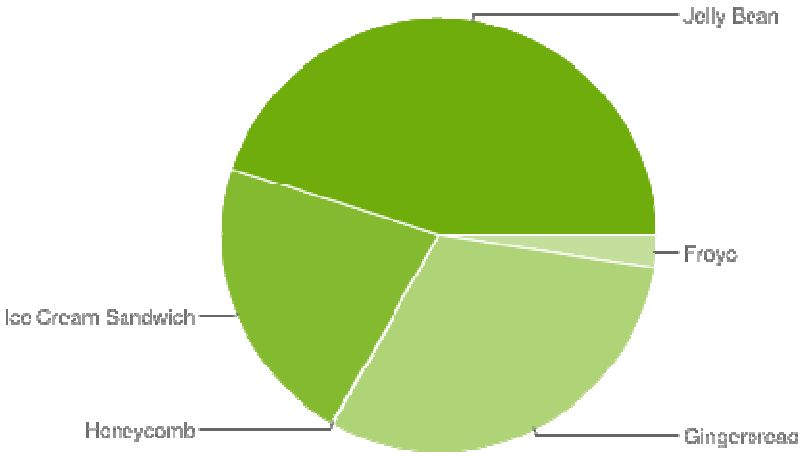


Gráfico de Estatística a respeito do S.O Android mais usado

O gráfico da estática acima foi feito em setembro de 2013, onde nele podemos observar que as versões do Android mais utilizadas são o Jelly Bean (versão 4.1 – 4.3), Gingerbread (versão 2.3) e Ice Cream Sandwich (versão 4.0). As versões mais antigas do Android como Eclair (versão 2.1) e Donut (versão 1.6) já nem são mais citadas e faladas hoje em dia, e durante a produção dessa apostila a versão 4.4 (KitKat) do Android havia acabado de ser lançada.

Hoje em dia, se fomos em alguma loja para comprar um aparelho (Smartphone ou Tablet Android) iremos adquiri-lo, naturalmente, com o S.O Android versão 4.0 para cima. Se olharmos por esse ponto de vista, devemos pensar em desenvolvermos nossa aplicação utilizando, como base, a versão 4.0.

1.3.2) O Publico

Um outro fator muito importante , e que destaco aqui , é a questão O PUBLICO. Nesse fator , a questão S.O deveria ser deixada "teoricamente" de lado, visto que muitos usuários ainda possuem aparelhos Android com uma versão mais antiga (como a versão 2.3 e 2.2), logo, devemos pensar também nesses usuários para "usufruir" das nossas aplicações desenvolvidas.

1.3.3) Qual prioridade devemos dar : Publico ou Versão do S.O ?

Agora a questão é : Como combinar a questão PUBLICO e VERSÃO do S.O para desenvolvermos a aplicação ? Se você pretende desenvolver uma aplicação Android simples (como um pequeno sistema de cadastro), podemos, se quisermos, dar prioridade a questão PUBLICO, procurando desenvolver sua aplicação Android para uma versão mais antiga, porém, ainda NÃO OBSOLETA . Agora se você desenvolver uma aplicação Android cheia de muitos recursos, cujos componentes só existem em versões mais atuais do sistema, devemos dar prioridade a questão VERSÃO do S.O.

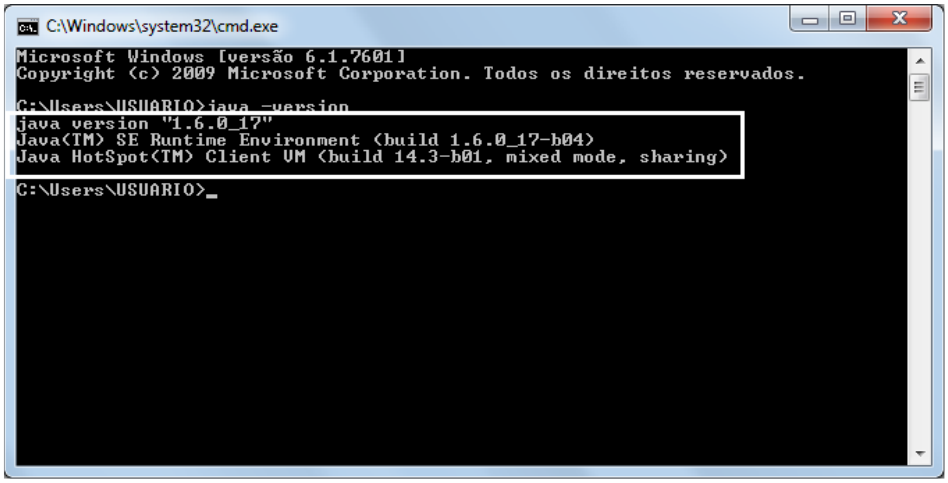
Capitulo 2 Instalando e Configurando o Android Developer Tools

Para a elaboração desse material eu fiz o uso do Android Developer Tools (que consiste no Eclipse 4.2 para Windows configurado com o Android SDK). Qualquer versão (de preferência superior) da ferramenta citada acima serve. Para que toda essa aplicação funcione é necessário que você tenha instalado, antes de tudo, a Máquina Virtual Java (de preferência a versão 6 ou posterior). Bom, mãos a obra.

Para saber se você possui uma Maquina virtual Java entre no prompt de comando de digite a seguinte linha:

java –version

Se mostrar algo parecido como demonstra a figura seguinte:



Maquina Virtual Java instalada no computador

Significa que você possui uma máquina virtual Java instalada no seu computador, caso contrário, instale o JRE. Você pode fazer o download do Java pelo link abaixo:

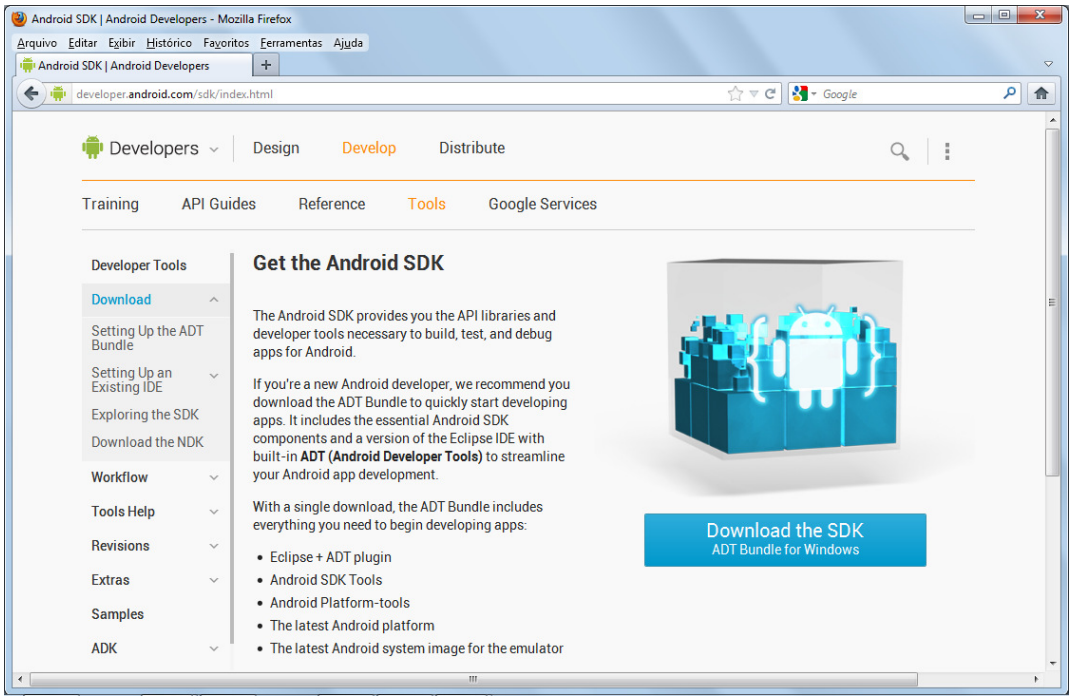
http://www.oracle.com/technetwork/java/javase/downloads/index.html

2.1) Efetuando o download e configurando o ADT

Conforme havia falado, a ferramenta que iremos utilizar aqui para a construção das nossas aplicações é o ADT (Android Developer Tools), que consiste no Eclipse configurado com o Android SDK. A linguagem que utilizaremos para criar as aplicações será a linguagem Java, na qual a ferramenta dá suporte. Para realizarmos do download desta ferramenta basta visitarmos o seguinte link abaixo:

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

Feito isso será carregado a página de download, conforme demonstra a figura abaixo:

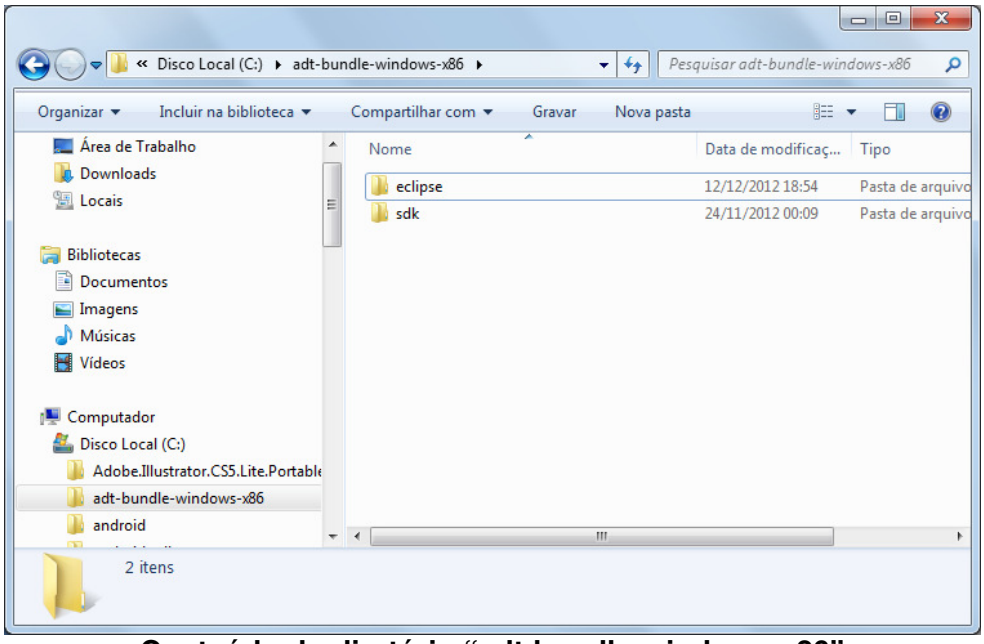


Página de download do Android Developer Tools

Para efetuarmos do download do Android Developer Tools basta clicarmos no botão “Download the SDK”. Feito isso será aberta uma tela de contrato de “termos e condições”, onde nela você vai confirmar os termos de contrato e selecionar a versão sistema operacional (32 bits ou 64 bits). Depois de escolher o sistema operacional, basta clicar no botão “Download the SDK ADT Bundle for Windows” para efetuar o download da ferramenta.

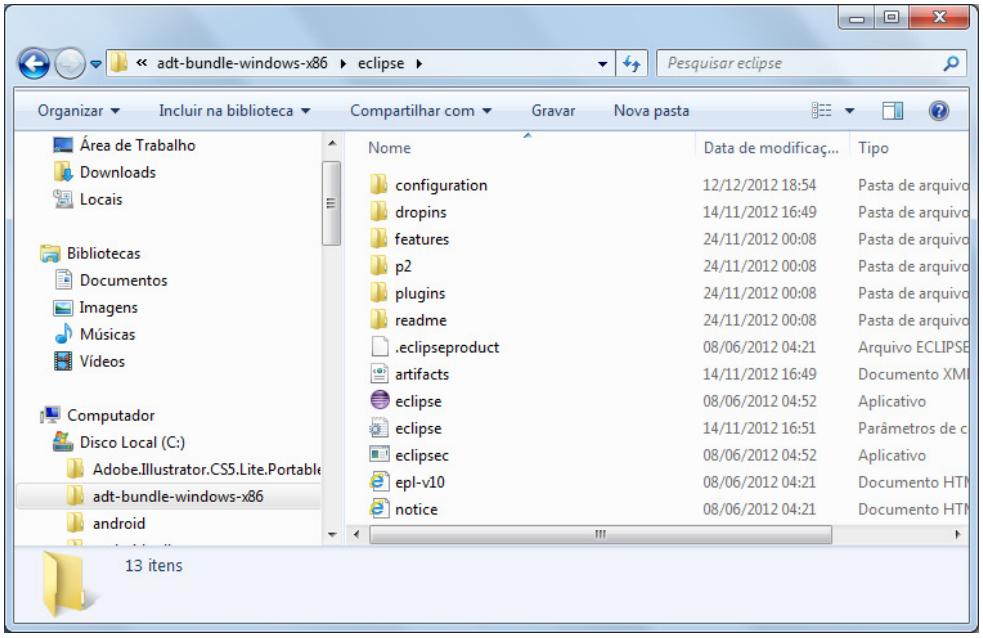
A instalação da ferramenta é bastante simples (como no Eclipse), bastando descompactar o arquivo em um diretório desejado. O diretório de instalação do ADT para esse caso será o diretório “raiz” “C:\”. Ao descompactar o arquivo da aplicação, certifique-se de que foi gerado o diretório “C:\ adt-bundle-windows-

x86” (ou “C:\ adt-bundle-windows-x86_64”, caso o sistema escolhido seja de 64 bits) e que o mesmo apresenta o conteúdo demonstrado na figura abaixo:



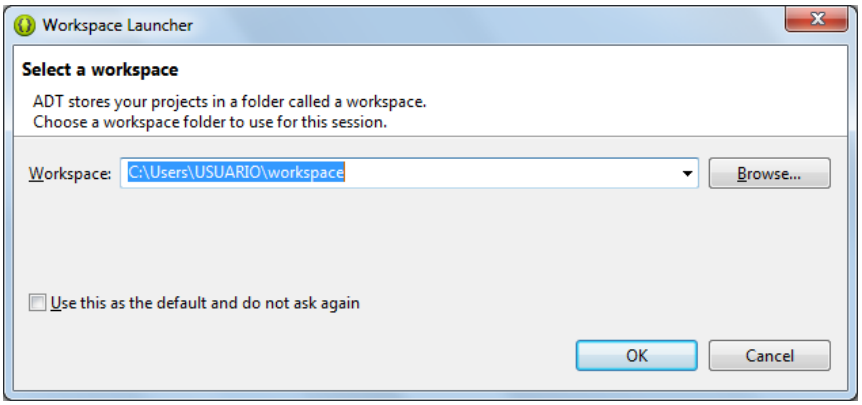
Conteúdo do diretório “adt-bundle-windows-x86”

A ferramenta de desenvolvimento se encontra dentro do diretório “eclipse”, basta isso basta acessar o diretório “eclipse”. Feito isso, você verá o seguinte conteúdo:



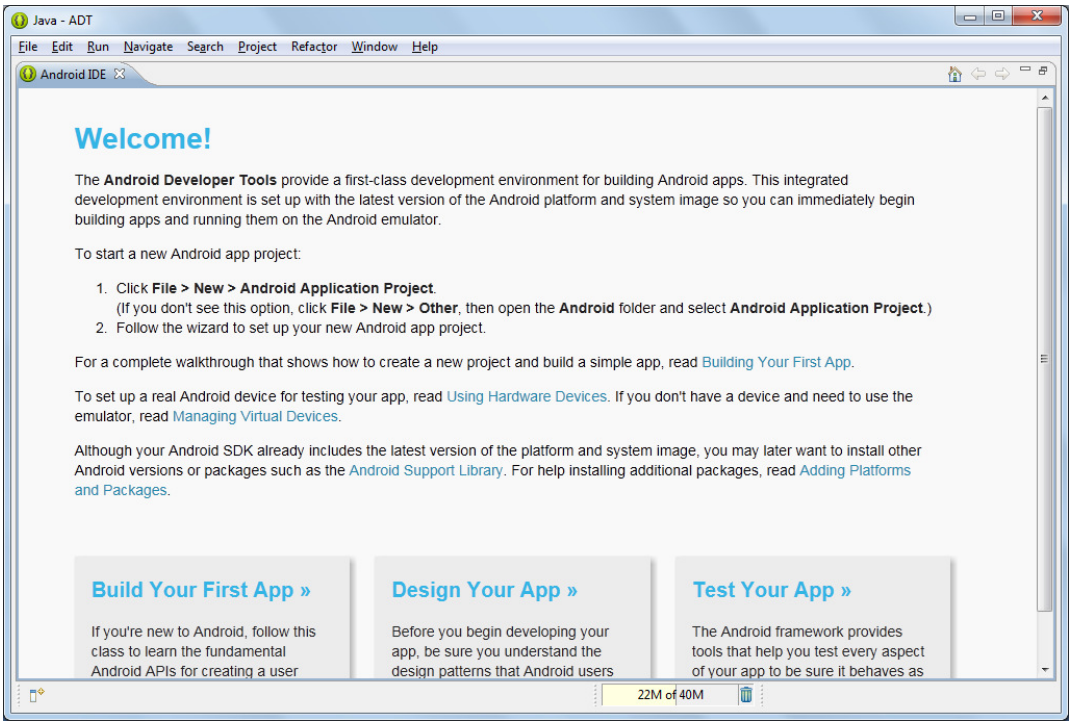
Conteúdo do diretório “eclipse”

Para executarmos o Android Developer Tools basta clicar no ícone (de cor "roxa") escrito "eclipse". Quando a ferramenta é executada pela primeira vez, será solicitado um diretório de "Workspace" (diretório de trabalho), que é o local no qual o Eclipse vai gerar os projetos, conforme você confere na figura seguinte:



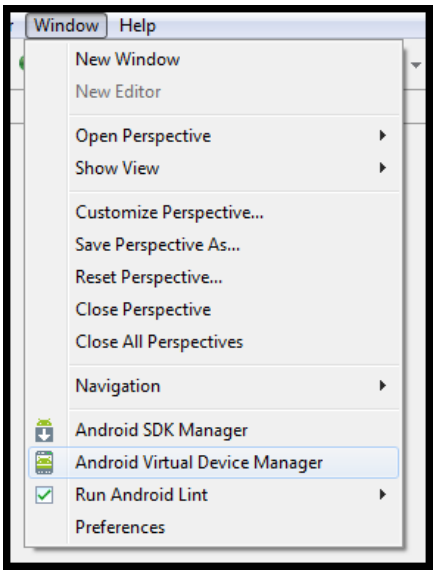
Diretório do Workspace

Escolha o diretório desejado para seu "Workspace" e , caso você deseje que ela seja definitivo, marque a opção "Use this as the default and do not ask again". Depois disso clique em "OK".
Feito isso, o ambiente de programação Eclipse será carregado, conforme demonstra a figura seguinte:



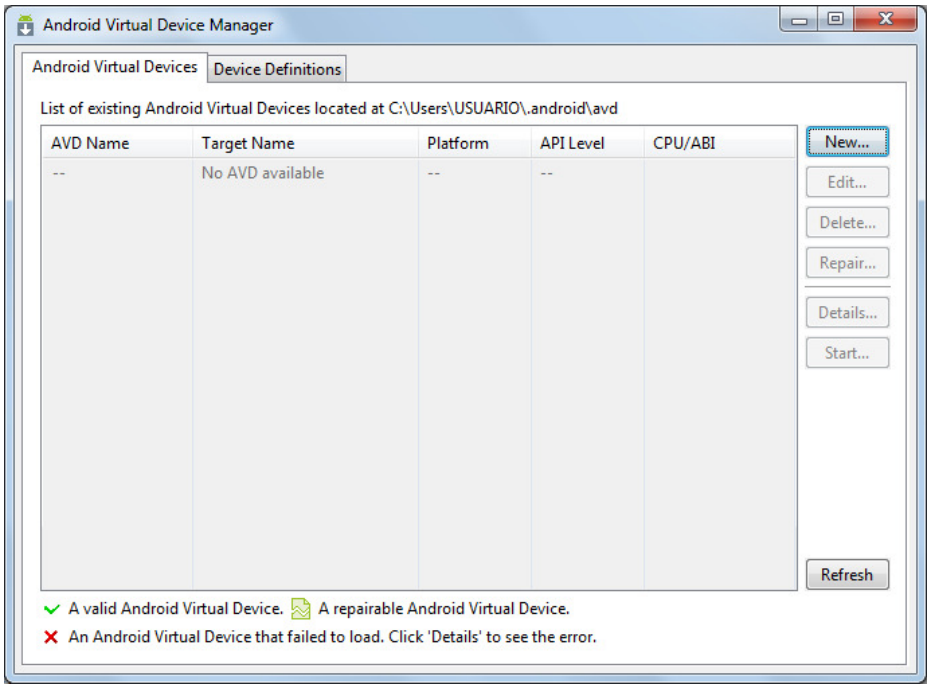
Ambiente de programação Eclipse (ADT)

