



**Universidade Federal de São Carlos**  
**Centro de Ciências Exatas e de Tecnologia**  
**Departamento de Computação**

---

**Introdução às tecnologias para desenvolvimento de  
aplicações em plataformas móveis Android**

Processo: **23112.003595/2012-35**

Coordenadores:

**Ricardo Menotti**

**Daniel Lucrédio**

Autor/Bolsista:

**Matheus Fernando Finatti**

**São Carlos - SP, 21 de outubro de 2013**



# Sumário

---

---

Lista de Figuras . . . . .	iv
Lista de Tabelas . . . . .	v
<b>Lista de Algoritmos</b>	<b>vii</b>
Lista de Algoritmos . . . . .	ix
Lista de Abreviaturas . . . . .	xi
Resumo . . . . .	xiii
<b>1 Introdução</b>	<b>1</b>
<b>2 Configuração do Ambiente</b>	<b>5</b>
<b>3 Linguagem do Android</b>	<b>7</b>
3.1 Linguagem . . . . .	7
3.2 Entendendo a estrutura de uma aplicação Android . . . . .	8
<b>4 Criando seu primeiro aplicativo</b>	<b>11</b>
<b>5 Design</b>	<b>27</b>
5.1 Activity . . . . .	27
5.2 Especifique a <i>activity</i> que inicia seu aplicativo . . . . .	28
5.3 Tipos de <i>Layout</i> . . . . .	29
5.4 Listas (ListView) . . . . .	32
5.5 Listas Compostas . . . . .	35
5.6 Listas expansíveis (ExpandableListView) . . . . .	38
5.7 Grades (GridView) e imagens ImageView . . . . .	45
5.8 Fragmentos . . . . .	50
5.9 Abas (Tabs) . . . . .	53
5.10 Trocar de página com gesto de arrastar usando ViewPager . . . . .	60
5.11 Abas com gesto de arrastar . . . . .	62

5.12	Menu lateral ( <i>Navigation Drawer</i> ) . . . . .	64
5.13	<i>ActionBar</i> . . . . .	65
<b>6</b>	<b>Comunicação</b>	<b>71</b>
6.1	<i>Internet</i> . . . . .	71
6.2	Telefone . . . . .	76
6.3	<i>Short Message Service (SMS)</i> . . . . .	77
<b>7</b>	<b>Armazenamento</b>	<b>79</b>
7.1	<i>Shared Preferences:</i> . . . . .	79
7.2	Armazenamento interno . . . . .	81
7.3	Armazenamento Externo . . . . .	85
7.4	Banco de dados . . . . .	86
<b>8</b>	<b>Câmera</b>	<b>91</b>
8.1	Usando a API . . . . .	91
<b>A</b>	<b>Especificação blá, blá, blá</b>	<b>97</b>

# Lista de Figuras

---

---

1.1	Distribuição das versões do Android . . . . .	2
4.1	Primeira janela de criação de novo aplicativo . . . . .	12
4.2	Segunda janela de criação de novo aplicativo . . . . .	12
4.3	Terceira janela de criação de novo aplicativo . . . . .	13
4.4	Quarta janela de criação de novo aplicativo . . . . .	13
4.5	Quinta janela de criação de novo aplicativo . . . . .	14
4.6	Selecionando o Hello world . . . . .	16
4.7	<i>activity</i> com os elementos colocados na tela . . . . .	17
4.8	Criando uma nova <i>activity</i> . . . . .	21
4.9	Primeira tela do primeiro aplicativo . . . . .	25
4.10	Primeira tela após escrever texto na caixa de texto . . . . .	25
4.11	Segunda tela mostrando a mensagem enviada . . . . .	25
5.1	Ciclo de vida de uma <i>activity</i> . . . . .	27
5.2	LinearLayout vertical (à esquerda) e horizontal (à direita) . . . . .	29
5.3	LinearLayout composto . . . . .	29
5.4	Exemplo de RelativeLayout . . . . .	30
5.5	FrameLayout com exemplo de posicionamento usando <code>layout_gravity</code> . . . . .	31
5.6	Exemplo de TableLayout . . . . .	31
5.7	Esquema de uma lista . . . . .	32
5.8	Detalhes de um elemento da lista . . . . .	32
5.9	Lista simples . . . . .	35
5.10	Lista Composta . . . . .	38
5.11	Exemplo de lista expansível rodando em um <i>smartphone</i> . . . . .	44
5.12	Esquema de um GridView . . . . .	45
5.13	Demonstração de um Grid view . . . . .	47
5.14	Exemplo GridView com imagem em tela cheia . . . . .	50
5.15	Esquema da interface com abas . . . . .	54

5.16	Figura mostrando as 3 abas criadas no exemplo . . . . .	59
5.17	Exemplo de <i>Sliding Menu</i> . . . . .	65
5.18	Exemplo de <i>ActionBar</i> no aplicativo Calendário . . . . .	65
5.19	Exemplo de busca na <i>ActionBar</i> . . . . .	69

# **Lista de Tabelas**

---

---

1.1 Tabela com as distribuições das versões do Android, todas as versões com menos de 0.1% de participação foram desconsideradas . . . . .	2
--	---



# Lista de Algoritmos

---

---

4.1	Exemplo de configuração de versão do SDK no arquivo <code>AndroidManifest.xml</code>	16
4.2	Código da caixa de texto no arquivo <code>activity_main.xml</code>	17
4.3	Código do botão	18
4.4	Arquivo de strings com as duas strings adicionadas	18
4.5	Adicionando método à classe <code>MainActivity</code>	19
4.6	Exemplo de import de uma classe <code>Android</code>	19
4.7	Adicionando uma <code>Intent</code>	19
4.8	Obtendo o conteúdo da caixa de texto e enviando para outra <i>activity</i>	19
4.9	Constante como chave para um extra	20
4.10	Obtendo a <i>string</i> passada como extra passados através do <code>Intent</code>	22
4.11	Método <code>onCreate()</code> recebendo um <i>Intent</i> e mostrando a mensagem	22
5.1	Exemplo de <i>Launcher activity</i>	28
5.2	<code>LinearLayout</code> no arquivo de <i>layout</i>	33
5.3	Código de uma <code>ListView</code>	33
5.4	<code>string-array</code> populada com elementos	33
5.5	Código de uma <i>activity</i> com lista clicável	34
5.6	Código do arquivo <code>item.xml</code>	36
5.7	Código da lista customizada	37
5.8	Código XML de uma Lista expansível	38
5.9	Layout <code>list_item_parent.xml</code>	39
5.10	Layout <code>list_item_child.xml</code>	39
5.11	Classe <code>Parent</code>	40
5.12	Classe <code>CustomAdapter</code>	42
5.13	Construindo a lista expandível na <i>activity</i>	43
5.14	Layout do <code>GridView</code>	45
5.15	Classe <code>ImageAdapter</code>	46
5.16	<i>activity</i> com grade	47

5.17	Layout full_image.xml . . . . .	48
5.18	Classe FullImageActivity . . . . .	49
5.19	Código da <i>activity</i> após as modificações . . . . .	49
5.20	Classe BasicFragment . . . . .	51
5.21	Layout da <i>activity</i> com um fragmento . . . . .	52
5.22	Layout da <i>activity</i> com o FrameLayout . . . . .	52
5.23	<i>activity</i> com adição dinâmica de fragmento . . . . .	53
5.24	Layout da <i>activity</i> TabHostLayout . . . . .	54
5.25	Layout do fragmento da aba. . . . .	55
5.26	Classe Tab1Fragment . . . . .	55
5.27	Classe TabInfo . . . . .	56
5.28	Primeira parte da classe TabLayoutActivity . . . . .	56
5.29	Classe TabFactory . . . . .	57
5.30	Método initialiseTabHost () . . . . .	57
5.31	Método onTabChanged () . . . . .	58
5.32	Método onSaveInstanceState () . . . . .	59
5.33	layout do ViewPager . . . . .	60
5.34	Classe PagerAdapter . . . . .	61
5.35	Activity com PagerAdapter . . . . .	62
5.36	Layout das abas com adição do ViewPager . . . . .	63
5.37	Método initialiseViewPager () . . . . .	63
5.38	Método onTabChanged () alterado . . . . .	64
5.39	Métodos da interface ViewPager.OnPageChangeListener . . . . .	64
5.40	Menu padrão dos exemplos . . . . .	66
5.41	Método padrão onCreateOptionsMenu () . . . . .	66
5.42	Método OnOptionsItemSelected () . . . . .	67
5.43	Adicionando novo item na <i>ActionBar</i> . . . . .	67
5.44	Configurando <i>ActionBar</i> no método onCreate () . . . . .	68
5.45	Criando a caixa de busca na <i>ActionBar</i> . . . . .	69
6.1	Atribuindo permissão de acesso à <i>Internet</i> no <i>Manifest</i> . . . . .	71
6.2	Classe RequestTask . . . . .	72
6.3	Classe RequestTask . . . . .	73
6.4	Modificando o método para requisições POST . . . . .	74
6.5	Classe JSONParser . . . . .	75
6.6	Criando JSON . . . . .	75
6.7	Fazendo uma chamada telefônica . . . . .	76
6.8	Permissão para fazer chamadas telefônicas . . . . .	76
6.9	Permissão para enviar mensagens SMS . . . . .	77
6.10	Método sendSMS () . . . . .	77

6.11	Chamando método <code>sendSMS()</code>	78
7.1	Utilizando <code>SharedPreferences</code> para salvar dados primitivos	80
7.2	Passos iniciais do <i>listener</i> , criando um <code>dialogo</code>	81
7.3	Salvando um arquivo e mostrando um <code>Toast</code>	82
7.4	Fechando o <code>dialogo</code> ao clicar em <i>close</i>	83
7.5	Criando um diálogo com os arquivos salvos	83
7.6	Criando um diálogo com os arquivos salvos	84
7.7	Verificando se o armazenamento externo está disponível	85
7.8	Classe <code>CarOpenHelper</code> do <code>SQLite</code>	87
7.9	Usando o <code>CarOpenHelper</code> na <i>activity</i>	88
7.10	Método <code>openCarList()</code>	89
7.11	Método <code>fetchCarList()</code>	90
8.1	Requisitando permissão de usar a camera	91
8.2	Requisitando permissão de gravar no armazenamento externo	91
8.3	Requisitando permissão de gravar áudio	92
8.4	Classe <code>CameraAccess</code>	93
8.5	Classe <code>CameraPreview</code>	94
8.6	<i>Layout</i> da <i>Activity</i> que irá conter a visualização	95
8.7	Configurando a orientação da <i>activity</i> no <i>Manifest</i>	96
8.8	Primeira parte da classe <code>CameraActivity</code>	96



# **Lista de Abreviaturas**

---

---



# Resumo

---

---

Esse material didático oferece uma visão geral de como programar para o sistema móvel Android e utilizar suas APIs nativas na criação de aplicativos. O material tentará cobrir desde o básico, como a configuração do ambiente de desenvolvimento, criação de layouts básicos e complexos, estrutura geral de um aplicativo e ir até a programação de aplicativos mais complexos que tentam utilizar uma ou várias APIs em conjunto.

O objetivo é dar apenas uma noção de como utilizar as ferramentas do Android, introduzir ao aluno os conceitos e não entrar em detalhes do sistema operacional ou em conceitos mais aprofundados, mas sim uma visão genérica. Após a leitura desse material e realização da prática o aluno deve estar preparado para construir seus próprios aplicativos nativos, e poderá até monetizar seus aplicativos se desejar.

## CAPÍTULO

# 1

# Introdução

---

---

O Android hoje está em centenas de milhões de dispositivos móveis ao redor do mundo, e vem crescendo. É uma plataforma para desenvolvimento em dispositivos móveis como *smartphones*, *tablets* e outros.

Construído em uma colaboração *open-source* com a comunidade de Linux, o Android se tornou a plataforma móvel mais utilizada e que mais cresce no mundo. Sua abertura o tornou o favorito de consumidores e desenvolvedores, levando a um rápido crescimento no número de aplicativos e jogos. Está disponível em centenas de dispositivos diferentes e de fabricantes diferentes em versões diferentes.

Atualmente<sup>1</sup> existem 4 principais versões do Android, são elas da mais atual para mais antiga:

- *Jelly Bean* versão 4.2 e 4.1 que trouxe otimizações de performance, uma nova interface do sistema e outros <sup>2</sup>
- *Ice Cream Sandwich* versão 4.0 trouxe uma interface refinada e unificada para *smartphones* e *tablets* além de facilidade com multitasking e outros <sup>3</sup>

<sup>1</sup>Data em que foi escrito: 06/2013

<sup>2</sup>*Jelly Bean*: <http://developer.android.com/about/versions/jelly-bean.html>

<sup>3</sup>*Ice Cream Sandwich*: <http://developer.android.com/about/versions/android-4.0-highlights.html>

- *Honeycomb* versão 3.0 desenvolvida exclusivamente para *tablets*<sup>4</sup>
- *Gingerbread* versão 2.3 introduziu refinamentos da interface, mais performance e tornou o sistema mais intuitivo<sup>5</sup>

O Google coletou os dados referentes a distribuição das versões do Android:

Versão	Codinome	API	Distribuição
1.6	Donut	4	0.1%
2.1	Eclair	7	1.5%
2.2	Froyo	8	3.2%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7	Gingerbread	10	36.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.6%
4.1.x	Jelly Bean	16	29.0%
4.2.x	Jelly Bean	17	4.0%

Tabela 1.1: Tabela com as distribuições das versões do Android, todas as versões com menos de 0.1% de participação foram desconsideradas

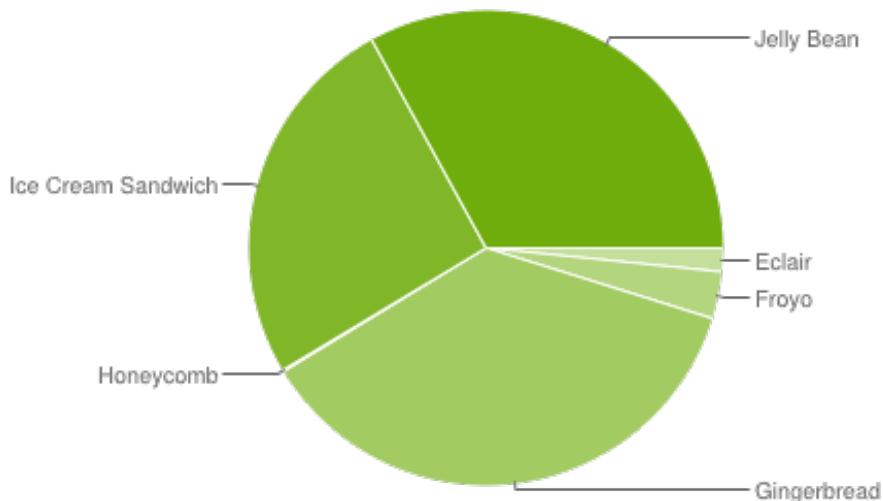


Figura 1.1: Distribuição das versões do Android

<sup>4</sup>*Honeycomb*: <http://developer.android.com/about/versions/android-3.0-highlights.html>

<sup>5</sup>*Gingerbread*: <http://developer.android.com/about/versions/android-2.3-highlights.html>

Esse material irá cobrir alguns tópicos no desenvolvimento de aplicativos para android, tais como:

- Configuração do ambiente de desenvolvimento: Como configurar o ambiente para começar a desenvolver aplicativos, os primeiros passos para criar seu primeiro aplicativo de maneira simples;
- Elementos da interface: Como projetar seu aplicativo para usar as principais interfaces. Listas, Listas compostas, Grades, Abas, Menus são as interfaces mais usadas nos diversos aplicativos no mercado; e
- Elementos de hardware: Como projetar seu aplicativo para usar as APIs de hardware: Bluetooth, GPS, SMS, Chamadas.

Para esse material, algumas convenções serão seguidas:

- Os códigos estarão sempre com a sintaxe colorida para facilitar a leitura;
- URLs das referências estarão nas notas de rodapé; e
- Dicas estarão envoltas por uma caixa para facilitar a visualização





CAPÍTULO  
2

## Configuração do Ambiente

---

---

A instalação e configuração do ambiente de desenvolvimento para Android é simples, o Google fornece um pacote chamado ADT (*Android Development Tools*) que contém o ambiente Eclipse com o *plugin* do Android, algumas ferramentas para instalação dos aplicativos nos *smartphones*, o gerenciador do SDK e as imagens para o emulador do Android. Essas ferramentas são suficientes para o desenvolvimento na plataforma. O pacote ADT pode ser encontrado em: [Android SDK<sup>1</sup>](#).

Basta fazer o download do pacote e extrair que tudo já está pré-configurado para iniciar o desenvolvimento, portanto não há muito o que configurar.

Caso opte por utilizar uma instalação já existente do ambiente Eclipse, você pode instalar o plugin do Android automaticamente através da ferramenta de instalação de plugins do ambiente. Após a instalação será necessário abrir o *SDK Manager* e instalar:

- *Android SDK Tools*;
- *Android SDK Platform-Tools*; e
- Para cada API que você irá utilizar, instalar o *SDK Platform* e opcionalmente o *Documentation for Android SDK* e o *Samples for SDK*.

---

<sup>1</sup><http://developer.android.com/sdk/>





CAPÍTULO

3

# Linguagem do Android

---

---

## 3.1 Linguagem

A linguagem usada para programar na plataforma Android é Java. Então antes de engajar no aprendizado Android é altamente recomendável estudar material Java e principalmente o paradigma de orientação a objetos.

O Android tem algumas particularidades na organização e configuração que é feita através de arquivos XML específicos do Android. Alguns arquivos XML servem para configurar o aplicativo, layout de cada tela e outros dão suporte a strings para facilitar o suporte a múltiplos idiomas. Felizmente o conjunto Eclipse com ADT já cuida disso automaticamente e possui uma série de facilidades alcançadas por meio de interfaces gráficas para os programadores. Por esse motivo, para qualquer iniciante nessa área é recomendável a utilização do ambiente Eclipse.

A criação de layouts dos aplicativos pode ser feita inteiramente através da interface gráfica disponível no ambiente, no estilo *drag and drop*.

## 3.2 Entendendo a estrutura de uma aplicação Android

Uma aplicação Android consiste de uma ou mais *activities*. Uma *activity* é uma tela com *views* que interagem com o usuário. Como o Android segue o padrão MVC (*Model-View-Control*) as *activities* são os *controllers* e as *views*, *views*. As *activities* são classes do Java, o *layout* e outros recursos são definidos em arquivos XML.

Dentre os diversos arquivos XML existentes na configuração de um aplicativo Android o mais importante é o `AndroidManifest.xml`<sup>1</sup> pois é nele que se exprimem as configurações gerais do aplicativo. Nesse texto não iremos adentrar muito nos detalhes das configurações, mas apenas deixar claro que é nesse arquivo que se colocam as versões do Android que seu aplicativo será compatível com, as permissões para usar os recursos do aparelho como Internet, GPS, Bluetooth, etc.

A pasta `src/` contém o pacote com as classes do seu aplicativo isto é, o código fonte do seu aplicativo. Tanto *activities* como classes de suporte devem estar dentro do pacote.

Dentro da pasta `res/` de recursos, encontram-se outros arquivos, referentes à disposição do layout, valores de strings e imagens que sua aplicação irá utilizar. A pasta `layout/` junto com as pastas `drawable-*/` servem para dispor o layout. Cada *drawable* comporta imagens para um tamanho diferente de tela, enquanto que a pasta de *layout* contém a disposição geral do layout. São nesses arquivos que se colocam os itens (*views*) que irão nas telas, como botões, caixas de texto, caixas de seleção, etc.

Na pasta `values/` o mais importante é o arquivo `strings.xml` que contém os valores das strings do aplicativo. Sempre que você quiser referenciar alguma string, a mesma deverá estar expressa nesse arquivo. Fica fácil dessa forma fazer o aplicativo suportar múltiplos idiomas, pois basta traduzir esse único arquivo para alterar todos os textos do aplicativo.

A pasta `menu/` contém os *layouts* do menus do aplicativo, esses são aqueles que podem ser acessados através da *Action Bar*<sup>2</sup> ou através dos botões físicos do aparelho.

---

<sup>1</sup>Documentação do `AndroidManifest`: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>

<sup>2</sup>*ActionBar*: <http://developer.android.com/design/patterns/actionbar.html>

**Resumindo:**

- `AndroidManifest.xml`: Configurações gerais do aplicativo;
- `src/`: Classes do aplicativo; e
- `res/`: Recursos do aplicativo tais que:
  - `strings/`: Todos os textos da sua aplicação, suporte a múltiplos idiomas;
  - `layout/`: Todos os *layouts* de suas telas (*activities*);
  - `drawable/`: Todas as imagens, separados por tamanho de tela; e
  - `menu/`: *layout* dos menus do aplicativo.





## CAPÍTULO

# 4

# Criando seu primeiro aplicativo

---

---

Para exemplificar a criação de um aplicativo, seguiremos o exemplo dado pelo próprio manual do Google sobre o Android (Ver original<sup>1</sup>). Trata-se de aplicativo simples do tipo "*Hello World*".

Iniciaremos criando um novo projeto no Eclipse acessando o menu: *File -> New -> Android Application Project*.

Na janela que apareceu você deve colocar o nome do aplicativo, do projeto e do pacote. O nome do pacote deve seguir a convenção do Java<sup>2</sup>.

- *Minimum Required SDK*: É a versão mínima do sistema operacional Android que sua aplicação irá suportar, o mais comum é a versão 8 do SDK que se refere ao Android 2.2. Alguns tipos de layouts mais complexos não são suportados em versões mais antigas;
- *Target SDK*: É a versão principal do Android para qual seu aplicativo está sendo desenvolvido;
- *Compile With*: Versão do Android com qual seu aplicativo será compilado; e
- *Theme*: Cores do layout.

