

Sumário

| | |
|---|-----------|
| Lista de Figuras | iii |
| Lista de Tabelas | v |
| Lista de Algoritmos | viii |
| Lista de Abreviaturas | ix |
| Resumo | xi |
| 1 Introdução | 1 |
| 2 Configuração do Ambiente | 5 |
| 3 Linguagem do Android | 7 |
| 3.1 Linguagem | 7 |
| 3.2 Entendendo a estrutura de uma aplicação Android | 8 |
| 4 Criando seu primeiro aplicativo | 11 |
| 5 Design | 27 |
| 5.1 Activity | 27 |
| 5.2 Especifique a <i>activity</i> que inicia seu aplicativo | 28 |
| 5.3 Listas (<code>ListView</code>) | 29 |
| 5.4 Listas Compostas | 32 |
| 5.5 Listas expandíveis (<code>ExpandableListView</code>) | 35 |
| 5.6 Grades (<code>GridView</code>) e imagens <code>ImageView</code> | 40 |
| 5.7 Fragmentos | 46 |
| 5.8 Abas (<i>Tabs</i>) | 49 |
| 5.9 Arrastar (<code>SwipeView</code>) com abas | 50 |
| 5.10 Menu lateral | 50 |
| A Especificação blá, blá, blá | 51 |

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 | Esquema de uma lista | 29 |
| 5.3 | Detalhes de um elemento da lista | 29 |
| 5.4 | Lista simples | 32 |
| 5.5 | Lista Composta | 34 |
| 5.6 | Exemplo de lista expandível | 40 |
| 5.7 | Esquema de um <code>Grid view</code> | 40 |
| 5.8 | Demonstração de um <code>Grid view</code> | 43 |
| 5.9 | Esquema da interface com abas | 50 |

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

Listings

1

de configuração de versão do SDK no arquivo `AndroidManifest.xml` 16 2

da caixa de texto no arquivo `activity_main.xml` 17 3

do botão 18 4

de strings com as duas strings adicionadas 18 5

método à classe `MainActivity` 19 6

de import de uma classe `Android` 19 7

uma `Intent` 19 8

o conteúdo da caixa de texto e enviando para outra *activity* 19 9

como chave para um extra 20 10

a *string* passada como extra passados através do `Intent` 22 11

`onCreate()` recebendo um *Intent* e mostrando a mensagem 22 12

de *Launcher activity* 28 13

Linear no `activity_main.xml` 30 14

XML de um `ListView` 30 15

populada com elementos 30 16

de uma *activity* com lista clicável 31 17

do arquivo `item.xml` 33 18

da lista customizada 34 19

expandível no `activity_main.xml` 35 20

`list_item_parent.xml` 35 21

`list_item_child.xml` 36 22

`Parent` 36 23

`CustomAdapter` 38 24

a lista expandível na *activity* 39 25

do `Grid View` 41 26

`ImageAdapter` 42 27

com grade 43 28

`full_image.xml`44 29
`FullImageActivity`45 30
com grade, melhorado46 31
`BasicFragment`47 32
da *activity* com um fragmento48 33
da *activity* com o `FrameLayout`48 34
com adição dinâmica de fragmento49

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.

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.

Atualmente¹ 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 ²
- *Ice Cream Sandwich* versão 4.0 trouxe uma interface refinada e unificada para *smartphones* e *tablets* além de facilidade com multitasking e outros ³