Agora vamos fechar a guia “Android IDE” para encerrarmos a tela de “Welcome!” e em seguida vamos configurar um dispositivo virtual (conhecido no Android como AVD ou “Android Virtual Device”) que usaremos para testar e executar as nossas aplicações (jogos). Para definimos um AVD basta irmos no menu “Windows” / “Android Virtual Device Manager”, como mostra a figura abaixo:



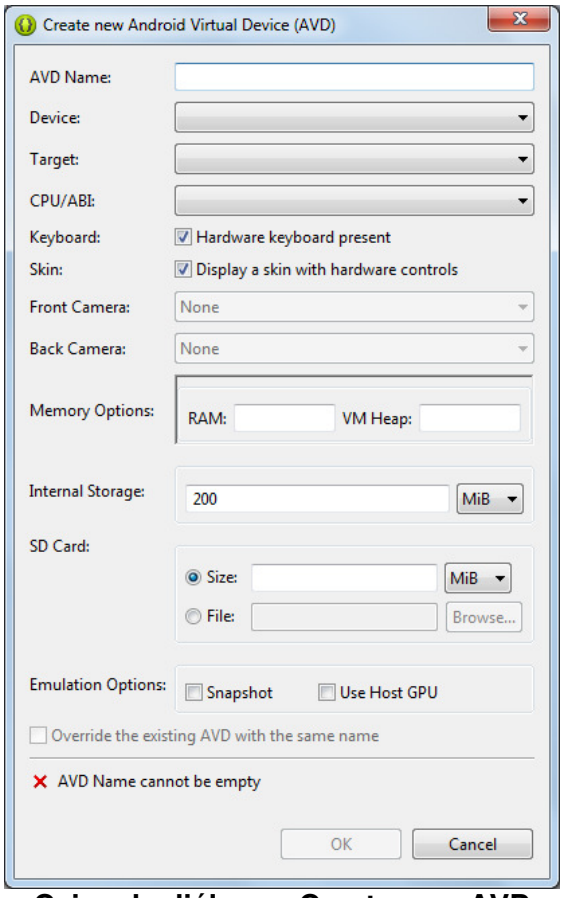
Android Virtual Device Manager

Feito isso, será aberta a seguinte tela abaixo:



Caixa de diálogo – Android Virtual Device Manager

Para criarmos um dispositivo virtual clique no botão “New”, e em seguida será aberta uma tela conforme mostra a figura seguinte:

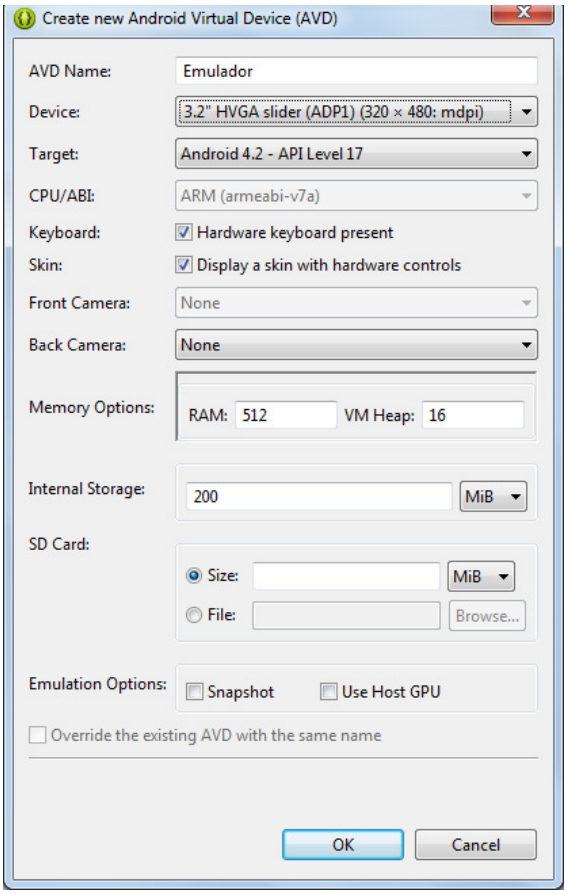


Caixa de diálogo - Create new AVD

No Android Developer Tools já existe uma plataforma que já acompanha a ferramenta (no meu caso é o Jelly Bean, versão 4.2), logo, vamos utilizar ela.

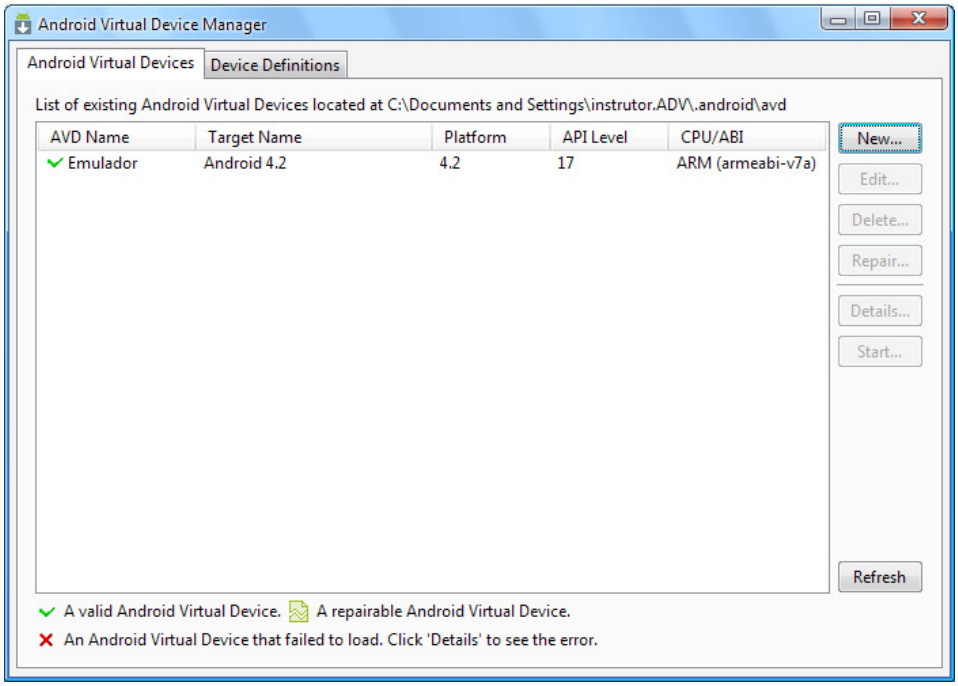
Inicialmente, vamos configurar o básico pra executarmos a nossa aplicação. Em “AVD Name” você define o nome do AVD, vamos chamá-lo de “Emulador”.

Em “Device” escolhemos o tipo de dispositivo e a sua resolução. Atualmente encontramos diversos dispositivos Android com os mais diversos tamanhos (resoluções) diferentes. A resolução que será utilizada para a construção dos nossos jogos será uma resolução “3.2 HVGA” (que é normalmente uma resolução padrão de Smartphone). Para o dispositivo vamos escolher a opção “3.12 HVGA slider (ADP1) (320 x 480 : mdpi).
Automaticamente quando selecionamos o tipo de dispositivo (Device), é definida a plataforma que será utilizada (a seção “Target”). Como havia falado, essa versão acompanha a plataforma Jelly Bean 4.2, logo, a plataforma escolhida será a Jelly Bean (Androd 4.2 – API Level 17).
Veja como ficou as configurações na figura abaixo:



Caixa de diálogo - Create new AVD

Para criarmos nosso AVD, clique no botão “OK” e pronto. O resultado você confere na figura seguinte:



Caixa de diálogo – Android Virtual Device Manager

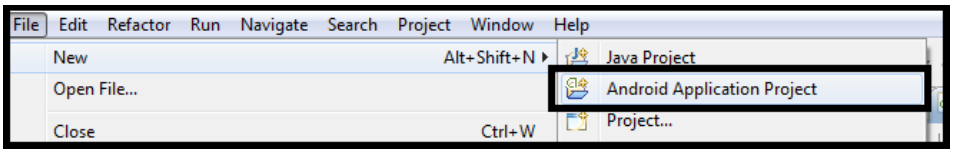
Bom, nosso ambiente de desenvolvimento está devidamente configurado. No próximo capítulo vamos aprender passo a passo como criar um projeto Android para desenvolvimento de aplicações.

Capitulo 3 Construindo nossas aplicações no Android

Vamos colocar a mão na massa ? A partir de agora iremos começar a desenvolver as nossas aplicações no Android utilizando os componentes descritos no capítulo anterior. Começaremos com aplicações simples e aos poucos iremos evoluir, criando aplicações mais ricas.

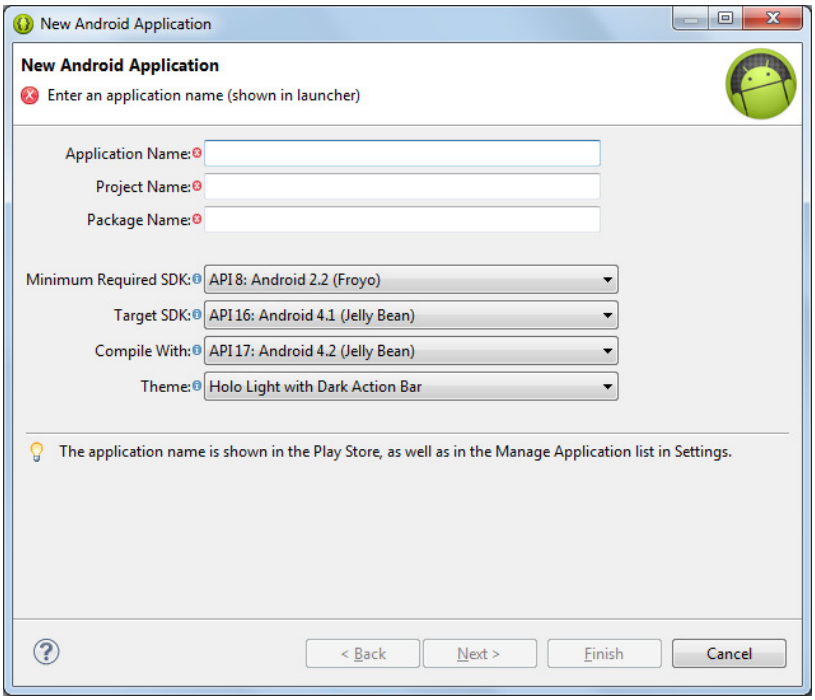
3.1) Desenvolvendo uma Calculadora Básica

Vamos construir a nossa primeira aplicação que vai consistir em uma calculadora básica com as quatro operações aritméticas. Para criar um projeto em Android (conforme já foi mostrado mas, mostro novamente aqui) vamos no menu “File”/”New / Android Aplication Project”. Confira na figura seguinte:



Android Application Project (pelo menu)

Seguido um dos passos descritos acima, irá se abrir a caixa de diálogo abaixo:



Criando o projeto Calculadora

Agora vamos preencher os campos, conforme abaixo:

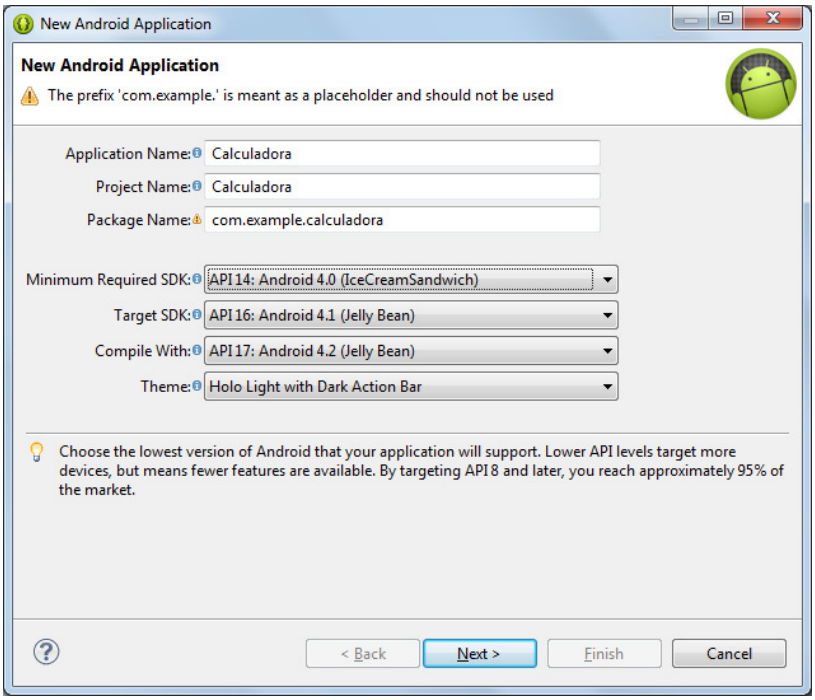
Application Name : Calculadora

Project Name : Calculadora

Package Name : com.example.calculadora

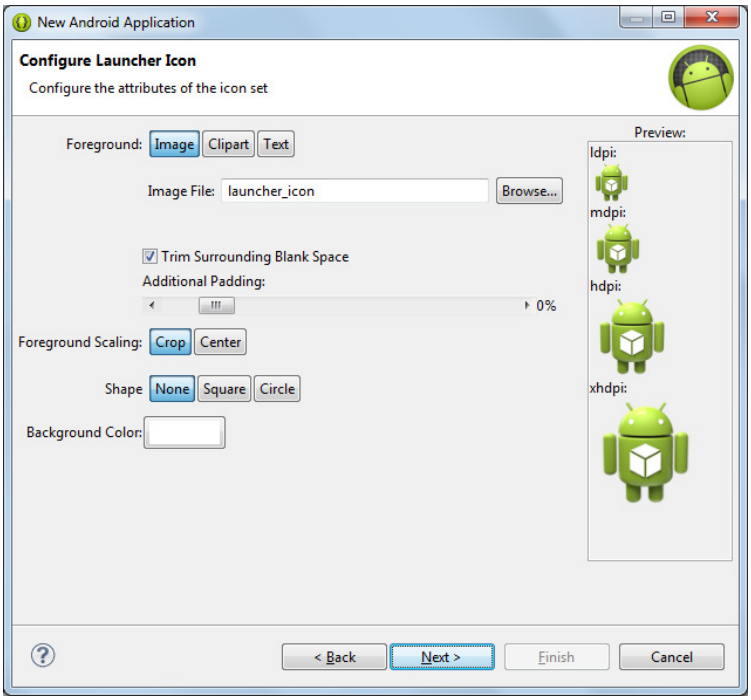
Mininum Required SDK : API 14 Android 4.0 (Ice Cream Sandwich)

Confira como ficou na figura seguinte:



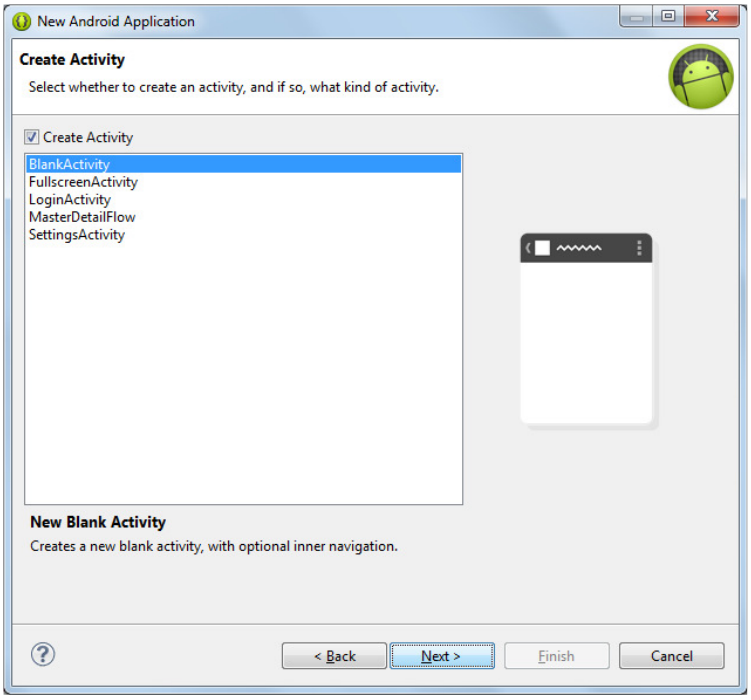
Criando o projeto Calculadora – Campos preenchidos

Agora na próxima seção (clicando em "Next") será aberta a tela de configurações do projeto (que já estão previamente definidas.Nela não iremos alterar nada, simplesmente vamos clicar em "Next". Na próxima tela escolhemos o ícone para a nossa aplicação (conforme demonstra a imagem seguinte). Fique a vontade para escolher seu ícone (Dica : escolha ícones que tenham de uma certa forma, relação com a aplicação em desenvolvimento).



Criando o projeto Calculadora – Definindo um ícone

Depois de definir seu ícone vamos para a próxima etapa, onde vamos escolher qual tipo de Activity iremos criar (por padrão, o BlackActivity), conforme demonstra a próxima imagem:



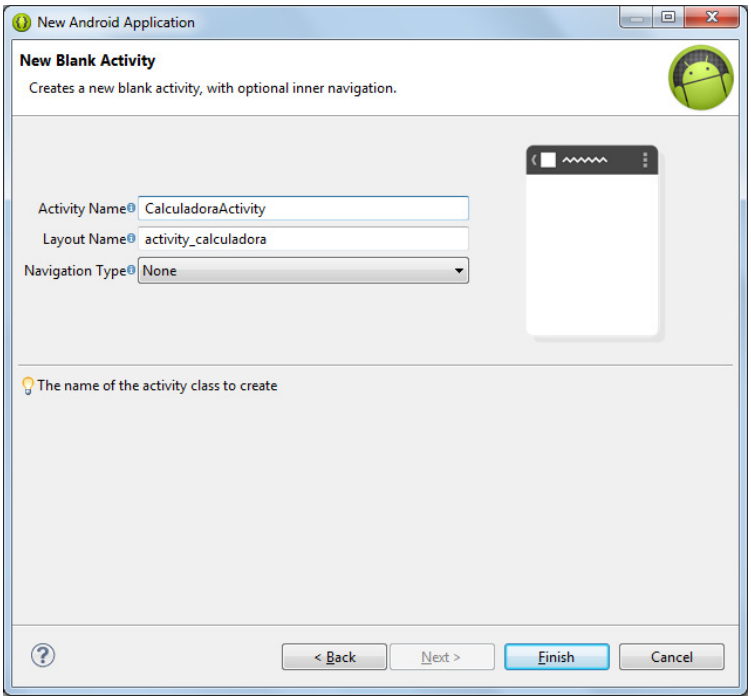
Criando o projeto Calculadora – Definindo a Activity

Agora na próxima seção (clicando em “Next”) vamos preencher as informações da Activity, conforme é mostrado abaixo:

Activity Name : CalculadoraActivity

Layout Name : activity_calculadora

Confira como ficou na figura seguinte:



Criando o projeto Calculadora – Informações preenchidas

Depois de preenchidas as informações, vamos criar o nosso projeto clicando no botão “Finish”. Feito isso o nosso projeto será criado.