<sup>1</sup>Original em: <http://developer.android.com/training/basics/firstapp/creating-project.html>

<sup>2</sup>Convenção sobre nome dos pacotes: <http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>

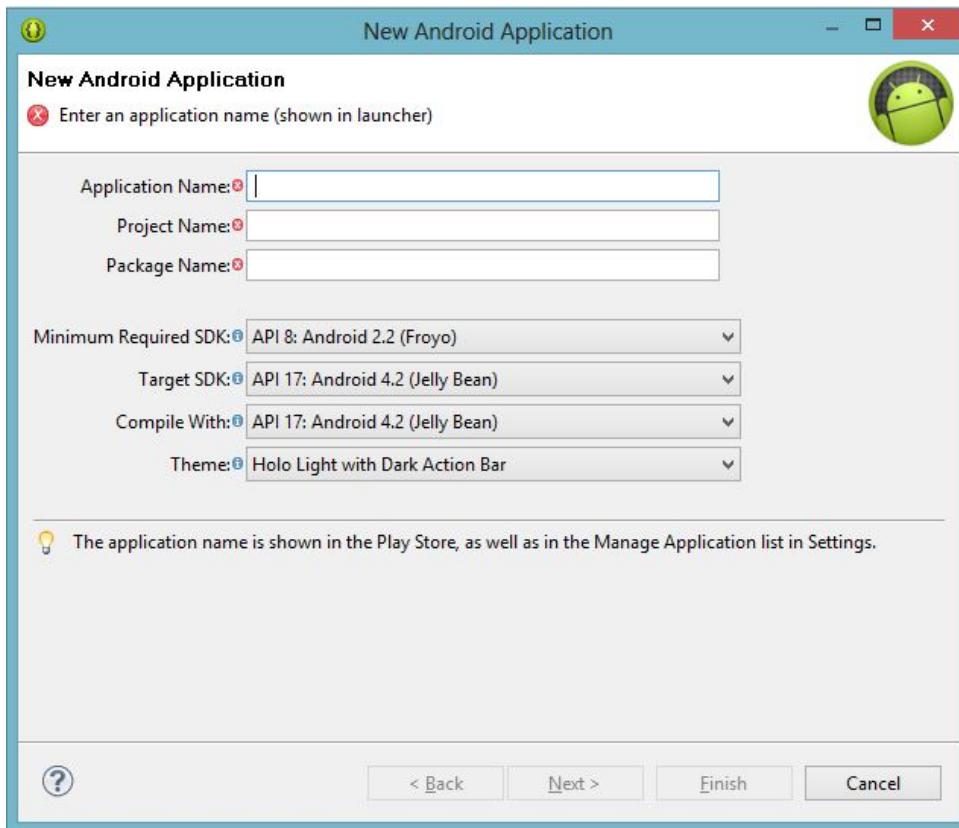


Figura 4.1: Primeira janela de criação de novo aplicativo

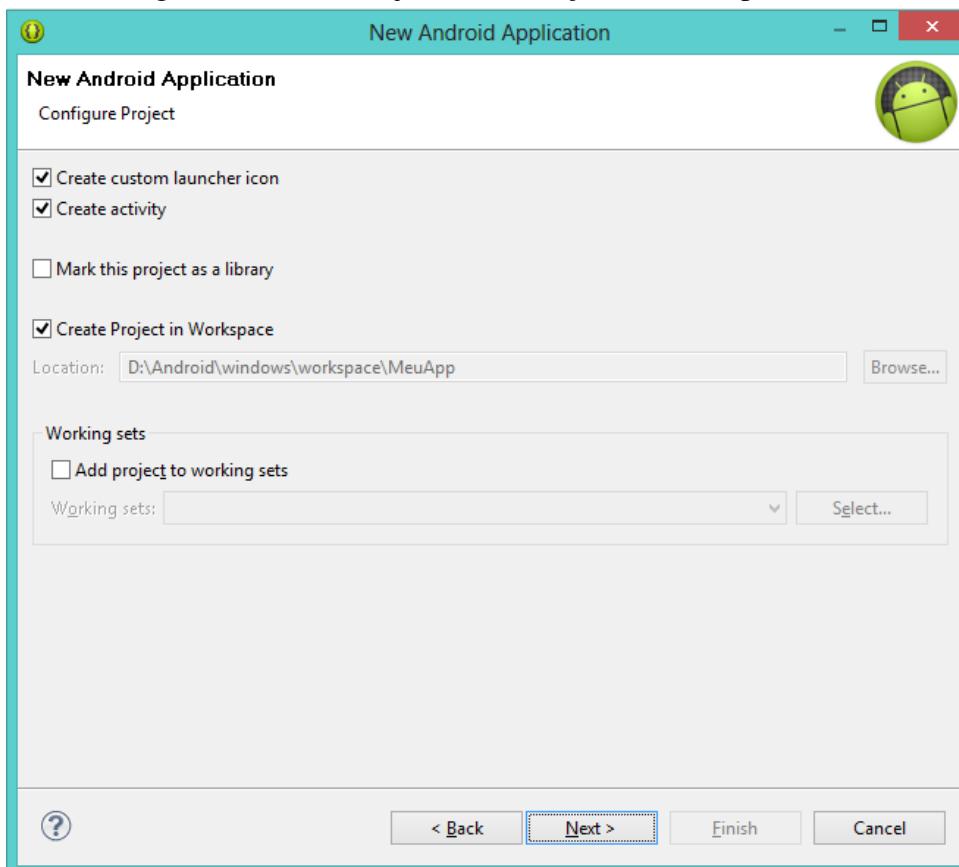


Figura 4.2: Segunda janela de criação de novo aplicativo

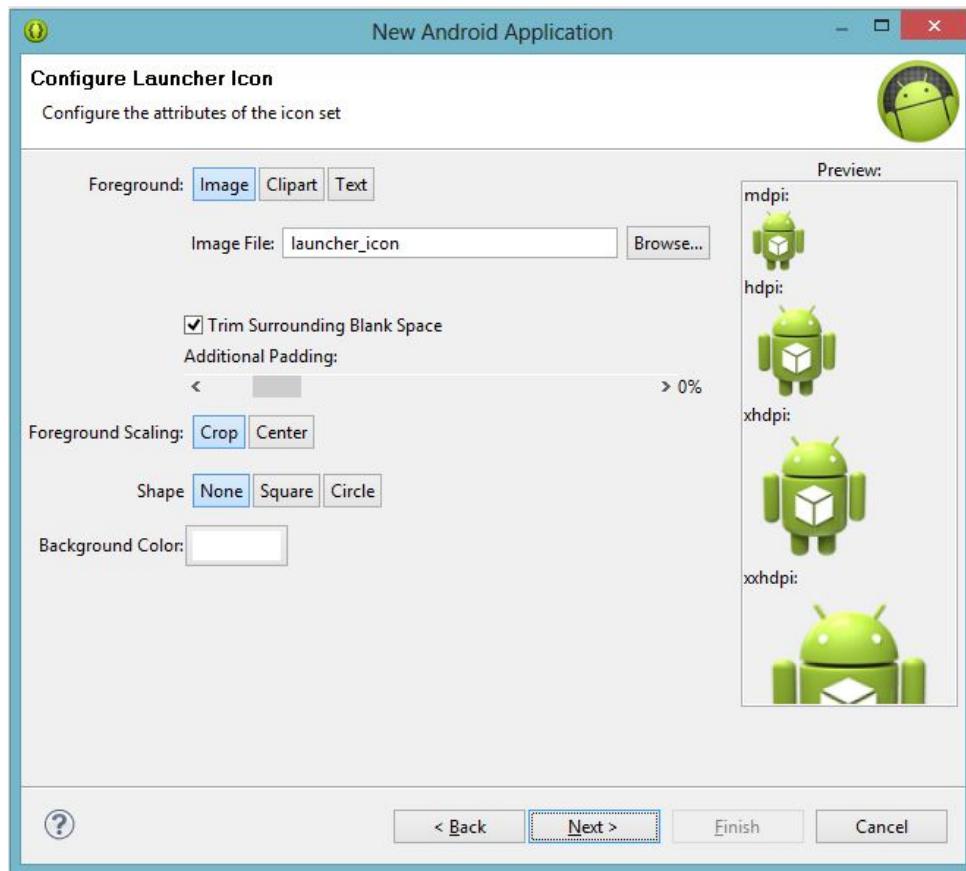


Figura 4.3: Terceira janela de criação de novo aplicativo

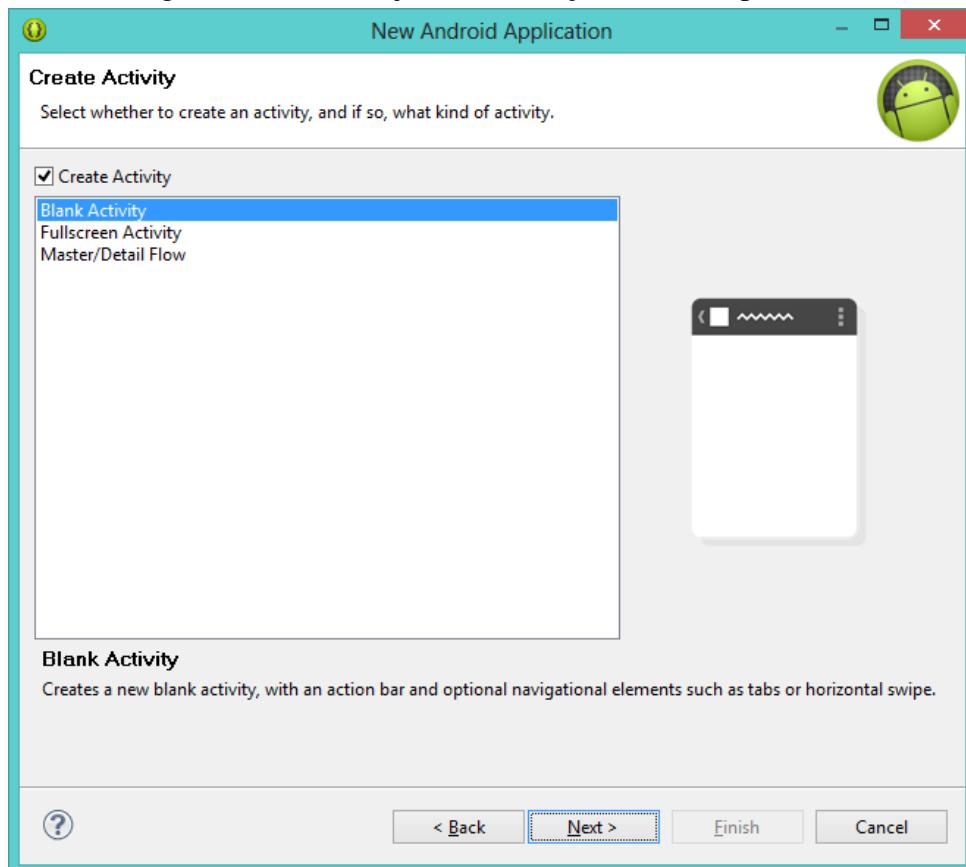


Figura 4.4: Quarta janela de criação de novo aplicativo

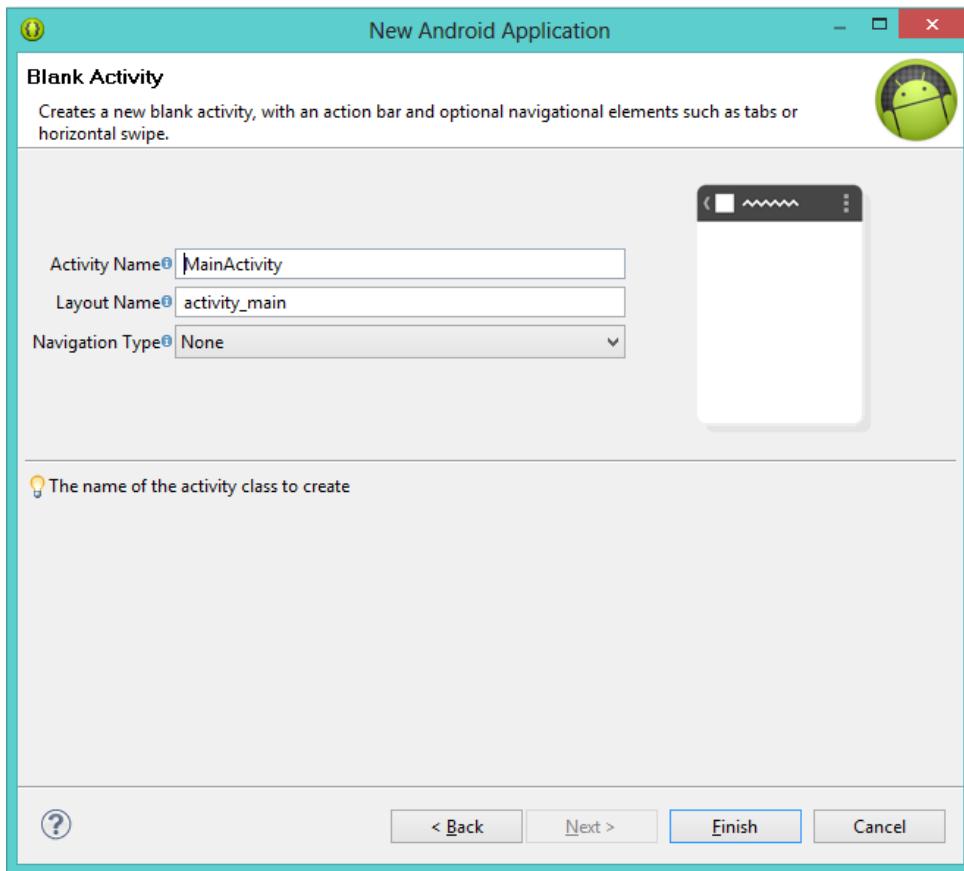


Figura 4.5: Quinta janela de criação de novo aplicativo

Observe na Figura 4.1 a janela de criação de uma nova aplicação Android. Em *Application Name* você deve colocar o nome do aplicativo, em *Project Name*, o nome do projeto e em *Package Name* o nome do pacote. Para esse exemplo utilizaremos como *Minimum Required SDK* a versão API 8, já que nesse exemplo não usaremos nenhum layout que não é suportado em versões mais antigas. Em *Target SDK* e *Compile With* optaremos pela versão mais nova, a API 17. Por final o *Theme* eu optei pelo *Holo Light with Dark Action Bar* que é um tema com fundo branco e barra superior preta, um dos padrões do Android.

**Dica:** Para obter o máximo de compatibilidade sempre procure utilizar *layouts* compatíveis com versões antigas, observe na figura 1.1 que versões antigas ainda tem uma fatia considerável do mercado.

A figura 4.2 mostra a segunda janela da configuração inicial do seu aplicativo. Você pode escolher um ícone personalizado se marcar a caixa *Create custom launcher icon* o que te levará para a janela da figura 4.3. Se marcar *Create Activity* o assistente de criação te levará para a janela da figura 4.4 onde poderá escolher qual *activity* vai ser criada para seu aplicativo. Em todos os exemplos escolheremos a opção *Blank Activity*. Como nosso projeto não é uma biblioteca não marcaremos *Mark this project as a library*. Se marcar *Create Project in Workspace* o assistente irá salvar o projeto na pasta que foi configurada para o *Workspace*, caso contrário ele irá pedir para escolher outro caminho. Como não trabalharemos com *Working Sets* do Eclipse, a opção *Add project to working sets* permanece desmarcada.

Finalmente a figura 4.5 mostra a janela para nomear a *activity* inicial, nesse exemplo mantive *MainActivity*. O nome do *layout* dessa *activity* mantive como *activity\_main* que é o padrão. Na caixa *Navigation Type* existem algumas opções de *layout* pré-definidas pelo Android. São elas:

- *None*: O *layout* vem apenas com uma *Action Bar*<sup>3</sup>
- *Fixed Tabs + Swipe*: O *layout* vem com algumas abas e com gesto de arrastar entre as abas (*activities*) pré-programados.<sup>4</sup>
- *Scrollable Tabs + Swipe*: O *layout* vem com algumas abas e com gesto de arrastar entre as abas pré-programados, porém nesse o estilo das abas é diferente, em vez de abas fixas,

<sup>3</sup>Documentação da *ActionBar*: <http://developer.android.com/guide/topics/ui/actionbar.html>

<sup>4</sup>*Tabs*: <http://developer.android.com/design/building-blocks/tabs.html>

são abas que movem para dar espaço a outras.

- *Dropdown*: O *layout* vem com a troca de *activities* através de um menu na *Action Bar*.

Você pode configurar a versão do SDK manualmente modificando os valores no *manifest*.

Como mostrado no exemplo abaixo:

---

```

1 <uses-sdk
2     android:minSdkVersion="8"
3     android:targetSdkVersion="17" />

```

---

Algoritmo 4.1: Exemplo de configuração de versão do SDK no arquivo *AndroidManifest.xml*

A tag *uses-sdk* serve apenas para o compilador saber quais versões do Android você pretende que seu aplicativo suporte. Dessa forma quando seu aplicativo for lançado na loja *Google Play* o aplicativo só será visível para aqueles usuários que possuem a versão mínima do Android indicada no atributo.

Primeiro vamos criar um *layout* para o aplicativo usando o construtor de interfaces presente no ambiente, primeiro abra o arquivo *res/layout/activity\_main.xml*, segundo o *manifest*, é essa *activity* que será aberta quando o aplicativo for iniciado, isso é configurado através do *intent-filter*<sup>5</sup>.

Selecione o "Hello world" e o remova da sua *activity*.

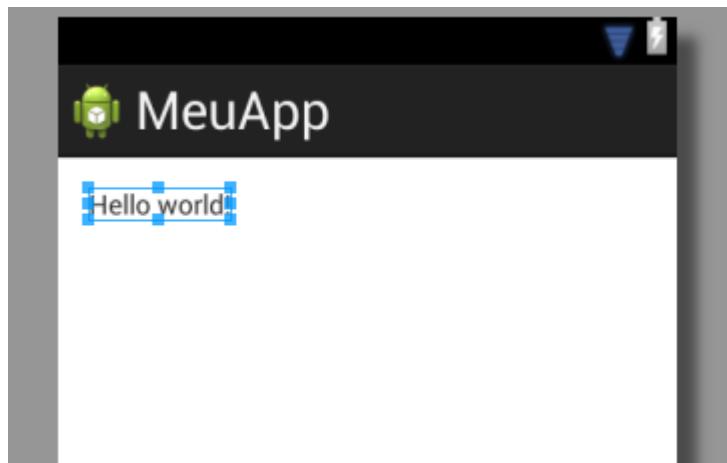


Figura 4.6: Selecionando o Hello world

A tela deverá ficar parecida com a da figura 4.4. Agora arraste um *Text Field -> Plain Text* e um *Form Widgets -> Button* para sua *activity*.

---

<sup>5</sup>Mais informações na seção 5.2

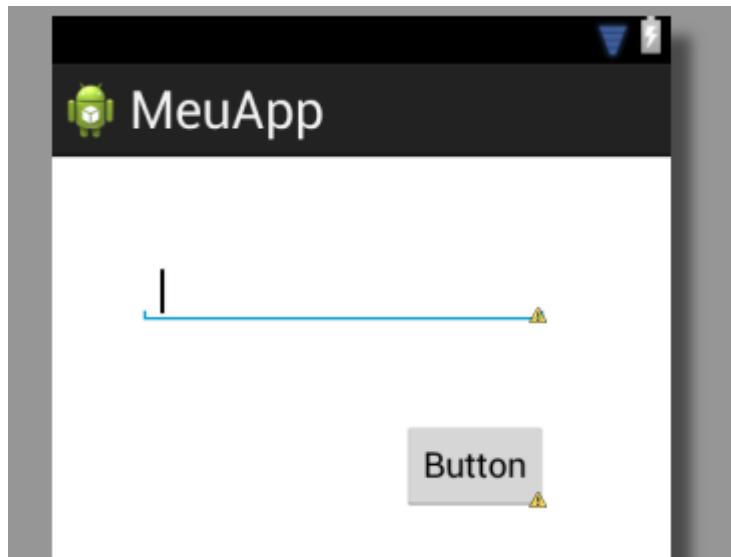


Figura 4.7: *activity* com os elementos colocados na tela

Ao clicar duas vezes no elemento no modo visual, você será levado ao marcador desse elemento no XML correspondente da *activity*. Clique duas vezes na caixa de texto, o seguinte código será exibido:

---

```
1 <EditText
2     android:id="@+id/nameField"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:layout_alignParentLeft="true"
6     android:layout_alignParentTop="true"
7     android:layout_marginLeft="28dp"
8     android:layout_marginTop="35dp"
9     android:ems="10"
10    android:hint="@string/name">
11    <requestFocus />
12 </EditText>
```

---

Algoritmo 4.2: Código da caixa de texto no arquivo `activity_main.xml`

Primeiro, na linha 2 modifique o *id* do *Text Field* para um nome mais intuitivo, nesse exemplo chamaremos apenas de *nameField*. O Android definiu que todo novo atributo *id* deve ser precedido de `@+id/`. O símbolo `@` diz para o compilador que estamos acessando os recursos do Android, esses recursos são compilados na classe R automaticamente. O símbolo `+` diz para o compilador que estamos criando um novo recurso. Por fim, *id* diz que estamos especificando um novo identificador para esse recurso e só então damos o nome a esse identificador.

**Dica:** Existem vários tipos de recursos, porém é importante salientar os diferentes tipos de *id*. Quando referimos aos recursos podemos usar `@android:id/` para acessar recursos que já estão definidos no sistema Android. Usamos `@id/` para acessar recursos que já foram definidos no seu projeto. Para criar um novo recurso, usamos `@+id/`.

Os outros atributos são para definir o tamanho, alinhamento e margem da caixa de texto. O valor `wrap_content` dos atributos `layout_width` e `layout_height` (largura e altura, linhas 3 e 4) força a `view` a mudar de tamanho automaticamente para abrigar seu conteúdo. Os atributos `layout_alignParentLeft` e `layout_alignParentTop` servem (linhas 5 e 6) para alinhar essa `view` com a `view` pai dela, dessa forma ficará alinhado com a borda esquerda e com a borda superior do pai. Os atributos `layout_marginLeft` e `layout_marginTop` (linhas 7 e 8) deslocam o elemento colocando uma margem entre a borda e a `view`, esses valores estarão diferentes pois são computados automaticamente quando a `view` é colocada através do construtor de interfaces. Note que isso só acontecerá caso esteja usando `RelativeLayout`<sup>6</sup> que é o nosso caso. Por último o atributo `ems` (linha 9) configura o tamanho da fonte através da unidade de medida Em.

Depois adicione uma `hint` para essa caixa de texto, uma `hint` é algo que vai estar escrito na caixa de texto quando ela estiver vazia, indicando que tipo de texto você pretende que seja escrito nessa caixa de texto. Neste exemplo (linha 10) a `hint` é uma referência a `string` chamada `name` que iremos definir depois.

Depois modifique o código do botão que está no mesmo arquivo, troque o `id` do botão (linha 2), também edite o atributo `text` (linha 8) para fazer uma referência a uma `string` definida no arquivo de `strings` que iremos chamar de `send_button`. Por último adicione um atributo `onClick` (linha 9) que define o método que será chamado quando esse botão for pressionado.

---

```

1 <Button
2     android:id="@+id/sendButton"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:layout_alignRight="@+id/nameField"
6     android:layout_below="@+id/nameField"
7     android:layout_marginTop="48dp"
8     android:text="@string/send_button"
9     android:onClick="sendMessage" />

```

---

Algoritmo 4.3: Código do botão

Agora iremos definir as `strings` usadas anteriormente no arquivo `res/values/strings.xml`. Abra ele e o modifique para que fique como mostrado no Algoritmo 4.4.

---

```

1 <resources>
2     <string name="app_name">MeuApp</string>
3     <string name="action_settings">Settings</string>
4     <string name="send_button">Enviar</string>
5     <string name="name">Nome</string>
6 </resources>

```

---

Algoritmo 4.4: Arquivo de strings com as duas strings adicionadas

Após terminar abra a classe `MainActivity.java` localizada na pasta `src/com.example.meuapp` do seu projeto e adicione um novo método que chamei de `sendMessage`, ele será responsável

<sup>6</sup>Mais informações na seção 5.3

por obter o conteúdo da caixa de texto e enviar para uma nova *activity* que irá mostrar esse conteúdo.

---

```
1 public void sendMessage(View view) {  
2     // Fazer alguma coisa em resposta ao clique do botao  
3 }
```

---

Algoritmo 4.5: Adicionando método à classe MainActivity

**Dica:** Isso vai requer você importe a classe View, você pode apertar Ctrl+Shift+O no Eclipse para importar classes que estejam faltando

---

```
1 import android.view.View;
```

---

Algoritmo 4.6: Exemplo de import de uma classe Android

Primeiro, crie um novo Intent<sup>7</sup>, um Intent é um objeto que provê uma facilidade para realizar uma ligação entre códigos de diferentes aplicações. O uso mais significante é a inicialização de novas *activities*.

---

```
1 public void sendMessage(View view) {  
2     // Fazer alguma coisa em resposta ao clique do botao  
3     Intent intent = new Intent(this, DisplayMessageActivity.class);  
4 }
```

---

Algoritmo 4.7: Adicionando uma Intent

Agora iremos obter o texto que está escrito na caixa para fazer algo com ele, no caso iremos enviar para outra *activity* que irá mostrar esse texto. Como é feito no algoritmo 4.7.

---

```
1 public void sendMessage(View view) {  
2     Intent intent = new Intent(this, DisplayMessageActivity.class);  
3     EditText textBox = (EditText) findViewById(R.id.nameField);  
4     String message = textBox.getText().toString();  
5     intent.putExtra(EXTRA_MESSAGE, message);  
6     startActivity(intent);  
7 }
```

---

Algoritmo 4.8: Obtendo o conteúdo da caixa de texto e enviando para outra *activity*

---

<sup>7</sup>Documentação Intent: <http://developer.android.com/reference/android/content/Intent.html>

O código na linha 3 está obtendo a referência a caixa de texto usando o método `findViewById()` passando o *id* da caixa de texto como parâmetro, esse *id* é obtido acessando uma variável estática da classe `R` (observe que esse é o mesmo *id* que você colocou no arquivo `xml` do layout da *activity*). Em seguida usando o método `getText()` da caixa de texto, obtem-se a *string* que foi escrita pelo usuário.

Por fim, essa *string* é colocada no `Intent` com o método `putExtra()`, uma `Intent` pode carregar consigo uma coleção de vários tipos de dados como pares chave-valor chamados *extras*, esse método toma a chave como primeiro parâmetro e o valor no segundo parâmetro. Para que a próxima *activity* consiga coletar esse valor, você deve definir uma chave para seu *extra* usando uma constante pública. Para isso adicione a definição de `EXTRA_MESSAGE` no topo da sua classe `MainActivity`.

---

```
1 public class MainActivity extends Activity {  
2     public final static String EXTRA_MESSAGE  
3         = "com.example.meuapp.MESSAGE";  
4     . . .  
5 }
```

---

Algoritmo 4.9: Constante como chave para um extra

Agora você deve criar uma nova *activity*, para isso vá em *File -> New -> Other -> Android Activity* e selecione *Blank Activity*. Preencha a próxima janela como na figura 4.8, depois clique *Finish*.