¹Data em que foi escrito: 06/2013

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

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

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

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

| Versão | Codínome | 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

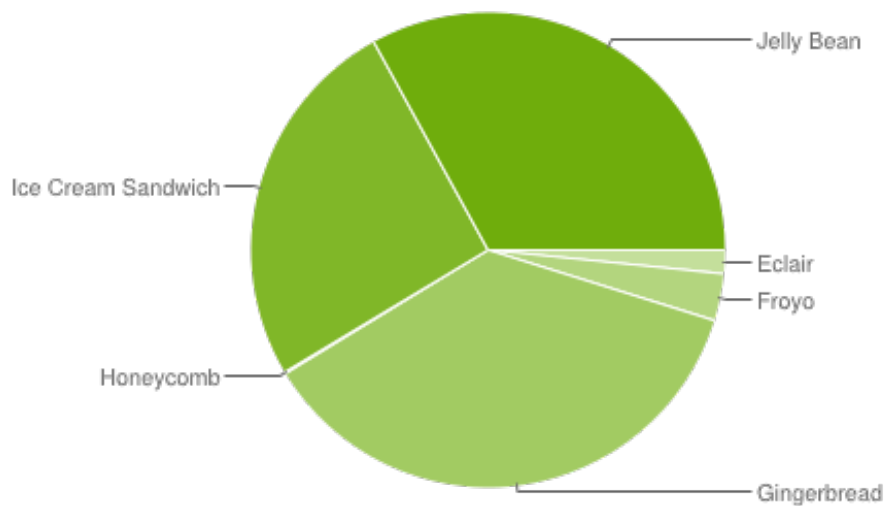


Figura 1.1: Distribuição das versões do Android

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

⁵*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

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](http://developer.android.com/sdk/)¹.

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*.

¹<http://developer.android.com/sdk/>

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`¹ 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*² ou através dos botões físicos do aparelho.

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

²*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.

Criando seu primeiro aplicativo

Para exemplificar a criação de um aplicativo, seguiremos o exemplo dado pelo próprio manual do Google sobre o Android (Ver original¹). 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 Java².

- *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.