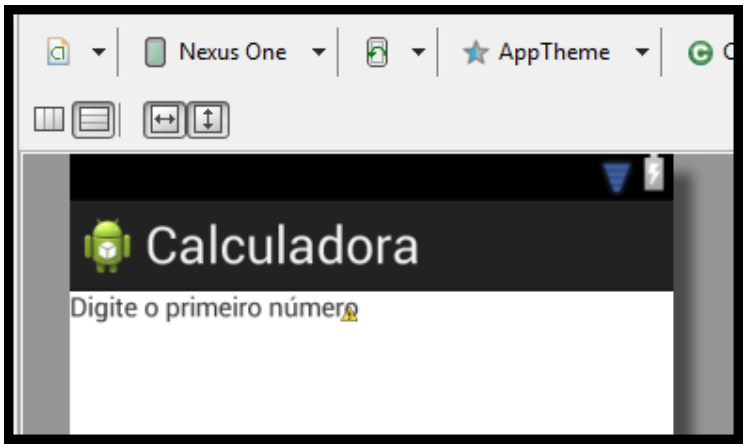
A primeira coisa que iremos fazer a alterar a estrutura de layout (trocar de **RelativeLayout** para **LinearLayout**, conforme já mostrado no Capítulo 3) que vai “comportar” nossos componentes.

Depois de alterar a estrutura de layout vamos selecionar o componente **TextView** na tela (cuja frase está escrito “Hello World”) e vamos alterar as seguintes propriedades, como segue:

TextView

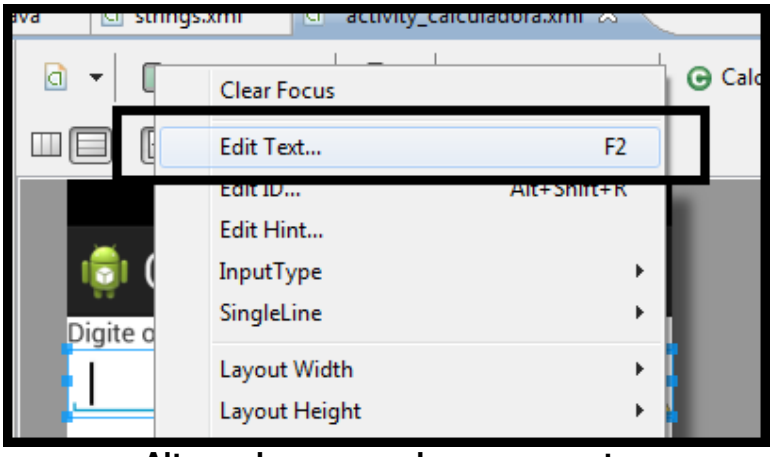
Propriedade	Valor
Text	Digite o primeiro número

Veja o resultado:



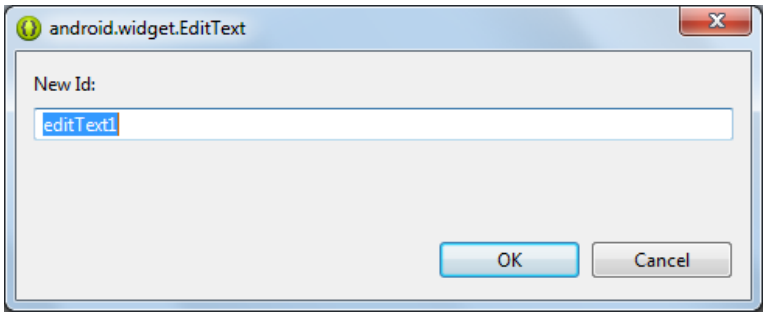
Tela da aplicação em desenvolvimento

Agora arraste e solte um componente “Plain Text” (**EditText**) abaixo do título, e em seguida, vamos atribuir um nome de componente para ele (por padrão, ele coloca “editText1”), clicando com o botão direito do mouse sobre ele e em seguida, selecione “Edit ID”. Confira na imagem abaixo:



Alterando o nome do componente

Feito isso, vai ser aberto uma caixa de diálogo conforme mostra a imagem seguinte:



Caixa para alteração do nome do componente

Conforme falei, o nome do componente é “editText1”. Agora vamos mudar o nome desse componente para “ednumero1” (sem aspas, é claro). Feito isso vamos clicar em “OK” para confirmar a alteração.

Porque alterar a sua ID ? Isso é necessário pois vamos “manipular” esse componente através do código Java, então nada mais justo do que trabalhar com componentes cujos nomes estejam de forma clara e organizada.

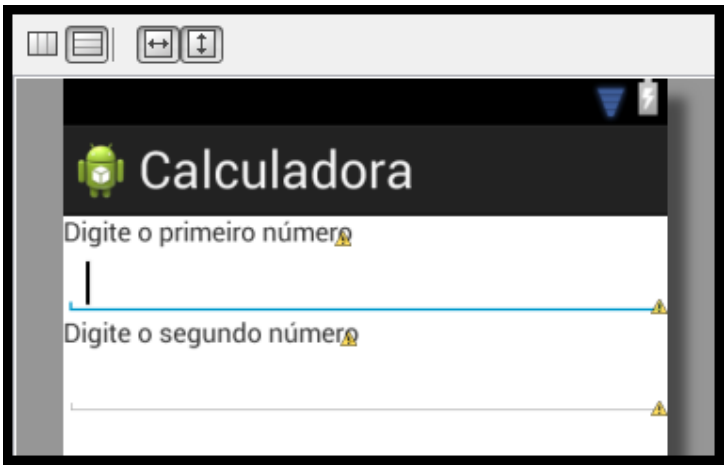
Agora arraste e solte um componente **TextView** abaixo da caixa de texto que inserimos, e em seguida altere as seguintes propriedades:

TextView

Propriedade	Valor
Text	Digite o segundo número

Logo após , arraste e solte um componente Plain Text (**EditText**) abaixo do componente acima inserido, e altere seu nome (ID) para “ednumero2” (conforme já foi mostrado).

Veja o resultado:



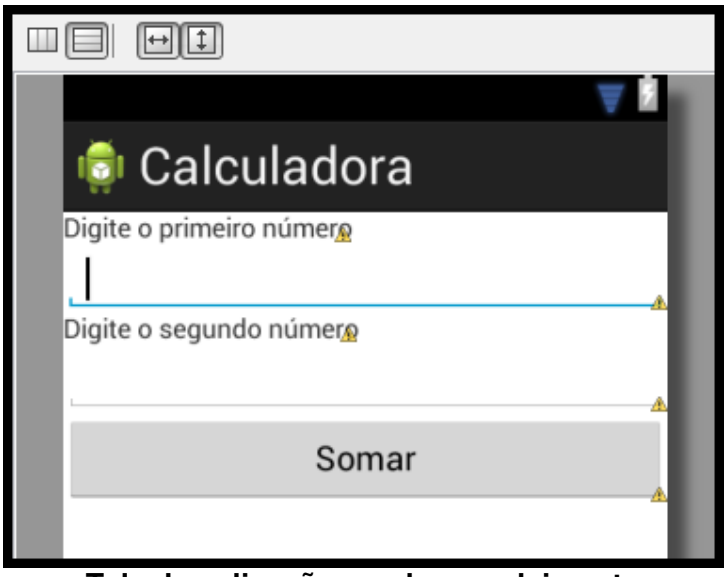
Tela da aplicação em desenvolvimento

Agora vamos adicionar um componente **Button** abaixo da caixa de texto, que vai ser o nosso botão de "somar" os números. Depois de adicionar, vamos alterar as suas propriedades, conforme é mostrado abaixo:

Button

Propriedade	Valor
Text	Somar
Width	fill_parent

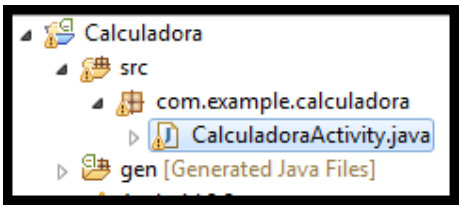
Depois disso vamos atribuir um nome (ID) para o componente, que vai se chamar "btsomar". Veja o resultado abaixo:



Tela da aplicação em desenvolvimento

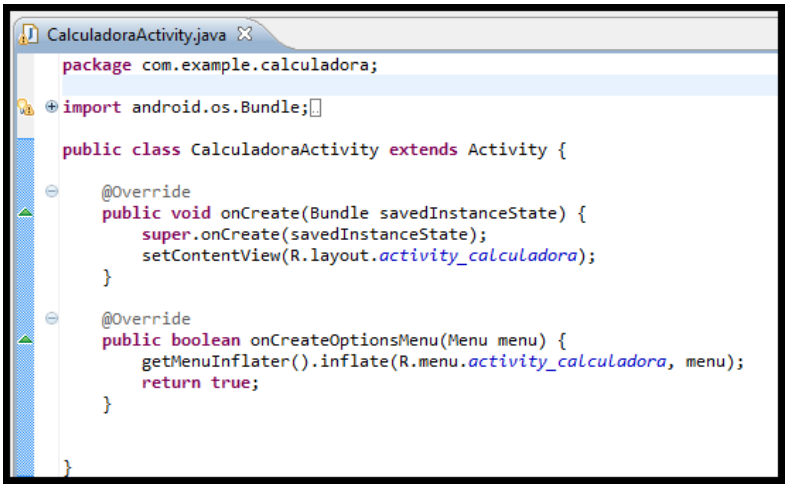
Para começarmos, vamos fazer o teste da nossa aplicação realizando somente soma dos números (implementaremos as outras operações restantes daqui a pouco). Agora salve o arquivo "activity_calculadora.xml" , para confirmar todas as alterações feitas, antes de trabalharmos com a parte da programação Java (que vai fazer uso dos componentes da tela da aplicação via código).

Depois de salvar o arquivo XML vamos abrir o arquivo "CalculadoraActivity.java" (situado no pacote "com.example.calculadora", que fica dentro do diretório "src"). Veja a imagem seguinte:



Arquivo "CalculadoraActivity.java"

Feito isso será aberto o seu conteúdo conforme é demonstrado na imagem seguinte:

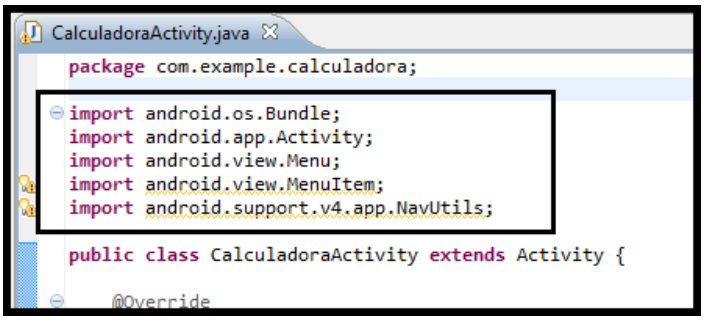


Conteúdo do arquivo "CalculadoraActivity.java"

Se você observar no código acima, na seção onde se declaram os pacotes, existe a seguinte instrução :

```
import android.os.Bunble;
```

Nessa linha se você observar (conforme demonstra a figura acima), existe um sinal de "+", que na verdade indica que há mais de uma importação (processo esse que o eclipse faz para "simplificar" e "organizar" a compreensão do código). Para você visualizar todos os pacotes utilizados basta clicar nesse sinal. Confira o resultado na próxima figura:



Visualizando todos os pacotes

Para começar, vamos importar alguns pacotes da plataforma Android que serão necessários para o desenvolvimento da nossa aplicação. Na seção onde se encontram os pacotes importados , vamos importar mais alguns pacotes digitando as seguintes linhas de comando abaixo:

```
import android.widget.*;
import android.view.*;
import android.app.*;
```

Agora no código do nosso programa, antes da linha:

```
@Override
```

Digite:

```
EditText ednumero1,ednumero2;
Button btsomar;
```

Agora vamos à explicação do código acima. Como você pode ver , os widgets também podem ser usados no nosso código Java. Se no código XML eu possuir um widget do tipo **EditText**, para acessar esse componente pelo Java é preciso fazer uso da classe **EditText**. Cada widget no XML possui o seu respectivo "em classe" Java, logo, se possui um widget **Button**, para acessá-lo devo fazer uso da classe **Button** e assim vai.

Agora dentro do método **onCreate** após a linha:

```
setContentView(R.layout.activity_calculadora);
```

Digite as seguintes linhas de código:

```
ednumero1 = (EditText) findViewById(R.id.ednumero1);
ednumero2 = (EditText) findViewById(R.id.ednumero2);
btsomar  = (Button) findViewById(R.id.btsomar);
```

Agora vou explicar as linhas de comando acima que adicionamos. A linha:

```
ednumero1 = (EditText) findViewById(R.id.numero1);
```

Faz referência ao primeiro **EditText**, através do método **findViewById** com o parâmetro *"R.id.numero1"*.

Se lembra do nome da primeira **EditText** que está no código XML? Ela se chama *"ednumero1"*.

Vamos entender. Observe que para fazer referência ao **EditText** pelo método **findViewById** eu passei o parâmetro "*R.id.numero1"*.

Na segunda instrução que digitamos, para fazer referência à segunda **EditText**, cujo nome é "*ednumero2"*, pelo método **findViewById**, passei o parâmetro "*R.id.numero2"*.

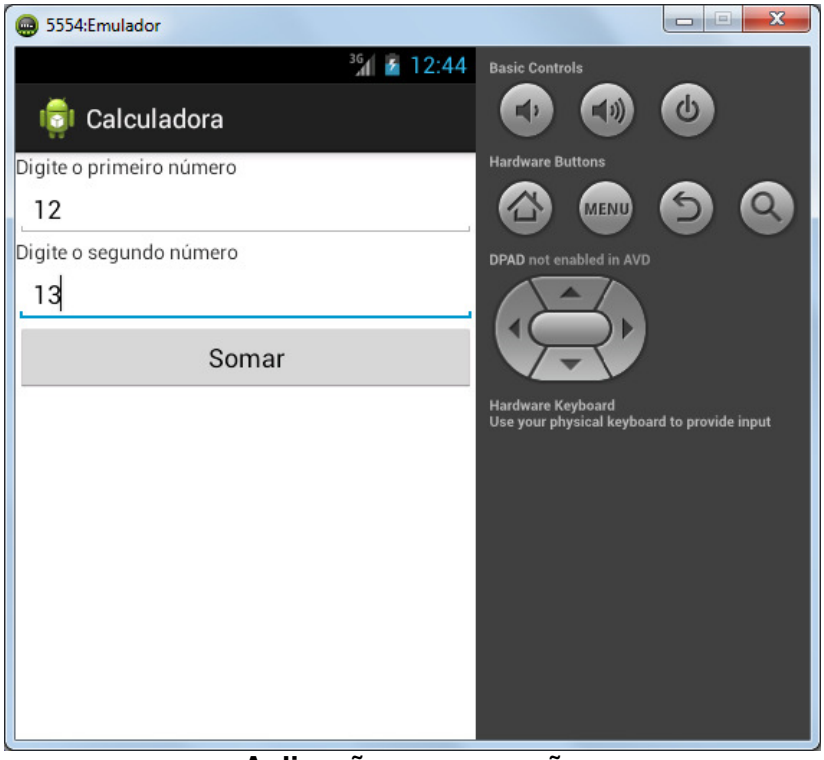
Como você pode ver, estou fazendo uso da classe **R** (situado dentro do diretório "gen", presente no pacote "com.example.calculadora") que funciona como interface entre o código Java e o arquivo XML. O procedimento é o mesmo para o componente **Button**.

Agora iremos adicionar um evento em nosso componente **Button** que será responsável por "detectar" toda vez que ele for "clicado", executando um conjunto de instruções após o evento (que vai consistir na soma dos números e na exibição do resultado). Para adicionarmos esse evento em nosso componente, basta escrevermos, após a última instrução que adicionamos, a seguinte linha de código:

```
btsomar.setOnClickListener(new View.OnClickListener() {

  @Override
  public void onClick(View arg0) {

      double num1 = Double.parseDouble(
      ednumero1.getText().toString());


      double num2 = Double.parseDouble(
      ednumero2.getText().toString());
      double soma = num1 + num2;

      AlertDialog.Builder dialogo = new
       AlertDialog.Builder(CalculadoraActivity.this);

      dialogo.setTitle("Resultado soma");

      dialogo.setMessage("A soma é " + soma);

      dialogo.setNeutralButton("OK", null);

      dialogo.show();

  }
});
```

Toda vez que eu clicar no botão ele irá mostrar o resultado da soma na tela através de uma caixa de mensagem. Ótimo!

Vamos executar a nossa aplicação? Para executar faça os mesmos procedimentos que já mostrei. O resultado da execução dessa aplicação você vê na figura seguinte:



Aplicação em execução

OBS: Provavelmente durante a execução da aplicação ao entrar com um número, deve ter surgido no dispositivo um teclado virtual (como mostra a figura acima), para ocultar ele é só pressionar ESC.

Irei descrever o código do evento de "clique". O método **setOnClickLisneter** serve para definir um evento de "clique" em um componente. Como parâmetro, criamos uma instância da interface **OnClickListener**, e dentro da mesma existe um método chamado **onClick**, que será disparado toda vez que o botão for clicado.

A linha:

```
double num1 =  Double.parseDouble(ednumero1.getText().toString());
```

Cria uma variável chamada *num1* e atribui a ela o valor que está contido dentro do componente identificado como *ednumero1*. Eu faço uso do método

parseDouble da classe **Double** pois o conteúdo é uma **String**. Observem que chamo o método **getText** de *ednumero1* para retornar o conteúdo. Diferente de muitos métodos de retorno **String**, esse método **getText** não retorna uma **String**, mais sim um tipo chamado **Editable**. Por isso chamei o método **toString** de **getText** para que me retornasse uma **String**. A descrição da próxima linha e a similar ao que já foi explicado.

Logo após a soma dos números que será armazenada na variável *soma* , vem o código em seguida:

```
AlertDialog.Builder dialogo = new
AlertDialog.Builder(CalculadoraActivity.this);

dialogo.setTitle("Resultado soma");

dialogo.setMessage("A soma é " + soma);

dialogo.setNeutralButton("OK", null);

dialogo.show();
```

Que mostra a soma dos números digitados na tela. Para conseguirmos exibir uma mensagem na tela, tivemos que fazer uso da classe **AlertDialog.Builder**, responsável por criar caixas de diálogo e exibi-las. Vamos aos comentários.
A linha de comando:

```
AlertDialog.Builder dialogo = new
AlertDialog.Builder(CalculadoraActivity.this);
```

Cria a instância da classe **AlertDialog.Builder** que será representada e guardada dentro da variável *dialogo*. Na linha seguinte:

```
dialogo.setTitle("Resultado soma");
```

Define o título da caixa de diálogo através do método **setTitle**. Na linha seguinte:

```
dialogo.setMessage("A soma é " + soma);
```

Define a mensagem a ser exibida através do método **setMessage**. Na linha seguinte:

```
dialogo.setNeutralButton("OK", null);
```

Define o botão “OK” da caixa de texto através do método **setNeutralButton**. O parâmetro **null** indica que nenhuma ação será executada quando o botão for clicado (simplesmente a caixa será fechada e nada mais). E para finalizar:

```
dialogo.show();
```

Que é responsável por “exibir” a mensagem na tela por imediato.