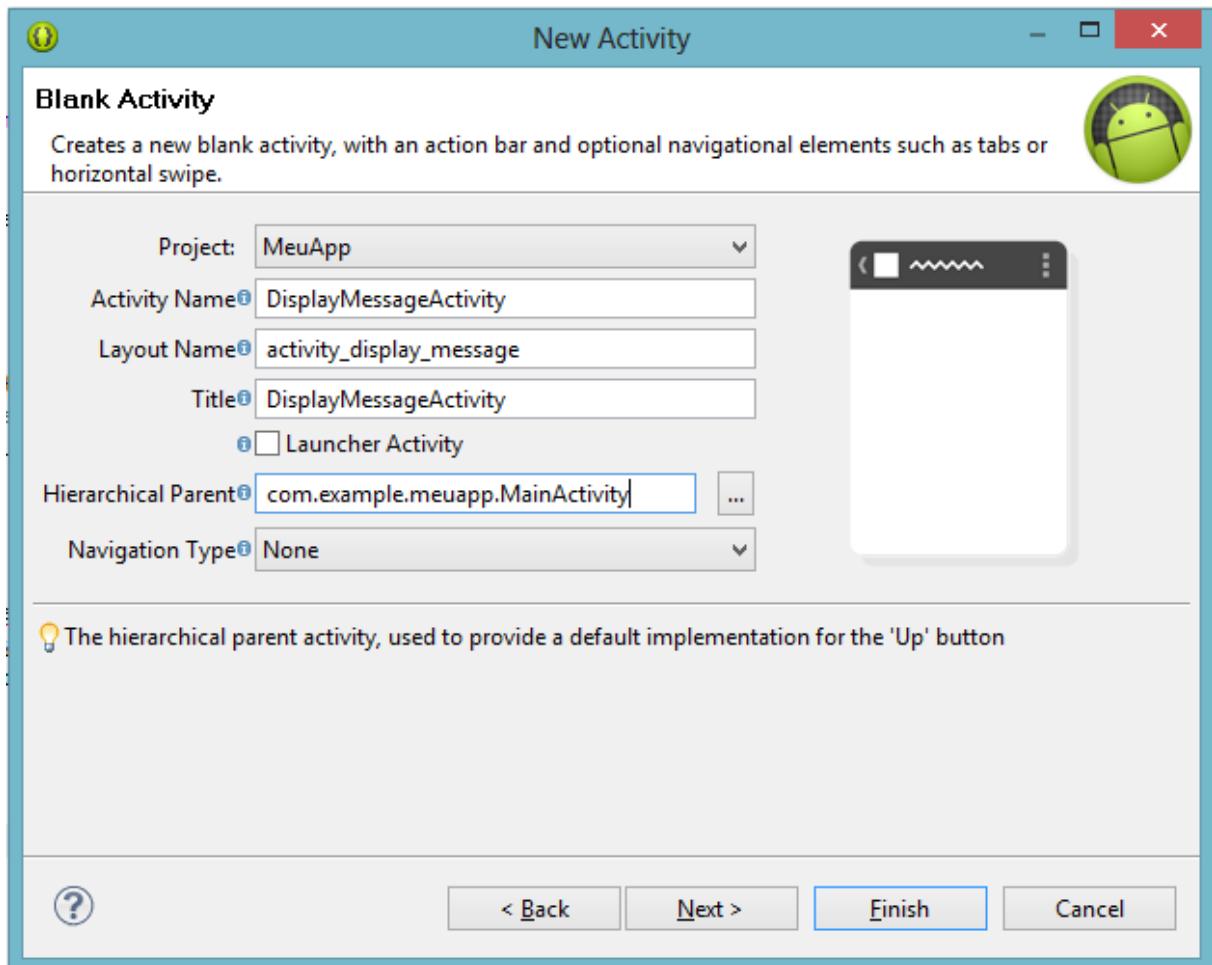


Figura 4.8: Criando uma nova *activity*

Observe a figura 4.8. Em *Project* você vai especificar o projeto em que a nova *activity* será adicionado. Em *Activity Name* especifique o nome da sua nova *activity*. Em *Layout Name* defina o nome do arquivo XML que contém o *layout* da nova *activity*. A opção *Title* define o título da *activity*, isso pode ser modificado posteriormente no arquivo de *strings* pois o título será definido ali após a criação da *activity*. A opção *Launcher Activity* ficará desmarcada pois essa *activity* não será usada para inicializar o aplicativo. Em *Hierarchical Parent* você vai definir o pai da nova *activity*, isso é usado para o Android implementar corretamente para qual *activity* o botão de voltar irá voltar. Por último *Navigation Type* deixe como *None* pois só queremos o *design* padrão. Clique em *Finish* para criar a nova *activity*.

**Dica:** O Eclipse adiciona automaticamente *activities* criadas por esse método no *Manifest*. Observe no *Manifest* como é feito caso você precise adicionar manualmente.

Abra a nova classe que foi criada junto com a *activity*. A classe já vem com alguns métodos implementados, alguns não serão necessários para esse aplicativo e serão explicados em outras seções, mas mantenha-os na classe. Todas as classes que são subclasses de *Activity* precisam implementar o método `onCreate()`<sup>8</sup> que define o procedimento a ser executado quando a *activity* é criada.

<sup>8</sup>[http://developer.android.com/reference/android/app/Activity.html#onCreate\(android.os.Bundle\)](http://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle))

Agora, precisamos extrair os dados enviados a essa *activity* através do *intent*, você pode obter a referência do *intent* que começou a *activity* chamando o método `getIntent()`<sup>9</sup>.

---

```
1 Intent intent = getIntent();
2 String mensagem = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

---

Algoritmo 4.10: Obtendo a *string* passada como extra passados através do Intent

Após obter a referência do *intent* que iniciou a *activity*, queremos coletar os extras que foram passado junto com ele. Criamos uma *string* que irá armazenar a mensagem que veio junto do *intent* e chamamos o método `getStringExtra()` passando como parâmetro a chave desse extra, que definimos na classe `MainActivity`. Agora para mostrar a mensagem na tela, você precisa criar um `TextView`<sup>10</sup>, essa *view* serve para mostrar texto.

---

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4
5     // Show the Up button in the action bar.
6     setupActionBar();
7
8     //Obtem o conteúdo da Intent
9     Intent intent = getIntent();
10    String mensagem = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
11
12    //Cria o TextView
13    TextView textView = new TextView(this);
14    textView.setTextSize(40);
15    textView.setText("Hello " + mensagem);
16
17    //Estabelece o text view como o layout da atividade
18    setContentView(textView);
19 }
```

---

Algoritmo 4.11: Método `onCreate()` recebendo um *Intent* e mostrando a mensagem

As linhas 3, 5 e 6 foram colocadas automaticamente na criação da *activity*, a linha 3 faz uma chamada ao método da superclasse, a linha 6 é um método que inicializa a *Action Bar*, que nesse aplicativo é a barra superior com o nome da *activity* e um menu de opções que já vem pré-programado. Deixamos isso como está.

O algoritmo 4.10 foi colocado nas linhas 9 e 10 para obter a referência ao *Intent*. Nas linhas 12-15 criamos um novo `TextView`, configuramos o tamanho da fonte e atribuímos o texto que será mostrado na tela a *view*, respectivamente.

Agora que o aplicativo está pronto, é necessário testar, caso tenha um smartphone Android você pode conectá-lo no seu computador e rodar diretamente, senão você deverá rodar em um

---

<sup>9</sup>[http://developer.android.com/reference/android/app/Activity.html#getIntent\(\)](http://developer.android.com/reference/android/app/Activity.html#getIntent())

<sup>10</sup><http://developer.android.com/reference/android/widget/TextView.html>

emulador. Lembrando que para ambos os casos é necessária a instalação do SDK primeiro, acesse *Android SDK Manager* e faça o download do SDK desejado.

Para rodar diretamente no smartphone:

1. Conecte seu smartphone no computador através do cabo USB. Se estiver desenvolvendo no Windows será preciso instalar os drivers USB do seu dispositivo. Se precisar de ajuda para instalar os drivers acesse: [OEM USB<sup>11</sup>](#)
2. Ative o modo *USB Debugging* no dispositivo
  - Para Android 3.2 ou mais antigos, a opção deve estar em *Configurações -> Aplicativos -> Desenvolvimento*
  - Para Android 4.0 e 4.1, a opção está em *Configurações -> Opções do desenvolvedor*
  - Para Android 4.2 e mais novos, a opção está escondida por padrão, para mostrar a opção você deve entrar em *Sobre o telefone* e clicar em *Número da versão* 7 vezes, ao retornar para tela anterior deverá aparecer *Opções do desenvolvedor*

**Dica:** Caso ocorra o erro *Launch error: adb rejected command: device not found*. Verifique se o aparelho está conectado e se os drivers estão instalados corretamente. Na área de notificações do aparelho deve ter uma notificação escrita: *Android debugging enabled*.

<sup>11</sup><http://developer.android.com/tools/extras/oem-usb.html>

Para rodar no emulador:

1. Abra o *SDK Manager* através do Eclipse em: *Window -> Android SDK Manager*
2. Verifique se, para Android 4.2.2 (API 17) ou outro desejado os seguintes pacotes estejam instalados
  - *SDK Platform* e;
  - *ARM EABI v7a System Image* ou;
  - *Intel x86 Atom System Image*
3. Verifique também se na aba *Tools*, os pacotes *Android SDK Tools* e *Android SDK Platform-tools* estão instalados
4. Agora é necessário criar um AVD (Android Virtual Device<sup>12</sup>). No Eclipse acesse o menu *Window -> Android Virtual Device Manager*
5. No AVD Manager clique em *New*
6. Complete as informações do AVD, especificando um aparelho, nome, plataforma, espaço de armazenamento, quantidade de memória RAM. Em *Device* haverá opções pré-configuradas de aparelhos do google, os *Nexus*, e opções genéricas de acordo com tamanho de tela. Em *Target* você deverá escolher a versão do sistema Android que deseja. Em alguns casos você poderá decidir pela CPU caso deseje ARM ou Intel Atom x86. A quantidade de RAM no Windows fica limitada a 768MB, mas que isso pode acarretar em erros no sistema.
7. Clique *Create AVD*
8. Ainda na janela *Android Virtual Device Manager* selecione o novo AVD e clique *Start*
9. Quando o emulador terminar de carregar, destrave a tela do emulador, usando o mouse.

Agora para rodar o aplicativo basta clicar em *Run* na barra de tarefas do Eclipse e selecionar *Android Application* na janela *Run as*. O Eclipse irá instalar o APK e abrir o aplicativo automaticamente, no dispositivo ou no emulador. As figuras 4.6, 4.7 e 4.8 mostram a execução do aplicativo.

---

<sup>12</sup><http://developer.android.com/tools/devices/index.html>

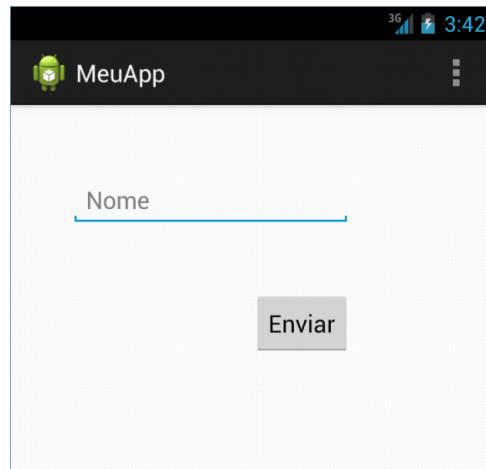


Figura 4.9: Primeira tela do primeiro aplicativo

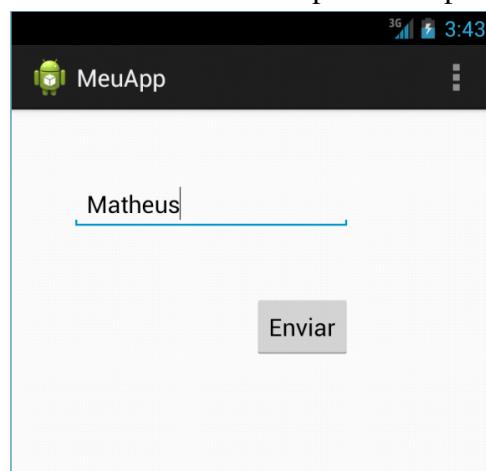


Figura 4.10: Primeira tela após escrever texto na caixa de texto

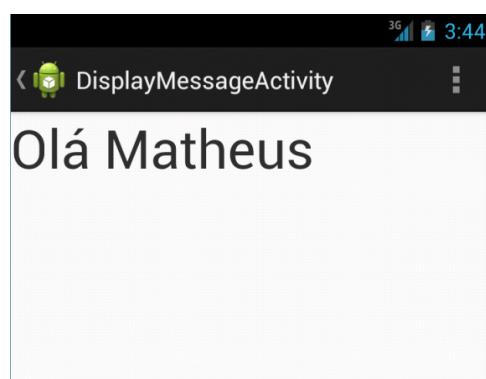


Figura 4.11: Segunda tela mostrando a mensagem enviada



# Design

## 5.1 Activity

Enquanto um usuário navega pelas variadas telas de um aplicativo, sai dele e volta depois, as instâncias de uma *activity* transitam dentre diferentes estados em seu ciclo de vida. Quando um aplicativo é iniciado, uma *activity* inicial é criada o sistema invoca métodos específicos que correspondem a criação dessa *activity*. Durante todo o ciclo de vida vários métodos são chamados, e todos eles correspondem a diferentes estágios desse ciclo de vida.

Observe na imagem abaixo os métodos correspondentes a cada estado da vida de uma *activity*, quando ela é criada o método `onCreate()` é o responsável pela configuração inicial. O sistema ao criar uma nova instância de uma *activity*, cada método muda o estado da *activity* um degrau pra cima na pirâmide.

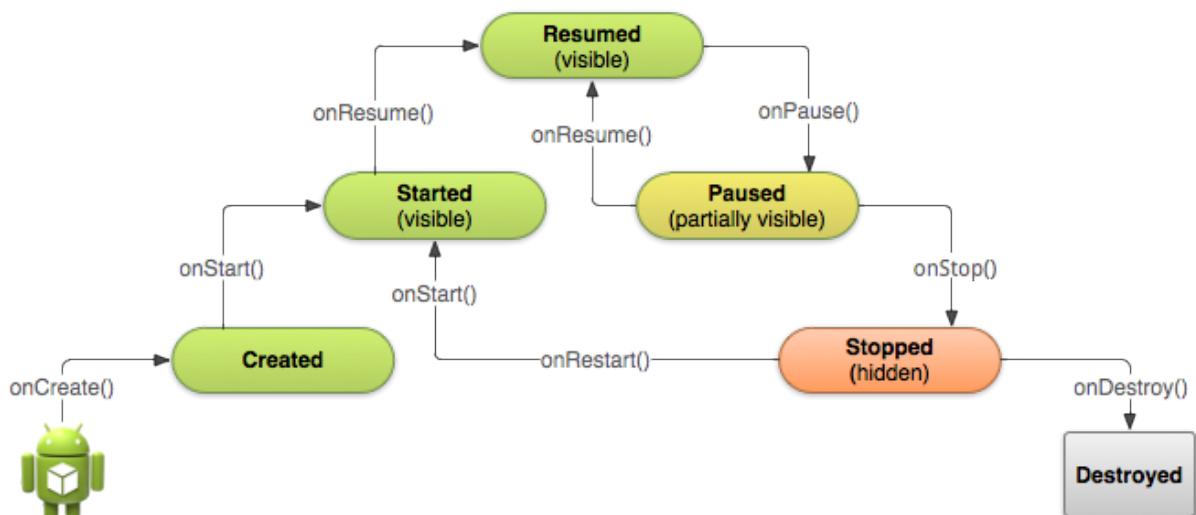


Figura 5.1: Ciclo de vida de uma *activity*

Assim que o usuário começa a sair da *activity*, o sistema invoca outros métodos que movem o estado para níveis mais baixos da pirâmide para começar a desmontar a *activity*. Em alguns

casos a *activity* irá apenas ir até certo ponto e esperar (por exemplo quando o usuário troca para outro aplicativo) tal que ela possa voltar de onde parou caso o usuário volte.

Não são todos métodos que precisam ser implementados pois isso irá depender da complexidade do seu aplicativo. É importante salientar porém que, implementar esses métodos irá garantir que seu aplicativo se comporte de maneira correta, por exemplo você deve garantir que:

- Seu aplicativo não falhe quando o usuário receber uma chamada telefônica ou quando o usuário troca de aplicativo;
- Seu aplicativo não consuma recursos do sistema enquanto não estiver sendo usado;
- Seu aplicativo não perca o progresso do usuário; e
- Seu aplicativo não falhe ou perca o progresso do usuário quando a tela rotaciona entre retrato e paisagem.

Apenas três dentre os estados são estáticos, isto é, a *activity* pode ficar nesse estado por um longo período de tempo:

#### Retomado (*Resumed*)

Nesse estado a *activity* está em primeiro plano e o usuário pode interagir com ela.

#### Pausado (*Paused*)

Nesse estado a *activity* está parcialmente obscurecida por outra *activity* - a outra *activity* que está em primeiro plano é semi-transparente ou não ocupa todo espaço da tela. A *activity* quando pausada não consegue interagir com o usuário e não executa nenhum código.

#### Parado (*Stopped*)

Nesse estado a *activity* está completamente oculto e não está visível para o usuário, está em plano de fundo. Quando está parada, uma instância de uma *activity* e toda informação de seu estado tais como variáveis são mantidos, porém a *activity* não executa nenhum código.

## 5.2 Especifique a *activity* que inicia seu aplicativo

Quando um usuário abre um aplicativo, o sistema chama o método `onCreate()` da *activity* que foi declarada como sendo a iniciadora do aplicativo. Você pode definir qual *activity* que vai iniciar seu aplicativo no arquivo `AndroidManifest.xml` que está no diretório raiz do seu projeto.

A *activity* que inicia seu aplicativo deve ser declarada no manifesto com um `<intent-filter>`<sup>1</sup> que inclui a `<action>` `MAIN` e a `<category>` `LAUNCHER`. Por exemplo:

---

```

1 <activity android:name=".MainActivity" android:label="@string/app_name">
2   <intent-filter>
3     <action android:name="android.intent.action.MAIN" />
4     <category android:name="android.intent.category.LAUNCHER" />
5   </intent-filter>
6 </activity>

```

---

Algoritmo 5.1: Exemplo de *Launcher activity*

<sup>1</sup>Documentação `<intent-filter>`: <http://developer.android.com/guide/topics/manifest/intent-filter-element.html>

**Dica:** Quando você cria um projeto Android no Eclipse, por padrão é incluída uma classe *activity* que está declarada no manifesto com esse filtro.

## 5.3 Tipos de Layout

Uma *Activity* contém *Views* e *ViewGroups*. Uma *view* é um elemento que têm presença na tela do dispositivo tais como botões, textos, imagens e etc. Um *ViewGroup* por sua vez é um elemento agrupador de *views* que provê um *layout* na qual você pode ajustar a ordem e aparição das *views*.

### 5.3.1 LinearLayout

O *LinearLayout* arranja *views* em uma única coluna ou uma única linha, desse modo as *views* podem ser arranjadas verticalmente ou horizontalmente. Como mostrado na figura 5.2:

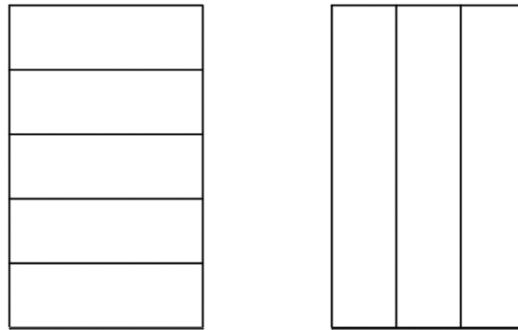


Figura 5.2: *LinearLayout* vertical (à esquerda) e horizontal (à direita)

*ViewGroups* também podem ser agrupados entre si para a criação de layouts mais complexos, por exemplo é possível agrupar um *LinearLayout* horizontal dentro de um *vertical* dessa forma é possível colocar *views* lado a lado em uma camadas do *LinearLayout* vertical, representada na figura 5.3.

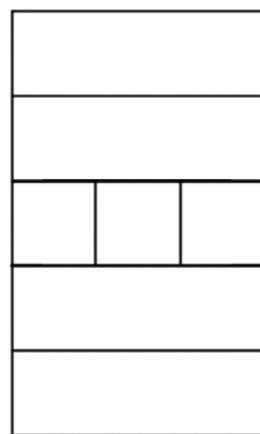


Figura 5.3: *LinearLayout* composto

### 5.3.2 RelativeLayout

O `RelativeLayout` permite especificar como as *views* são posicionadas uma em relação a outra. Cada *view* embutida no interior de um `RelativeLayout` tem atributos que permitem o seu alinhamento com outras *views*. Esses atributos podem ser encontrados na [documentação](#)<sup>2</sup>

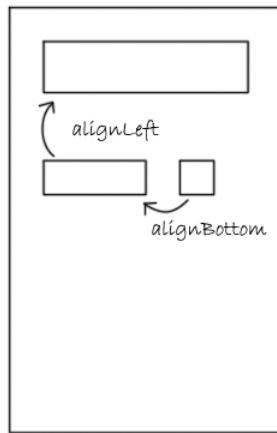


Figura 5.4: Exemplo de `RelativeLayout`

Novamente cabe comentar que é possível aninhar diferentes *ViewGroups* para formar um *layout* com maior complexidade.

### 5.3.3 FrameLayout

O `FrameLayout` é o mais simples e eficiente tipo de *layout*, pode ser usado apenas para mostrar uma *view* ou *views* que se sobrepõem. Geralmente é usado como um recipiente para os *Fragments*<sup>3</sup>.

Uma *view* definida em um `FrameLayout` sempre será colocado no canto superior esquerdo da tela do dispositivo ou do *ViewGroup* a que pertence o `FrameLayout`. Se mais de uma *view* foi definida elas serão empilhadas uma em cima da outra. Isso significa que a primeira *view* adicionada ao `FrameLayout` será mostrada na base da pilha, e a última adicionada será mostrada no topo.

Você pode fazer com que as *views* não sobreponham as outras usando o atributo `layout_gravity`<sup>4</sup>, dessa forma uma *view* pode ficar posicionada na borda inferior e outra na borda superior e não ficarem sobrepostas.

É possível posicionar as *views* dentro de um `FrameLayout` usando parâmetros diferentes no `layout_gravity`, no exemplo da figura 5.5 existe um `FrameLayout` com 3 elementos e cada um com parâmetros diferentes. É possível combinar os parâmetros utilizando a barra reta '|'.

<sup>2</sup><http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html>

<sup>3</sup>Mais informações na seção XXXXXX

<sup>4</sup><http://developer.android.com/reference/android/widget/FrameLayout.LayoutParams.html>

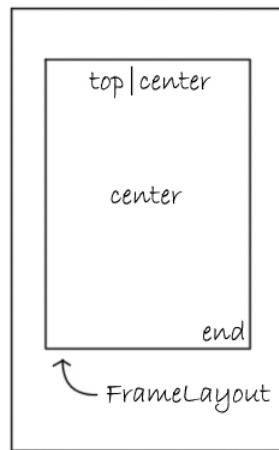


Figura 5.5: FrameLayout com exemplo de posicionamento usando `layout_gravity`

### 5.3.4 TableLayout

TableLayouts podem ser usadas para apresentar dados tabulados ou alinhar conteúdo como tabelas HTML em uma página web. Um TableLayout é composto de TableRows, uma cada para linha da tabela. Os conteúdos das TableRows são as *views* que vão em cada célula da tabela. Cada linha terá zero ou mais células e cada célula pode conter uma *view*.

O aspecto da TableLayout vai depender de alguns fatores. Primeiro, o número de colunas da tabela inteira vai depender do número de colunas da linha que contém mais colunas. Segundo, a largura de cada coluna é definida como a largura do conteúdo mais largo da coluna. Você pode combinar colunas para formar uma célula maior, mas não pode combinar linhas. Leia mais na [documentação](#)<sup>5</sup>

Embora TableLayouts possam ser usados para projetar interfaces, geralmente não é a melhor opção já que são derivadas de LinearLayouts. Se você tem dados que já estão em formato de tabela, como planilhas, então pode ser uma boa opção.

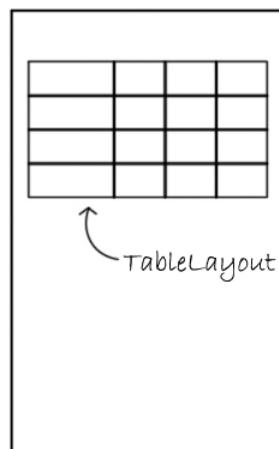


Figura 5.6: Exemplo de TableLayout

<sup>5</sup><http://developer.android.com/reference/android/widget/TableLayout.html>

## 5.4 Listas (ListView)

<sup>6</sup> Listas são uma das formas mais simples e poderosas de se mostrar informações ao usuário de forma objetiva. A ListView é capaz de apresentar uma lista rolável de itens.

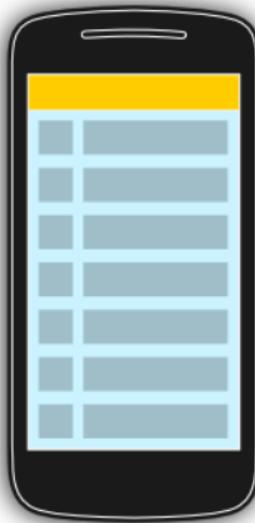


Figura 5.7: Esquema de uma lista

Um item individual da lista pode ser selecionado, essa seleção pode acionar uma outra tela com detalhes do item.