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

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

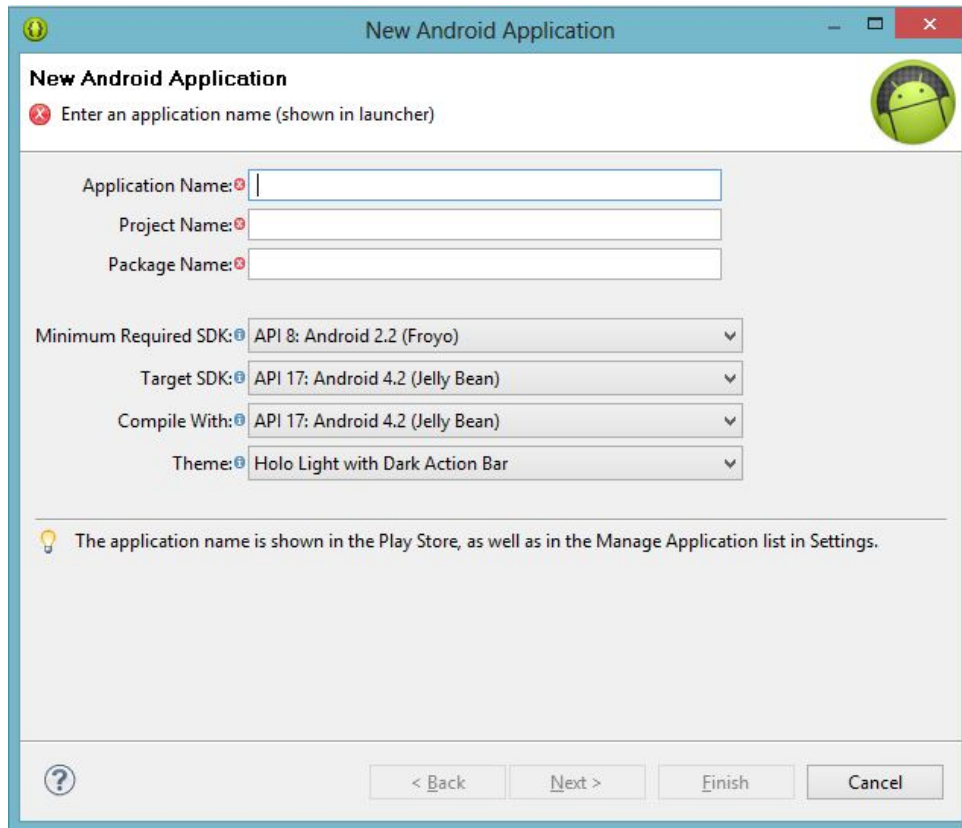


Figura 4.1: Primeira janela de criação de novo aplicativo

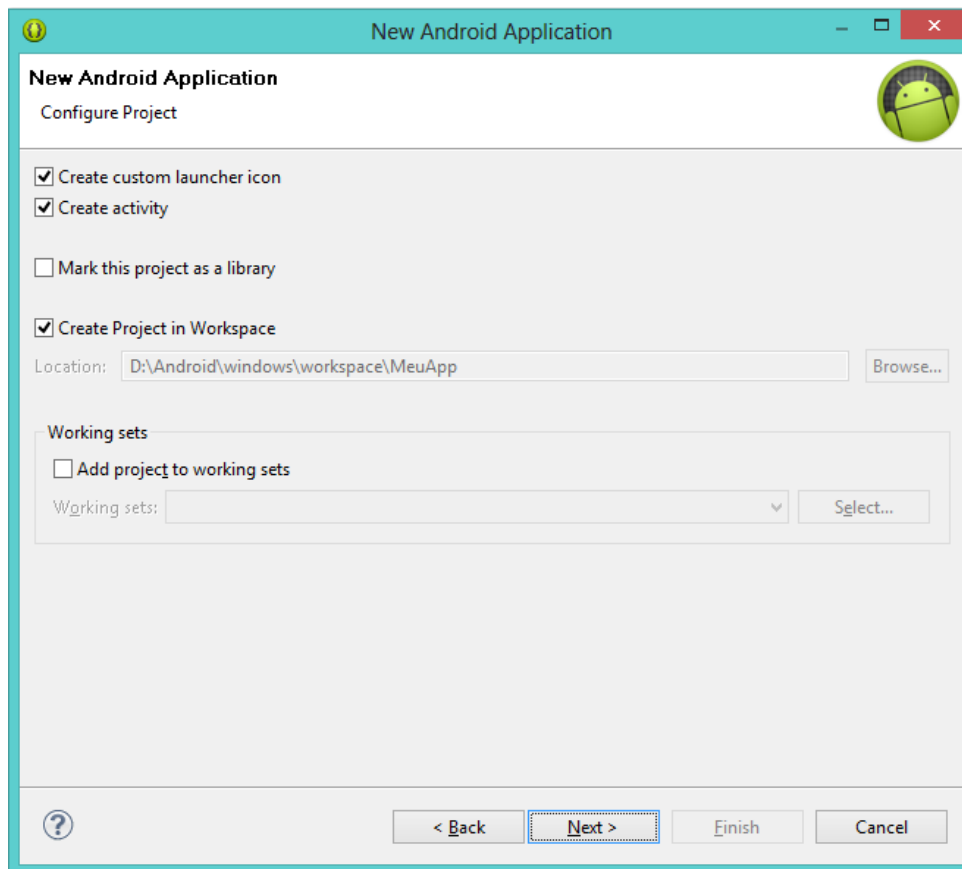


Figura 4.2: Segunda janela de criação de novo aplicativo

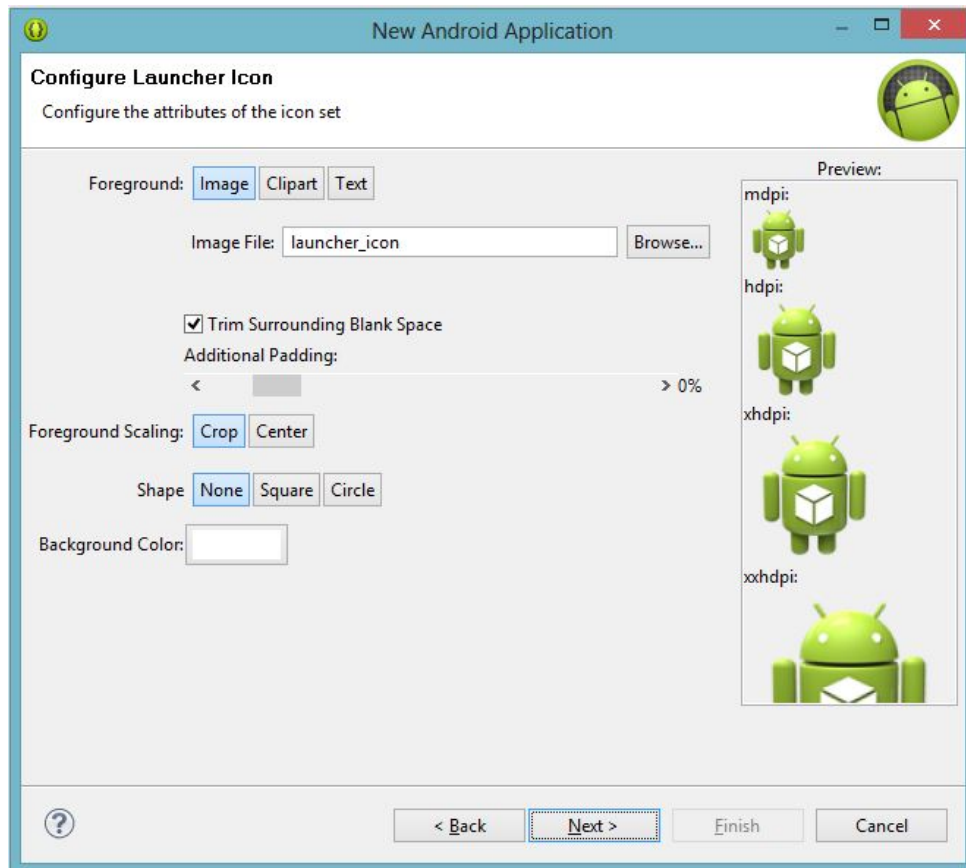


Figura 4.3: Terceira janela de criação de novo aplicativo