Agora vamos continuar as outras operações certo ? Retornaremos então para a tela da nossa aplicação e vamos adicionar mais 3 botões referentes as operações restantes. Vamos adicionar na tela mais três botões como segue (um em baixo do outro, conforme a sequência abaixo):

Button

Propriedade	Valor
Id	btsubtrair
Text	Subtrair
Width	fill_parent

Button

Propriedade	Valor
Id	btmultiplicar
Text	Multiplicar
Width	fill_parent

Button

Propriedade	Valor
Id	btdividir
Text	Dividir
Width	fill_parent

Depois de “finalizado” o que foi se pedido acima, veja como ficou a tela da nossa aplicação:

Tela da aplicação da calculadora

Conforme já havia falado, a tela da aplicação nada mais é do que uma estrutura XML. Vamos ver agora a estrutura XML que existe por trás dessa tela que acompanhamos na figura acima:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o primeiro número"
        tools:context=".CalculadoraActivity" />
    <EditText
        android:id="@+id/ednumero1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite o segundo número" />
```

```
    <EditText
        android:id="@+id/ednumero2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" />

    <Button
        android:id="@+id/btsomar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Somar" />

    <Button
        android:id="@+id/btsubtrair"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Subtrair" />

    <Button
        android:id="@+id/btmultiplicar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Multiplicar" />

    <Button
        android:id="@+id/btdividir"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Dividir" />

</LinearLayout>
```

Agora retornando para o código do arquivo “CalculadoraActivity.java”, vamos declarar mais três atributos (variáveis) que vão corresponder aos botões que representam as operações restantes, conforme destaca a linha em negrito:

```
:
Button btsomar, btsubtrair,btmultiplicar, btdividir;
:
```

Agora vamos atribuir para cada botão um evento de clique, fazendo com que eles efetuem a sua respectiva operação aritmética. Vamos continuar a codificação do método onCreate , digitando o seguinte código abaixo:

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

btmultiplicar=(Button)findViewById(R.id.btmultiplicar);

btdividir = (Button) findViewById(R.id.btdividir);

btsubtrair.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View arg0) {

      double num1 = Double.parseDouble
      (ednumero1.getText().toString());

      double num2 = Double.parseDouble
      (ednumero2.getText().toString());

       double soma = num1 - num2;

       AlertDialog.Builder dialogo = new
       AlertDialog.Builder(CalculadoraActivity.this);

       dialogo.setTitle("Resultado subtração");

       dialogo.setMessage("A subtração é " + soma);

       dialogo.setNeutralButton("OK", null);

       dialogo.show();

     }
    });


 btmultiplicar.setOnClickListener(new View.
 OnClickListener() {

    @Override
     public void onClick(View arg0) {

         double num1 = Double.parseDouble
         (ednumero1.getText().toString());

        double num2 = Double.parseDouble
        (ednumero2.getText().toString());

        double soma = num1 * num2;

         AlertDialog.Builder dialogo = new
         AlertDialog.Builder(CalculadoraActivity.this);

        dialogo.setTitle("Resultado multiplicação");

        dialogo.setMessage("A multiplicação é " + soma);

        dialogo.setNeutralButton("OK", null);

        dialogo.show();

       }
});

btdividir.setOnClickListener(new View.OnClickListener() {

     @Override
     public void onClick(View arg0) {
```

```
        double num1 = Double.parseDouble
        (ednumero1.getText().toString());

      double num2 = Double.parseDouble
      (ednumero2.getText().toString());

      double soma = num1 / num2;

      AlertDialog.Builder dialogo = new
      AlertDialog.Builder(CalculadoraActivity.this);

      dialogo.setTitle("Resultado divisão");

      dialogo.setMessage("A divisão é " + soma);

      dialogo.setNeutralButton("OK", null);

      dialogo.show();

} });
```

Depois de escrever o código acima, salve o arquivo e em seguida teste a aplicação. Veja o resultado na figura seguinte:

Aplicação da calculadora em execução

3.2) Desenvolvendo uma aplicação simples de compras

Agora para aprimorarmos o nosso conhecimento no desenvolvimento de aplicações para Android, vamos criar um outro aplicativo que consiste em um sistema de compras, bem simples. Em nossa aplicação terei disponível cinco produtos: Arroz (R$ 2,69) , Leite (R$ 5,00) , Carne (R$ 10,00), Feijão (R$ 2,30) e Refrigerante Coca-Cola (R$ 2,00). Nessa aplicação eu marco os itens que quero comprar e no final o sistema mostra o valor total das compras.

Na aplicação que iremos desenvolver vamos utilizar os seguintes widgets : **TextView**, **CheckBox** e **Button**.

Bom, vamos criar um novo projeto no Eclipse para Android chamado "SistemaDeCompras". Siga os dados do projeto abaixo:

Application Name: SistemaDeCompras

Project Name: SistemaDeCompras

Package Name : com.example.sistemadecompras

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: ComprasActivity

Layout Name : activity_compras

Depois de carregado e criado o projeto, vamos alterar a estrutura de layout padrão (**RelativeLayout**) para **LinearLayout**. Em seguida, modifique o componente **TextView** situado na tela, de acordo com a tabela abaixo:

TextView

Propriedade	Valor
Text	Escolha seu produto

Feito o que se foi pedido, adicione os seguintes componentes na sequência:

CheckBox

Propriedade	Valor
Text	Arroz (R$ 2,69)
Id	Chkarroz

CheckBox

Propriedade	Valor
Text	Leite (R$ 5,00)
Id	Chkleite

CheckBox

Propriedade	Valor
Text	Carne (R$ 9,70)
Id	Chkcarne

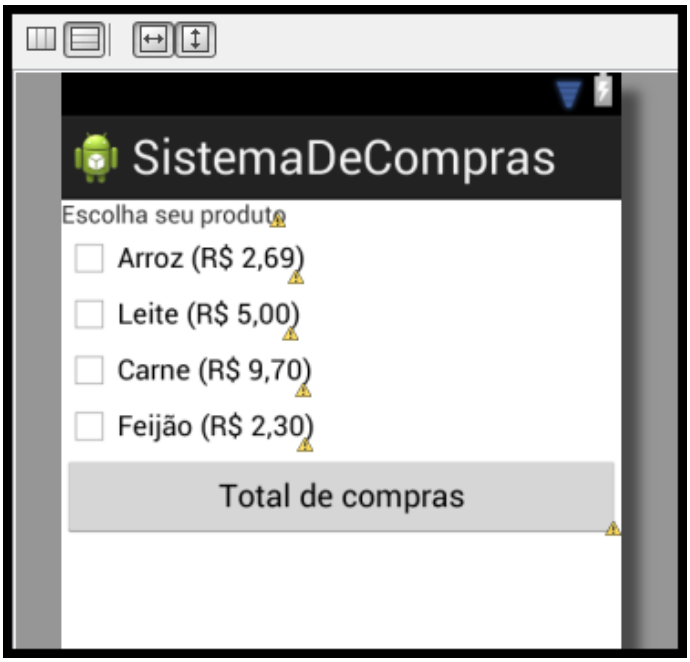
CheckBox

Propriedade	Valor
Text	Feijão (R$ 2,30)
Id	Chkfeijao

Button

Propriedade	Valor
Text	Total das compras
Id	Bttotal
Width	fill_parent

Ao final, o layout da nossa aplicação deve estar de acordo com a figura seguinte:



Layout da tela da aplicação

Vamos ver agora a estrutura XML da tela desta aplicação:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Escolha seu produto"
        tools:context=".ComprasActivity" />

    <CheckBox
        android:id="@+id/chkarroz"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Arroz (R$ 2,69)" />

    <CheckBox
        android:id="@+id/chkleite"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Leite (R$ 5,00)" />

    <CheckBox
        android:id="@+id/chkcarne"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Carne (R$ 9,70)" />
```

```
    <CheckBox
        android:id="@+id/chkfeijao"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Feijão (R$ 2,30)" />
    <Button
        android:id="@+id/bttotal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Total das compras" />

</LinearLayout>
```

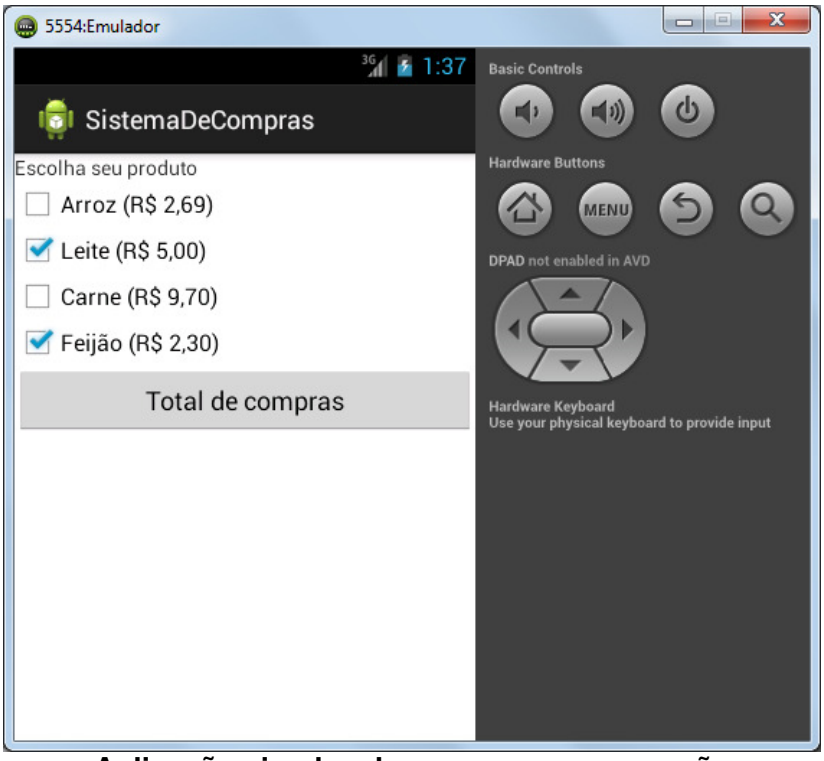
Agora vamos modificar o arquivo “ComprasActivity.java“. O código “completo” desse arquivo será como o código que é exibido abaixo:

```
package com.example.sistemadecompras;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.*;
import android.view.*;
import android.app.*;

public class ComprasActivity extends Activity {

      CheckBox chkarroz, chkleite, chkcarne, chkfeijao;

      Button bttotal;


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

        chkarroz = (CheckBox)
        findViewById(R.id.chkarroz);

        chkleite = (CheckBox)
        findViewById(R.id.chkleite);

        chkcarne = (CheckBox)
        findViewById(R.id.chkcarne);

        chkfeijao = (CheckBox)
        findViewById(R.id.chkfeijao);


        Button bttotal = (Button) findViewById(R.id.bttotal);

        bttotal.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View arg0) {
```

```
            double total =0;

            if(chkarroz.isChecked())
              total += 2.69;

            if(chkleite.isChecked())
              total += 5.00;

            if(chkcarne.isChecked())
             total += 9.7;

            if(chkfeijao.isChecked())
              total += 2.30;

            AlertDialog.Builder dialogo = new
            AlertDialog.Builder(ComprasActivity.this);

            dialogo.setTitle("Aviso");
            dialogo.setMessage("Valor total da compra :" +
            String.valueOf(total));

            dialogo.setNeutralButton("OK", null);

            dialogo.show();



             }
      });

    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.
        activity_compras, menu);
        return true;
    }


}
```

Agora vou descrever o código situado no método **onClick**. Dentro do método eu crio uma variável chamada *total* que armazena o valor total da compra. Observe que eu tenho quatro estruturas *if's* onde cada uma verifica se um determinado item foi marcado, se foi, incrementa o valor do item na variável *total*. No final é exibido o valor total das compras na tela.

Vamos roda nossa aplicação? O resultado você confere na figura seguinte:



Aplicação simples de compras em execução

3.3) Desenvolvendo uma aplicação de cálculo de salário

Agora vamos desenvolver uma nova aplicação que vai consistir em um sistema onde nós vamos digitar o salário de um funcionário permitindo escolher o seu percentual de aumento, que pode ser de 40% , 45% e 50%. Ao final de tudo, o sistema irá mostrar o salário reajustado com o novo aumento.

Para essa aplicação vamos utilizar os seguintes widgets : **TextView, EditText, RadioButton** e **Button**.

Bom, vamos lá! Crie um novo projeto Android com os seguintes dados abaixo:

Application Name: CalculoDeSalario

Project Name: CalculoDeSalario

Package Name : com.example.calculodesalario

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: SalarioActivity

Layout Name : activity_salario

Depois de carregado e criado o projeto, vamos alterar a estrutura de layout padrão (**RelativeLayout**) para **LinearLayout**. Em seguida, modifique o componente **TextView** situado na tela, de acordo com a tabela abaixo:

TextView

Propriedade	Valor
Text	Digite seu salário

Em seguida, adicione os seguintes componentes na sequência:

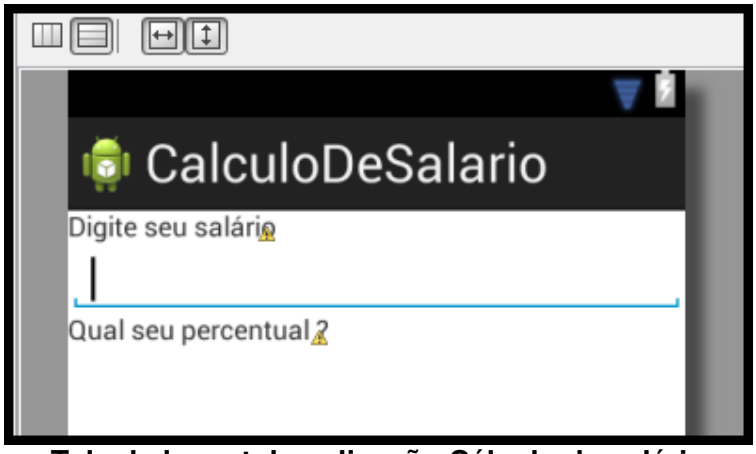
EditText (Number Decimal)

Propriedade	Valor
Text	
Id	Edsalario
Width	fill_parent

TextView

Propriedade	Valor
Text	Qual é o seu percentual ?

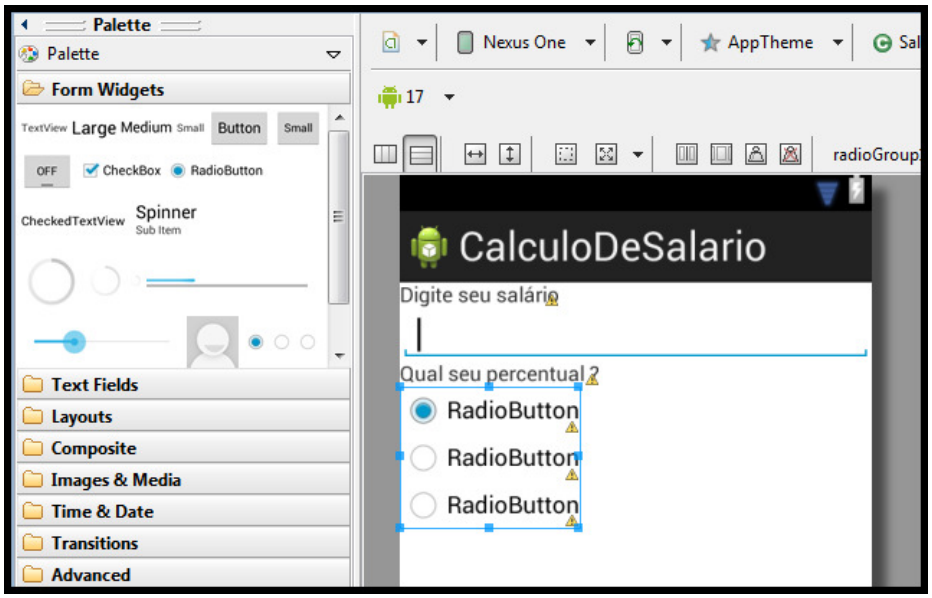
Seguindo os passos acima ate aqui, a aplicação deve estar de acordo com o dá figura abaixo:



Tela de layout da aplicação Cálculo de salário

Bom, agora vamos adicionar um componente, ou melhor, uma estrutura que será responsável por agrupar as RadioButtons dentro dela, que se chama **RadioGroup** (Para mais informações veja o Capítulo 4). O **RadioGroup** já

oferece por padrão três RadioButtons, que é quantidade necessária para a nossa aplicação. Clique a arraste o componente abaixo do ultimo widget adicionado. O resultado você confere na figura abaixo:



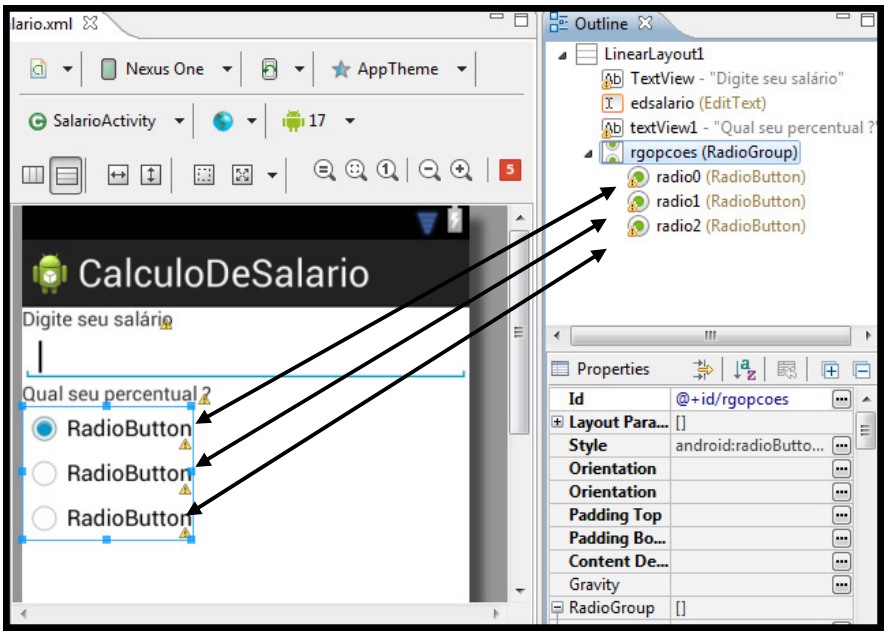
Estrutura RadioGroup inserida

Com o **RadioGroup** selecionado, modifique as propriedades abaixo:

RadioGroup

Propriedade	Valor
Width	fill_parent
Id	Rgopcoes

Observe que dentro do **RadioGroup** existem três elementos, cada um deles do tipo **RadioButton** e identificados por um nome. Se você observar no Eclipse, à direita da tela da aplicação, existe uma seção chamada “Outline”, onde nela posso visualizar toda a estrutura dos componentes que estão na minha aplicação. Confira na figura abaixo:



Guia "Outline"

Agora modifique as propriedades das RadioButons de acordo com as indicações abaixo:

radio0

Propriedade	Valor
Text	40%
Id	rb40

radio1

Propriedade	Valor
Text	45%
Id	rb45

radio2

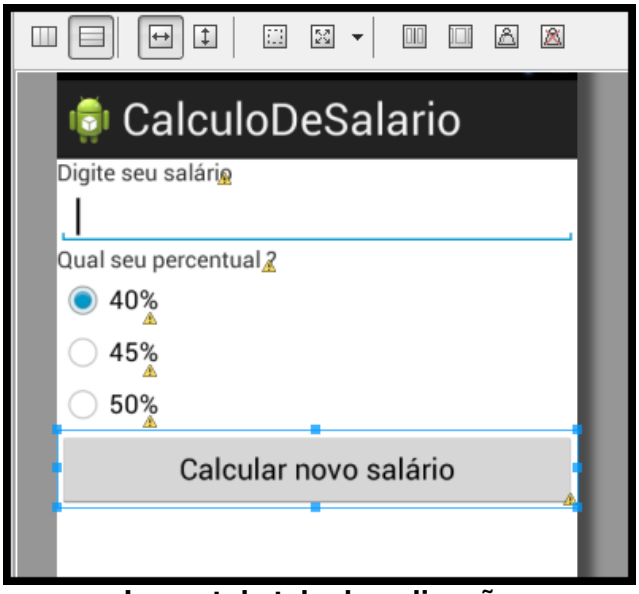
Propriedade	Valor
Text	50%
Id	rb50

Agora, vamos adicionar uma **Button**, simplesmente clicando e arrastando o componente na tela. Agora um detalhe, é para colocar esse componente na tela do dispositivo mas FORA da área do **RadioGroup**.

Depois de colocar o **Button**, modifique as propriedades abaixo:

Propriedade	Valor
Text	Calcular novo salário
Id	Btcalcular
Width	fill_parent

Depois de inserir todos os componentes citados, o layout da aplicação deve ficar de acordo com a figura abaixo:



Layout da tela da aplicação

Vamos analisar agora parte de um trecho de código produzido. Como havia falado acima, as **RadioButtons** precisam ficar dentro de uma estrutura chamada **RadioGroup** certo ? Vamos ver como isso é estruturado dentro de um código XML, como você confere abaixo:

```
<RadioGroup
        android:id="@+id/rgopcoes"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >