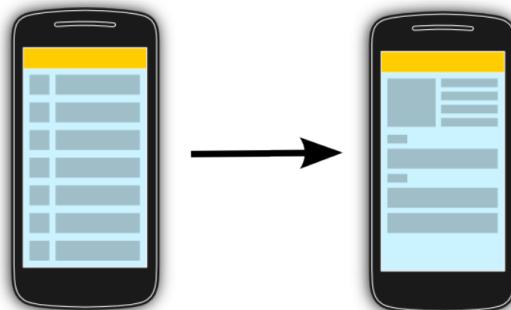


Figura 5.8: Detalhes de um elemento da lista

### 5.4.1 Adaptadores

Adaptadores são usados para providenciar dados a *views*. O adaptador também define como item da *view* será mostrada. Para ListViews o adaptador define como cada linha será mostrada.

Um adaptador deve extender a classe base `BaseAdapter`. O Android já tem alguns adaptadores padrão, os mais importantes são o  `ArrayAdapter` e o  `CursorAdapter`.

O  `ArrayAdapter` é usado para manipular dados em *arrays* ou listas (`java.util.List`). Já o  `SimpleCursorAdapter` consegue manipular dados em banco de dados.

<sup>6</sup>Documentação ListView:<http://developer.android.com/reference/android/widget/ListView.html>

### 5.4.2 Construção

A construção desse tipo de design é simples. No arquivo de *layout* da *activity* use o *LinearLayout* para conter a *ListView*.

---

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical" >
6 </LinearLayout>
```

---

Algoritmo 5.2: *LinearLayout* no arquivo de *layout*

Se estiver usando o construtor de interface gráfica, pode arrastar uma *ListView* para dentro do *layout*. Caso contrário pode construir manualmente no arquivo XML do *layout* da *activity*.

Você deve colocar o *LinearLayout* como raíz do arquivo XML, o elemento raiz sempre deve conter o atributo `xmlns:android` como mostrado na linha 2 do algoritmo 5.2, não entraremos em detalhes sobre os outros atributos.

Adicione uma *ListView*, escreva o código abaixo dentro do *LinearLayout*.

---

```

1 <ListView
2   android:id="@+id/listView1"
3   android:layout_width="match_parent"
4   android:layout_height="wrap_content" >
5 </ListView>
```

---

Algoritmo 5.3: Código de uma *ListView*

Você precisa popular a lista, para isso você pode criar um *string-array* no arquivo `strings.xml` com os elementos que deseja colocar na lista. Nesse exemplo do algoritmo 5.4 foi criado uma lista com nome `listString` e 4 itens que serão mostrados em forma de lista pela *ListView*.

---

```

1 <string-array name="listString">
2   <item>Menu 1</item>
3   <item>Menu 2</item>
4   <item>Menu 3</item>
5   <item>Menu 4</item>
6 </string-array>
```

---

Algoritmo 5.4: *string-array* populada com elementos

Finalmente, você deve escrever o código que irá preencher a lista com as *strings*. Como é feito no algoritmo 5.5 abaixo.

---

```

1 public class MainActivity extends Activity {
2     private ListView lv;
3
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_main);
8
9         //Obtem o array de strings para popular a lista
10        String listStr[] = getResources().getStringArray(R.array.listString);
11
12        //Obtem a lista
13        ListView lv = (ListView) findViewById(R.id.listView1);
14
15        //Adaptador das strings para a lista
16        lv.setAdapter(new ArrayAdapter<String>
17            (this, android.R.layout.simple_list_item_1, listStr));
18
19        /* Acao para quando clica num elemento da lista
20         * precisa criar um listener e programa-lo para
21         * realizar uma acao. */
22        lv.setOnItemClickListener(new OnItemClickListener() {
23
24            @Override
25            public void onItemClick(AdapterView<?> parent,
26                View view, int position, long id) {
27                //Quando clicado, mostra um Toast
28                Toast.makeText(getApplicationContext(),
29                    ((TextView) view).getText(), Toast.LENGTH_SHORT).show();
30            }
31        );
32    }
33    ...

```

---

Algoritmo 5.5: Código de uma *activity* com lista clicável

Primeiro, na linha 2, foi criada uma variável do tipo *ListView* para guardar um ponteiro para a *view* já definida no *layout*.

No método *onCreate()* você precisa criar e inicializar a lista na sua *activity*. Na linha 10 obtemos as *strings* do *string-array* e o guardamos na variável *listStr*. Usamos o método *getResources()* para poder adquirir o ponteiro para os recursos do aplicativo. Na linha 13 conseguimos o ponteiro pra lista e o guardamos na variável criada.

Usamos o adaptador ao chamar o método *ListView.setAdapter()* nas linhas 16-17 e passamos como parâmetro a criação de um novo adaptador do tipo *ArrayAdapter*. Para o construtor<sup>7</sup> desse adaptador está sendo passado o contexto atual da *activity*, um *layout* pré-definido do sistema, o *simple\_list\_item\_1*, e os dados na forma de *array*.

Na linha 22 usamos o método *ListView.setOnItemClickListener* para configurar uma ação a ser executada quando um item da lista for clicado. Neste exemplo é criado um

<sup>7</sup><http://developer.android.com/reference/android/widget/ArrayAdapter.html>

Toast, o Toast mostra uma mensagem em uma caixa de texto na parte inferior da tela por um curto período de tempo, nesse caso irá mostrar o mesmo texto do item da lista que foi clicado. Uma das aplicações mais comuns é fazer com que ao se clicar em um item da lista, uma nova *activity* seja aberta com detalhes do item.

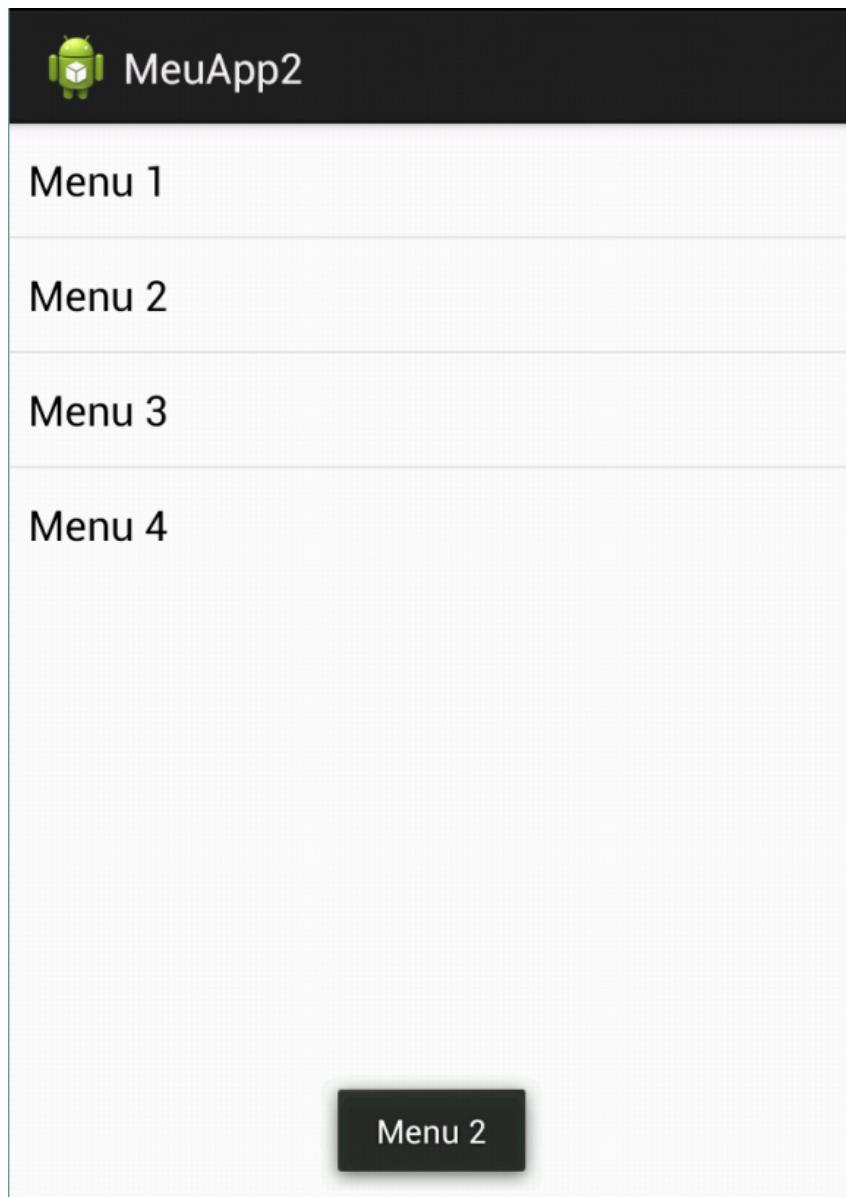


Figura 5.9: Lista simples

A figura 5.9 mostra como ficou o exemplo ao ser executado em um *smartphone*, o item "Menu 2" foi clicado e um *Toast* foi mostrado no momento do clique.

## 5.5 Listas Compostas

É possível compor um item da lista colocando mais elementos além de um texto. Para isso você precisa criar um novo arquivo XML que irá definir a customização de cada linha da *ListView*, nesse exemplo iremos definir um arquivo chamado `item.xml`, mostrado abaixo.

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="wrap_content"
4   android:layout_height="wrap_content"
5   android:orientation="horizontal" >
6
7   <ImageView
8     android:id="@+id/userIcon"
9     android:layout_width="wrap_content"
10    android:layout_height="wrap_content"
11    android:layout_margin="8dp" >
12  </ImageView>
13
14  <LinearLayout
15    android:layout_width="fill_parent"
16    android:layout_height="wrap_content"
17    android:layout_marginBottom="5dp"
18    android:layout_marginTop="5dp"
19    android:orientation="vertical"
20    android:paddingLeft="0px"
21    android:paddingRight="5dp" >
22
23  <TextView
24    android:id="@+id/username"
25    android:layout_width="wrap_content"
26    android:layout_height="wrap_content"
27    android:layout_alignParentLeft="true"
28    android:textColor="#FFF38585"
29    android:textSize="15sp" >
30  </TextView>
31
32  <TextView
33    android:id="@+id/usertext"
34    android:layout_width="wrap_content"
35    android:layout_height="wrap_content"
36    android:layout_marginTop="4dp"
37    android:textColor="#FF444444"
38    android:textSize="13sp" >
39  </TextView>
40
41  </LinearLayout>
42 </LinearLayout>

```

---

Algoritmo 5.6: Código do arquivo item.xml

O algoritmo 5.6 mostra como você pode fazer a customização de um item da lista. Nesse exemplo há uma pequena imagem à esquerda e dois textos de cores e tamanhos diferentes. Para isso primeiro criamos um LinearLayout que irá conter uma ImageView para mostrar a imagem e outro LinearLayout para colocar os dois textos. As cores do textos são configuradas com o atributo `textColor` e usa o padrão HTML de cores.

Agora você precisa usar um adaptador para mostrar esse layout customizado em cada linha da lista, usaremos a classe `SimpleAdapter`<sup>8</sup>. Essa classe faz a adaptação de um `ArrayList` de `Maps` para um *layout* definido.

---

```

1 public class MainActivity extends Activity {
2     private ListView lv;
3
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_main);
8
9         //Obtem a lista
10        ListView lv = (ListView) findViewById(R.id.listView1);
11
12        //Cria uma lista de maps(key->value) dos views de cada item do ListView
13        List<Map> list = new ArrayList<Map>();
14        Map map = new HashMap();
15        map.put("userIcon", R.drawable.miku);
16        map.put("userName", "Hatsune Miku");
17        map.put("userText", "Texto exemplo para o adaptador");
18        list.add(map);
19        map = new HashMap();
20        map.put("userIcon", R.drawable.luka);
21        map.put("userName", "Megurine Luka");
22        map.put("userText", "Texto exemplo para o adaptador");
23        list.add(map);
24
25        //Cria um adaptador pro layout customizado
26        SimpleAdapter adapter = new SimpleAdapter(this,
27                (List<? extends Map<String, ?>>) list, R.layout.item,
28                new String[] {"userIcon", "userName", "userText"},
29                new int[] {R.id.userIcon, R.id.username, R.id.usertext});
30
31        lv.setAdapter(adapter);
32    }

```

---

Algoritmo 5.7: Código da lista customizada

Observando o algoritmo 5.7. Na linha 13 criamos um `ArrayList` de `Maps`. Na linha 14 e 19 criamos um `HashMap` onde a chave é uma *string* que identifica o conteúdo, essas chaves serão *userIcon*, *userName* e *userText* respectivamente. Em *userIcon* colocamos uma imagem, essa imagem deve ser colocada nas subpastas da pasta `drawable` e é acessada através da classe `R`. Em *userName* colocamos um nome de usuário, por exemplo. Em *userText* poderia ser colocada uma descrição, ou uma frase customizada do usuário mas nesse exemplo foi colocado uma sentença qualquer. Nas linhas 18 e 23 adicionamos o `Map` criado no `ArrayList`.

Criamos o `SimpleAdapter` nas linhas 26-29. Para o construtor passamos o `ArrayList` de `Maps` que contém os dados, passamos também o *layout* que definimos anteriormente

---

<sup>8</sup><http://developer.android.com/reference/android/widget/SimpleAdapter.html>

`R.layout.item`. Passamos um *array* de *strings* que contém as chaves que serão usadas para obter os dados e por último um *array* de inteiros que contém os *ids* das *views* em que os conteúdos dos Maps serão colocados.

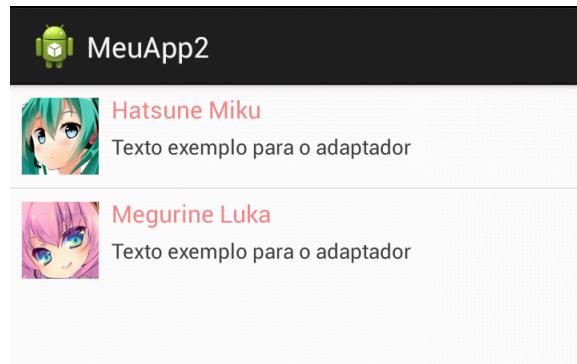


Figura 5.10: Lista Composta

A figura 5.10 mostra como ficou o exemplo acima ao ser executado em um *smartphone*.

## 5.6 Listas expansíveis (`ExpandableListView`)

Listas expansíveis são úteis para agrupar conjuntos de itens semelhantes, funcionam da mesma maneira que as listas comuns e podem ser customizadas. Comece colocando sua lista no *layout* da *activity* desejada.

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent"
5   android:orientation="vertical" >
6
7   <ExpandableListView
8     android:id="@+id/expandableList"
9     android:layout_width="match_parent"
10    android:layout_height="wrap_content"
11    android:transcriptMode="alwaysScroll"
12    android:listSelector="@android:color/holo_green_light">
13   </ExpandableListView>
14
15 </LinearLayout>
```

Algoritmo 5.8: Código XML de uma Lista expansível

**Dica:** O atributo `transcriptMode="alwaysScroll"` vai fazer com que a lista sempre role até o final quando você expande ou contrai um grupo. O atributo `listSelector` coloca o item da lista quando este é clicado.

Agora crie 2 novos arquivos XML, um chamado `list_item_parent.xml` e o outro chamado `list_item_child.xml` dentro da pasta `res/layout`.

```
1 <LinearLayout  
2   xmlns:android="http://schemas.android.com/apk/res/android"  
3   android:id="@+id/list_item"  
4   android:orientation="horizontal"  
5   android:layout_width="fill_parent"  
6   android:layout_height="fill_parent">  
7  
8   <TextView  
9     android:id="@+id/list_item_text_view"  
10    android:layout_width="0dp"  
11    android:layout_height="wrap_content"  
12    android:textSize="20sp"  
13    android:padding="10dp"  
14    android:layout_weight="1"  
15    android:layout_marginLeft="35dp" />  
16  
17 </LinearLayout>
```

---

Algoritmo 5.9: Layout list\_item\_parent.xml

Nesses dois *layouts* teremos apenas uma *TextView* para abrigar um texto.

```
1 <LinearLayout  
2   xmlns:android="http://schemas.android.com/apk/res/android"  
3   android:id="@+id/list_item_child"  
4   android:orientation="vertical"  
5   android:layout_width="fill_parent"  
6   android:layout_height="fill_parent"  
7   android:gravity="center_vertical">  
8  
9   <TextView  
10    android:layout_width="wrap_content"  
11    android:layout_height="wrap_content"  
12    android:id="@+id/list_item_text_child"  
13    android:textSize="20sp"  
14    android:padding="10dp"  
15    android:layout_marginLeft="5dp" />  
16  
17 </LinearLayout>
```

---

Algoritmo 5.10: Layout list\_item\_child.xml

Em seguida precisamos criar uma classe que irá abrigar os dados dos elementos pai, elementos estes que serão expandidos quando clicados. Nesse exemplo criamos uma classe *Parent*, como mostrado no algoritmo 5.11

```

1 public class Parent {
2     private String mTitle;
3     private ArrayList<String> mArrayChildren;
4
5     public String getTitle() {
6         return mTitle;
7     }
8
9     public void setTitle(String mTitle) {
10        this.mTitle = mTitle;
11    }
12
13    public ArrayList<String> getArrayChildren() {
14        return mArrayChildren;
15    }
16
17    public void setArrayChildren(ArrayList<String> mArrayChildren) {
18        this.mArrayChildren = mArrayChildren;
19    }
20}

```

---

Algoritmo 5.11: Classe Parent

Essa classe contém o texto do item, que será guardado na *string* `mTitle` e um *ArrayList* que irá comportar os sub-itens desse item. Os métodos *get* e *set* são simples.

Em seguida, crie uma nova classe, `CustomAdapter` que será o adaptador da lista expansível para os dados, para esse exemplo estaremos adaptando apenas para o uso de texto. Essa classe deve extender a classe `BaseExpandableListAdapter`.

```

1 public class CustomAdapter extends BaseExpandableListAdapter {
2     private LayoutInflator inflater;
3     private ArrayList<Parent> parent;
4
5     public CustomAdapter(Context context, ArrayList<Parent> parent) {
6         this.parent = parent;
7         inflater = LayoutInflator.from(context);
8     }
9
10    @Override
11    //Obtem o nome de cada item
12    public Object getChild(int groupPosition, int childPosition) {
13        return parent.get(groupPosition).getArrayChildren().
14            get(childPosition);
15    }
16
17    @Override
18    public long getChildId(int groupPosition, int childPosition) {
19        return childPosition;
20    }

```

```
21
22     @Override
23     //Nesse metodo voce seta os textos para ver os filhos na lista
24     public View getChildView(int groupPosition, int childPosition,
25         boolean isLastChild, View view, ViewGroup viewGroup) {
26
27         if(view == null) {
28             view = inflater.inflate(R.layout.list_item_child, viewGroup,
29                             false);
30         }
31
32         TextView textView = (TextView)
33             view.findViewById(R.id.list_item_text_child);
34
35         textView.setText(parent.get(groupPosition).getArrayChildren() .
36             get(childPosition));
37
38         return view;
39     }
40
41     @Override
42     public int getChildrenCount(int groupPosition) {
43         //retorna o tamanho do array de filhos
44         return parent.get(groupPosition).getArrayChildren().size();
45     }
46
47     @Override
48     //Obtem o titulo de cada pai
49     public Object getGroup(int groupPosition) {
50         return parent.get(groupPosition).getTitle();
51     }
52
53     @Override
54     public int getGroupCount() {
55         return parent.size();
56     }
57
58     @Override
59     public long getGroupId(int groupPosition) {
60         return groupPosition;
61     }
62
63     @Override
64     //Nesse metodo voce seta o texto para ver os pais na lista
65     public View getGroupView(int groupPosition, boolean isExpanded,
66         View view, ViewGroup viewGroup) {
67
68         if(view == null) {
69             //Carrega o layout do parent na view
70             view = inflater.inflate(R.layout.list_item_parent, viewGroup,
71                             false);
```

```

72     }
73
74     //Obtem o textView
75     TextView textView = (TextView)
76         view.findViewById(R.id.list_item_text_view);
77
78     textView.setText(getGroup(groupPosition).toString());
79
80     return view;
81 }
82
83 @Override
84 public boolean hasStableIds() {
85     return true;
86 }
87
88 @Override
89 public boolean isChildSelectable(int groupPosition,
90         int childPosition) {
91     return true;
92 }
93 }
```

---

#### Algoritmo 5.12: Classe CustomAdapter

Primeiro precisamos de um `LayoutInflater`<sup>9</sup> que irá instanciar o *layout XML* nas *views* correspondentes, e um *array* da classe `Parents` que criamos anteriormente, esses serão os itens principais da lista.

Na linha 7 no construtor da classe, usamos o método `LayoutInflater.from()` para obter o *inflater* do contexto da *activity*.

Ao extender a classe `BaseExpandableListAdapter` temos que programar alguns métodos. O método `getChild()` deve adquirir o ponteiro para um subitem de um item na lista. O método  `getChildId()` deve obter o *id* de um subitem, porém nesse exemplo não temos nada configurado então usamos a própria posição desse subitem como *id* e retornamos `childPosition`.

O método `getChildView` na linha 24 vai atribuir o *layout* dos subitens na linha 28. Na linha 32 obtemos o `TextView` desse subitem e com o método `TextView.setText()` atribuímos seu respectivo texto. Esse texto está guardado no *array* chamado `mArrayChildren` da classe `Parent`, então a fim de obter esse texto devemos obter o `Parent` correto. Quando você clica em um item da lista, o Android guarda qual item você clicou no parâmetro `groupPosition`. Em seguida se obtém o texto de cada subitem pelo parâmetro `childPosition`.

Outro método importante é o `getGroupView`, funciona da mesma maneira que `getChildView` mas configurando os *views* dos itens pai em vez dos subitens.

Para finalizar, você deve construir os objetos na classe da *activity*, nesse exemplo para popular a lista eu coloquei no arquivo de `strings` alguns fabricantes e modelos de carros, você pode obtê-los no repositório do projeto.

<sup>9</sup><http://developer.android.com/reference/android/view/LayoutInflator.html>

```
1 public class MainActivity extends Activity {
2     private ExpandableListView mExpandableList;
3
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_main);
8
9         mExpandableList = (ExpandableListView)
10            findViewById(R.id.listaExpandivel);
11
12        ArrayList<Parent> arrayParents = new ArrayList<Parent>();
13        ArrayList<String> arrayChildren;
14
15        //Array de fabricantes no arquivo de strings
16        String parentsNames[] = getResources().
17            getStringArray(R.array.Fabricantes);
18
19        for(int i = 0; i < parentsNames.length; i++) {
20            /*Para cada pai "i" criar um novo objeto
21            Parent para setar o nome e os filhos */
22            Parent parent = new Parent();
23            parent.setTitle(parentsNames[i]);
24
25            arrayChildren = new ArrayList<String>();
26            /* Obtem os carros daquele fabricante
27             * primeiro obtendo o resource id (passando o nome do fabricante)
28             * depois usando esse resource id para obter o array de strings
29             */
30            int resId = getResources().
31                getIdentifier(parentsNames[i], "array", getPackageName());
32            String childrenNames[] = getResources().getStringArray(resId);
33
34            for(int j = 0; j < childrenNames.length; j++) {
35                arrayChildren.add(childrenNames[j]);
36            }
37
38            parent.setmArrayChildren(arrayChildren);
39            arrayParents.add(parent);
40        }
41
42        mExpandableList.setAdapter(
43            new CustomAdapter(MainActivity.this, arrayParents));
44    }
45    ...
46 }
```

---

Algoritmo 5.13: Construindo a lista expandível na *activity*

O algoritmo 5.13 é a construção da lista expansível na *activity*, na linha 9-10 obtemos a *view* usando `findViewById()`. A linha 30-31 são um pouco mais complicadas, primeiro é preciso obter o *id* do *string-array* que é subitem do item atual no laço de repetição. Para isso usamos `getResources().getIdentifier()` para obter o *id* do subitem a partir do nome do item pai. Em seguida podemos acessar o *string-array* normalmente como é feito na linha 31.

Na linha 41 usamos o *CustomAdapter* que criamos anteriormente.

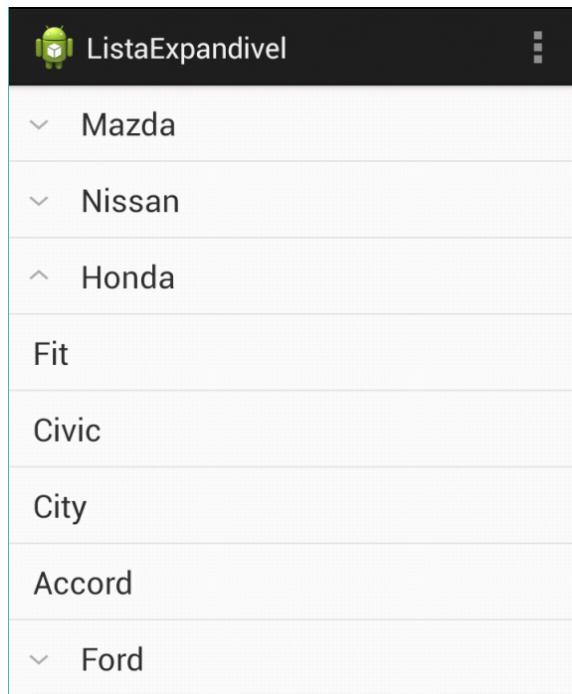


Figura 5.11: Exemplo de lista expansível rodando em um *smartphone*

## 5.7 Grades (GridView) e imagens ImageView

Grades são úteis para mostrar imagens e fotos como uma galeria, ou permitir a seleção de categorias semelhante a uma lista. A idéia é ter elementos lado a lado para mostrar ou para selecionar e mostrar mais detalhes. Basicamente funciona como uma grade bi-dimensional que pode ser arrastada para os lados ou de cima pra baixo.

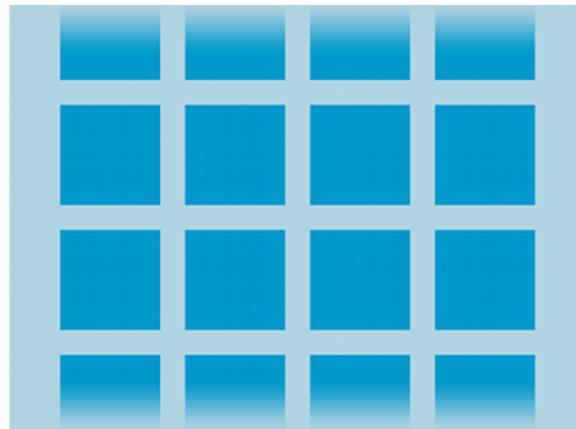


Figura 5.12: Esquema de um GridView

Comece colocando um GridView<sup>10</sup> no *layout* de sua *activity*.

---