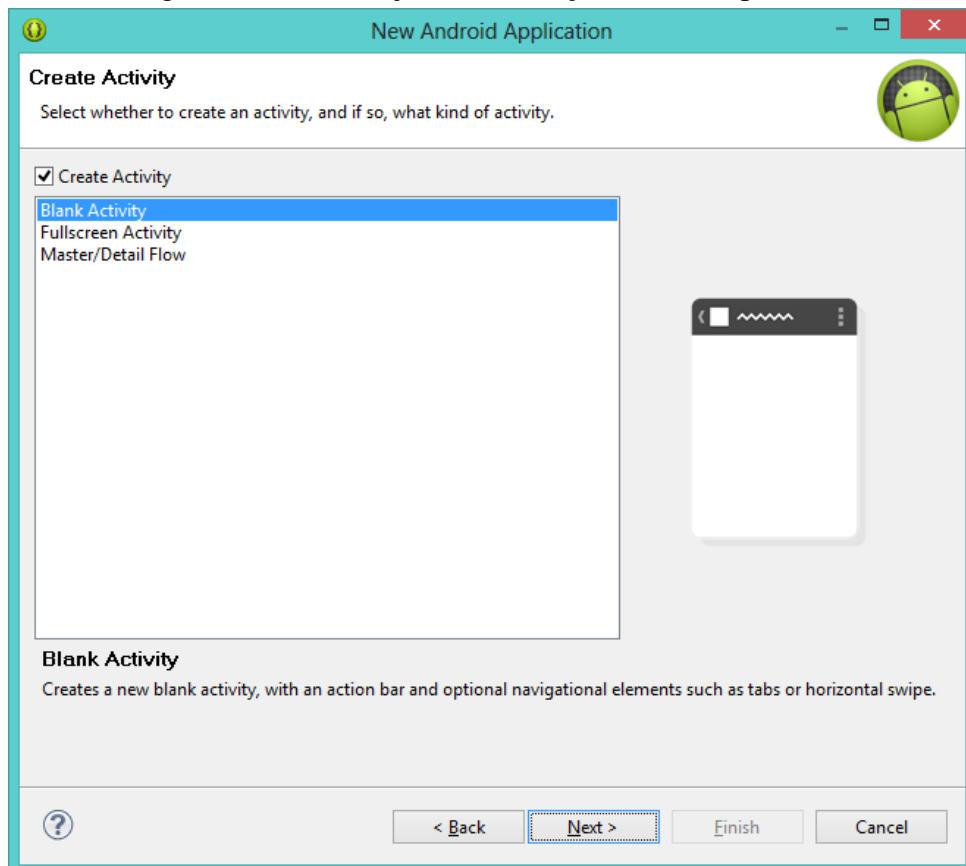


Figura 4.4: Quarta janela de criação de novo aplicativo

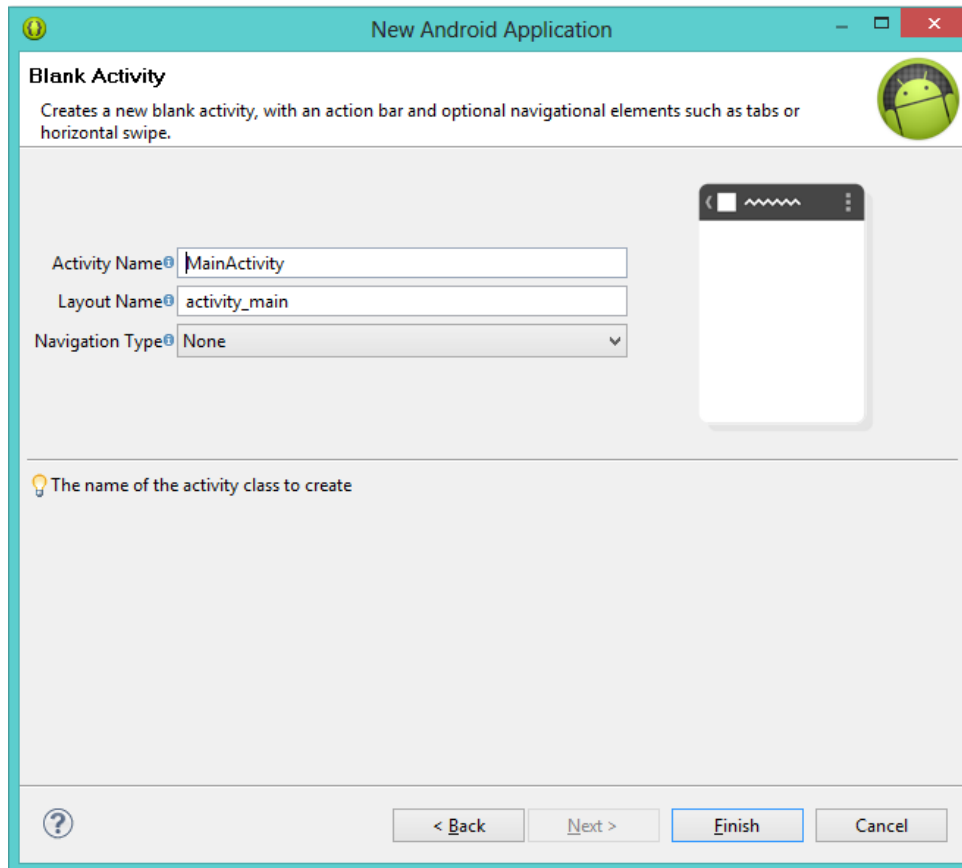


Figura 4.5: Quinta janela de criação de novo aplicativo

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*³
- *Fixed Tabs + Swipe*: O *layout* vem com algumas abas e com gesto de arrastar entre as abas (*activities*) pré-programados.⁴
- *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,