        <RadioButton
            android:id="@+id/rb40"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true"
            android:text="40%" />

        <RadioButton
            android:id="@+id/rb45"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="45%" />
```

```
        <RadioButton
            android:id="@+id/rb50"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="50%" />

    </RadioGroup>
```

Observe acima que logo após a definição da estrutura **RadioGroup**, existe dentro dela as RadioButtons, que serão utilizadas na aplicação.

Agora confira a estrutura XML da tela da aplicação em desenvolvimento:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite seu salário"
        tools:context=".SalarioActivity" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberDecimal" >

        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Qual seu percentual ?" />

    <RadioGroup
        android:id="@+id/rgopcoes"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >

       <RadioButton
            android:id="@+id/rb40"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true"
            android:text="40%" />
```

```
        <RadioButton
             android:id="@+id/rb45"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="45%" />

        <RadioButton
             android:id="@+id/rb50"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="50%" />

    </RadioGroup>


    <Button
        android:id="@+id/btcalcular"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Calcular novo salário" />

</LinearLayout>
```

No arquivo “SalarioActivity.java” vamos colocar o seguinte código abaixo:

```
package com.example.calculodesalario;

import android.os.Bundle;
import android.app.Activity;
import android.widget.*;
import android.view.*;
import android.app.*;

public class SalarioActivity extends Activity {

  RadioGroup rgopcoes;
  Button btcalcular;

  EditText edsalario;

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

    edsalario = (EditText)findViewById
    (R.id.edsalario);

    rgopcoes = (RadioGroup) findViewById
    (R.id.rgopcoes);

    btcalcular = (Button) findViewById(R.id.btcalcular);

    btcalcular.setOnClickListener(new View.
    OnClickListener() {
```

```
        @Override
        public void onClick(View arg0) {

        double salario = Double.parseDouble
        (edsalario.getText().toString());

        int op = rgopcoes.getCheckedRadioButtonId();

        double novo_salario = 0;

        if(op==R.id.rb40)
            novo_salario = salario + (salario * 0.4);
        else if(op==R.id.rb45)
            novo_salario = salario + (salario * 0.45);
        else
            novo_salario = salario + (salario * 0.5);


        AlertDialog.Builder dialogo = new
        AlertDialog.Builder(SalarioActivity.this);

        dialogo.setTitle("Novo salário");

        dialogo.setMessage("Seu novo salário é : R$" +
        String.valueOf(novo_salario));

        dialogo.setNeutralButton("OK", null);

        dialogo.show();


                    }
        });
    }


}
```

Vamos à explicação de alguns códigos interessantes. Dentro do método **onClick**, eu realizo o cálculo do novo salário do funcionário. Os primeiros códigos do evento são similares de programas anteriores que já foram devidamente explicados. A linha:

```
int op = rg.getCheckedRadioButtonId();
```

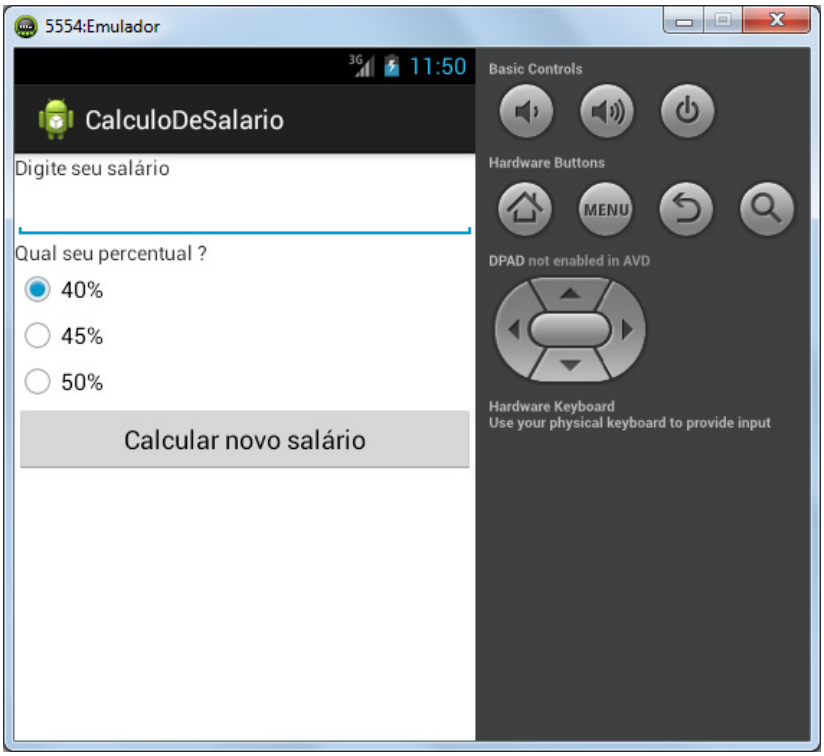
Cria uma variável *op* e retorna para ela o Id da opção selecionada, ou seja, qual **RadioButton** foi selecionada.

Agora na condição:

```
if(op==R.id.rb40)
```

Verifico se a opção de 40% foi selecionada, se foi selecionada, realiza o cálculo do salário com o reajuste de 40%. A mesma explicação e válida para o cálculo

dos outros reajustes. Agora vamos executar a nossa aplicação. O resultado você vê na figura seguinte:



Aplicação de cálculo de salário em execução

3.4) Desenvolvendo uma aplicação de lista de contatos

Agora vamos fazer uma nova aplicação em Android que consiste em uma aplicação de lista de contatos. Para essa aplicação iremos utilizar um componente chamado **ListView** , que seja bastante útil para esse tipo de situação (quando queremos exibir itens). Toda vez que clicarmos (ou melhor "tocarmos") em um contato na lista, será exibida uma mensagem com o nome do contato selecionado.

Vamos criar agora um novo projeto no Android Developer Tools, conforme os dados abaixo:

Application Name: ListaDeContatos

Project Name: ListaDeContatos

Package Name : com.example.listadecontatos

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: ListaContatosActivity

Layout Name : activity_lista_contatos

Vamos trocar a estrutura de layout padrão para o **LinearLayout**, e no componente **TextView**, dentro da sua propriedade “Text” digite a seguinte frase : “Escolha um contato:”.

Em seguida vamos adicionar o componente **ListView (**que se encontra na seção “Composite”). Seguindo o que foi se pedido, a tela da aplicação ficará de acordo com a seguinte figura:

Layout da tela da aplicação em desenvolvimento

Agora vamos criar um objeto (String Array) que vai armazenar os contatos que serão exibidos no componente, que iremos chamado de “contatos” (criar no arquivo “strings.xml”). Os contatos que estarão nessa lista são : "Aline","Lucas","Rafael","Gabriela" e "Silvana".

Depois de criar os contatos, selecione o objeto **ListView** que você adicionou e altere as seguintes propriedades.

ListView

Propriedade	Valor
Id	lista_contatos
Entries	@array/contatos

Agora vamos visualizar a estrutura XML do arquivo que forma a tela da aplicação acima:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Escolha um contato:"
        tools:context=".ListaContatosActivity" />

    <ListView
        android:id="@+id/lista_contatos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

    </ListView>

</LinearLayout>
```

Agora vamos no arquivo “ListaContatosActivity.java” para colocar o seguinte código abaixo (lembre-se antes de salvar o arquivo “activity_lista_contatos.xml”):

```
package com.example.listadecontatos;

import android.os.Bundle;
import android.widget.*;
import android.view.*;
import android.app.*;

public class ListaContatosActivity extends Activity {

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

      setContentView(R.layout.activity_lista_contatos);
      lista_contatos = (ListView) findViewById
      (R.id.lista_contatos);
```

```
        lista_contatos.setOnItemClickListener(new
        AdapterView.OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> parent,
        View view, int position,long id) {

         String nome = ((TextView) view).getText()
        .toString();

         AlertDialog.Builder dialogo = new
         AlertDialog.Builder(ListaContatosActivity.this)
         dialogo.setTitle("Contato");
         dialogo.setMessage("Contato selecionado: " +
         nome);
         dialogo.setNeutralButton("OK", null);
         dialogo.show();
        }
    });
   }
}
```

Como havia falado (e também como vocês podem conferir no código acima), quando se clica em um item, o sistema mostra uma mensagem do item selecionado (no caso, o nome contato selecionado). Isso é conseguido fazendo uso da interface **OnItemClickListener**, como mostra a instrução abaixo:

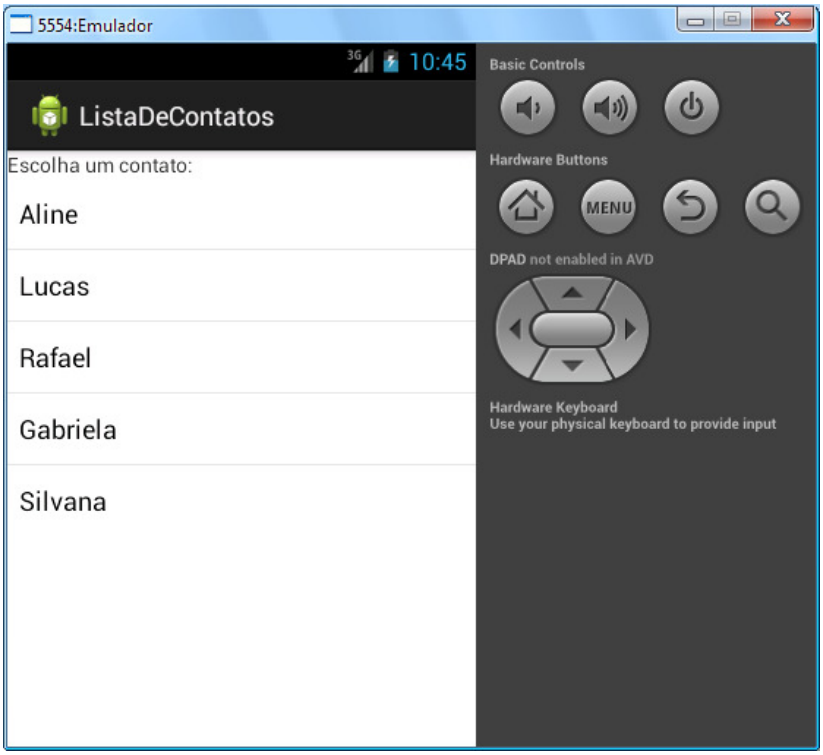
```
lista_contatos.setOnItemClickListener(new
AdapterView.OnItemClickListener() {

@Override
public void onItemClick(AdapterView<?> parent,
View view, int position,long id) {

   String nome = ((TextView) view).getText()
   .toString();

   AlertDialog.Builder dialogo = new
   AlertDialog.Builder(ListaContatosActivity.this)
   dialogo.setTitle("Contato");
   dialogo.setMessage("Contato selecionado: " + nome);
   dialogo.setNeutralButton("OK", null);
   dialogo.show();
      }
   });
```

Vamos analisar alguns trechos do código. A linha de comando:

```
String nome = ((TextView) view).getText().toString();
```

Guarda na variável *"nome"* o conteúdo retornado pelo objeto *"view"* (que contém o contato selecionado). Como o conteúdo precisa ser retornado para a variável que é do tipo **String**, foi preciso converte-lo em **TextView** para que o conteúdo fosse retornado em uma **String** (através do método **toString** situado em **getText**).

Vamos executar a aplicação. O resultado você vê na figura abaixo:



Aplicação de lista de contatos em execução

3.5) Desenvolvendo uma aplicação que visualiza imagens (com ImageView)

Agora vamos desenvolver uma aplicação básica que visualiza imagens através do uso o componente **ImageView**. Vamos criar um projeto com os seguintes dados abaixo:

Application Name: VisualizadorDeImagens

Project Name: VisualizadorDeImagens

Package Name : com.example.visualizadordeimagens

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: VisualizadorImagensActivity

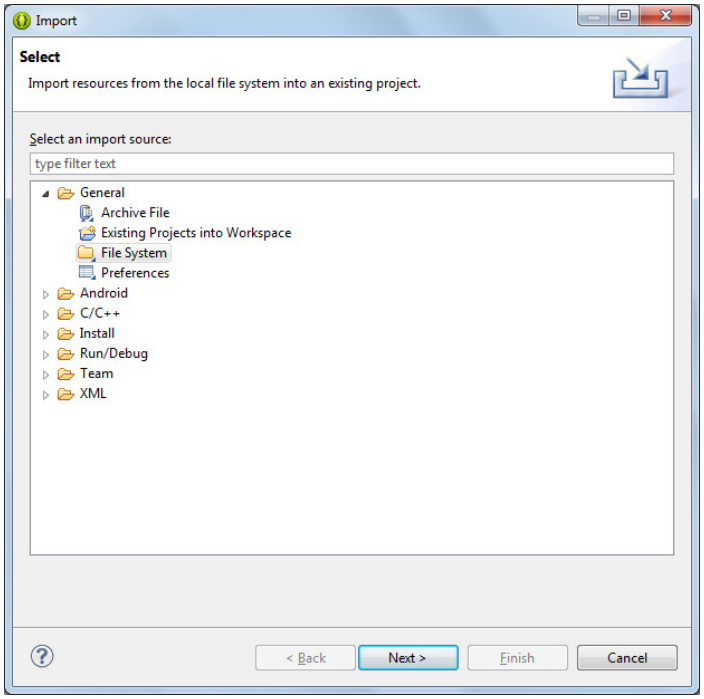
Layout Name : activity_visualizador_imagens

Depois de criado o projeto vamos trocar a estrutura de layout para o **LinearLayout** em seguida "apague" o componente **TextView** da tela.

Antes de iniciarmos a codificação do programa, quero que você coloque duas imagens JPEG (com a extensão .jpg), dentro da pasta "res/drawable-mdpi" (para esse projeto usei duas imagens chamadas "foto1.jpg" e "foto2.jpg").

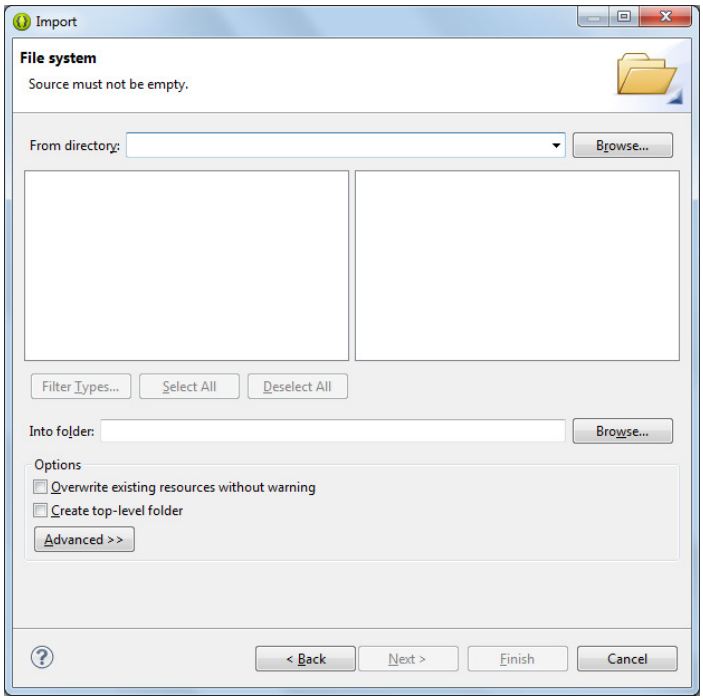
Irei mostrar aqui duas formas de se realizar essa tarefa. Começarei pela primeira forma, que considero a mais tradicional de todas.

Para importar um arquivo, clique com o botão direito do mouse sobre a pasta "res/drawable-mdpi" e selecione "Import", depois selecione "File System" (Que se encontra dentro da pasta "General", conforme mostra a figura abaixo) e em seguida clique em "Next".



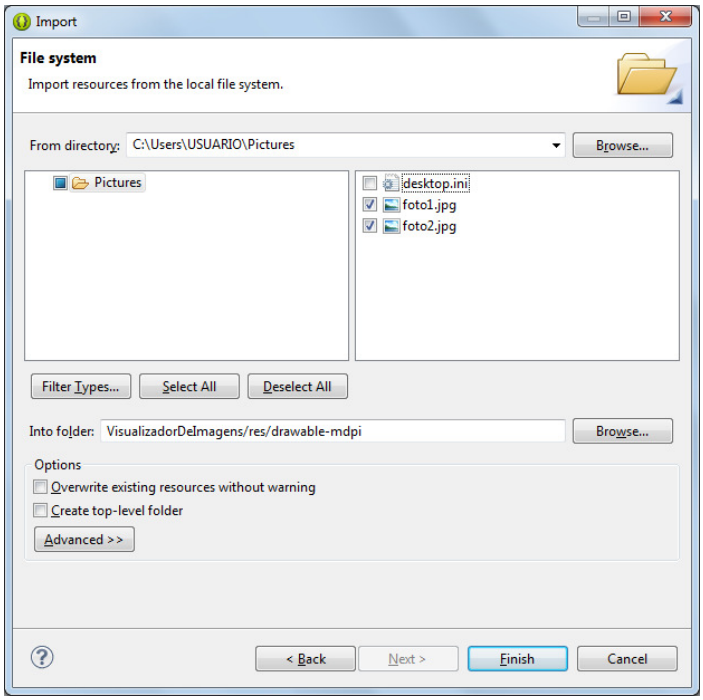
Selecionando a opção "File System"

Após clicar em "Next" será exibida a caixa de diálogo como demonstra a figura abaixo:



Caixa de diálogo "File System"

Clique no botão "Browse..." para selecionar o diretório onde se encontram as imagens. Feito isso, marque os dois arquivos (imagens) para que eles sejam importados para a pasta "res/drawable-mdpi" . Veja a figura abaixo:

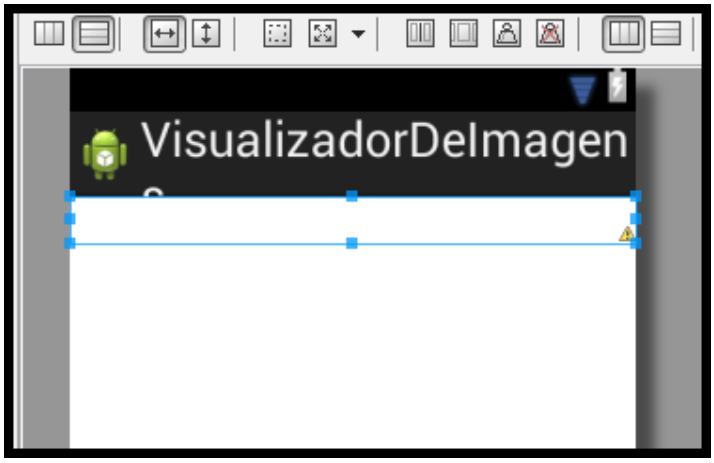


Importando as imagens para o projeto

Depois disso, é só clicar em “Finish” para importar as imagens para o projeto.

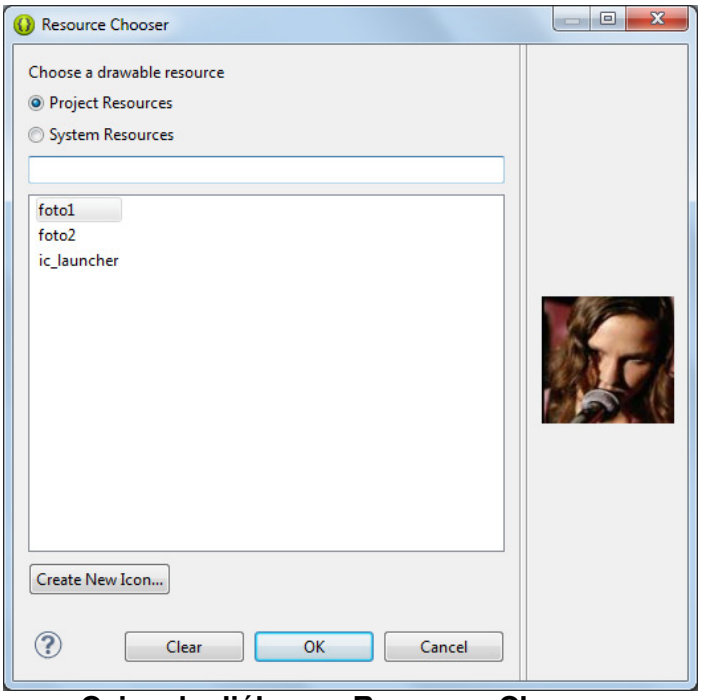
A segunda forma, que diria que é a mais fácil de todas, é você ir até o diretório onde se encontram as imagens , para que você em seguida possa selecioná-las, e logo após copiá-las (o famoso “Ctrl+C”). Feito isso vá até o projeto que criamos para selecionarmos o diretório “drawable-mpdi” para colarmos as imagens dentro da pasta (simples não ?).

Vamos adicionar dentro da tela da nossa aplicação uma estrutura **LinearLayout** (Horizontal), que se encontra na guia “Layouts”, simplesmente arrastando o componente para a tela da aplicação. O resultado você confere na figura abaixo:



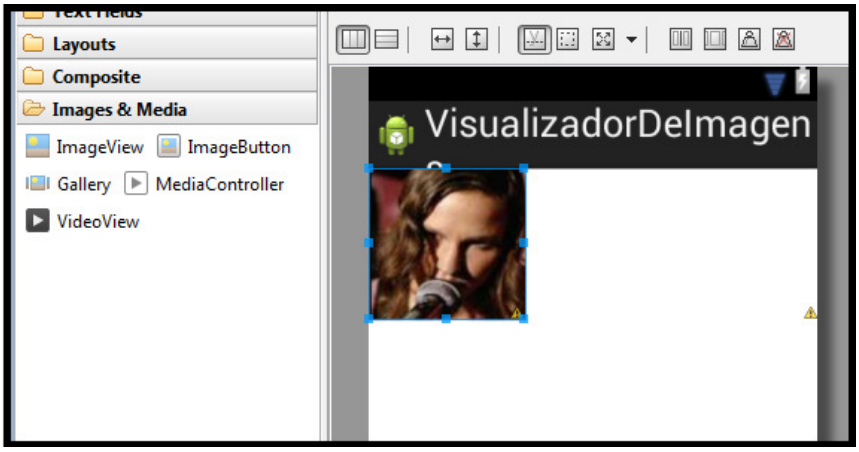
Estrutura LinearLayout inserida

Agora dentro da estrutura **LinearLayout** que adicionamos acima, vamos inserir o componente **ImageView** (que se encontra na guia “Images & Media”). Quando colocamos o componente no local desejado da tela, é exibido a seguinte caixa de diálogo:



Caixa de diálogo – Resource Chooser

Nesta caixa de diálogo escolhemos a imagem que o nosso componente vai assumir inicialmente. Iremos escolher a imagem chamada "foto1" (conforme é demonstrado acima). Depois de escolher a imagem clique em "OK". Veja o resultado em seguida:



Resultado da operação

Agora vamos alterar a propriedade do componente **ImageView** conforme abaixo:

ImageView

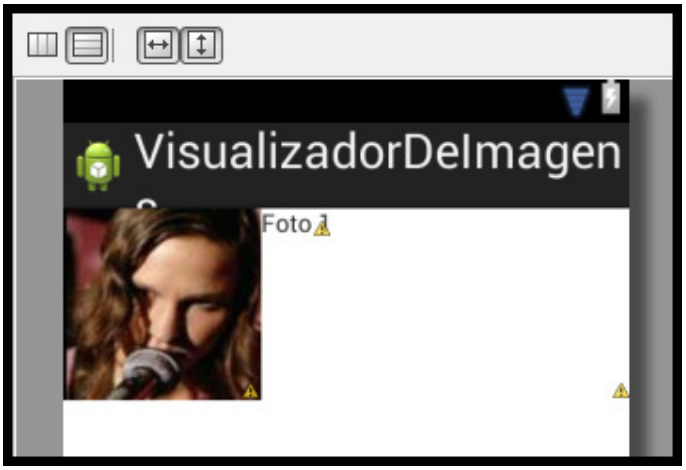
Propriedade	Valor
Id	imgfoto

Agora vamos adicionar um componente **TextView** que ficará ao lado da imagem. Altere suas propriedades conforme a tabela abaixo:

TextView

Propriedade	Valor
Id	txtinformacao
Text	Foto 1

Seguindo os passos acima, o resultado do layout deve ficar de acordo com a figura abaixo:



Layout da aplicação

Agora vamos adicionar na sequência dois componentes do tipo **Button**, só que esses dois componentes vão estar dentro da tela da aplicação e fora (e também abaixo) da estrutura de layout que adicionamos. Segue abaixo as propriedades que precisam ser modificadas:

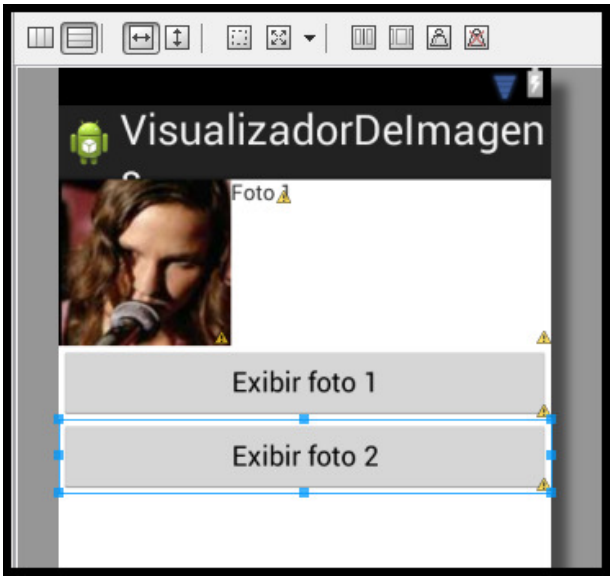
Button

Propriedade	Valor
Id	btfoto1
Text	Exibir foto 1
Width	fill_parent

Button

Propriedade	Valor
Id	Btfoto2
Text	Exibir foto 2
Width	fill_parent

Depois de seguir todos os passos descritos acima, a aplicação tem que estar de acordo com a figura abaixo:



Layout da aplicação

Agora vamos no arquivo "VisualizadorImagensActivity.java" para colocarmos o código em seguida (lembre-se de salvar o arquivo "activity_visualizador_imagens.xml" antes de escrever o código):