```
1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent"
5   android:orientation="vertical" >
6
7   <GridView
8     android:id="@+id/gridview"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent"
11    android:numColumns="auto_fit"
12    android:columnWidth="90dp"
13    android:horizontalSpacing="10dp"
14    android:verticalSpacing="10dp"
15    android:gravity="center"
16    android:stretchMode="columnWidth" >
17   </GridView>
18
19 </LinearLayout>
```

---

Algoritmo 5.14: Layout do GridView

Vamos criar uma nova classe que será o adaptador de imagens para o GridView, chamemos a classe de `ImageAdapter`, ela é mostrada no algoritmo 5.15.

---

<sup>10</sup><http://developer.android.com/reference/android/widget/GridView.html>

```

1 public class ImageAdapter extends BaseAdapter {
2     private Context mContext;
3
4     //Mantendo todos os ids num array
5     public Integer[] thumbIds = {
6         R.drawable.sample_0, R.drawable.sample_1,
7         R.drawable.sample_2, R.drawable.sample_3,
8         R.drawable.sample_4, R.drawable.sample_5,
9         R.drawable.sample_6, R.drawable.sample_7
10    };
11
12    //Construtor
13    public ImageAdapter(Context c) {
14        mContext = c;
15    }
16
17    @Override
18    //Retorna o tamanho do array
19    public int getCount() {
20        return thumbIds.length;
21    }
22
23    @Override
24    //Retorna um elemento do array
25    public Object getItem(int position) {
26        return thumbIds[position];
27    }
28
29    @Override
30    //Nao sera usado
31    public long getItemId(int position) {
32        return 0;
33    }
34
35    @Override
36    public View getView(int position, View convertView,
37    ViewGroup parent) {
38        ImageView imageView = new ImageView(mContext);
39        imageView.setImageResource(thumbIds[position]);
40        imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
41        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
42        return imageView;
43    }
44}

```

---

Algoritmo 5.15: Classe ImageAdapter

A classe ImageAdapter deve ser subclasse da classe BaseAdapter<sup>11</sup>. Temos, como

<sup>11</sup><http://developer.android.com/reference/android/widget/BaseAdapter.html>

variável pública, um *array* das imagens que queremos colocar na grade. Como todo *id* de um recurso da classe R é um inteiro, criamos um *array* de inteiros. Note que estamos considerando que todas as imagens já foram devidamente colocadas na pasta drawable.

O método mais importante é o método `getView()`. Nele criamos uma nova `ImageView`<sup>12</sup> para abrigar a imagem que queremos colocar na grade. Em seguida configuramos alguns parâmetros desse `ImageView`, o método `ImageView.setImageResource()` é responsável por estabelecer um `drawable` como conteúdo do `ImageView`. Já o método `View.setLayoutParams()` configura os parâmetros de *layout* associados com essa *view*, note que para esse método passamos parâmetros de *layout* de uma `GridView`, que por sua vez recebe (200, 200) como largura e altura de um elemento da grade.

`ImageView.setScaleType` controla como a imagem deve ser redimensionada para condizer com o tamanho do `ImageView`, `ImageView.ScaleType`<sup>13</sup> são as formas disponíveis para escalar a imagem.

Agora basta criar a grade em sua *activity*.

---

```

1 public class MainActivity extends Activity {
2
3     @Override
4     public void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.activity_main);
7
8         GridView gridView = (GridView) findViewById(R.id.gridview);
9
10        // Instance of ImageAdapter Class
11        gridView.setAdapter(new ImageAdapter(this));
12    }
13    ...
14 }
```

---

Algoritmo 5.16: *activity* com grade

Exemplo acima rodando em um *smartphone* na figura 5.13:

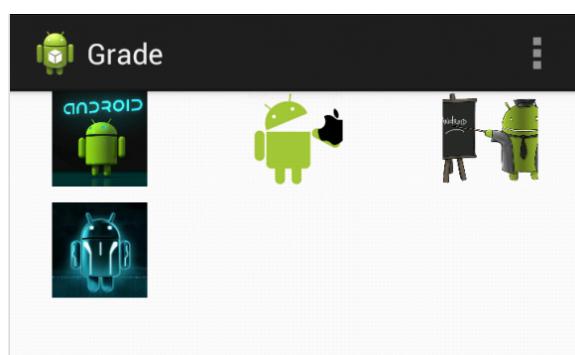


Figura 5.13: Demonstração de um Grid view

<sup>12</sup><http://developer.android.com/reference/android/widget/ImageView.html>

<sup>13</sup><http://developer.android.com/reference/android/widget/ImageView.ScaleType.html>

Para complementar, você pode fazer com que a imagem abra em tela cheia quando clicada na view, para isso é necessário que você passe o *id* do recurso do *GridView* para uma nova *activity* que irá mostrar a imagem em tela cheia. Para isso precisamos criar um novo *layout XML*, a qual chamaremos de *full\_image.xml*, nele teremos apenas uma *ImageView* e um *TextView* que será uma pequena legenda da imagem.

---

```

1 <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:id="@+id/relativelayout"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent" >
6
7   <ImageView
8     android:id="@+id/full_image_view"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent" />
11
12  <TextView
13    android:id="@+id/myImageViewText"
14    android:layout_width="fill_parent"
15    android:layout_height="40dp"
16    android:gravity="center"
17    android:background="#55555555"
18    android:textSize="16sp"
19    android:textColor="#FFFFFF" />
20
21 </RelativeLayout>

```

---

Algoritmo 5.17: Layout *full\_image.xml*

Note que para o *TextView* usaremos o atributo *layout\_width* como *fill\_parent* e *layout\_height* como *40dp*, dessa forma criamos um pequeno retângulo de altura fixa mas de forma que a largura preecha a tela completamente. O atributo *background* com o valor *#55555555* faz com que a cor do retângulo seja cinza com transparência, já que o parâmetro *alfa* também tem valor *0x55*. Também deixamos o texto com cor branca com o atributo *textColor*.

Em seguida, crie uma nova classe chamada *FullImageActivity*, essa é a *activity* que vai mostrar a imagem em tela cheia. A construção da classe é simples, você deve apenas obter o *id* da imagem passado como *extra* através do *intent* e então obter essa imagem da classe *ImageAdapter*.

---

```

1 public class FullImageActivity extends Activity {
2     public void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.full_image);
5
6         //Obtem os dados do intent
7         Intent intent = getIntent();
8
9         //Seleciona o id da imagem
10        int id = intent.getExtras().getInt("id");
11        ImageAdapter imageAdapter = new ImageAdapter(this);
12
13        //Configura o ImageView para mostrar a imagem correspondente
14        ImageView imageView = (ImageView) findViewById(R.id.full_image_view);
15        imageView.setImageResource(imageAdapter.thumbIds[id]);
16
17        //Configura o TextView para mostrar uma descrição da imagem
18        TextView textView = (TextView) findViewById(R.id.myImageViewText);
19        textView.setText("Image id: " + id);
20    }
21 }
```

---

Algoritmo 5.18: Classe FullImageActivity

---

```

1 @Override
2 public void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_main);
5     GridView gridView = (GridView) findViewById(R.id.gridview);
6     gridView.setAdapter(new ImageAdapter(this));
7
8     //Cria um listener para o evento de clique em um elemento da grade
9     gridView.setOnItemClickListener(new OnItemClickListener() {
10
11         @Override
12         public void onItemClick(AdapterView<?> parent, View view,
13             int pos, long id) {
14             //Envia o id da imagem para o FullImageActivity
15             Intent intent = new Intent(getApplicationContext(),
16                 FullImageActivity.class);
17             intent.putExtra("id", pos);
18             startActivity(intent);
19         }
20     });
21 }
```

---

Algoritmo 5.19: Código da *activity* após as modificações

No algoritmo 5.19, configuramos um `View.setOnClickListener()` de forma que quando uma imagem da grade for clicada, um Intent seja enviado a uma nova *activity* que por sua vez ficará encarregada de mostrar a imagem em tela cheia. Quando um item é clicado conseguimos obter a posição dele na grade com o parâmetro `pos` da função `onItemClick()`, essa posição é equivalente ao *id* da imagem no array criado na classe `ImageAdapter`. A `FullImageActivity` por sua vez recebe esse Intent que possui o *id* da imagem que deve ser mostrada e configura o `ImageView` de acordo.

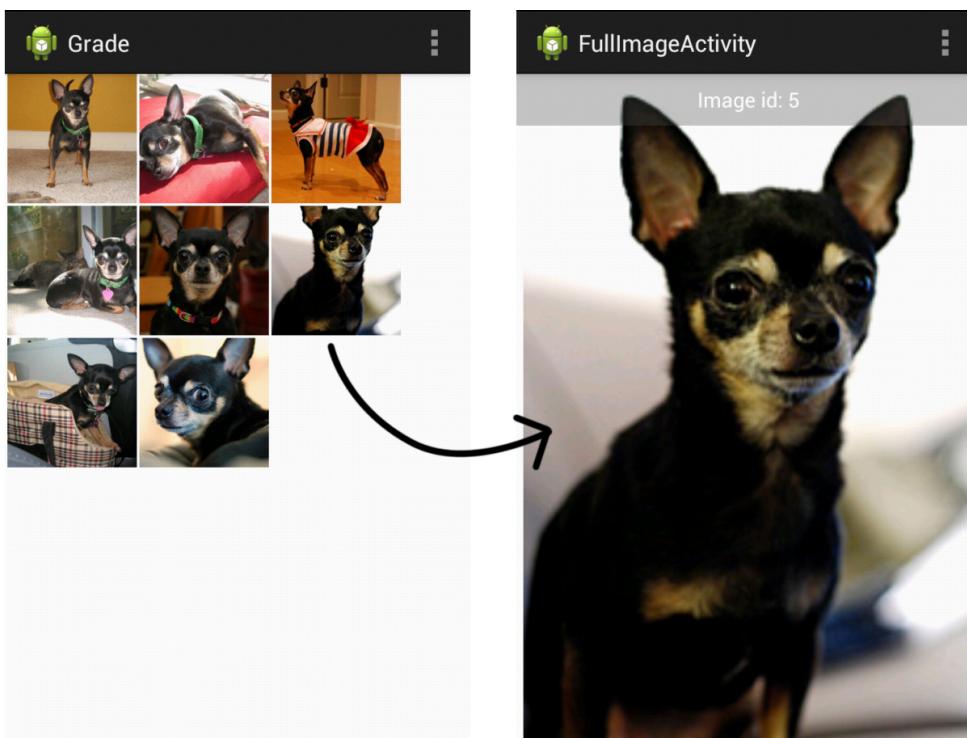


Figura 5.14: Exemplo GridView com imagem em tela cheia

## 5.8 Fragmentos

Fragmentos são a solução do Android para criar interfaces de usuário modulares, eles vivem dentro das *activity* e uma *activity* pode conter vários fragmentos. Assim como as *activity* os fragmentos possuem um ciclo de vida.

Dentre as vantagens de um fragmento estão:

- Modularidade e reuso de código
- Habilidade de construir interfaces com múltiplos painéis
- Facilidade de construir aplicativos para celulares e tablets

O primeiro conceito a ser coberto é como construir um fragmento, comece definindo o *layout* do fragmento.

Um layout bem simples, apenas com um botão para efeito de demonstração. Agora crie uma classe `BasicFragment`

```
1 public class BasicFragment extends Fragment {
2
3     @Override
4     public View onCreateView(LayoutInflater inflater,
5             ViewGroup container, Bundle savedInstanceState) {
6
7         //Obtem o layout do fragmento em uma view
8         View view = inflater.inflate(R.layout.fragment, container, false);
9
10        //Obtem o botao da view
11        Button button = (Button) view.findViewById(R.id.fragment_button);
12
13        //Um listener simples para o botao
14        button.setOnClickListener(new OnClickListener() {
15
16            @Override
17            public void onClick(View v) {
18                Activity activity = getActivity();
19
20                if(activity != null){
21                    Toast.makeText(activity,
22                        "A toast to a fragment", Toast.LENGTH_SHORT).show();
23                }
24            }
25        });
26        return view;
27    }
28 }
```

---

Algoritmo 5.20: Classe BasicFragment

Caso você esteja desenvolvendo para API menores que 11 (HoneyComb 3.0) você vai precisar usar a API de retrocompatibilidade que o Google providenciou para essas APIs, você precisa importar a classe de suporte:

```
import android.support.v4.app.Fragment;
```

Agora para incluir o fragmento na *activity* existem duas opções. A primeira é incluir o fragmento no XML da *activity* como você faria com qualquer *view*.

---

```

1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="fill_parent"
3     android:layout_height="fill_parent"
4     android:orientation="vertical" >
5
6     <fragment
7         android:id="@+id/fragment_content"
8         android:name="com.example.fragmento.BasicFragment"
9         android:layout_width="fill_parent"
10        android:layout_height="fill_parent" >
11    </fragment>
12 </LinearLayout>

```

---

Algoritmo 5.21: Layout da *activity* com um fragmento

Você pode usar o `<fragment>` quantas vezes quiser para incluir múltiplos fragmentos. Note que você precisa usar um nome qualificado em `android:name`, veja mais na documentação oficial: [activity-element<sup>14</sup>](#)

Novamente, caso esteja desenvolvendo para APIs menores que 11, você vai precisar fazer a *activity* extender a classe `FragmentActivity` e importar a classe de suporte:

```

import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity

```

Simplesmente configurando a *activity* para usar o fragmento vai fazer com que o fragmento seja adicionado e renderizado na tela, entretanto você deve querer ter mais controle de quando e como seus fragmentos serão adicionados durante o curso do seu aplicativo. Para isso existe uma maneira alternativa de adicionar o fragmento em tempo de execução. A fim de adicionar o fragmento em tempo de execução você precisa fazer uma mudança no layout da *activity*:

---

```

1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="fill_parent"
3     android:layout_height="fill_parent"
4     android:orientation="vertical" >
5
6     <FrameLayout
7         android:id="@+id/fragment_content"
8         android:layout_width="fill_parent"
9         android:layout_height="fill_parent" />
10
11 </LinearLayout>

```

---

Algoritmo 5.22: Layout da *activity* com o `FrameLayout`

E uma mudança na *activity* que vai mostrar o fragmento:

<sup>14</sup><http://developer.android.com/guide/topics/manifest/activity-element.html#nm>

---

```

1 public class MainActivity extends FragmentActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.activity_main);
7
8         //Como estamos usando o pacote de suporte
9         //Precisamos usar o Manager desse pacote
10        FragmentManager fm = getSupportFragmentManager();
11        //Voce pode obter um fragmento da mesma forma que obtém
12        //qualquer outra view usando o FragmentManager
13        Fragment fragment = fm.findFragmentById(R.id.fragment_content);
14
15        if(fragment == null) {
16            //Começa uma transação de fragmentos
17            FragmentTransaction ft = fm.beginTransaction();
18            //Adiciona o fragmento
19            ft.add(R.id.fragment_content, new BasicFragment());
20            //"/Commita" a transação
21            ft.commit();
22        }
23    }
24    ...

```

---

Algoritmo 5.23: *activity* com adição dinâmica de fragmento

E dessa forma obtemos o mesmo resultado, porém com a adição dinâmica do fragmento, você pode experimentar e fazer com que o botão remova um fragmento e coloca outro diferente no lugar.

## 5.9 Abas (*Tabs*)

Existem diversas maneiras de criar uma interface com abas no Android, uma delas é usando as interfaces TabHost e TabWidget, outra é imitando o comportamento usando apenas Fragments.

### 5.9.1 Usando TabHost e TabWidget

Abas usando essas interfaces são suportados por todas as versões do android Vamos criar uma interface com abas seguindo esse esquema:

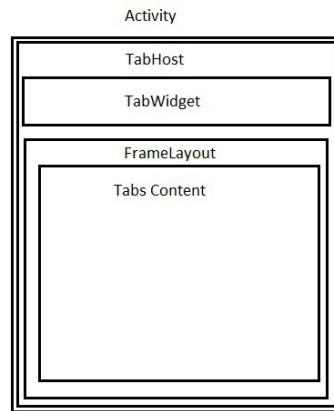


Figura 5.15: Esquema da interface com abas

Primeiro precisamos criar uma *activity* que servirá como recipiente para as abas e seu conteúdo. O *TabWidget*<sup>15</sup> é o controle de seleção das abas. Todo conteúdo das abas ficará contido dentro do *FrameLayout*, é nele que as respectivas *activities* serão mostradas. O *TabHost*<sup>16</sup> por sua vez serve como um recipiente para o *TabWidget* e o *FrameLayout*.

Crie uma nova *activity*, a chamaremos de *TabLayoutActivity*. No XML que define o *layout* da *activity*, insira o *TabHost*, o *TabWidget* e *FrameLayout* como mostrado no algoritmo abaixo.

---

```

1 <TabHost
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:id="@+id/tabhost"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent">
6
7   <LinearLayout
8     android:orientation="vertical"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent" >
11
12     <TabWidget
13       android:id="@+id/tabs"
14       android:layout_width="fill_parent"
15       android:layout_height="wrap_content" />
16
17     <FrameLayout
18       android:id="@+id/tabcontent"
19       android:layout_width="fill_parent"
20       android:layout_height="fill_parent" />
21
22   </LinearLayout>
23 </TabHost>
  
```

---

Algoritmo 5.24: Layout da activity TabHostLayout

<sup>15</sup><http://developer.android.com/reference/android/widget/TabWidget.html>

<sup>16</sup><http://developer.android.com/reference/android/widget/TabHost.html>

Agora precisamos definir o *layout* dos fragmentos, isto é, o *layout* de cada aba. Para simplificar o exemplo, as abas só terão um fundo colorido, de cores diferentes. Para isso usa-se o atributo `background`.

---

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent"
5   android:background="#FF0000" />

```

---

Algoritmo 5.25: *Layout* do fragmento da aba.

Precisamos definir as classes de cada fragmento de aba. Cada classe deverá extender a classe `Fragment` e inflar seu *layout* correspondente. Depois cada fragmento será instanciado pela nossa *activity* principal, `TabLayoutActivity` usando o *fragment manager*. No algoritmo 5.26 está definido a classe `Tab1Fragment` (as classes `Tab2Fragment` e `Tab3Fragment`) são exatamente iguais, exceto que elas inflam seus respectivos *layouts*.

**Dica:** Você deve importar a classe `android.support.v4.app.Fragment` para suportar versões mais antigas do Android!

---

```

1 public class Tab1Fragment extends Fragment {
2     public View onCreateView(LayoutInflater inflater,
3         ViewGroup container, Bundle savedInstanceState) {
4
5         if(container == null) {
6             return null;
7         }
8
9         return (LinearLayout) inflater.
10            inflate(R.layout.tab_fragment1, container, false);
11     }
12 }

```

---

Algoritmo 5.26: Classe `Tab1Fragment`

Na classe `TabLayoutActivity`, note que estamos extendendo a classe `FragmentActivity` para poder usufruir das funcionalidades dos fragmentos. É necessário configurar o método `onCreate()`, esse é o ponto de início da nossa *activity*. O primeiro passo é inflar o *layout* com abas definido no algoritmo 5.24. O segundo passo é inicializar as abas, para isso invocamos o método `TabHost.setup()`, adicionar as abas e suas informações em um mapa e determinar a primeira aba como ativa.

Primeiro criaremos uma classe que servirá de suporte para guardar as informações relevantes sobre as nossas abas.

```

1 public class TabInfo {
2     private String tag;
3     private Class klass;
4     private Bundle args;
5     private Fragment fragment;
6
7     TabInfo(String tag, Class klass, Bundle args) {
8         this.tag = tag;
9         this.klass = klass;
10        this.args = args;
11    }
12 }
```

---

Algoritmo 5.27: Classe TabInfo

Em seguida, começaremos a escrever nossa classe TabLayoutActivity.

```

1 public class TabLayoutActivity extends Activity
2             implements TabHost.OnTabChangeListener {
3     private TabHost mTabHost;
4     private HashMap<String, TabInfo> mapTabInfo =
5             new HashMap<String, TabInfo>();
6     private TabInfo mLastTab = null;
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         //Estabelece o layout da activity
12         setContentView(R.layout.activity_tab_layout);
13
14         //Metodo para inicializar as abas
15         initialiseTabHost(savedInstanceState);
16         if(savedInstanceState != null){
17             //Determina a aba que esta selecionada
18             mTabHost.setCurrentTabByTag
19                 (savedInstanceState.getString("tab"));
20         }
21 }
```

---

Algoritmo 5.28: Primeira parte da classe TabLayoutActivity

Antes de criar o método initialiseTabHost () precisamos criar outra classe suporte, essa classe é necessária para criar o conteúdo de uma aba sob demanda. Crie uma classe que chamaremos de TabFactory e ela deve implementar a interface TabContentFactory<sup>17</sup>.

<sup>17</sup><http://developer.android.com/reference/android/widget/TabHost.TabContentFactory.html>

---

```

1 public class TabFactory implements TabContentFactory{
2     private final Context mContext;
3
4     public TabFactory(Context context) {
5         mContext = context;
6     }
7
8     @Override
9     public View createTabContent(String tag) {
10        View v = new View(mContext);
11        v.setMinimumHeight(0);
12        v.setMinimumWidth(0);
13        return v;
14    }
15 }
```

---

Algoritmo 5.29: Classe TabFactory

O parâmetro *tag* do método `createTabContent()` é que define qual aba foi selecionada. O método retorna uma *view* para mostrar os elementos da aba selecionada.

Na classe `TabLayoutActivity` precisamos criar o método `initialiseTabHost()`. Siga o algoritmo 5.30 abaixo. Note o uso do método `onTabChanged()`. Precisamos implementar esse método em seguida através da interface `TabHost.OnTabChangeListener`.

---

```

1 private void initialiseTabHost(Bundle args) {
2     mTabHost = (TabHost)findViewById(android.R.id.tabhost);
3     mTabHost.setup();
4     TabInfo tabInfo = null;
5     String tag;
6
7     //Cria Tab1
8     tabSpec = mTabHost.newTabSpec("Tab1");
9     tabSpec.setIndicator("Tab 1");
10    tabSpec.setContent(new TabFactory(this));
11    tag = tabSpec.getTag();
12    tabInfo = new TabInfo("Tab1", Tab1Fragment.class, args);
13    tabInfo.fragment = getSupportFragmentManager().
14        findFragmentByTag(tag);
15    mTabHost.addTab(tabSpec);
16    mapTabInfo.put(tabInfo.tag, tabInfo);
17    /* Repete para Tab2 e Tab3 */
18
19    //Ajusta primeira aba como default
20    onTabChanged("Tab1");
21    mTabHost.setOnTabChangedListener(this);
22 }
```

---

Algoritmo 5.30: Método `initialiseTabHost()`

Primeiro obtemos a *view* TabHost usando o método `findViewById()`. Observe que estamos pegando um recurso já existente do sistema Android, já que estamos chamando a classe R do sistema, e não do nosso aplicativo. Em seguida chamamos o método `setup()`, a documentação diz que é necessário invocar esse método antes de adicionar abas se carregamos o TabHost usando `findViewById()`.