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

⁴*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 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*⁵.

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

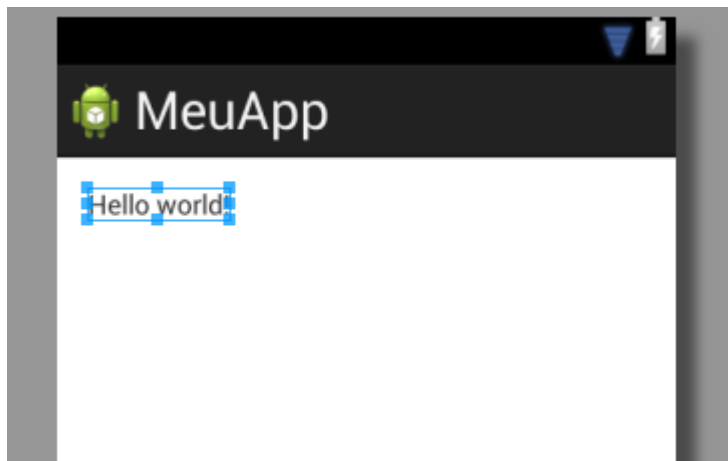
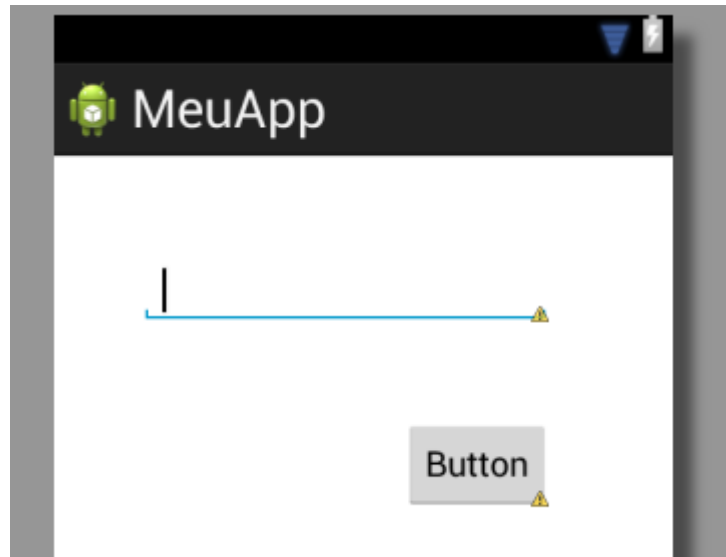


Figura 4.6: Selecionando o Hello world

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*.