```
package com.example.visualizadordeimages;

import android.os.Bundle;
import android.app.Activity;
import android.widget.*;
import android.view.*;


public class VisualizadorImagensActivity extends Activity {

  ImageView imgfoto;

  Button btfoto1,btfoto2;

  TextView txtinformacao;
```

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

   setContentView
   (R.layout.activity_visualizador_imagens);

   imgfoto = (ImageView) findViewById(R.id.imgfoto);

   btfoto1 = (Button) findViewById(R.id.btfoto1);
   btfoto2 = (Button) findViewById(R.id.btfoto2);

   txtinformacao = (TextView) findViewById
   (R.id.txtinformacao);

   btfoto1.setOnClickListener(new View.
   OnClickListener() {

    @Override
    public void onClick(View arg0) {

      imgfoto.setImageResource
      (R.drawable.foto1);

      txtinformacao.setText("Foto 1");

     }
    });

  btfoto2.setOnClickListener(new View.OnClickListener()
   {

     @Override
     public void onClick(View arg0) {

        imgfoto.setImageResource
        (R.drawable.foto2);
        txtinformacao.setText("Foto 2");

      }
     });

   }


}
```

Agora vamos analisar alguns trechos de códigos. Vamos no evento Click referente a abertura da primeira imagem. O código:

```
imgfoto.setImageResource(R.drawable.foto1);
```

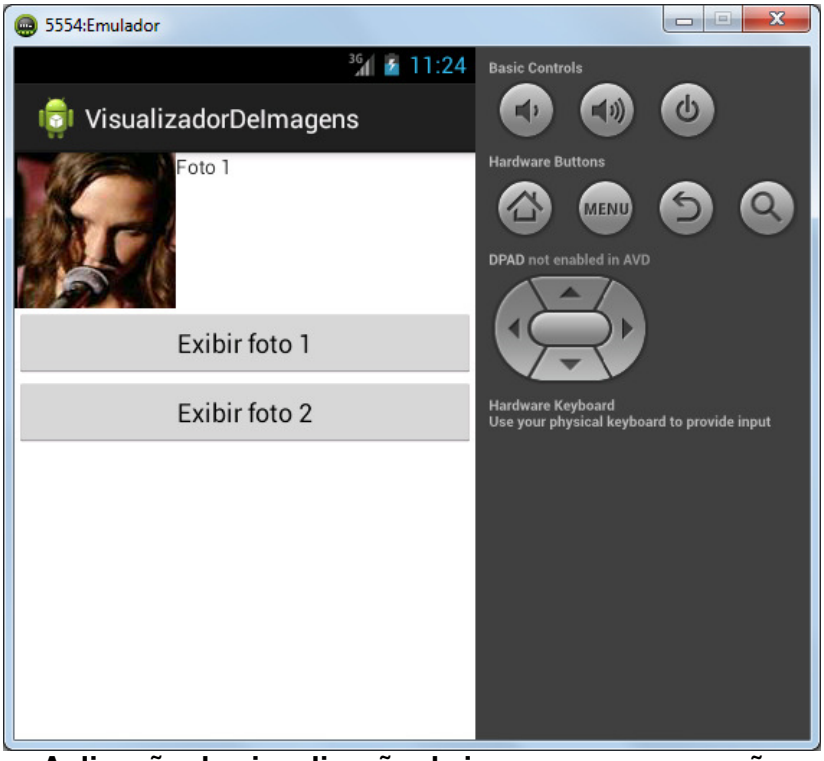
É responsável por abrir a imagem "foto1.jpg" e exibi-la no componente. Observe que foi passado o parâmetro *"R.drawable.foto1"* onde *"drawable"* corresponde a pasta e "*foto1"* corresponde ao arquivo *"foto1.jpg"*. Logo após vem o código:

```
txtinformacao.setText("Foto 1");
```

Cuja finalidade é mudar o título da **TextView** , de acordo com a String passada como parâmetro.

O comentário acima é o mesmo para o segundo botão referente à abertura da segunda imagem.

Vamos executar a nossa aplicação. O resultado você vê nas imagens abaixo:



Aplicação de visualização de imagens em execução

Capitulo 4 Trabalhando com mais de uma tela em uma aplicação

Até agora as aplicações que desenvolvemos tinham somente uma única tela, mas, sabemos que algumas aplicações possuem normalmente mais de uma tela. A partir de agora iremos aprender como inserir e gerenciar várias telas em uma aplicação Android através dos exemplos que serão demonstrados nesse capítulo.

Para começarmos, vamos criar um novo projeto Android com os seguintes dados abaixo:

Application Name: TrocaDeTelas

Project Name: TrocaDeTelas

Package Name : com.example.trocadetelas

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: TrocaTelasActivity

Layout Name : tela_principal

Altere a estrutura de layout da sua aplicação para o **LinearLayout** e em seguida altere o componente **TextView** de acordo com a tabela abaixo.

TextView

Propriedade	Valor
Text	Você está na tela principal

Agora adicione um componente **Button** e modifique as seguintes propriedades:

Button

Propriedade	Valor
Id	bttela2
Layout width	fill_parent
Text	Ir pata tela 2

Seguindo os passos acima, a aplicação deve estar de acordo com a figura abaixo:

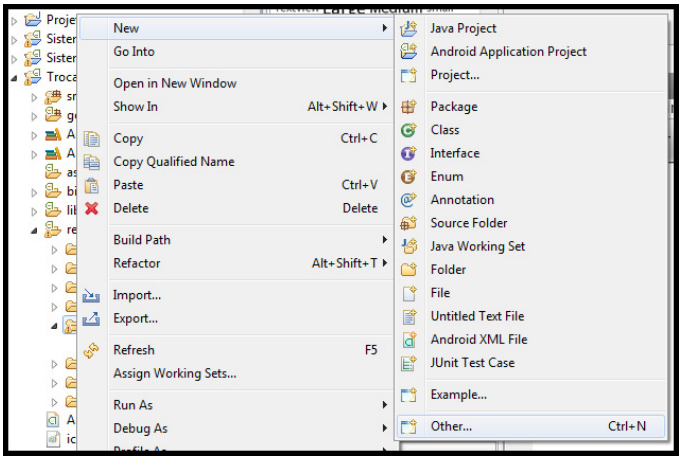
Layout da tela 1

Vejamos agora o código XML da tela da nossa aplicação:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Você está na tela principal"
        tools:context=".TrocaTelasActivity" />


    <Button
        android:id="@+id/bttela2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Ir para tela 2" />

</LinearLayout>
```

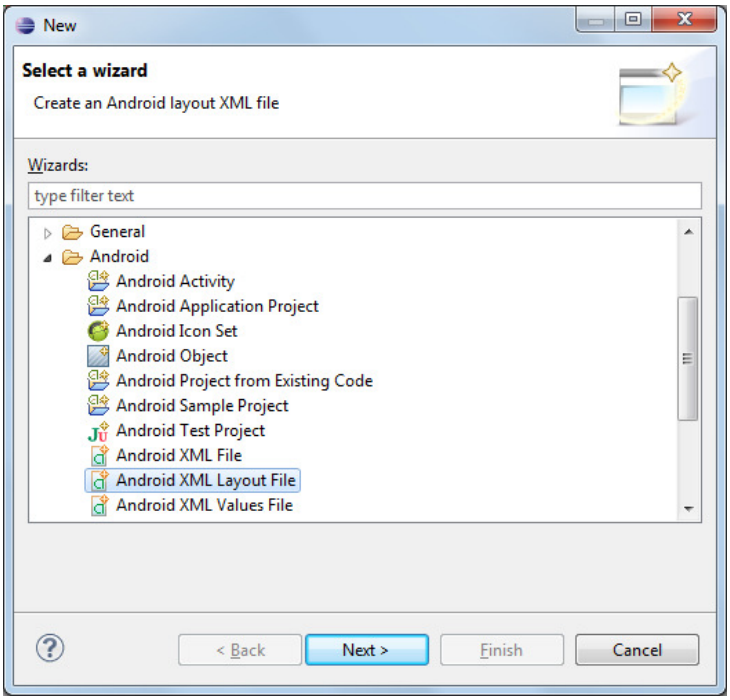
Nossa primeira tela está pronta, muito bem! Agora vamos criar uma nova tela para a nossa aplicação. O nome do arquivo que vai representar a segunda tela da nossa aplicação vai se chamar "tela2.xml" (um arquivo XML). Conforme já foi explicado (e explico novamente aqui), todos os arquivos que representam a tela da aplicação devem estar dentro do diretório "layout" (situado dentro da pasta "res" do projeto), logo, vamos criar o nosso arquivo dentro desse diretório.

Para criarmos um novo arquivo XML dentro do diretório "layout" basta clicar com o botão direito sobre a pasta e em seguida clicar em "New" e logo após "Other", confira na figura abaixo:



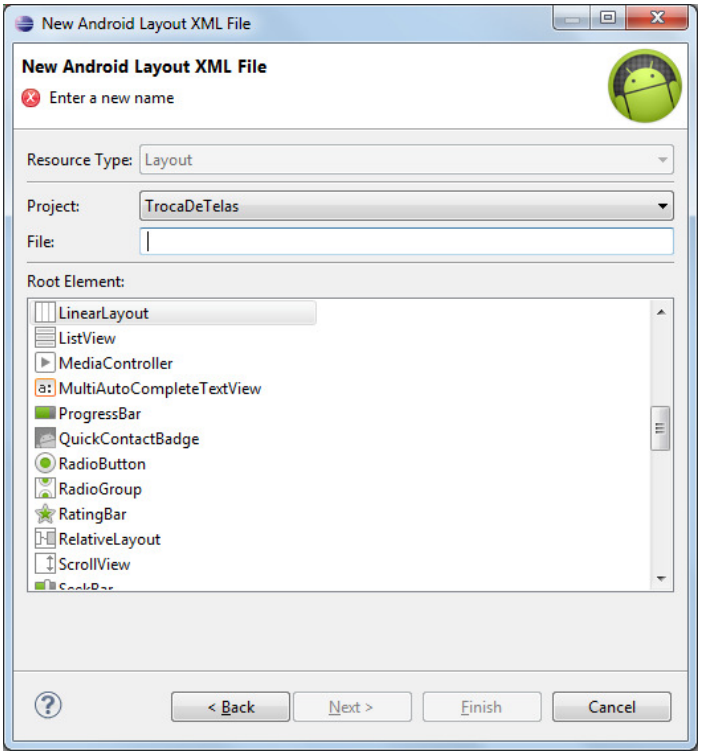
Criando um arquivo XML

Feito isso será aberto uma tela com várias pastas, você irá expandir a pasta "Android" e em seguida vai selecionar a opção "Android Layout XML File", conforme mostra a figura seguinte:



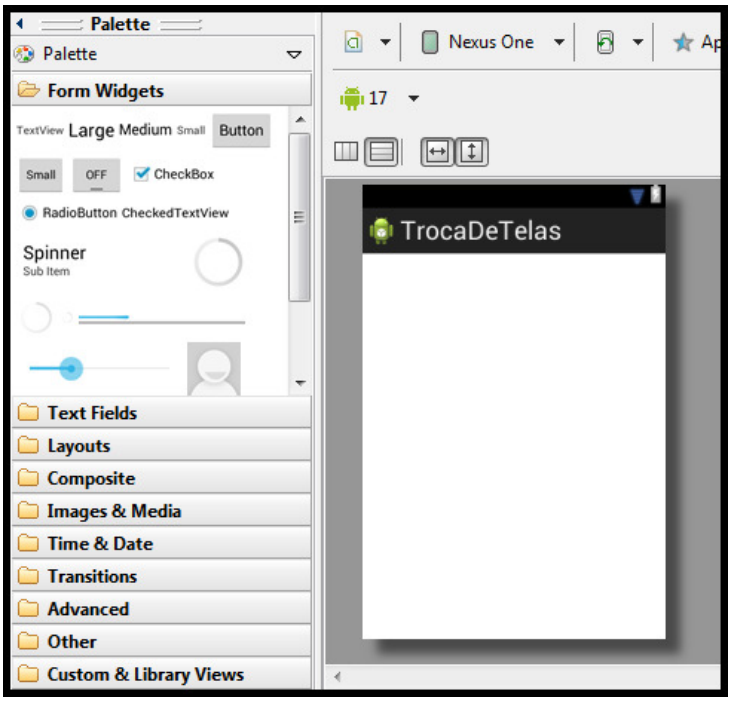
Android XML Layout File

Após selecionar a opção acima clique em “Next” que em seguida será aberta a tela seguinte:



New Android XML Layout File

Por padrão a estrutura de layout padrão selecionada é **LinearLayout** (que será a estrutura que iremos usar). Agora no campo "File" digite "tela2" e em seguida clique em "Finish" para que o arquivo seja gerado. Veja o resultado na figura seguinte:



Tela de layout em branco

Por padrão quando criamos a tela da aplicação utilizando o **LinearLayout**, ele cria com a orientação "vertical" (essa é que vamos utilizar, portanto, não vamos mexer na estrutura).

Agora vamos adicionar os seguintes componentes, na sequência:

TextView

Propriedade	Valor
Text	Você está na tela 2

Button

Propriedade	Valor
Id	bttelaprincipal
Width	fill_parent
Text	Ir pata tela principal

Seguindo os passos acima, o layout do arquivo “tela2.xml” deve estar de acordo com a figura abaixo:

Layout da tela 2

Vejamos agora o código XML da tela da nossa aplicação:

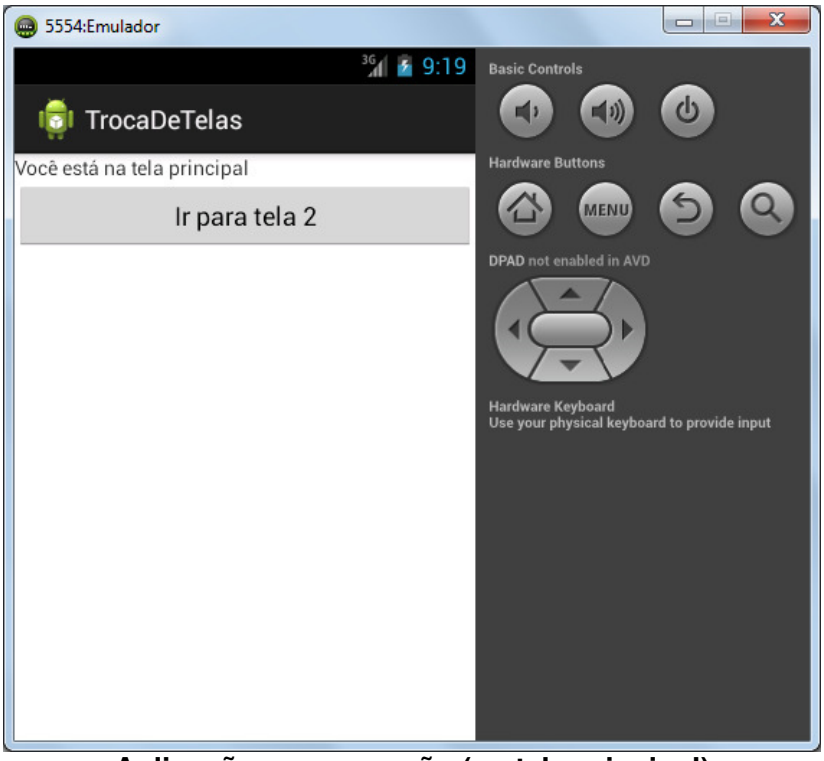
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Você está na tela 2" />

    <Button
        android:id="@+id/bttelaprincipal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Ir para tela principal" />

</LinearLayout>
```