Criamos um TabInfo e um TabSpec para nos auxiliar na adição das abas. Para adicionar as abas, primeiro chamamos o método `TabHost.newTabSpec()` para obtermos um novo TabSpec associado a esse TabHost, colocamos a tag "Tab1" nele, como pode ser observado na linha 9. Em seguida determinados um indicador (que será mostrado ao usuário) a essa aba usando `TabSpec.setIndicator()`, na linha 10. Criamos um novo TabInfo e passamos a tag criada, a classe com o conteúdo da aba e uma série de argumentos que podem ser passados entre *activities*. Na linha 12 usamos o método `addTab()` criado anteriormente. Na linha 13 adicionamos a tag e um ponteiro para o recém-criado TabInfo no HashMap.

Repetimos o mesmo procedimento para as abas 2 e 3, porém mudando os valores da tag, do indicador e da classe. Por último definimos a primeira aba como *default* e determinamos o argumento *this* para o método `setOnTabChangedListener()` pois iremos implementar o método `onTabChanged` em seguida.

---

```

1  @Override
2  public void onTabChanged(String tag) {
3      TabInfo newTab = mapTabInfo.get(tag);
4
5      if (mLastTab != newTab) {
6          FragmentTransaction ft =
7              getSupportFragmentManager().beginTransaction();
8
9          if (mLastTab != null) {
10             if (mLastTab.fragment != null) {
11                 ft.detach(mLastTab.fragment);
12             }
13         }
14
15         if (newTab != null) {
16             if (newTab.fragment == null) {
17                 newTab.fragment = Fragment.instantiate(this,
18                     newTab.klass.getName(), newTab.args);
19                 ft.add(android.R.id.tabcontent, newTab.fragment, newTab.tag);
20             } else {
21                 ft.attach(newTab.fragment);
22             }
23         }
24         mLastTab = newTab;
25         ft.commit();
26         getSupportFragmentManager().
27             executePendingTransactions();
28     }
29 }
```

---

Algoritmo 5.31: Método `onTabChanged()`

Primeiro obtemos as informações da aba que queremos do mapa com `mapTabInfo.get(tag)`, usamos a `tag` para obter o objeto que queremos. Em seguida testamos para saber se a aba selecionada é a mesma que a anterior, pois não faria sentido recarregar a mesma aba. Na linha 9 testado para saber se a última aba não é nula, isso deve ser feito para evitar uma falha do aplicativo, testamos também se o fragmento é nulo para então usar `detach()` para retirar esse fragmento do `layout`.

Fazemos o mesmo com a nova aba, caso o fragmento seja nulo isso quer dizer que ele não foi instanciado ainda, isto é, é a primeira vez que o usuário seleciona essa aba nesse ciclo de vida do aplicativo. Caso isso ocorra, então usamos `instantiate()` para instânciar esse novo fragmento e `add` para adicioná-lo ao `layout`. Caso ele já tenha sido instânciado, então apenas usamos `attach()` para coloca-lo de volta no `layout`.

Por final é preciso salvar a aba que estavamos caso o aplicativo fique em segundo plano. O método `onSaveInstanceState` fica encarregado disso.

---

```
1 protected void onSaveInstanceState(Bundle outState) {  
2     outState.putString("tab", mTabHost.getCurrentTabTag());  
3     super.onSaveInstanceState(outState);  
4 }
```

---

Algoritmo 5.32: Método `onSaveInstanceState()`

A figura abaixo mostra o resultado.

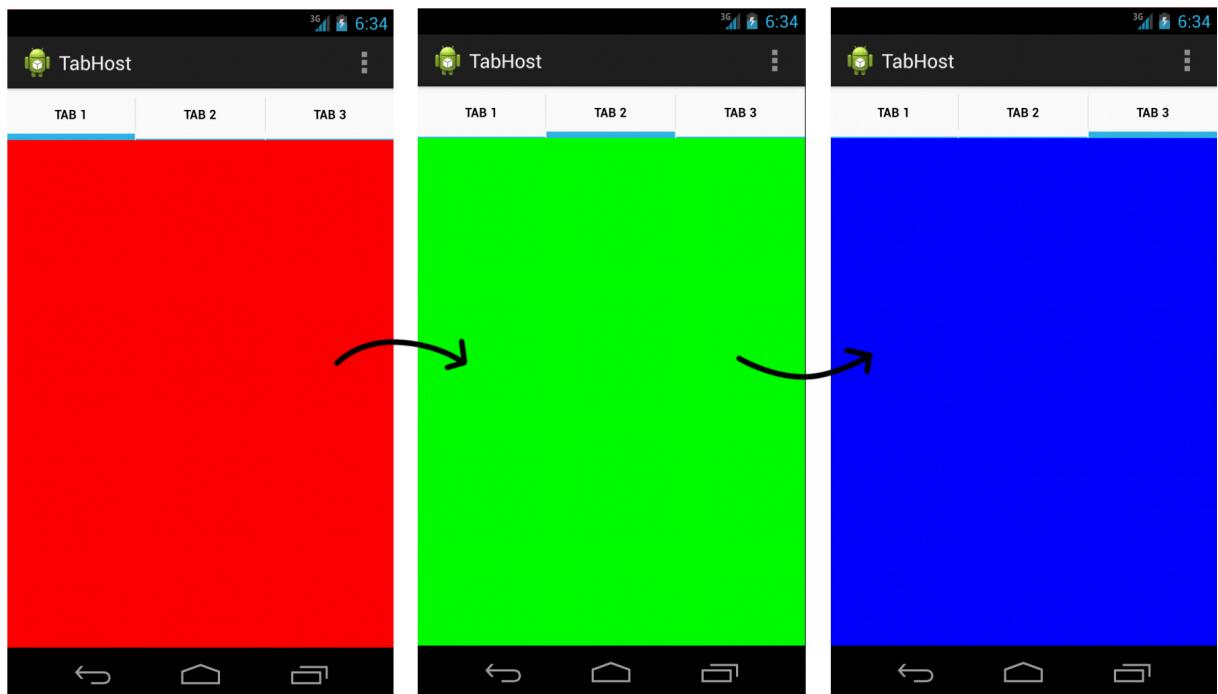


Figura 5.16: Figura mostrando as 3 abas criadas no exemplo

## 5.10 Trocar de página com gesto de arrastar usando ViewPager

É possível trocar entre fragmentos usando o gesto de arrastar, isto é, arrastando a tela de um lado para o outro acionará a troca entre os fragmentos.

Primeiro iremos definir o *layout* do ViewPager<sup>18</sup>. Depois iremos definir o PagerAdapter<sup>19</sup>. Por último precisamos definir a *activity* que irá conter o visualizador de páginas.

---

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent"
5   android:orientation="vertical" >
6
7   <android.support.v4.view.ViewPager
8     android:id="@+id/viewpager"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent" />
11
12 </LinearLayout>

```

---

Algoritmo 5.33: *layout* do ViewPager

Agora defina uma nova classe chamada PagerAdapter que irá extender a classe FragmentPagerAdapter. Primeiro crie uma lista que irá conter os fragmentos, isto é, as páginas que serão exibidas. Ao extender essa classe precisamos implementar dois métodos: getItem() e getCount(). O método getItem(), na linha 10 do algoritmo 5.34, deve retornar o item que será selecionado pelo parâmetro position. O método getCount(), na linha 15, deve retornar a quantidade de páginas. Depois crie o construtor como mostrado nas linhas 4-7.

---

<sup>18</sup><http://developer.android.com/reference/android/support/v4/view/ViewPager.html>

<sup>19</sup><http://developer.android.com/reference/android/support/v4/view/PagerAdapter.html>

---

```
1 public class PagerAdapter extends FragmentPagerAdapter{
2     private List<Fragment> fragments;
3
4     public PagerAdapter(FragmentManager fm, List<Fragment> fragments) {
5         super(fm);
6         this.fragments = fragments;
7     }
8
9     @Override
10    public Fragment getItem(int position) {
11        return this.fragments.get(position);
12    }
13
14    @Override
15    public int getCount() {
16        return this.fragments.size();
17    }
18 }
```

---

Algoritmo 5.34: Classe PagerAdapter

Por último devemos construir a *activity* que irá conter o `PagerAdapter` e será responsável por mostrar as páginas. Neste exemplo iremos reutilizar os fragmentos que fizemos na seção anterior quando trabalhamos com abas.

**Dica:** Você pode importar as classes e os arquivos de *layout*. No *Package Explorer* na IDE Eclipse, clique com o botão direito sobre a pasta que quer importar os arquivos, depois clique em *Import* e selecione *General -> File System*. Agora selecione o caminho da pasta que contém os arquivos no campo *From directory*. Selecione os arquivos que deseja importar e clique em *Finish*.

Note que ao importar classes Java será necessário trocar o pacote a que a classe pertence.

Outra opção é usar o `import` do java para importar as classes sem ter elas no pacote.

Essa *activity*, como mostrada no algoritmo 5.35 abaixo, apenas precisa instanciar os fragmentos (linhas 10,11 e 12), criar e determinar o adaptador, linhas 14, 16 e 17.

```

1 public class ViewPagerLayout extends FragmentActivity {
2     private PagerAdapter mPagerAdapter;
3
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_viewpager_layout);
8
9         List<Fragment> fragments = new ArrayList<Fragment>();
10        fragments.add(Fragment.instantiate
11                      (this, Tab1Fragment.class.getName()));
12        fragments.add(Fragment.instantiate
13                      (this, Tab2Fragment.class.getName()));
14        fragments.add(Fragment.instantiate
15                      (this, Tab3Fragment.class.getName()));
16
17        mPagerAdapter = new PagerAdapter(getSupportFragmentManager(), fragments);
18
19        ViewPager pager = (ViewPager) findViewById(R.id.viewpager);
20        pager.setAdapter(mPagerAdapter);
21    }
22    ...
23 }
```

---

Algoritmo 5.35: *Activity com PagerAdapter*

## 5.11 Abas com gesto de arrastar

Ao juntar os dois conceitos, o de *layout* com abas e o gesto de arrastar, podemos fazer o controle das abas arrastando a tela. Esse tipo de *design* é comum em muitos aplicativos pela facilidade e rapidez com o que o usuário pode visualizar vários conteúdos. Nesse exemplo iremos re-utilizar o código dos exemplos anteriores com algumas modificações. Serão reutilizadas classes: TabInfo, TabFactory, PagerAdapter, Tab1Fragment, Tab2Fragment, Tab3Fragment.

Primeiro modificaremos o *layout* das abas adicionando o ViewPager após FrameLayout. Note que esse é o mesmo *layout* do algoritmo 5.24, por isso o algoritmo 5.36 não está completo.

---

```

1 ...
2 <FrameLayout
3   android:id="@+id/tabcontent"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent" >
6
7   <android.support.v4.view.ViewPager
8     android:id="@+id/viewpager"
9     android:layout_width="fill_parent"
10    android:layout_height="0dp"
11    android:layout_weight="1" />
12
13 </FrameLayout>
14 ...

```

---

Algoritmo 5.36: Layout das abas com adição do ViewPager

Agora iremos usar a classe TabLayoutActivity do exemplo anterior e fazer algumas alterações. Nesse exemplo irei mudar o nome da classe para SwipeTabActivity. A primeira alteração é a adição de algumas variáveis para serem usadas na classe, adicione uma variável PageAdapter e uma ViewPager. Depois devemos alterar o método `onCreate()`, adicione uma chamada ao método `initialiseViewPager()`. A implementação é a mesma do método `onCreate()` do exemplo anterior com a adição da chamada ao método `setOnPageChangeListener()`. Devemos também implementar a interface `OnPageChangeListener`. No método `initialiseTabHost()` você precisa remover a linha `onTabChanged("Tab1");`.

---

```

1 private void initialiseViewPager() {
2   List<Fragment> fragments = new ArrayList<Fragment>();
3   fragments.add(Fragment.instantiate
4     (this, Tab1Fragment.class.getName()));
5   fragments.add(Fragment.instantiate
6     (this, Tab2Fragment.class.getName()));
7   fragments.add(Fragment.instantiate
8     (this, Tab3Fragment.class.getName()));
9
10  mPageAdapter = new PagerAdapter(getSupportFragmentManager(), fragments);
11
12  mViewPager = (ViewPager) findViewById(R.id.viewpager);
13  mViewPager.setAdapter(mPageAdapter);
14  mViewPager.setOnPageChangeListener(this);
15 }

```

---

Algoritmo 5.37: Método initialiseViewPager()

Em seguida precisamos alterar o método `onTabChanged()`, não necessitamos mais fazer verificações já que o próprio *design* irá limitar as falhas que poderiam ocorrer. Precisamos apenas determinar para o ViewPager o item atual.

---

```

1 @Override
2 public void onTabChanged(String tag) {
3     int pos = mTabHost.getCurrentTab();
4     mViewPager.setCurrentItem(pos);
5 }
```

---

Algoritmo 5.38: Método `onTabChanged()` alterado

Por fim, devemos implementar os métodos da interface `ViewPager.OnPageChangeListener`, somente o método `onPageSelected()` será usado, para selecionar a aba correspondente à pagina atual.

---

```

1 @Override
2 public void onPageScrollStateChanged(int arg0) {
3     //Nada
4 }
5
6 @Override
7 public void onPageScrolled(int arg0, float arg1, int arg2) {
8     //Nada
9 }
10
11 @Override
12 public void onPageSelected(int position) {
13     mTabHost.setCurrentTab(position);
14 }
```

---

Algoritmo 5.39: Métodos da interface `ViewPager.OnPageChangeListener`

**Dica:** Pela dificuldade em acompanhar passo a passo a construção desse *design*, é recomendável obter o código do projeto no repositório. Está sob o nome *SwipeableTabs*.

## 5.12 Menu lateral (*Navigation Drawer*)

O *Sliding Menu* não é padrão no SDK do Android, mas é um tipo de *design* que está sendo usado por muitos aplicativos famosos hoje em dia, tais como: Facebook, Linkedin, Foursquare, e outros. Consiste de um menu que fica escondido na lateral esquerda da tela e que pode ser acessado utilizando um gesto ”puxando” o menu ou com um botão instalado na *ActionBar*.

Existe um projeto *Open Source* que é bem conhecido, acessado por esse link no [GitHub](#)<sup>20</sup>.

A imagem abaixo mostra o menu sendo usado no aplicativo do Facebook.

---

<sup>20</sup><https://github.com/jfeinstein10/SlidingMenu>

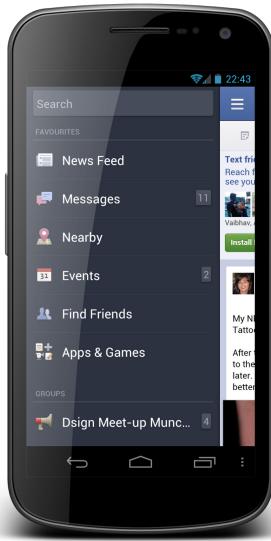


Figura 5.17: Exemplo de *Sliding Menu*

## 5.13 ActionBar

A *ActionBar* é aquela barra presente em todos os aplicativos que fizemos de exemplo até agora. Ela pode mostrar o nome da *activity*, ícones, ações que podem ser acionadas, outras *views* ou botões interativos. Também pode ser usada para navegar entre as *activities* do seu aplicativo.

Dispositivos Android mais antigos possuem um botão físico chamado *Option* que abre um menu na parte inferior do aplicativo. A *ActionBar* é melhor que esse menu pois está claramente visível para o usuário, enquanto que o menu antigo era escondido e o usuário pode não reconhecer que as opções estão disponíveis.

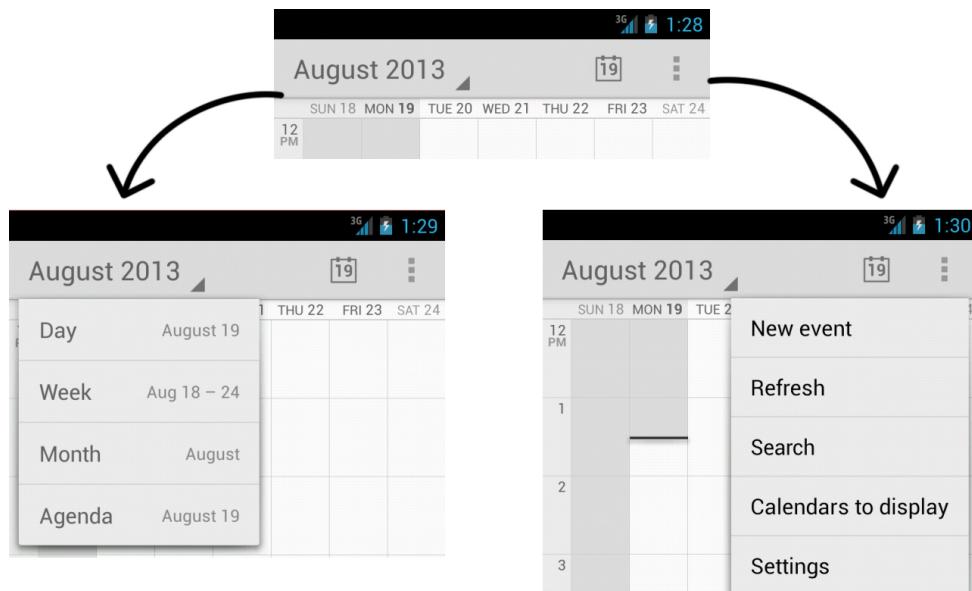


Figura 5.18: Exemplo de *ActionBar* no aplicativo Calendário

A figura 5.18 mostra o uso da *ActionBar* no aplicativo Calendário, padrão dos aparelhos Android mais atuais. É possível observar três principais componentes. O primeiro é um menu *drop-down* que permite ao usuário mudar o modo de visualização do calendário. O segundo é um botão com o dia atual, 19, que ao ser pressionado faz com que o calendário posicione um cursor no dia e hora atuais. O terceiro é um outro menu *drop-down* com algumas opções que podem ser interessantes ao usuário.

### 5.13.1 Implementando a *ActionBar*

A *activity* popula a *ActionBar* em seu método `onCreateOptionsMenu()`. Entradas na *ActionBar* são chamadas de ações (*actions*).

As ações para a *ActionBar* são definidas em arquivos XML posicionados na pasta `menu/`. O algoritmo abaixo mostra o menu padrão dos exemplos que construímos até agora. Ele só contém um item *"Settings"* que está no *dropdown* menu que pode ser acessado através da *ActionBar*, à direita. O fato dele estar escondido se deve ao atributo `showAsAction` estar com valor *never*, ao ser mudado para *always* o acesso ao *Settings* será diretamente através da *ActionBar*. Existe também o valor *ifRoom* que irá mostrar apenas se houver espaço disponível.

---

```

1 <menu
2   xmlns:android="http://schemas.android.com/apk/res/android" >
3
4   <item
5     android:id="@+id/action_settings"
6     android:orderInCategory="100"
7     android:showAsAction="always"
8     android:title="@string/action_settings"/>
9
10 </menu>
```

---

Algoritmo 5.40: Menu padrão dos exemplos

---

```

1 @Override
2 public boolean onCreateOptionsMenu(Menu menu) {
3     getMenuInflater().inflate(R.menu.main, menu);
4     return true;
5 }
```

---

Algoritmo 5.41: Método padrão `onCreateOptionsMenu()`

Se uma ação é selecionada, o método `onOptionsItemSelected()` é chamado. Ele recebe a ação selecionada como parâmetro `MenuItem`. Baseando-se nessa informação você pode decidir o que fazer. Nesse exemplo iremos abrir uma nova *activity* que seria a tela de configuração do aplicativo.

No seu projeto, crie uma nova *Activity* do tipo *Settings Activity*.

**Dica:** Para criar uma nova *activity*, clique com o botão direito sob o projeto selecione *New -> Other* (ou pressione *Ctrl+N*). Selecione *Android Activity* e selecione o tipo desejado.

Agora você deve fazer com que o método `OnCreateOptionsMenu()` abra essa nova *activity*.

```
1 public boolean onOptionsItemSelected(MenuItem item) {  
2     if(item.getItemId() == R.id.action_settings) {  
3         startActivity(new Intent(this, SettingsActivity.class));  
4     }  
5     return true;  
6 }
```

Algoritmo 5.42: Método `OnOptionsItemSelected()`

Para melhorar nossa *ActionBar* vamos adicionar um campo para pesquisa. Você pode adicionar *views* em sua *ActionBar*. Para isso você deve usar o método `setCustomView()` da classe *ActionBar* e passar uma *view* como parâmetro. Você também precisa ativar a exibição de *views* com o método `setDisplayOptions()` e passar a flag `ActionBar.DISPLAY_SHOW_CUSTOM`.

Primeiro vamos adicionar um ícone de busca na nossa *ActionBar*, voltando ao arquivo do *layout* da *ActionBar*, adicione um novo item acima do primeiro como mostrado no algoritmo abaixo.

```
1 <menu xmlns:android="http://schemas.android.com/apk/res/android" >  
2  
3     <item  
4         android:id="@+id/action_search"  
5         android:orderInCategory="100"  
6         android:showAsAction="always"  
7         android:title="Search"  
8         android:icon="@android:drawable/ic_menu_search" />  
9  
10    <item  
11        android:id="@+id/action_settings"  
12        android:showAsAction="never"  
13        android:title="@string/action_settings"/>  
14  
15 </menu>
```

Algoritmo 5.43: Adicionando novo item na *ActionBar*

Observe o atributo `android:icon`, estamos obtendo um *drawable* que já existe no sistema Android, e se chama *ic\_menu\_search*. Esse é o ícone da busca, a lupa. Depois vamos adicionar uma ação a ser executada quando esse ícone for clicado. Iremos criar nesse exemplo,

uma caixa de texto de forma programática, isto é, em vez de defini-la no XML iremos cria-la com código na *Activity*.

No método `onCreate()` você precisa configurar a *ActionBar* para mostrar *views*. Use o método `getActionBar()` para obter uma referência da *ActionBar* e `setDisplayOptions()` para configura-la. Depois, no método `onOptionsItemSelected()` você vai programar a ação do novo botão. Inicialmente cria-se uma nova *view* do tipo `EditText`, colocamos algumas configurações e a adicionamos na *ActionBar*. Por não termos uma busca devidamente implementada, vamos mostrar um *Toast* com o conteúdo da busca, a fim de demonstração.

---

```

1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3   super.onCreate(savedInstanceState);
4   setContentView(R.layout.activity_main);
5
6   getActionBar().setDisplayOptions
7     (ActionBar.DISPLAY_SHOW_CUSTOM |
8     ActionBar.DISPLAY_SHOW_HOME);
9 }
```

---

Algoritmo 5.44: Configurando *ActionBar* no método `onCreate()`

No algoritmo 5.45, abaixo, criamos um `EditText` e o configuramos caso o botão com o ícone de busca seja pressionado. Na linha 5-6 criamos um `LayoutParams` que configura o tamanho da *view*. Na linha 8 o método `EditText.setImeOptions()` é responsável por configurar o teclado para uma busca, junto com a linha 9 que diz para o `EditText` que a entrada será texto, essa combinação faz com que o teclado mostre o ícone da lupa no lugar da tecla *Enter*. A linha 10 configura a cor do texto para branca. Adicionamos a *view* na *ActionBar* com o método `ActionBar.setCustomView()` e passamos como parâmetro a *view* criada e os parâmetros de *layout* criados. O método `EditText.requestFocus()` faz com que o foco seja dado à nova caixa de texto, para que possamos editá-la. Precisamos ainda fazer com que o teclado abra para que possamos editar a caixa de texto, é isso que as linhas 13-14 e 15 estão fazendo. O método `getSystemService()` obtém a referência de um serviço do Android, e nesse caso estamos pedindo pelo serviço de método de entrada, o teclado. O método `InputMethodManager.showSoftInput()` abre o *Soft Input*, ou seja, o teclado virtual para edição da *view*.

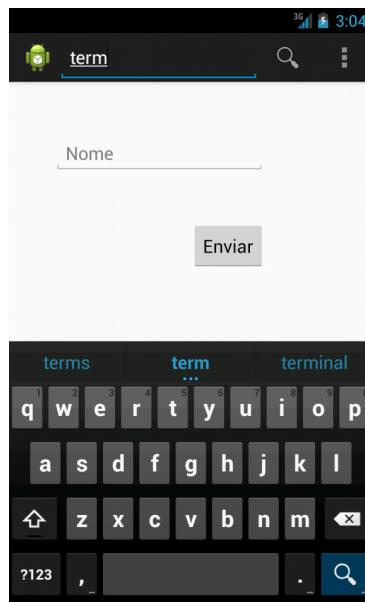
Finalmente fazemos com que ao botão de busca no teclado ser clicado, um *Toast* mostre para o usuário o conteúdo da caixa de texto. É isso que a interface `onEditorActionListener` faz com o método `onEditorAction()`.

A intenção desse exemplo é mostrar como você pode adicionar novas *views* na *ActionBar* de forma dinâmica.