⁵Mais 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 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`⁶ 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 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.

```

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: 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

⁶Mais 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 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 6: Exemplo de import de uma classe Android

Primeiro, crie um novo Intent⁷, um Intent é um objeto que provê uma facilidade para realizar uma ligação entre códigos de diferentes aplicações. O uso mais significativo 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 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 8.

```
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 8: Obtendo o conteúdo da caixa de texto e enviando para outra *activity*

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

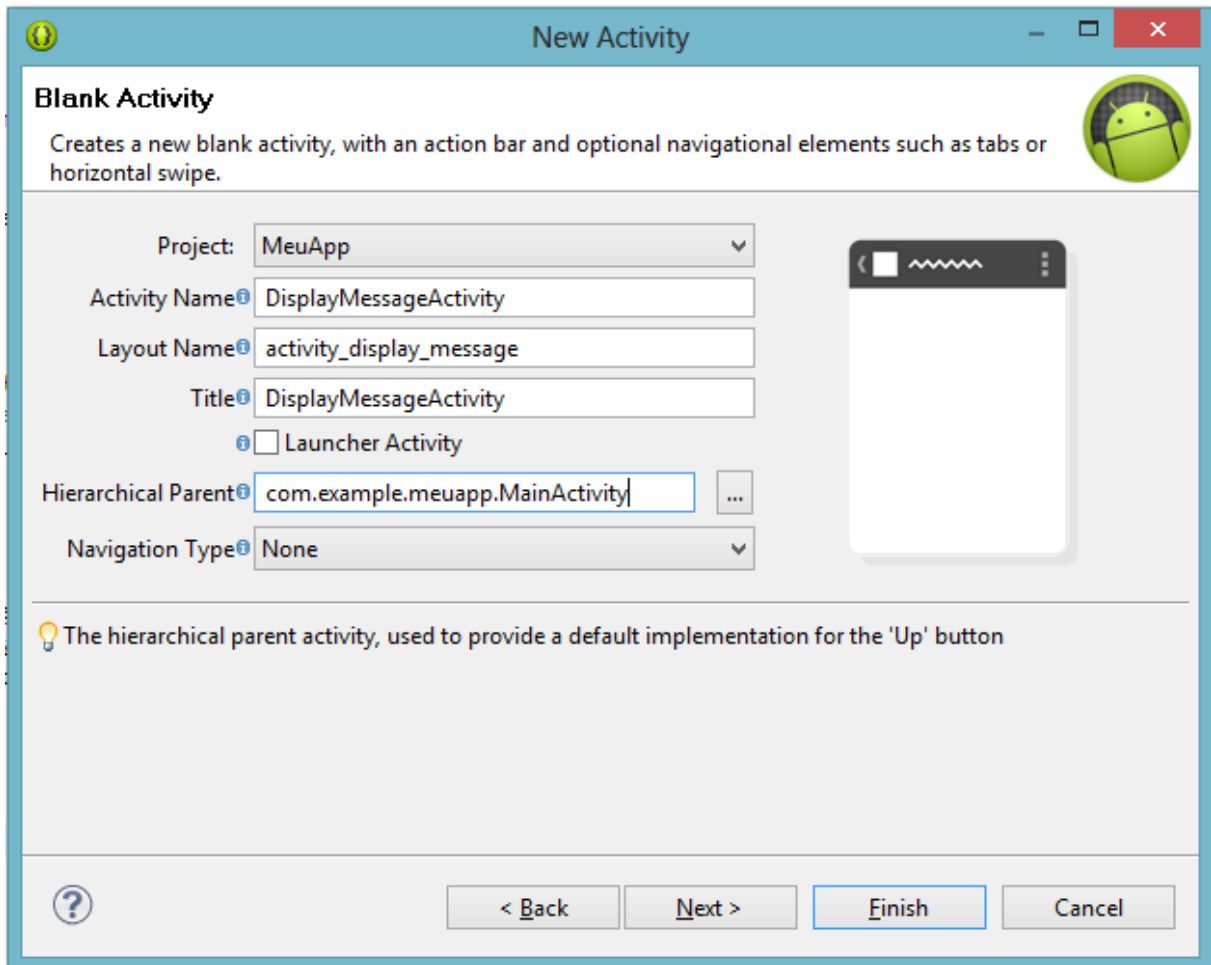
O código na linha 3 está obtendo a referência a caixa de texto usando o método `findViewById()` passando o *id* da caixa de texto como parâmetro, esse *id* é obtido acessando uma variável estática da classe `R` (observe que esse é o mesmo *id* que você colocou no arquivo xml do layout da *activity*). Em seguida usando o método `getText()` da caixa de texto, obtém-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 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()`⁸ que define o procedimento a ser executado quando a *activity* é criada.