Agora vamos no arquivo “TrocaTelasActivity.java” para digitarmos o código abaixo:

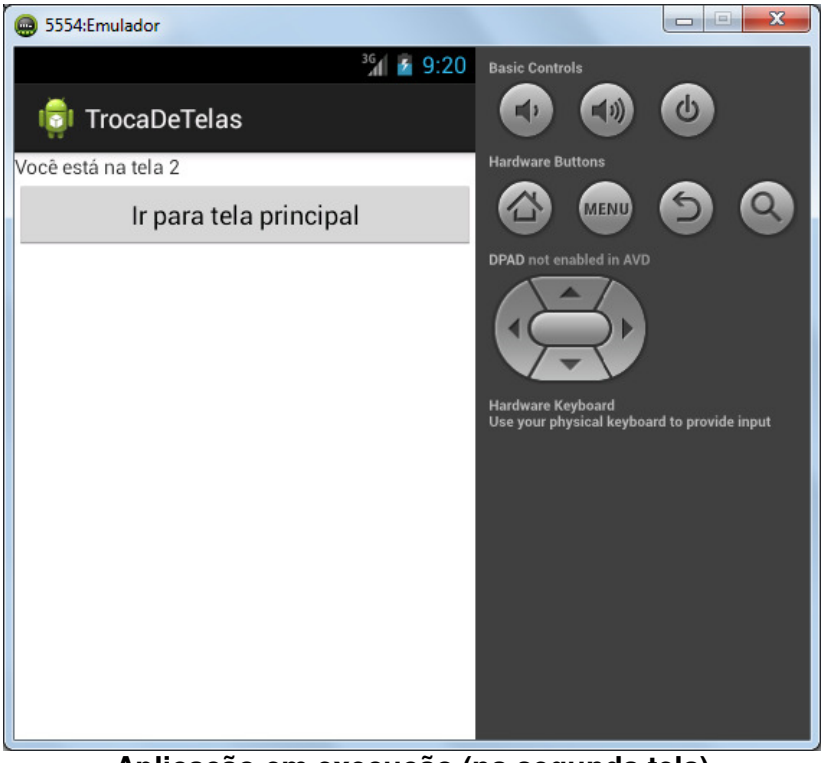
```
package com.example.trocadetelas;

import android.os.Bundle;
import android.app.Activity;
import android.widget.*;
import android.view.*;
```

```
public class TrocaTelasActivity extends Activity {

      Button bttelaprincipal, bttela2;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        CarregarTelaPrincipal();
    }
    public void CarregarTelaPrincipal()
    {
      setContentView(R.layout.tela_principal);
      bttela2 = (Button) findViewById(R.id.bttela2);
      bttela2.setOnClickListener(new
       View.OnClickListener() {

         @Override
         public void onClick(View v) {

               CarregarTela2();

           }
         });
    }

    public void CarregarTela2()
    {
      setContentView(R.layout.tela2);
      bttelaprincipal = (Button) findViewById
      (R.id.bttelaprincipal);
      bttelaprincipal.setOnClickListener(new
       View.OnClickListener() {

           @Override
            public void onClick(View v) {

               CarregarTelaPrincipal();

           }
         });
    }
}
```

Observem que nesta classe eu criei dois métodos : **CarregaTelaPrincipal** e **CarregaTela2**. Para toda aplicação que utilize mais de um layout (tela), o carregamento dos layouts e de seus respectivos widgets devem estar separados em funções desenvolvidas para esse propósito. Logo, o método **CarregaTelaPrincipal** carrega o layout principal e seus respectivos componentes, o mesmo válido para o método **CarregaTela2**, que carrega o layout da tela 2 e seus respectivos componentes. Feito isso, execute a aplicação. Veja o resultado abaixo:



Aplicação em execução (na tela principal)



Aplicação em execução (na segunda tela)

4.1) Desenvolvendo uma aplicação de cadastro

Com o que já aprendemos neste capítulo (e também nós capítulos e tópicos anteriores) já podemos desenvolver uma aplicação mais interessante. Para tornar o aprendizado mais interessante vamos criar uma aplicação de cadastro de pessoas. Nessa aplicação o usuário vai informar os seguintes dados : nome, profissão e idade.

Essa aplicação vai possuir três telas, cada uma com as seguintes funcionalidades:

Tela principal (tela com opções): Nessa tela da aplicação teremos um menu que dará acesso ao cadastro do usuário e a visualização dos usuários cadastrados.

Tela de cadastro: Nesse tela o usuário irá preencher os campos solicitados pela aplicação e em seguida o mesmo poderá cadastrar para que as informações sejam registradas.

Tela de visualização de dados: Nessa tela poderão ser visualizados os dados (usuários) cadastrados. Se nada foi cadastrado na aplicação, será exibida uma mensagem informando essa situação.

Nessa aplicação para armazenar os dados eu faço uso de uma estrutura de dados do tipo “lista” (duplamente encadeada). Nessa estrutura os dados são armazenados em sequência, e acessados tanto sequencialmente e na sua ordem inversa de cadastro (avançando e retrocedendo pelos registros).

Bom, vamos construir a nossa aplicação. Crie um novo projeto com os seguintes dados abaixo:

Application Name: AplicacaoDeCadastro

Project Name: AplicacaoDeCadastro

Package Name : com.example.aplicacaodecadastro

Minimum Required SDK : API 14: Android 4.0 (Ice Cream Sandwich)

Activity Name: AplicacaoCadastroActivity

Layout Name : activity_aplicacao_cadastro

Dentro da pasta “res/drawable-mdpi” coloque uma figura que que represente o ícone da nossa aplicação. Para esta aplicação utilizei uma imagem de extensão PNG chamada “profile.png”.

Altere a estrutura de layout da sua aplicação para o **LinearLayout** e em "APAGUE" o componente **TextView**. Feito isso adicione os seguintes componentes na sequência:

ImageView

Propriedade	Valor
Src (imagem)	profile.png

TextView

Propriedade	Valor
Text	Bem vindo a Aplicação de Cadastro de Pessoas.Este é um pequeno programa de demonstração de cadastro.Selecione uma das opções abaixo:

Button

Propriedade	Valor
Id	btcadastrarpessoas
Width	fill_parent
Text	Cadastrar pessoa

Button

Propriedade	Valor
Id	btlistarpessoas
Width	fill_parent
Text	Listar pessoas cadastradas

Seguindo os passos acima, a aplicação deve estar de acordo com a figura seguinte:

Layout da tela principal da aplicação

Vejamos agora a estutura XML da tela principal da nossa aplicação:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/profile" />
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bem vindo a Aplicação
        de Cadastro de Pessoas.Este é um pequeno
        programa de demonstração de cadastro.
        Selecione uma das opções abaixo:" />
    <Button
        android:id="@+id/btcadastrarpessoa"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Cadastrar pessoa" />
    <Button
        android:id="@+id/btlistarpessoas"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Listar pessoas cadastradas" />

</LinearLayout>
```

Agora vamos criar mais uma tela (arquivo de layout XML) para nossa aplicação. Para isso vamos clicar com o botão direito sobre a pasta "layout" e em seguida vamos chamar o recurso de criação de arquivos de layout XML para Android (o "Android XML Layout File", conforme já foi mostrado anteriormente no último exemplo). O nome do nosso arquivo de layout XML vai ser "cadastro" (a estrutura que vamos utilizar para essa tela será o **LinearLayout**).

Depois de criado o arquivo vamos adicionar os seguintes componentes na sequência:

ImageView

Propriedade	Valor
Src (Imagem)	profile

TextView

Propriedade	Valor
Text	Módulo de cadastro.Digite seus dados abaixo:

TextView

Propriedade	Valor
Text	Nome:

EditText (Plain Text)

Propriedade	Valor
Id	ednome
Text	(Deixar em branco)

TextView

Propriedade	Valor
Text	Profissão:

EditText (Plain Text)

Propriedade	Valor
Id	edprofissao
Width	fill_parent
Text	(Deixar em branco)

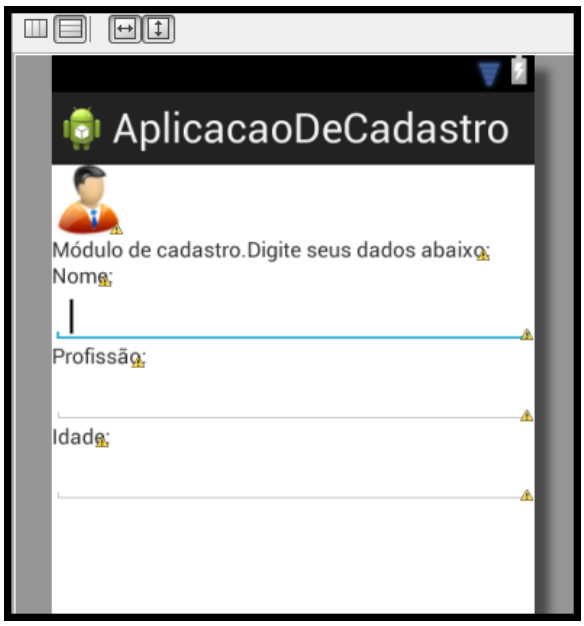
TextView

Propriedade	Valor
Text	Idade:

EditText (Plain Text)

Propriedade	Valor
Id	edidade
Width	fill_parent
Text	(Deixar em branco)

Seguindo os passos acima, a aplicação deve estar de acordo com a figura seguinte:



Layout da tela de cadastro

Bom, ainda não acabou. Agora vamos adicionar uma estrutura **LinearLayout** (Horizontal) que será responsável por organizar os botões de forma horizontal logo após a **EditText** que representa o campo "idade". Após adicionar essa estrutura, modifique a seguintes propriedades dela, conforme abaixo:

Propriedade	Valor
Gravity	center

A propriedade "Gravity" , similar a propriedade "Orientation", determina o alinhamento dos componentes dentro da estrutura, que no caso acima está alinhando os componentes de forma centralizada, ou seja, os componentes vão estar dispostos de forma horizontal (um ao lado do outro) e todos esses componentes estarão organizados de forma centralizada.

Seguindo o mesmo procedimento acima, vamos adicionar dois componentes do tipo **Button** dentro dessa estrutura, e vamos mudar as seguintes propriedades citadas abaixo.

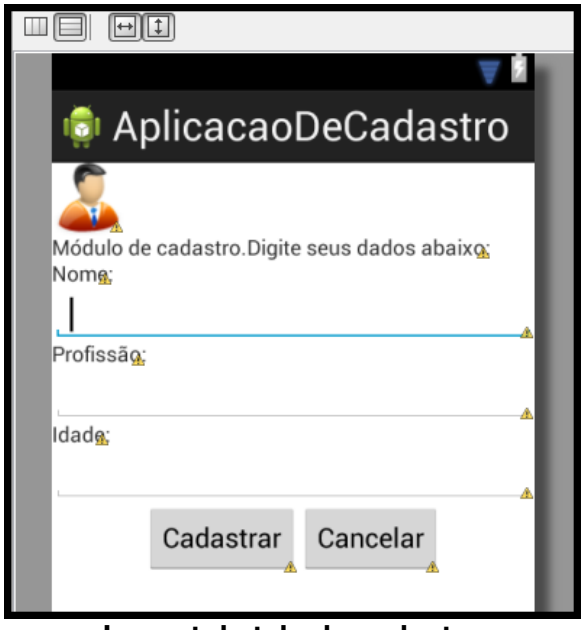
Button

Propriedade	Valor
Id	btcadastrar
Text	Cadastrar pessoa

Button

Propriedade	Valor
Id	btcancelar
Text	Cancelar

Seguindo os passos acima, o Layout de nossa aplicação deve estar de acordo com a figura abaixo:



Layout da tela de cadastro

Vamos ver agora a estrutura XML da tela de cadastro da aplicação:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/profile" />


    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Módulo de cadastro.
        Digite seus dados abaixo:" />


    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nome:" />


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

        <requestFocus />
    </EditText>


    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Profissão:" />


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


    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Idade:" />
```

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" >


        <Button
            android:id="@+id/btcadastrar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cadastrar" />



        <Button
            android:id="@+id/btcancelar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cancelar" />

    </LinearLayout>

</LinearLayout>
```

Agora vamos criar mais uma tela (arquivo de layout XML) para nossa aplicação. O nome da nossa tela vai se chamar “lista_pessoas_cadastradas”.

Após criar a tela da nossa aplicação, vamos adicionar os seguintes componentes na sequência:

ImageView

Propriedade	Valor
Src (Imagem)	profile.png

TextView

Propriedade	Valor
Text	Lista de pessoas cadastradas.
Text size	20sp

Seguindo os passos acima, o layout da aplicação deve estar de acordo com a figura seguinte:

Layout da tela de listagem de cadastrados

Ainda não acabou. Vamos adicionar após o texto uma nova estrutura de layout do tipo **LinearLayout** (Horizontal), e em seguida vamos mudar as seguintes propriedades dela, conforme abaixo:

LinearLayout

Propriedade	Valor
Id	layoutNome

Agora dentro da estrutura "layoutNome" vamos adicionar os seguintes componentes na sequência:

TextView

Propriedade	Valor
Text	Nome.
Text color	#0000ff
Text size	20sp

TextView

Propriedade	Valor
Text	(Deixar em branco com espaços)
Text size	20sp
Id	txtnome

Logo após a estrutura **LinearLayout** que chamamos de "layoutNome", vamos adicionar uma nova estrutura **LinearLayout** (Horizontal), e em seguida modifique as seguintes propriedades abaixo:

Propriedade	Valor
Id	layoutProfissao

Agora dentro da estrutura "layoutProfissao" vamos adicionar os seguintes componentes na sequência:

TextView

Propriedade	Valor
Text	Profissão.
Text color	#0000ff
Text size	20sp

TextView

Propriedade	Valor
Text	(Deixar em branco com espaços)
Text size	20sp
Id	txtprofissao

Agora vamos adicionar uma nova estrutura **LinearLayout** (Horizontal) e em seguida modifique as seguintes propriedades abaixo:

Propriedade	Valor
Id	layoutIdade

Agora dentro da estrutura "layoutIdade" vamos adicionar os seguintes componentes na sequência:

TextView

Propriedade	Valor
Text	Idade.
Text color	#0000ff
Text size	20sp

TextView

Propriedade	Valor
Text	(Deixar em branco com espaços)
Text color	#ffffff
Text size	20sp
Id	txtidade

Agora vamos adicionar uma nova estrutura **LinearLayout** (Horizontal), como já foi mostrado acima. E depois, modifique as seguintes propriedades abaixo:

Propriedade	Valor
Id	layoutBotoes
Gravity	Center

Agora dentro da estrutura "layoutBotoes" vamos adicionar os seguintes componentes na sequência:

Button

Propriedade	Valor
Id	btvoltar
Text	Voltar

Button

Propriedade	Valor
Id	@+lista/btavancar
Text	Avançar

Agora vamos adicionar um **Button** logo após a estrutura "layoutBotoes" que criamos com as seguintes propriedades abaixo:

Button

Propriedade	Valor
Id	btmenu_principal
Text	Menu principal
Width	fill_parent

Seguindo todos os passos acima, o layout da aplicação deve estar de acordo com a figura abaixo:

Layout da tela de lista de pessoas cadastradas

Vejamos agora a estrutura em XML da tela de listagem de pessoas cadastradas:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/profile" />


    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Lista de pessoas cadastradas"
        android:textSize="20sp" />

    <LinearLayout
        android:id="@+id/layoutNome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="Nome :"
            android:textColor="#0000ff"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/txtnome"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="    "
            android:textSize="20sp" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/layoutProfissao"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >


        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Profissão:"
            android:textColor="#0000ff"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/txtprofissao"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="   "
            android:textSize="20sp" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/layoutIdade"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >


        <TextView
            android:id="@+id/textView4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Idade:"
            android:textColor="#0000ff"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/txtidade"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="    "
            android:textSize="20sp" />

    </LinearLayout>
```

```
    <LinearLayout
        android:id="@+id/layoutBotoes"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" >

        <Button
            android:id="@+id/btvoltar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Voltar" />

        <Button
            android:id="@+id/btavancar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Avançar" />
    </LinearLayout>

    <Button
        android:id="@+id/btmenu_principal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Menu principal" />

</LinearLayout>
```

Conforme já havia falado, para esse programa vamos trabalhar com uma estrutura de dados do tipo “lista” (duplamente encadeada). Para organizarmos de uma forma mais clara o armazenamento dos dados na memória, irei criar uma classe que vai representar o “registro” das informações na memória. A classe que iremos criar vai se chamar **Registro**. Essa classe vai estar dentro do mesmo pacote onde se encontra o arquivo “AplicacaoCadastroActivity.java” (neste caso, o pacote “com.example.aplicacaodecadastro”). Depois de criarmos a classe, vamos escrever o seu código conforme é mostrado em seguida:

Para criamos uma classe basta clicar com o botão direito sobre o pacote citado acima e em seguida selecionar “New/Class”. Feito isso vai se abrir aquela caixa de diálogo de criação de classes e no campo “Name” vamos digitar o nome da classe que vai se chamar “Registro” (digitar sem aspas, é claro). Feito isso clique no botão “Finish” para que a classe possa ser criada. Depois de gerada a classe coloque o seguinte código como é demonstrado abaixo:

```
package com.example.aplicacaodecadastro;

public class Registro {

      String nome;
      String profissao;
      String idade;

      Registro prox;
      Registro ant;

}
```