```

1 public boolean onOptionsItemSelected(MenuItem item) {
2     if(item.getItemId() == R.id.action_settings) {
3         startActivity(new Intent(this, SettingsActivity.class));
4     } else if(item.getItemId() == R.id.action_search) {
5         LayoutParams lp = new LayoutParams
6             (LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
7         EditText search = new EditText(this);
8         search.setImeOptions(EditorInfo.IME_ACTION_SEARCH);
9         search.setTextColor(Color.WHITE);
10        search.setInputType(InputType.TYPE_CLASS_TEXT);
11        getActionBar().setCustomView(search, lp);
12        search.requestFocus();
13        InputMethodManager imm = (InputMethodManager)
14            getSystemService(Context.INPUT_METHOD_SERVICE);
15        imm.showSoftInput(search, InputMethodManager.SHOW_IMPLICIT);
16        search.setOnEditorActionListener(new OnEditorActionListener() {
17
18            @Override
19            public boolean onEditorAction
20                (TextView v, int actionId, KeyEvent event) {
21                Toast.makeText
22                    (MainActivity.this, v.getText(), Toast.LENGTH_SHORT).show();
23                return true;
24            }
25        });
26    }
27    return true;
28 }

```

Algoritmo 5.45: Criando a caixa de busca na *ActionBar*Figura 5.19: Exemplo de busca na *ActionBar*



# Comunicação

Agora iremos explorar as funcionalidades de comunicação dos aparelhos Android como acesso a *Internet* no caso de *Tables* e *Smartphones* e envio de SMS e chamadas de voz com os os *Smartphones*.

## 6.1 *Internet*

Antes de iniciar sua aplicação que faz acesso à *Internet*, devemos dar permissão ao aplicativo através do arquivo de *Manifest*. Logo antes da tag `uses-sdk` você deve adicionar a tag `uses-permission`, como mostrado abaixo:

---

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

---

Algoritmo 6.1: Atribuindo permissão de acesso à *Internet* no *Manifest*

Nesta seção, veremos como se faz requisições HTTP para obter páginas HTML, saídas de um *script server-side* como PHP ou ASP.NET e *parsing* de respostas JSON ou XML.

### 6.1.1 HTTP GET

Para fazer uma requisição HTTP GET usaremos as classes da biblioteca *Apache*, tais como *HttpClient*, *HttpGet* e *HttpResponse*. Primeiros crie uma classe que iremos chamar de `RequestTask` e ela deve extender a classe `AsyncTask`<sup>1</sup>, ela vai permitir que façamos a requisição na *thread* da interface do usuário sem precisar criar e manipular uma *thread* diferente.

---

<sup>1</sup><http://developer.android.com/reference/android/os/AsyncTask.html>

---

```

1 public class RequestTask extends AsyncTask<URI, Integer, String>{
2
3     @Override
4     protected String doInBackground(URI... uri) {
5         HttpClient httpclient = new DefaultHttpClient();
6         HttpResponse response;
7         String responseString = null;
8
9         try {
10             response = httpclient.execute(new HttpGet(uri[0]));
11             StatusLine statusLine = response.getStatusLine();
12
13             if(statusLine.getStatusCode() == HttpStatus.SC_OK) {
14                 ByteArrayOutputStream out = new ByteArrayOutputStream();
15                 response.getEntity().writeTo(out);
16                 out.close();
17                 responseString = out.toString();
18             } else {
19                 response.getEntity().getContent().close();
20                 throw new IOException(statusLine.getReasonPhrase());
21             }
22         } catch (ClientProtocolException e) {
23             e.printStackTrace();
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27         return responseString;
28     }
29 }
```

---

Algoritmo 6.2: Classe RequestTask

O método `doInBackground()` é o responsável por realizar a operação sem o usuário perceber. Existem outros dois métodos da classe `AsyncTask` que podem ser utilizados: `onProgressUpdate()`, caso queira mostrar o progresso de um *download* para o usuário e `onPostExecute()` para executar alguma ação após o término do *download*.

Observe o algoritmo 6.2, na linha 5-7 criamos um novo `HttpClient`, `HttpResponse` e uma *string* para armazenar a resposta do servidor. Depois, envolto por um bloco `try-catch` temos uma chamada ao método `HttpClient.execute()` e é passado para ele um novo `HttpGet` com o parâmetro `uri[0]`, `uri` é um dos parâmetros do método e contém uma `URI`<sup>2</sup> que identifica o destino da requisição.

Na linha 11 foi criado um novo `StatusLine`<sup>3</sup> para guardar a resposta da requisição HTTP que obtemos com o método `HttpResponse.getStatusLine()`. Depois comparamos o código do *status* com `HttpStatus.SC_OK` (equivalente ao código 200), o *status OK* significa que a requisição e a resposta ocorreram como esperado e temos a resposta correta vindo do servidor.

<sup>2</sup><http://developer.android.com/reference/java/net/URI.html>

<sup>3</sup><http://developer.android.com/reference/org/apache/http/StatusLine.html>

Em seguida criamos um novo `ByteArrayOutputStream` que é responsável por guardar a resposta do servidor na chamada ao método `HttpResponse.getEntity().writeTo()`. Por último convertemos o *byte stream* em uma string usando o método `toString()` e a retornamos.

Agora precisamos de uma *Activity* para mostrar a resposta, nesse exemplo obteremos o código HTML da página do DC UFSCar: <http://www.dc.ufscar.br>. Mostraremos o código HTML como uma página *Web* usando o *WebView*.

---

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_main);
5
6     String uri = "http://www.dc.ufscar.br";
7     AsyncTask<String, String, String> task;
8     String response = null;
9
10    try {
11        task = new RequestTask(this).execute(uri);
12        response = task.get();
13    } catch (Exception e) {
14        e.printStackTrace();
15    }
16
17    if(response != null) {
18        WebView webview = new WebView(this);
19        setContentView(webview);
20
21        webview.loadData(response, "text/html; charset=UTF-8", null);
22    }
23 }
```

---

Algoritmo 6.3: Classe `RequestTask`

Primeiro criamos uma nova variável `task` do tipo `AsyncTask` nos moldes da classe que criamos, dentro de um bloco `try-catch` criamos uma nova URI e atribuímos a ela o endereço da página do DC UFSCar. Em seguida iniciamos a variável `task` criando uma nova `RequestTask` e chamamos o método `execute()`, que ao ser chamado irá executar o método `doInBackground()`.

Por fim, criamos uma nova `WebView`<sup>4</sup> e carregamos o conteúdo da string `response` nela usando o método `WebView.loadData()`. A página é mostrada sem estilo pois não estamos puxando as folhas de estilo junto, somente o código HTML.

**Dica:** Usamos a `WebView` para mostrar o código HTML como página apenas para demonstração. Caso queira carregar uma página corretamente deve fornecer a *URL* da página diretamente para a `WebView`.

<sup>4</sup><http://developer.android.com/reference/android/webkit/WebView.html>

## 6.1.2 HTTP POST

Diferentemente do *GET*, onde os parâmetros para o servidor vão codificados na *URL*, no *POST* os parâmetros vão codificados no final do cabeçalho HTTP. Para enviar uma requisição do tipo *POST*, usaremos a classe `HttpPost`. O algoritmo abaixo é reaproveitado da classe `RequestTask` e do método `doInBackground()`, onde mudaremos apenas algumas linhas. Agora estaremos enviando a requisição para essa página: <http://httpbin.org/post>, ela só aceita requisições do tipo *POST*, se tentar com *GET* irá receber uma mensagem de erro.

---

```

1 try {
2     HttpPost post = new HttpPost(uri[0]);
3     List<NameValuePair> nvp = new ArrayList<NameValuePair>();
4     nvp.add(new BasicNameValuePair("testing", "Post"));
5     nvp.add(new BasicNameValuePair("user", "You"));
6
7     post.setEntity(new UrlEncodedFormEntity(nvp));
8     response = httpclient.execute(post);
9     ...
10    ...
11 }
```

---

Algoritmo 6.4: Modificando o método para requisições *POST*

Criamos um novo `HttpPost`<sup>5</sup> passando a URI como parâmetro. Criamos uma lista de tuplas chave-valor que representa a variável e seu valor no cabeçalho HTTP e adicionamos dois valores. Em seguida chamamos o método `HttpPost.setEntity()` e passamos um objeto do tipo `UrlEncodedFormEntity`<sup>6</sup> que recebe a lista como parâmetro, esse objeto irá codificar a lista em variáveis aceitas pelo padrão de uma requisição *POST*.

Por final, executamos a requisição como feito anteriormente. O resto do código é igual.

Note que se colocar a resposta dessa requisição em uma `WebView` ou `TextView` irá observar que está codificado no formato `JSON`<sup>7</sup>.

## 6.1.3 Decodificando JSON

Vamos obter dados do objeto `JSON` retornado pelo exemplo anterior. Se você observar a `string` na tela, irá perceber os dados que foram enviado via *POST* dentro de um objeto `JSON` chamado *form*. Queremos obter esses dados, para isso precisamos criar uma classe que será responsável por obter especificamente esses dados do *form*.

**Dica:** Em `JSON`, { representa um objeto `JSON` e [ representa um *array* dentro de um objeto `JSON`

Crie uma classe `JSONParser`, como mostrado abaixo.

<sup>5</sup><http://developer.android.com/reference/org/apache/http/client/methods/HttpPost.html>

<sup>6</sup><http://developer.android.com/reference/org/apache/http/client/entity/UrlEncodedFormEntity.html>

<sup>7</sup>O que é `JSON`: <http://www.json.org/>

---

```

1 public class JSONParser {
2
3     public String getFormData(String jsonstr) {
4         String formData = null;
5
6         try {
7             JSONObject jObj = new JSONObject(jsonstr);
8             JSONObject form = jObj.getJSONObject("form");
9             formData = form.getString("testing");
10            formData += "\n" + form.getString("user");
11
12        } catch (JSONException e) {
13            e.printStackTrace();
14        }
15
16        return formData;
17    }
18 }
```

---

Algoritmo 6.5: Classe JSONParser

Para obter dados do *form*, criamos um método `getFormData()`. Primeiro obtemos o objeto JSON dado pela *string* retornada pela requisição *POST* do exemplo anterior ao criar um novo `JSONObject` e passando a *string* como parâmetro. Em seguida queremos obter outro objeto JSON, aquele cujo nome é *form*, para isso chamamos o método `JSONObject.getJSONObject()` passando o nome do objeto junto. Após obter o objeto desejado, usamos o método `JSONObject.getString()` e passamos o nome do valor para obter o valor. Então ao passar "*testing*" e "*user*" esperamos como retorno *POST* e *You*, respectivamente.

### 6.1.4 Codificando JSON

Podemos também codificar objetos JSON, e é simples. Basta criar um `JSONObject` ou `JSONArray` e usar o método `toString()`. Por exemplo:

---

```

1 public void writeJSON() {
2     JSONObject object = new JSONObject();
3     try {
4         object.put("name", "Matheus");
5         object.put("age", new Integer(22));
6         object.put("university", "UFSCar");
7     } catch (JSONException e) {
8         e.printStackTrace();
9     }
10 }
```

---

Algoritmo 6.6: Criando JSON

## 6.2 Telefone

É possível fazer uma chamada telefônica, porém você terá que usar o discador padrão do Android uma vez que ele não provê uma API pública para fazer chamadas diretamente pelo seu aplicativo. A única forma é através do intermediador ACTION\_CALL. O Exemplo abaixo mostra como fazer uma chamada usando esse intermediador.

---

```

1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5
6      Button call = (Button) findViewById(R.id.button1);
7      call.setOnClickListener(new OnClickListener() {
8
9          @Override
10         public void onClick(View v) {
11             EditText tv = (EditText) findViewById(R.id.editNumber);
12             String phone = tv.getText().toString();
13
14             Intent callIntent = new Intent(Intent.ACTION_CALL);
15             callIntent.setData(Uri.parse("tel:" + phone));
16             startActivity(callIntent);
17         }
18     });
19 }
```

---

Algoritmo 6.7: Fazendo uma chamada telefônica

Para esse exemplo, foi criado uma caixa de texto e um botão, o número de telefone é colocado na caixa de texto e o botão *Call* inicia a *activity* responsável por realizar a chamada. Observe que é necessário fazer um *parse* do número de telefone para uma *URI* com formato tel:0123456789. Cria-se uma *Intent* para o ACTION\_CALL, coloca-se os dados na *Intent* e chama o método *startActivity()* para fazer a chamada telefônica. É preciso dar permissão ao aplicativo para fazer chamadas adicionando essa linha ao *Manifest*.

---

```

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

---

Algoritmo 6.8: Permissão para fazer chamadas telefônicas

Usando a classe *TelephonyManager*<sup>8</sup>, no entanto, o Android nos dá a possibilidade obter dados do *Sim Card* e também um *listener* para saber o status atual do telefone, se ele está fazendo uma ligação ou não.

Você pode usar os métodos *getDeviceId()* para obter o número IMEI do aparelho, *getSimSerialNumber()* para obter o número de série do *Sim Card*, *isNetworkRoaming()* para saber se está em modo *roaming*, etc.

---

<sup>8</sup><http://developer.android.com/reference/android/telephony/TelephonyManager.html>

## 6.3 Short Message Service (SMS)

O Android provê uma API pública para mandar mensagens SMS, através da classe `SMSManager`<sup>9</sup>. Assim como fazer ligações, enviar SMS também precisa configurar a permissão no *Manifest*.

---

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

---

Algoritmo 6.9: Permissão para enviar mensagens SMS

Agora vamos criar um aplicativo simples que chama a classe `SMSManager` para enviar uma mensagem para um número de telefone. Teremos duas `EditText` views para abrigar a mensagem e o número, e um botão para enviar. Inicialmente crie um método `sendSMS()`.

---

```
1 private void sendSMS(String message, String phone) {
2     try{
3         SmsManager smsMan = SmsManager.getDefault();
4         smsMan.sendTextMessage(phone, null, message, null, null);
5     } catch(InvalidArgumentException e) {
6         e.printStackTrace();
7         Toast.makeText(this, "Error sending message", Toast.LENGTH_SHORT)
8             .show();
9     }
10 }
```

---

Algoritmo 6.10: Método `sendSMS()`

Dentro de um bloco `try-catch` crie um novo `SMSManager` como mostrado na linha 3 do algoritmo 6.11, use o método `SmsManager.sendTextMessage()` para enviar uma mensagem. O primeiro parâmetro é o número de telefone para qual a mensagem será enviada, o segundo é o telefone de origem, colocando `null` o valor será o número do próprio aparelho ou serviço, o terceiro parâmetro é o texto da mensagem.

Completanto, com um algoritmo simples para chamar o método quando o botão for pressionado.

---

<sup>9</sup><http://developer.android.com/reference/android/telephony/SmsManager.html>

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_main);
5
6     Button btnSend = (Button) findViewById(R.id.buttonSend);
7     btnSend.setOnClickListener(new OnClickListener() {
8
9         @Override
10        public void onClick(View v) {
11            EditText editMsg = (EditText) findViewById(R.id.editMessage);
12            String message = editMsg.getText().toString();
13            EditText editPhone = (EditText) findViewById(R.id.editPhone);
14            String phone = editPhone.getText().toString();
15
16            sendSMS(message, phone);
17
18            //Clean text fields and show toast
19            editMsg.setText("");
20            editPhone.setText("");
21            Toast.makeText
22                (getApplicationContext(), "Message sent", Toast.LENGTH_SHORT)
23                .show();
24        }
25    });
26 }
```

---

Algoritmo 6.11: Chamando método sendSMS ()



# Armazenamento

O Android provê algumas opções para que você possa salvar dados persistentes de sua aplicação. A escolha da opção de armazenamento vai depender das necessidades da sua aplicação, isto é, se os dados vão ser visíveis somente pela aplicação ou se o usuário terá acesso a eles, quanto de espaço será necessário, que tipo de dados você pretende salvar. As opções são:

- *Shared Preferences*: Usada principalmente para salvar preferências do usuário, esse método guarda primitivas em duplas chave-valor;
- Armazenamento Interno: Salva dados privados na memória do dispositivo;
- Armazenamento Externo: Salva dados públicos na memória externa compartilhada;
- Banco de Dados *SQLite*: Salva dados estruturados em um banco de dados privado.
- Conexão com a rede: Salva dados na *web* em seu próprio servidor na rede.

## 7.1 *Shared Preferences*:

A classe `SharedPreferences`<sup>1</sup> fornece um framework que te permite salvar e recuperar dados primitivos persistentes no formato chave-valor. Você pode salvar qualquer tipo de dado primitivo: *booleans*, *floats*, *inteiros*, *longs* e *strings*.

Você pode usar essa classe para armazenar dados durante o ciclo de vida de uma *activity*. Isto é, antes da *activity* ser destruída na função `onDestroy()`, os dados podem ser salvos durante a execução de `onStop()` e recuperados na execução de `onCreate()`.

Para usar a `SharedPreferences` na sua aplicação você deve chamar `getSharedPreferences()` se precisar de vários arquivos de preferencias que serão identificados pelo nome ou `getPreferences()` se precisar de apenas um arquivo para sua *activity*, esse não necessita de nome pois será único para a *activity*. Em seguida para escrever valores você deve chamar o método `edit()` para obter um `SharedPreferences.Editor`, e usar os métodos como `putString()` para adicionar novos valores. Por último `commit()` deve ser

<sup>1</sup><http://developer.android.com/reference/android/content/SharedPreferences.html>

chamado para salvar os valores. Para ler os valores, você deve chamar métodos como `getBoolean()` ou `getString()`.

O exemplo abaixo salva os dados de uma caixa de texto, uma barra de progresso e uma chave em uma `SharedPreferences` e depois os lê quando o aplicativo é aberto novamente.

---

```

1 public class MainActivity extends Activity {
2     private EditText mEditText;
3     private SeekBar mSeekBar;
4     private Switch mSwitch;
5
6     @Override
7     protected void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.activity_main);
10
11         mEditText = (EditText) findViewById(R.id.editText1);
12         mSeekBar = (SeekBar) findViewById(R.id.seekBar1);
13         mSwitch = (Switch) findViewById(R.id.switch1);
14
15         SharedPreferences prefs = getPreferences(MODE_PRIVATE);
16         String text = prefs.getString("text", "");
17         int progress = prefs.getInt("seek", 0);
18         boolean checked = prefs.getBoolean("switch", false);
19
20         mEditText.setText(text);
21         mSeekBar.setProgress(progress);
22         mSwitch.setChecked(checked);
23     }
24
25     @Override
26     protected void onStop() {
27         super.onStop();
28
29         SharedPreferences prefs = getPreferences(MODE_PRIVATE);
30         SharedPreferences.Editor editor = prefs.edit();
31
32         editor.putString("text", mEditText.getText().toString());
33         editor.putInt("seek", mSeekBar.getProgress());
34         editor.putBoolean("switch", mSwitch.isChecked());
35         editor.commit();
36     }
37     ...
38 }
```

---

Algoritmo 7.1: Utilizando `SharedPreferences` para salvar dados primitivos

Observe no algoritmo 7.1 a utilização da classe `SharedPreferences`, no método `onStop()` chamamos o método `getPreferences(MODE_PRIVATE)` para obter uma instância da classe no modo privado, isto é, os dados somente são acessíveis por esta *activity* deste

aplicativo. Na linha 30 obtemos o `SharedPreferences.Editor` ao chamar o método `SharedPreferences.edit()`.

O Editor é usado então para adicionar os valores que obtemos das *views*. Na linha 32 usamos `putString()` para guardar o conteúdo da caixa de texto, `putInt()` para guardar a posição da SeekBar e `putBoolean()` para guardar o estado da chave. Finalmente invocamos `commit()` para salvar os dados.

No método `onCreate()` usamos os métodos `get` para obter os dados que foram salvos. Esses métodos requerem a passagem de um valor *default* para quando não há dados previamente salvos. Observe então nas linhas 16-18 que para a caixa de texto usamos uma *string* vazia, para a SeekBar o valor 0 e a chave desligada.

## 7.2 Armazenamento interno

Você pode salvar arquivos diretamente na memória interna do dispositivo. Por padrão os arquivos salvos no armazenamento interno são privados a seu aplicativo e não podem ser acessados por outros aplicativos.

Para criar um novo arquivo você deve invocar o método `openFileOutput()` passando o nome do arquivo e o modo de operação. Esse método irá retornar um `FileOutputStream`. Você poderá escrever nesse arquivo chamando o método `FileOutputStream.write()` e `FileOutputStream.close()` para fechar o arquivo.

Para exemplificar, faremos um aplicativo do tipo bloco de notas. Será composto de dois botões, um para abrir um arquivo e outro para salvar um arquivo, e uma caixa de texto para escrever. Tudo será feito no método `onCreate()`, primeiro usamos `findViewById()` para obter as *views*. O botão de salvar irá abrir um dialogo perguntando o nome do arquivo, e o botão de abrir mostrará um dialogo permitindo ao usuário escolher dentre os arquivos já salvos anteriormente.

No código abaixo criamos o *listener* do botão de salvar, dentro dele criamos um dialogo e precisamos de um *listener* para confirmar e um para cancelar.

---

```
1 saveBtn.setOnClickListener(new OnClickListener() {
2
3     @Override
4     public void onClick(View v) {
5
6         AlertDialog.Builder builder =
7             new AlertDialog.Builder(MainActivity.this);
8         LayoutInflater inflater = getLayoutInflater();
9         final View fnameEntry =
10            inflater.inflate(R.layout.save_dialog, null);
11         builder.setView(fnameEntry).setTitle("Save as...")
12             .setPositiveButton("Save", new DialogInterface.OnClickListener() {
13
14             ...
15         });
16     }
17 }
```

---

Algoritmo 7.2: Passos iniciais do *listener*, criando um dialogo.

Com o `AlertDialog.Builder` começamos a construir um novo diálogo. Usamos o `LayoutInflater` para carregar o *layout* com uma `EditText`. Chamamos `setPositiveButton()` para criar um *listener* que irá salvar o arquivo quando o botão for clicado.

---

```

1 ...
2 .setPositiveButton("Save", new DialogInterface.OnClickListener() {
3
4     @Override
5     public void onClick(DialogInterface dialog, int which) {
6         EditText fnameEt = (EditText) fnameEntry.findViewById(R.id.saveas);
7         String fname = fnameEt.getText().toString();
8         if(fname.isEmpty())
9             fname = "untitled";
10        String text = textEt.getText().toString();
11
12        try {
13            FileOutputStream fos = openFileOutput(fname, MODE_PRIVATE);
14            fos.write(text.getBytes());
15            fos.close();
16        } catch (FileNotFoundException e) {
17            e.printStackTrace();
18        } catch (IOException e) {
19            e.printStackTrace();
20        }
21
22        Toast.makeText(getApplicationContext(),
23                fname + " saved", Toast.LENGTH_SHORT).show();
24    }
25 }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
26 ...

```

---

Algoritmo 7.3: Salvando um arquivo e mostrando um `Toast`

Vamos analisar o algoritmo 7.3. Na linha 5 queremos a referência ao `EditText` contido no diálogo, para isso precisamos usar a variável `fnameEntry` criada ao chamar o método `LayoutInflater.inflate()`. Depois, na linha 7-8 certificamos que o nome do arquivo não será vazio (se for vazio, o arquivo não é salvo), para isso atribuimos o nome *untitled* a ele. Obtemos também o texto escrito na caixa de texto da *activity* na linha 9.

Agora que já temos o nome e os dados do arquivo, precisamos efetivamente salvá-lo no armazenamento interno. Com o método `openFileOutput()` teremos um `FileOutputStream` que servirá para escrever os dados no arquivo. Isso é feito nas linhas 13-14 com o método `write()` e em seguida o método `close()`. Para certificar ao usuário que o arquivo foi salvo, mostramos um `Toast` com a mensagem que o arquivo foi salvo.

Se o usuário clicar em *cancel* no diálogo, precisamos fechá-lo, basta invocar `dialog.close()` no *listener*.

---

```
1 ...
2 }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
3
4     @Override
5     public void onClick(DialogInterface dialog, int which) {
6         dialog.cancel();
7     }
8 } );
9 ...
```

---

Algoritmo 7.4: Fechando o dialogo ao clicar em *close*

Depois chame `builder.create().show()` no listener do botão *Save* para mostrar o diálogo.

Para o botão *Open* iremos mostrar um dialogo com os nomes dos arquivos salvos anteriormente, quando um desses for clicado o arquivo se abrirá preenchendo a caixa de texto.