⁸[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()`⁹.

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

Algoritmo 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`¹⁰, 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 conteudo 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 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 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

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

¹⁰<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](http://developer.android.com/tools/extras/oem-usb.html)¹¹
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*.

¹¹<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 Device¹²). 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, mais que isso pode acarretar em erros no sistema.
7. Clique *Create AVD*
8. Selecione o novo AVD no *Android Virtual Device Manager* e clique *Start*
9. Quando o emulador terminar de carregar, destrave a tela do emulador

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.

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

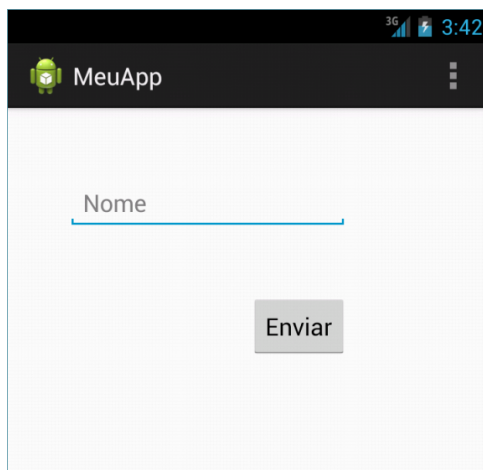


Figura 4.9: Primeira tela do primeiro aplicativo

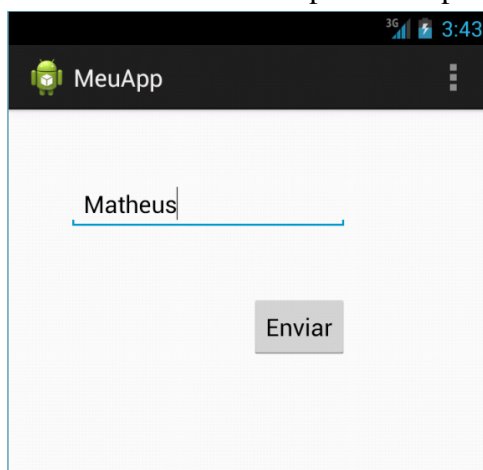


Figura 4.10: Primeira tela após escrever texto na caixa de texto

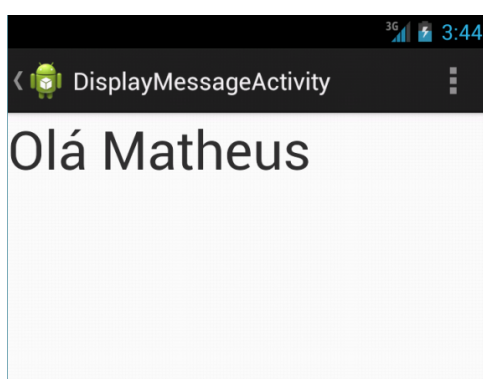


Figura 4.11: Segunda tela mostrando a mensagem enviada

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.