Agora no arquivo “AplicacaoCadastroActivity.java” vamos colocar o seguinte código abaixo:

```
package com.example.aplicacaodecadastro;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.widget.*;
import android.view.*;

public class AplicacaoCadastroActivity extends Activity {

    Button btcadastarpessoa, btlistarpessoas;

    Button btcadastrar, btcancelar;

    Button btvoltar, btavancar, btmenu_principal;

    TextView txtnome,txtprofissao,txtidade;

    EditText ednome, edprofissao, edidade;

    Registro pri,ult, reg, aux, anterior;

    int numero_registros = 0;
    int posicao;

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

    public void CarregarTelaPrincipal()
    {
      setContentView
       (R.layout.activity_aplicacao_cadastro);

      btcadastarpessoa = (Button) findViewById
      (R.id.btcadastrarpessoa);

      btlistarpessoas = (Button) findViewById
      (R.id.btlistarpessoas);

      btcadastarpessoa.setOnClickListener(new
       View.OnClickListener() {

         @Override
         public void onClick(View arg0) {

            CarregarTelaCadastro();

          }
       });
```

```
        btlistarpessoas.setOnClickListener(new
        View.OnClickListener() {

          @Override
          public void onClick(View arg0) {

             CarregarTelaListagemPessoas();

             }
         });

}

public void CarregarTelaCadastro()
{
   setContentView(R.layout.cadastro);

   ednome = (EditText) findViewById(R.id.ednome);

   edprofissao = (EditText) findViewById
   (R.id.edprofissao);

   edidade = (EditText) findViewById(R.id.edidade);

   btcadastrar = (Button) findViewById
   (R.id.btcadastrar);

   btcancelar = (Button) findViewById(R.id.btcancelar);

   btcadastrar.setOnClickListener(new
   View.OnClickListener() {

    @Override
    public void onClick(View arg0) {

      anterior = reg;

      reg = new Registro();

      reg.nome = ednome.getText().toString();

      reg.profissao = edprofissao.getText().toString();

      reg.idade = edidade.getText().toString();

      if(pri == null)
         pri = reg;

      if(ult == null)
        ult = reg;
      else {
        ult.prox = reg;
        ult = reg;
        ult.ant = anterior;
      }

      numero_registros++;

      ExibirMensagem("Registro cadastrado com sucesso");
```

```
        CarregarTelaPrincipal();
       }
    });

    btcancelar.setOnClickListener(new
     View.OnClickListener() {

       @Override
       public void onClick(View arg0) {
         CarregarTelaPrincipal();

        }
    });

}

public void CarregarTelaListagemPessoas()
{

  if(numero_registros == 0)
  {
     ExibirMensagem("Nenhum registro cadastrado");
     CarregarTelaPrincipal();
     return;


      }

  else {

    setContentView(R.layout.lista_pessoas_cadastradas);
    btvoltar = (Button) findViewById(R.id.btvoltar);
    btavancar = (Button) findViewById(R.id.btavancar);
    btmenu_principal = (Button)
    findViewById(R.id.btmenu_principal);

    txtnome = (TextView) findViewById(R.id.txtnome);
    txtprofissao = (TextView)
    findViewById(R.id.txtprofissao);
    txtidade = (TextView) findViewById(R.id.txtidade);

    aux = pri;

    posicao = 1;

    txtnome.setText(aux.nome);
    txtprofissao.setText(aux.profissao);
    txtidade.setText(aux.idade);

    btvoltar.setOnClickListener(new
    View.OnClickListener() {

      @Override
      public void onClick(View v) {

        if(posicao == 1)
            return;
```

```
          posicao--;

          aux = aux.ant;

           txtnome.setText(aux.nome);
           txtprofissao.setText(aux.profissao);
           txtidade.setText(aux.idade);

            }
         });

        btavancar.setOnClickListener(new
         View.OnClickListener() {

          @Override
          public void onClick(View v) {

            if(posicao == numero_registros)
              return;

            posicao++;

            aux = aux.prox;

            txtnome.setText(aux.nome);
            txtprofissao.setText(aux.profissao);
            txtidade.setText(aux.idade);

            }
         });


        btmenu_principal.setOnClickListener(new
        View.OnClickListener() {

          @Override
          public void onClick(View v) {
              CarregarTelaPrincipal();
           }
        });

    }
     }

     public void ExibirMensagem(String mensagem){

       AlertDialog.Builder dialogo = new AlertDialog.
        Builder(AplicacaoCadastroActivity.this);
       dialogo.setTitle("Aviso");
       dialogo.setMessage(mensagem);
       dialogo.setNeutralButton("OK", null);
       dialogo.show();
     }


}
```

Agora vamos analisar aos poucos os códigos dessa aplicação. Observe que nessa aplicação ou possuo três métodos: o método **CarregarTelaPrincipal** (responsável por carregar a tela principal), o método **CarregarTelaCadastro** (responsável por carregar a tela de cadastro) e o método **CarregarTelaListagemPessoas** (responsável por carregar a tela de listagem dos usuários cadastrados).

Vamos analisar alguns trechos importantes de código do método **CarregarTelaCadastro**. Se você observar nessa aplicação, foram declarados cinco variáveis : *pri,ult* , *reg* , *aux* e *anterior* (todas elas do tipo **Registro**). Irei descrever aqui a finalidade de cada variável:

A variável *pri* serve para apontar para o endereço do primeiro registro armazenado na memória.

A variável *ult* aponta para o endereço do último registro.

A variável *reg* armazena os dados do registro corrente, ou seja, a cada instância de um novo registro, ela irá armazenar a instância corrente.

A variável *aux* funciona como uma variável auxiliar, que será utilizada durante a navegação pelos registros na tela de listagem de usuários cadastrados.

A variável *anterior* guarda sempre o "último" registro anterior antes da criação de uma nova instância de dados, que será armazenada em *reg*. Isso significa que antes de *reg* receber uma nova instância de dados, a variável *anterior* irá armazenar o endereço de dados guardado atualmente por *reg*.

Vamos analisar a linha de comando dentro do método **onClick** do botão *"btcadastrar"*. A linha:

```
reg = new Registro();
```

Cria uma nova instância da classe **Registro** e guarda essa instância na variável *reg*. As linhas:

```
reg.nome = ednome.getText().toString();
reg.profissao = edprofissao.getText().toString();
reg.idade = edidade.getText().toString();
```

Gravam os dados dos campos no objeto *reg*. Já as linhas abaixo:

```
if(pri == null)
 pri = reg;
if(ult == null)
 ult = reg;
 else {
   ult.prox = reg;
   ult = reg;
   ult.ant = anterior;
 }
```

Fazem todo o processo de armazenamento dos dados.

Agora vamos para o método **CarregarTelaListagemPessoas**. Quando esse método é chamado, é feita uma verificação se há dados cadastrados, se não houver dados cadastrados será exibida uma mensagem indicando essa situação e você será retornado a tela principal. Vou comentar algumas linhas desse método. A instrução:

```
aux=pri;
```

Retorna para a variável *aux* o endereço do primeiro registro, que está armazenado em *pri* . Já as linhas:

```
txtnome.setText(aux.nome);
txtidade.setText(aux.idade);
txtprofissao.setText(aux.profissao);
```

Joga as informações obtidas (nome, idade e profissão) para os campos (todos do tipo **TextView**), para que elas possam ser exibidas.

Vamos agora para o método **onClick** do botão "*btvoltar"*. Esse botão mostra os registros anteriores. Antes de voltar um registro, verifico se eu me encontro no primeiro registro pela condição:

```
if(pos==1)
```

Se a condição for verdadeira, saio do evento, senão, continuo executando as instruções. A linha:

```
aux=aux.ant;
```

Retorna para *aux* o endereço do registro anterior. Depois disso são executados instruções para que os dados possam ser exibidos.

Já no método **onClick** do botão *btavancar*, antes de passar para o próximo registro, verifico se já está no último registro pela instrução:

```
if(pos==numero_registros)
```

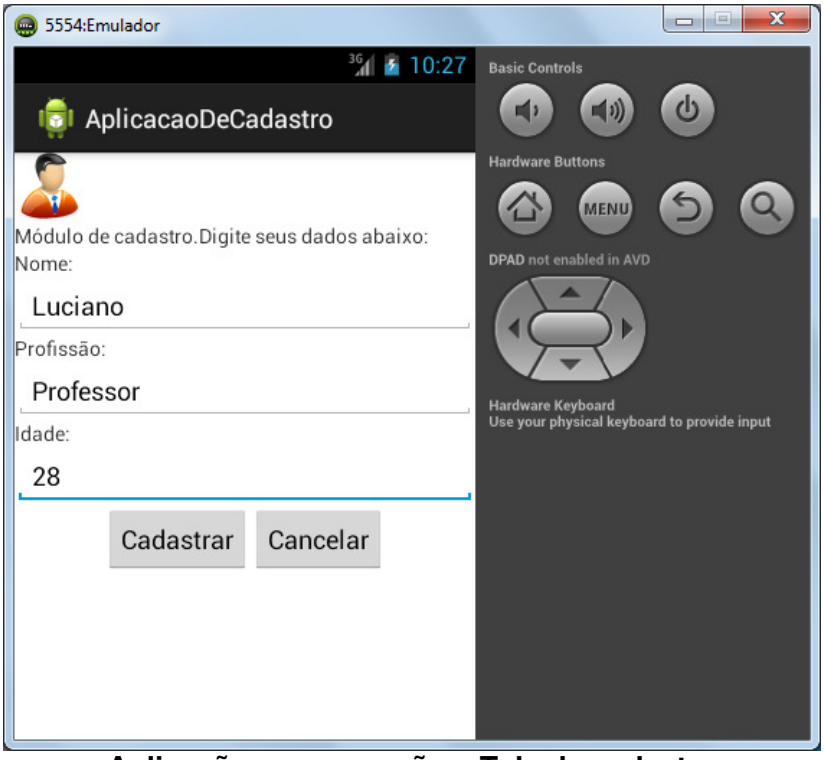
Se a condição for verdadeira , o evento é encerrado, senão, continuo executando as instruções. A linha:

```
aux=aux.prox;
```

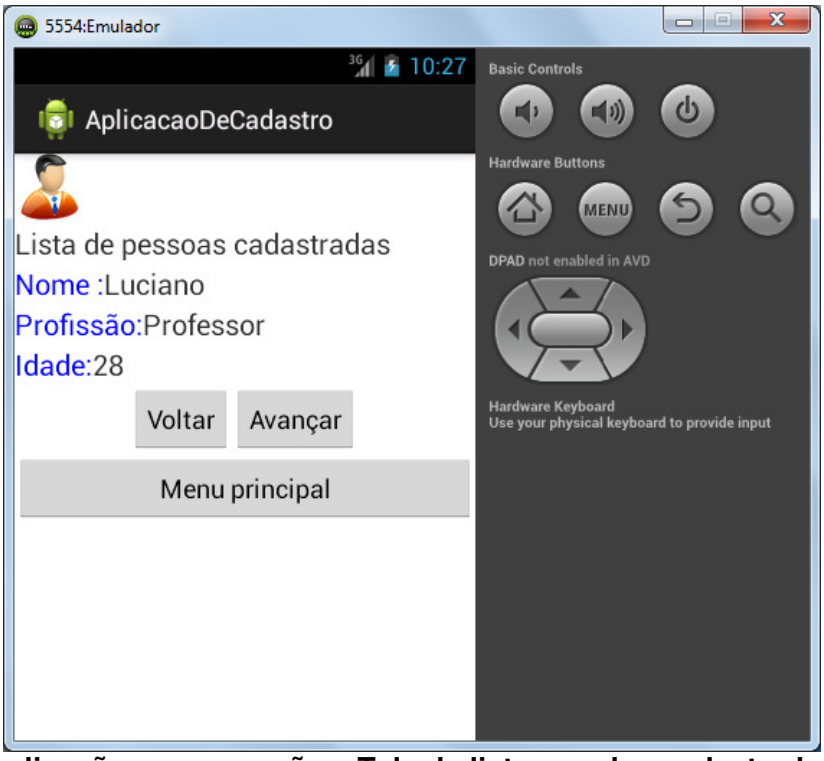
Retorna para *aux* o endereço do próximo registro. Depois disso são executados instruções para que os dados possam ser exibidos.

Bom, vamos executar a nossa aplicação ? O resultado você confere nas figuras seguintes:

Aplicação em execução – Tela principal



Aplicação em execução – Tela de cadastro



Aplicação em execução – Tela de listagem dos cadastrados

Capitulo 5 Propriedades e eventos dos componentes trabalhados

Neste capitulo eu irei mostrar e descrever as propriedades e eventos de todos os componentes que trabalhamos neste material.

Widget TextView

- Propriedades

Propriedade	Em XML	Em Java
`Text`	`android:text`	`setText(CharSequence c)`
Nessa propriedade, você define o texto a ser exibido na tela.		

Propriedade	Em XML	Em Java
`Text Color`	`android:textColor`	`setTextColor(Color c)`
Nessa propriedade, você define a cor de texto.		

Propriedade	Em XML	Em Java
`Background`	`android:background`	`setBackGroundColor(Color c)`
Nessa propriedade , você define o cor de fundo do componente exibido. Valor: #000000 até #FFFFFF.		

Propriedade	Em XML	Em Java
`Text Size`	`android:textSize`	`setTextSize(float tamanho)` ou `setTextSize(int unidade, int tamanho)`
Define o tamanho do texto. O tamanho da fonte pode ser especificado em várias notações : px (pixels),sp(scaled-pixels) , mm(milímetros), in (polegadas) e etc.		

Propriedade	Em XML	Em Java
Typeface	android:typeface	setTypeface(Typeface fonte)
Essa propriedade serve para definir uma fonte ao texto (normal,sans,serif,monospace).		

- Eventos

Método que define o evento	Evento	Métodos relacionados ao evento
setOnClickListener	OnClickListener	onClick(View v)
Esse evento é disparado toda vez que o componente for clicado, disparando o método onClick.		

Widget EditText

- Propriedades

Propriedade	Em XML	Em Java
Text	android:text	setText(CharSequence c)
Nessa propriedade, você define o texto a ser exibido na tela.		

Propriedade	Em XML	Em Java
Text Color	android:textColor	setTextColor(Color c)
Nessa propriedade, você define a cor do texto.		

Propriedade	Em XML	Em Java
Background	android:background	setBackGroundColor(Color c)
Nessa propriedade , você define o cor de fundo do componente exibido. Valor: #000000 até #FFFFFF.		

Propriedade	Em XML	Em Java
Capitalize	android:capitalize	
Essa propriedade serve para definir o tipo capitalização das palavras. Por padrão, o valor e "none"(nenhum). Os possíveis valores para essa propriedade são : "words","sentences" e "characters"		

Propriedade	Em XML	Em Java
Password	android:password	
Com essa propriedade você habilita a digitação de senhas. O valor padrão desse atributo é "false".		

Propriedade	Em XML	Em Java
Text Size	android:textSize	setTextSize(float tamanho) ou setTextSize(int unidade, int tamanho)
Define o tamanho do texto. O tamanho da fonte pode ser especificado em várias notações : px (pixels),sp(scaled-pixels) , mm(milímetros), in (polegadas) e etc.		

Propriedade	Em XML	Em Java
Typeface	android:typeface	setTypeface(Typeface fonte)
Essa propriedade serve para definir uma fonte ao texto. Os possíveis valores são : "normal","monospace","sans" e "serif".		

Propriedade	Em XML	Em Java
Hint	android:hint	setHint(CharSequence c)
define uma mensagem que aparecerá quando a EditText estiver vazia.		

- Eventos

Método que define o evento	Evento	Métodos relacionados ao evento
setOnClickListener	OnClickListener	onClick(View v)
Esse evento é disparado toda vez que o componente for clicado, disparando o método onClick.		

Método que define o evento	Evento	Métodos relacionados ao evento
setOnKeyListener	OnKeyListener	onKey(View v,int KeyCode, KeyEvent event)
Esse evento é disparado toda vez que a tecla é acionada, disparando o método onKey.		

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnFocusChangeListener`	`OnFocusChangeListener`	`onFocusChange(View v, boolean hasFocus)`
Esse método é disparado toda vez quando um componente EditText ganha ou perde foco.		

Widget Button

- Propriedades

Propriedade	Em XML	Em Java
`Text`	`android:text`	`setText(CharSequence c)`
Nessa propriedade, você define o texto a ser exibido na tela.		

Propriedade	Em XML	Em Java
`Text Color`	`android:textColor`	`setTextColor(Color c)`
Nessa propriedade, você define a cor do texto.		

Propriedade	Em XML	Em Java
`Text size`	`android:textSize`	`setTextSize(float tamanho)` ou `setTextSize(int unidade, int tamanho)`
Define o tamanho do texto. O tamanho da fonte pode ser especificado em várias notações : px (pixels),sp(scaled-pixels) , mm(milímetros), in (polegadas) e etc.		

Propriedade	Em XML	Em Java
`Typeface`	`android:typeface`	`setTypeface(Typeface fonte)`
Essa propriedade serve para definir uma fonte ao texto. Os possíveis valores são : “normal”,”monospace”,”sans” e “serif”.		

- Eventos

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnClickListener`	`OnClickListener`	`onClick(View v)`
Esse evento é disparado toda vez que o componente for clicado, disparando o método onClick.		

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnKeyListener`	`OnKeyListener`	`onKey(View v,int KeyCode, KeyEvent event)`
Esse evento é disparado toda vez que a tecla é acionada, disparando o método onKey.		

Widget CheckBox/RadioButton

- Propriedades

Propriedade	Em XML	Em Java
`Text`	`android:text`	`setText(CharSequence c)`
Nessa propriedade, você define o texto a ser exibido na tela.		

Propriedade	Em XML	Em Java
`Text color`	`android:textColor`	`setTextColor(Color c)`
Nessa propriedade, você define a cor do texto.		

Propriedade	Em XML	Em Java
`Checked`	`android:checked`	`setChecked(boolean estado)`
Nessa propriedade você define o estado do CheckBox, se estará marcado (true) ou não (false).		

- Eventos

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnClickListener`	`OnClickListener`	`onClick(View v)`
Esse evento é disparado toda vez que o componente for clicado, disparando o método onClick.		

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnCheckedChangeListener`	`OnCheckedChangeListener`	`onCheckedChanged (CompoundButton cb,boolean b)`
Esse evento será disparado toda vez que o estado do CheckBox for modificado, ou seja, marcado ou desmarcado, disparando o método onCheckedChanged.		

Widget Spinner / ListView

- Propriedades

Método	Descrição
`setAdapter(SpinnerAdapter a)`	Nesse método você define os elementos que irão compor esse componente através de um vetor (array).
`int getSelectedPosition()`	Essa função retorna a posição do elemento selecionado. Por exemplo, se for o primeiro elemento, retorna 0, se for o segundo, retorna 1 e assim sucessivamente.
`Object getSelectedItem()`	Essa função retorna em um tipo Object, o item selecionado.
`Object getItemAtPosition(int posicao)`	Retorna em um tipo Object o elemento de uma determinada posição, passada como parâmetro.

- Eventos

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnClickListener`	`OnClickListener`	`onClick(View v)`
Esse evento é disparado toda vez que o componente for clicado, disparando o método onClick.		

Método que define o evento	Evento	Métodos relacionados ao evento
`setOnItemClickListener`	`OnItemClickListener`	`onItemClick (AdapterView<?> a, View v, int I, long l)`
Esse evento será disparado toda vez que um derminado item for clicado, disparando o método onItemClick.		

Método que define o evento	Evento	Métodos relacionados ao evento

`setOnItemSelectedListener`	`OnItemSelectedListener`	`onItemSelected(AdapterView av, View v,int posição,long id)`
		
		`onNothingSelected(AdapterView av)`

Esse evento será disparado toda vez que um derminado item for selecionado, disparando o método onItemSelected. Caso nenhum item seja selecionado, será disparado o método onNothingSelected.		

Widget ImageView

- Propriedades

Propriedade	Em XML	Em Java
`Src`	`android:src`	`setImageResource(int Id)`
Nessa propriedade, você define a imagem que será exibida na tela.		

Método	Descrição
`setImageURI(Uri link)`	Esse método é similar ao método acima, sendo que aqui você especifica o Uri (como se fosse um link de internet) como caminho de localização da imagem.

Conclusão a respeito do material

Nesta apostila vimos de forma bem básica e introdutória como desenvolver aplicações para Android para algumas situações . Começamos vendo um pouco sobre a plataforma Android, como ela surgiu e tudo mais. Aprendemos a instalar e configurar Android Developer Tools , que é a ferramenta de desenvolvimento para a criação de Aplicações Android, e em seguida aprendemos a construir algumas pequenas aplicações para Android,como uma calculadora básica, um aplicativo de compras, um aplicativo de cálculo de salário e etc.

Se você quiser uma abordagem "mais completa" de como desenvolver aplicações para Android , adquira a "Apostila de Android – Programando Passo a Passo " Completa, efetuando o pagamento do seu valor através do **PagSeguro**. Visite o site **www.apostilaandroid.net** para mais informações à respeito da Apostila de Android "completa"

Espero que esse material lhe tenha sido útil.

Abraços