---

```
1 openBtn.setOnClickListener(new OnClickListener() {
2
3     @Override
4     public void onClick(View v) {
5
6         AlertDialog.Builder builder =
7             new AlertDialog.Builder(MainActivity.this);
8
9         builder.setTitle("Choose a File")
10        .setItems(fileList(), new DialogInterface.OnClickListener() {
11
12            ...
13        });
14
15        builder.create().show();
16    }
17 }) ;
```

---

Algoritmo 7.5: Criando um diálogo com os arquivos salvos

Dessa vez usamos o método `builder.setItems()` para construir uma lista no diálogo. O método `fileList()` retorna um array de nomes de arquivo que foram armazenados internalmente no aplicativo.

```

1 ...
2 .setItems(fileList(), new DialogInterface.OnClickListener() {
3
4     @Override
5     public void onClick(DialogInterface dialog, int which) {
6
7         try {
8             File f = getFilesDir().listFiles()[which];
9             BufferedReader in = new BufferedReader(new FileReader(f));
10            StringBuilder text = new StringBuilder();
11
12            try{
13                String line = null;
14                while ((line = in.readLine()) != null) {
15                    text.append(line);
16                    text.append(System.getProperty("line.separator"));
17                }
18            } finally {
19                in.close();
20            }
21
22            textEt.setText(text);
23            in.close();
24        } catch(FileNotFoundException e){
25            e.printStackTrace();
26        } catch (IOException e) {
27            e.printStackTrace();
28        }
29    }
30 } );
31 ...

```

---

Algoritmo 7.6: Criando um diálogo com os arquivos salvos

Na linha 7 do algoritmo 7.6 queremos obter a referência do arquivo selecionado. Chamando o método `getFilesDir()` nos é fornecido o diretório contendo os arquivos, com `listFiles()` temos um *array* de `File`s e o selecionamos com a variável `which`, que é a posição clicada na lista do diálogo. Para ler os dados desse arquivo, usamos a classe `BufferedReader` e guardamos as linhas sendo lidas em um objeto do tipo `StringBuilder`. Nas linhas 13-16 lemos cada linha do arquivo em um loop *while* e usamos o método `append()` duas vezes, uma para adicionar a linha lida na *string* e outra para adicionar o separador de linha (comumente `\n`). Ao fim invocamos `close()` para fechar o `BufferedReader`. E colocamos o texto lido de volta na caixa de texto.

**Dica:** Você também pode optar por salvar seus arquivos no *cache*, isto é, na pasta *cache* da sua aplicação. Esses arquivos são apagados automaticamente quando o sistema necessita de memória. Para isso use o método `getCacheDir()` para obter o caminho do diretório de *cache*.

## 7.3 Armazenamento Externo

Todo dispositivo Android suporta um armazenamento externo que é compartilhado e que pode ser usado para salvar arquivos. Esse armazenamento pode ser tanto um cartão SD que está conectado ao aparelho como a própria memória interna dele. Os arquivos salvos no armazenamento externo podem ser lidos por todos outros aplicativos e modificados quando o usuário conecta ao computador para transferir os arquivos via USB.

Se um dispositivo usa uma partição da memória interna do aparelho como armazenamento externo e ainda oferecer um cartão SD, então a partição do cartão SD não estará disponível para armazenamento, sua aplicação não conseguirá acessar o cartão SD.

Por isso, o armazenamento externo pode ficar indisponível caso o usuário monte o armazenamento externo no computador ou remova o cartão SD. Os arquivos são abertos e podem ser abertos, modificados ou removidos pelo usuário ou por outras aplicações.

Para poder usar o armazenamento externo, primeiro você deve verificar sua disponibilidade usando o método `getExternalStorageState()`. A mídia pode estar conectada a um computador, faltando ou em estado de apenas leitura. Você pode verificar a disponibilidade da mídia como mostrado no exemplo abaixo, retirado da documentação oficial<sup>2</sup>.

---

```
1 boolean mExternalStorageAvailable = false;
2 boolean mExternalStorageWriteable = false;
3 String state = Environment.getExternalStorageState();
4
5 if (Environment.MEDIA_MOUNTED.equals(state)) {
6     mExternalStorageAvailable = mExternalStorageWriteable = true;
7 } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
8     mExternalStorageAvailable = true;
9     mExternalStorageWriteable = false;
10} else {
11    mExternalStorageAvailable = mExternalStorageWriteable = false;
12}
```

---

Algoritmo 7.7: Verificando se o armazenamento externo está disponível

O algoritmo 7.7 verifica se o armazenamento externo está disponível. Temos duas variáveis de controle que guardarão o estado da mídia, `mExternalStorageAvailable` e `mExternalStorageWriteable`. A primeira nos diz se o armazenamento externo está disponível e a segunda se é possível escrever nele. Na variável `state` é guardado o estado que é obtido através do método `getExternalStorageState()`. Uma comparação é feita com o resultado obtido, se for equivalente à `Environment.MEDIA_MOUNTED` então sabemos que a mídia está montada e é possível escrever nela, então setamos os estados das variáveis de controle para `true`. Caso o resultado seja equivalente à `Environment.MEDIA_MOUNTED_READ_ONLY` então sabemos que a mídia está montada porém está somente com permissão de leitura, nesse caso setamos apenas a variável de disponibilidade como `true` e mantemos a outra como `false`. Se o resultado não for esses dois estados, mas sim algum outro estado possível, então setamos as duas variáveis para `false`.

Após verificar a disponibilidade, você deve usar o método `getExternalFilesDir()` para obter uma referência ao caminho do diretório onde você deve salvar seus arquivos. Esse

<sup>2</sup><http://developer.android.com/guide/topics/data/data-storage.html#filesExternal>

método recebe um parâmetro que especifica o tipo de subdiretório que você quer usar, tais como DIRECTORY\_MUSIC e DIRECTORY\_RINGTONES, ou passe null para obter a raiz dos diretórios da sua aplicação. O método irá criar o diretório apropriado se necessário. Ao especificar o tipo, você se assegura que o Android irá categorizar seus arquivos de forma apropriada. Se o usuário desinstalar seu aplicativo, todos os dados serão deletados.

Se você quer salvar arquivos que não são específicos da sua aplicação e que não devem ser deletados quando desinstalado, salve-os em um dos diretórios públicos que estão no raiz da mídia de armazenamento externo. São eles os diretórios Music/, Pictures/, etc. Use o método getExternalStoragePublicDirectory() passando o tipo de diretório desejado, da mesma forma que anteriormente.

O sistema classifica os arquivos encontrados nessas pastas na forma:

- Music/ - Músicas do usuário;
- Podcasts/ - podcasts;
- Ringtones/ - Toques;
- Alarms/ - Toques do despertador;
- Notifications/ - Toques das notificações;
- Pictures/ - Fotos (exceto aquelas tiradas com a camera);
- Movies/ - Filmes (exceto aquelas gravadas com a camera); e
- Download/ - Arquivos baixados.

## 7.4 Banco de dados

O Android fornece suporte à bancos de dados SQLite. Qualquer banco que você criar será acessível apenas pela sua aplicação e não fora dela.

A maneira recomendada de criar um banco de dados é criando uma subclasse da classe SQLiteOpenHelper e sobrescrever o método onCreate(). Para ler e escrever no banco, chame os métodos getReadableDatabase() e getWritableDatabase(). Ambas retornam um objeto do tipo SQLiteDatabase que fornece métodos para operar sobre o banco.

Usando o método SQLiteDatabase.query() podemos realizar uma consulta, o método aceita vários parâmetros, tais como: tabela, seleção, colunas, agrupamento e etc. Para consultas mais complexas você pode usar a classe SQLiteQueryBuilder que contém vários métodos para construir consultas.

Toda consulta vai retornar um Cursor que aponta para as tuplas do resultado da consulta. Você deverá usá-lo para navegar através dos resultados.

Para exemplificar, vamos fazer um mecanismo simples de busca. O banco será populado com fabricantes de carros e alguns de seus modelos. Para o *design* desse exemplo, optei por ter dois Spinners (equivalente ao *Combo Box*) e um botão. Os Spinners serão populados com os dados do Banco de Dados de forma que o primeiro contenha todos os fabricantes de carros e o segundo todos os anos de fabricação de seus modelos, esses dados serão obtidos através de consultas ao banco.

Para começar, devemos criar o BD e isso é feito com uma classe que iremos criar que irá ser subclasse de SQLiteOpenHelper<sup>3</sup>. Chamaremos de CarOpenHelper. Essa classe tem algumas funções básicas, como criar e popular o BD, configurar a ação a ser tomada quando o BD é atualizado ou desatualizado.

<sup>3</sup><http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

O código 7.8 abaixo demonstra a criação do BD usado no exemplo:

---

```

1 public class CarOpenHelper extends SQLiteOpenHelper {
2     private static final int DATABASE_VERSION = 1;
3     private static final String DATABASE_NAME = "MyCars";
4     private static final String DATABASE_TABLE_NAME = "Cars";
5     private static final String DATABASE_TABLE_CREATE =
6         "CREATE TABLE " + DATABASE_TABLE_NAME + " (" +
7             "SN INT PRIMARY KEY, " +
8             "Manufacturer TEXT, " +
9             "Model TEXT, " +
10            "Year INT);";
11
12     public CarOpenHelper(Context context) {
13         super(context, DATABASE_NAME, null, DATABASE_VERSION);
14     }
15
16     @Override
17     public void onCreate(SQLiteDatabase db) {
18         db.execSQL(DATABASE_TABLE_CREATE);
19
20         db.execSQL("INSERT INTO " + DATABASE_TABLE_NAME +
21                 " SELECT '1' AS SN, 'Mazda' AS Manufacturer, " +
22                 "'MX-5' AS Model, '1991' AS Year" +
23                 " UNION SELECT '2', 'Mazda', 'RX-8', '2001'" +
24                 " UNION SELECT '3', 'Mazda', 'Speed3', '2007'" +
25                 " UNION SELECT '4', 'Subaru', 'Impreza', '2010'" +
26                 " UNION SELECT '5', 'Fiat', '500', '2012'" +
27                 " UNION SELECT '6', 'Ford', 'Focus', '2008'" +
28                 " UNION SELECT '7', 'Fiat', 'Punto', '2012'" +
29                 " UNION SELECT '8', 'Ford', 'Fiesta', '2006'" +
30                 " UNION SELECT '9', 'Honda', 'Civic', '2013'" +
31                 " UNION SELECT '10', 'Honda', 'Fit', '2010'" +
32                 " UNION SELECT '11', 'Toyota', 'Corolla', '2010'" +
33                 " UNION SELECT '12', 'Chevrolet', 'Celta', '2009'" +
34                 " UNION SELECT '13', 'Chevrolet', 'Cruze', '2012');");
35
36     }
37 }
```

---

Algoritmo 7.8: Classe CarOpenHelper do SQLite

Primeiro criamos algumas *Strings* para ajudar na criação do banco. Em seguida, criamos o construtor da classe, os argumentos passados são: o contexto da *activity* que instanciou a classe, o nome do banco, um *CursorFactory* caso esteja usando (não é o caso), e a versão do banco que começa de 1 e deve incrementar a medida que você o atualiza.

O método `onCreate()` do banco é responsável pela criação do banco de o popular. Fazemos isso executando consultas SQL diretamente ao banco com o método `execSQL()`. No código 7.8 você pode observar a chamada de `execSQL()` duas vezes. Não se confunda com

a segunda consulta, que adiciona as tuplas ao banco, é só uma maneira de inserir várias tuplas em uma única consulta.

Agora na nossa *activity* precisamos realizar as consultas para popular os *Spinners*. No método `onCreate()` faça:

---

```

1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      ...
4
5      mDatabase = new CarOpenHelper(this);
6      String[] mColumns = {"Manufacturer", "Year"};
7
8      SQLiteDatabase mSQLite = mDatabase.getReadableDatabase();
9      Cursor cursor =
10         mSQLite.query("Cars", mColumns, null, null, null, null, null);
11
12     Set<String> set1 = new HashSet<String>();
13     Set<String> set2 = new HashSet<String>();
14     while(cursor.moveToNext()) {
15         set1.add(cursor.getString(0));
16         set2.add(cursor.getString(1));
17     }
18     List<String> list1 = new ArrayList<String>(set1);
19     List<String> list2 = new ArrayList<String>(set2);
20
21     ArrayAdapter<String> manuAdapter =
22         new ArrayAdapter<String>
23             (this, android.R.layout.simple_spinner_item, list1);
24     manuAdapter.
25         setDropDownViewResource
26             (android.R.layout.simple_spinner_dropdown_item);
27     mSpManufacturer.setAdapter(manuAdapter);
28
29     ArrayAdapter<String> yearAdapter =
30         new ArrayAdapter<String>
31             (this, android.R.layout.simple_spinner_item, list2);
32     manuAdapter.
33         setDropDownViewResource
34             (android.R.layout.simple_spinner_dropdown_item);
35     mSpYear.setAdapter(yearAdapter);

```

---

Algoritmo 7.9: Usando o `CarOpenHelper` na *activity*

Estou "pulando" a parte de obter as referências dos *spinners* e botões e partindo para o que interessa no código 7.9. Na linha 5 instanciamos o `CarOpenHelper` na variável `mDatabase`. Na linha 6 criamos um *array* que contém as colunas que iremos fazer consulta, isto é necessário para o método `query` na linha 10.

Ao chamar `mDatabase.getReadableDatabase()` obtemos um objeto do tipo `SQ-`

LiteDatabase<sup>4</sup> mas que só permite a leitura. Já o método query retorna um Cursor que é usado para iterar sobre os resultados. Na linha 10 observe os parâmetros que usamos para fazer a consulta: A tabela que vamos fazer a consulta, um array de strings com as colunas que queremos obter, uma cláusula WHERE (observe que passar null aqui implica em retornar todas as tuplas da coluna), argumentos da seleção (para aqueles que conhecem *prepared statements* onde "?" é substituído pelos valores, uma cláusula GROUP BY (null significa não agrupar), uma cláusula HAVING, uma cláusula ORDER BY e uma cláusula LIMIT.

Em seguida criamos dois Set para armazenar os resultados e omitir os repetidos (poderia ter feito duas consultados e usado DISTINCT, claramente um banco mal projetado). Usamos o Cursor para iterar sobre o resultado e adiciona-los aos conjuntos. O set1 contém os fabricantes e o set2 contém os anos de fabricação. Usamos Cursor.getString(int index) para obter o elemento do resultado que está naquele índice, nesse caso o índice 0 é a coluna de fabricantes e o índice 1 é a coluna de anos de fabricação. Com esses dois conjuntos em mão podemos criar duas ArrayList para serem usadas com o ArrayAdapter que irá popular os Spinners.

Depois é necessário criar dois listeners para os elementos dos Spinners que guarda numa variável o fabricante e o ano de fabricação selecionados (aqui irei omitir código, mas pode ser obtido no repositório). E um listener para o botão que irá chamar o método openCarList().

---

```
1 public void openCarList() {
2     Intent intent = new Intent(this, CarDetailActivity.class);
3     List<String> list = fetchCarList();
4     String[] carList = list.toArray(new String[list.size()]);
5     intent.putExtra(CARLIST, carList);
6     startActivity(intent);
7 }
```

---

Algoritmo 7.10: Método openCarList()

Esse método é responsável por obter a lista de carros da seleção feita nos Spinners e mandar essa lista para outra activity. O método fetchCarList() é que irá fazer a chamada ao BD, da mesma forma com que foi feito anteriormente, mas dessa vez estamos obtendo a coluna Model do banco com WHERE sendo os parâmetros obtidos.

---

<sup>4</sup><http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

```
1 public List<String> fetchCarList() {
2     SQLiteDatabase mSQLite = mDatabase.getReadableDatabase();
3     String[] column = {"Model"};
4     Cursor cursor =
5         mSQLite.query("Cars", column,
6             "Year='"+selYear+"' AND Manufacturer='"+selManufacturer+"',
7             null, null, null, null);
8
9     List<String> list1 = new ArrayList<String>();
10    while(cursor.moveToNext()) {
11        list1.add(cursor.getString(0));
12    }
13
14    return list1;
15 }
```

---

Algoritmo 7.11: Método fetchCarList ()

O resultado de `fetchCarList()` é enviado para outra *activity* que irá mostrar os modelos em forma de uma lista.

Nesse capítulo podemos aprender apenas o básico de acesso ao banco de dados SQLite, é um assunto que tem muito mais o que aprender e recomendo olhar a documentação para futuras consultas.

# CAPÍTULO

# 8

## Câmera

---

O Android fornece um *framework* para acesso às diferentes cameras e funcionalidades dessas cameras disponíveis no dispositivo. É possível tirar fotos e gravar vídeos nos aplicativos. Há duas formas de acessar a camera, a primeira sendo pela API e a segunda pelo *intent* da camera. Nesse capítulo iremos discutir brevemente sobre como usar a API e um exemplo de chamar o aplicativo padrão da camera através de um *intent*.

### 8.1 Usando a API

Para usar a API da camera, primeiro devemos requisitar a permissão de usar a camera no *Manifest*.

---

```
1 <uses-permission android:name="android.permission.CAMERA" />
2
3 <uses-feature android:name="android.hardware.camera" />
```

---

Algoritmo 8.1: Requisitando permissão de usar a camera

Você pode ainda requisitar outras funcionalidades como *flash*, foco automático, usar a camera da frente ao requisitar outras *features* usando `<uses-feature>`. A lista das funcionalidades está presente na [documentação](#)<sup>1</sup>.

Se você for guardar as fotos no armazenamento externo, precisa requisitar a permissão de escrita:

---

```
1 <uses-permission android:name=
2   "android.permission.WRITE_EXTERNAL_STORAGE" />
```

---

Algoritmo 8.2: Requisitando permissão de gravar no armazenamento externo

<sup>1</sup><http://developer.android.com/guide/topics/manifest/uses-feature-element.html#hw-features>

Se for gravar áudio quando estiver gravando vídeo, precisa requisitar a permissão de gravar áudio:

---

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

---

Algoritmo 8.3: Requisitando permissão de gravar áudio

Segundo a documentação<sup>2</sup> existem alguns passos gerais que são seguidos para criar um aplicação usando a API da camera. São eles:

- **Detectar e acessar a camera:** Código que verifica a existência de uma camera e requisita o acesso.
- **Criar uma classe de pré-visualização:** Criar uma classe que extende `SurfaceView` e implementa `SurfaceHolder` para mostrar na tela o que a camera está vendo.
- **Construir o *layout* da pré-visualização:** Com a classe pronta, crie uma *view* que irá incorporar a classe de pré-visualização e mostrar os controles da interface.
- **Configurar os *listeners* para a captura:** Criar *listeners* para os botões da interface para gravar a imagem.
- **Capturar e salvar os arquivos:** Criar o código que irá capturar a imagem e salvar no armazenamento.
- **Liberar a camera:** Depois de usar a camera, a aplicação deve liberar o uso para outras aplicações.

### 8.1.1 Acessando a camera

Primeiro vamos criar uma classe `CameraAccess` que irá requisitar o acesso à camera. A classe fica responsável apenas por verificar se o dispositivo tem camera e por instanciar a camera para o uso.

---

<sup>2</sup><http://developer.android.com/guide/topics/media/camera.html>

---

```
1 public class CameraAccess {
2     Context c;
3     Camera cam;
4
5     public CameraAccess(Context c) {
6         this.c = c;
7         cam = null;
8     }
9
10    private boolean checkCameraHardware() {
11        if(c.getPackageManager() .
12            hasSystemFeature(PackageManager.FEATURE_CAMERA) )
13            return true;
14        return false;
15    }
16
17    public Camera getCameraInstance() {
18        Camera cam = null;
19        if(checkCameraHardware()) {
20            try{
21                cam = Camera.open();
22            } catch (Exception e) {
23                e.printStackTrace();
24            }
25        } else {
26            return null;
27        }
28
29        return cam;
30    }
31
32    public void releaseCamera() {
33        cam.release();
34    }
35 }
```

---

Algoritmo 8.4: Classe CameraAccess

### 8.1.2 Pré-visualização

Em seguida iremos criar a classe da visualização da camera. Essa classe extende SurfaceView e implementa a interface SurfaceHolder.Callback, iremos chamar essa classe de CameraPreview. No código 8.5 abaixo começamos a implementar essa classe. Precisamos de duas variáveis SurfaceHolder e Camera. O construtor recebe o contexto e a camera previamente instanciada como parâmetro. Obtemos o SurfaceHolder usando o método getHolder(), adicionamos a classe como callback. Por último, na linha 11 configuramos o tipo do SurfaceHolder como SURFACE\_TYPE\_PUSH\_BUFFERS isso já é depreciado mas é necessário para acessar a camera em versões do Android mais antigas que 3.0.

```
1 public class CameraPreview extends SurfaceView implements Callback {
2     private SurfaceHolder mHolder;
3     private Camera mCamera;
4
5     public CameraPreview(Context c, Camera cam) {
6         super(c);
7         mCamera = cam;
8
9         mHolder = getHolder();
10        mHolder.addCallback(this);
11        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
12    }
13
14    @Override
15    public void surfaceCreated(SurfaceHolder holder) {
16        try{
17            mCamera.setPreviewDisplay(holder);
18            mCamera.startPreview();
19        } catch (IOException e) {
20            Log.d("CAM", "Error setting camera preview: " + e.getMessage());
21        }
22    }
23
24    @Override
25    public void surfaceChanged(SurfaceHolder holder, int format, int width,
26        int height) {
27        if(mHolder.getSurface() == null){
28            return;
29        }
30
31        try {
32            mCamera.stopPreview();
33        } catch (Exception e) {
34            // Irrelevante
35        }
36
37        try{
38            mCamera.setPreviewDisplay(mHolder);
39            mCamera.startPreview();
40        } catch (Exception e) {
41            Log.d("CAM", "Error setting camera preview: " + e.getMessage());
42        }
43    }
44
45    @Override
46    public void surfaceDestroyed(SurfaceHolder holder) {
47        // Nada, tomar conta de liberar a camera na activity
48    }
49 }
```

---

Algoritmo 8.5: Classe CameraPreview

Continuando a analisar o algoritmo 8.5. Precisamos sobrestrar os métodos da interface. Esses são: `surfaceCreated()`, `surfaceChanged()` e `surfaceDestroyed()`. O primeiro, `surfaceCreated()` é responsável por instanciar a pré-visualização. Para isso chame o método `setPreviewDisplay()` passando o `holder` como parâmetro para a câmera saber onde a visualização será desenhada. Depois é só chamar `startPreview()` para iniciar a visualização.

O segundo, `surfaceChanged` serve para parar e recomeçar a visualização quando o usuário rotaciona a tela, por exemplo. Para isso é necessário fechar a visualização e depois iniciar novamente. Na linha 27 testamos para saber se o `mHolder` não tem nada desenhado, se for o caso então apenas retorne do método pois não há nada para ser fechado. Caso contrário então precisamos parar a visualização com `stopPreview()` e depois iniciar novamente da mesma forma como é feito no método `surfaceCreated()`.

O terceiro, `surfaceDestroyed()` é irrelevante nesse caso pois estaremos tomando conta de liberar a camera na `activity` e não nessa classe.

### 8.1.3 Layout e Activity da visualização

A classe da visualização precisa ser instanciada por uma `activity` que vai efetivamente mostrar o conteúdo. Para o `layout` da `activity` iremos criar `FrameLayout` que irá conter a visualização e um botão pra capturar a imagem.

---

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="horizontal"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent" >
6
7     <FrameLayout
8         android:id="@+id/camera_preview"
9         android:layout_width="fill_parent"
10        android:layout_height="fill_parent"
11        android:layout_weight="1" />
12
13    <Button
14        android:id="@+id/button_capture"
15        android:text="Capture"
16        android:layout_width="wrap_content"
17        android:layout_height="wrap_content"
18        android:layout_gravity="center" />
19 </LinearLayout>
```

---

Algoritmo 8.6: *Layout* da *Activity* que irá conter a visualização

Você deve também especificar a orientação da `activity` como *landscape* no *Manifest*. Como mostrado na linha 5 do código 8.7. Apesar disso não ser obrigatório, é normal que tiremos fotos no modo paisagem e não retrato, mas isso não é regra.

```
1 ...
2 <activity
3     android:name="com.example.camera1.MainActivity"
4     android:label="@string/app_name"
5     android:screenOrientation="landscape" >
6 ...
```

---

Algoritmo 8.7: Configurando a orientação da *activity* no *Manifest*

### 8.1.4 Criando a *activity*

Para começar vamos apenas fazer com que a *activity* mostre a visualização da camera na tela.

```
1 public class CameraActivity extends Activity {
2     private Camera mCam;
3     private CameraPreview mPreview;
4
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         setContentView(R.layout.activity_camera);
9
10        CameraAccess ca = new CameraAccess(this);
11        mCam = ca.getCameraInstance();
12
13        mPreview = new CameraPreview(this, mCam);
14        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
15        preview.addView(mPreview);
16    }
```

---

Algoritmo 8.8: Primeira parte da classe CameraActivity

---

APÊNDICE

A

## Especificação blá, blá, blá

---

---

Isto é um apêndice...