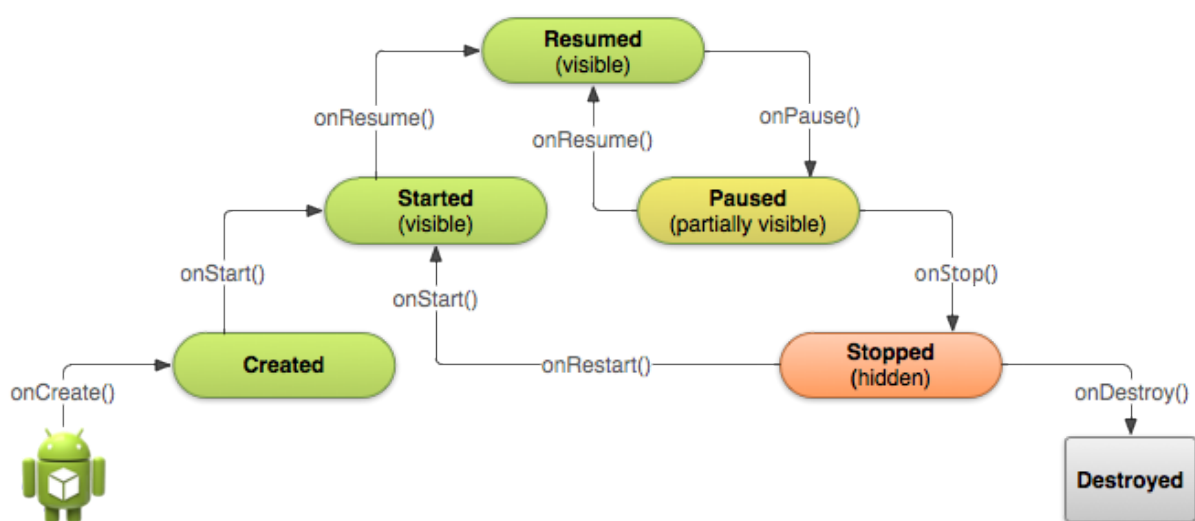


Figura 5.1: Ciclo de vida de uma *activity*

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>`¹ 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 12: Exemplo de *Launcher activity*

¹Documentação `<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 Listas (ListView)

² Listas são uma das formas mais simples e mais poderosas de se mostrar informações ao usuário de forma simples e objetiva, a `ListView` é altamente customizável através de adaptadores.

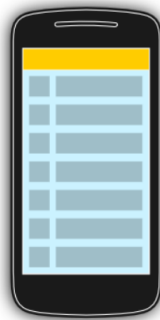


Figura 5.2: Esquema de uma lista

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

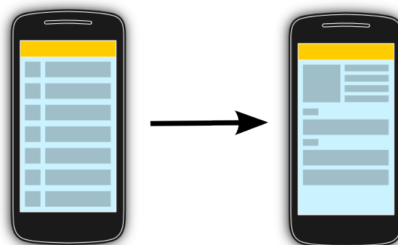


Figura 5.3: Detalhes de um elemento da lista

²Documentação `ListView`: <http://developer.android.com/reference/android/widget/ListView.html>

A construção é simples, você pode começar mudando o layout de `RelativeLayout` para `LinearLayout`, a diferença entre eles é que o `RelativeLayout` do exemplo passado permite você posicionar elementos uns relativos aos outros enquanto que o `LinearLayout` segue uma estrutura, vertical ou horizontal. Ambas podem ser aninhadas uma dentro de outra.

```
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 13: Layout *Linear* no `activity_main.xml`

Agora, você pode arrastar um `ListView` para o layout ou criar um manualmente com o seguinte código XML

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

Algoritmo 14: Código XML de um `ListView`

Agora para popular a lista, você precisa criar um `string-array` no arquivo `strings.xml` com os elementos que deseja colocar na lista.

```
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 15: `string-array` populada com elementos

E por final, escrever o código que irá preencher a lista com as *strings* desse *array*.

```

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 16: Código de uma *activity* com lista clicável

Adaptadores são usados para prover dados para o `ListView`, ele define como cada linha será mostrada. Um adaptador estende a classe `BaseAdapter`, o Android provê alguns adaptadores padrões, no caso estamos usando o `ArrayAdapter` que é para manusear dados em *arrays*.

Dentro da interface `OnItemClickListener` você pode configurar a ação de clicar em um dos itens da lista, nesse exemplo é criado um `Toast` que mostra uma mensagem com o texto do item na lista porém, qualquer ação pode ser executada incluindo abrir uma nova *activity* que esteja relacionada a esse item.

Você pode também herdar da classe `ListActivity` para uma forma mais simples de manusear `ListsViews`. Você não precisa atribuir um layout a `ListActivity` contém uma `ListView` padrão. Caso voce precise colocar mais Views em seu layout, você *deve* colocar a `ListView` em seu layout com o id `"@android:id/list"`.

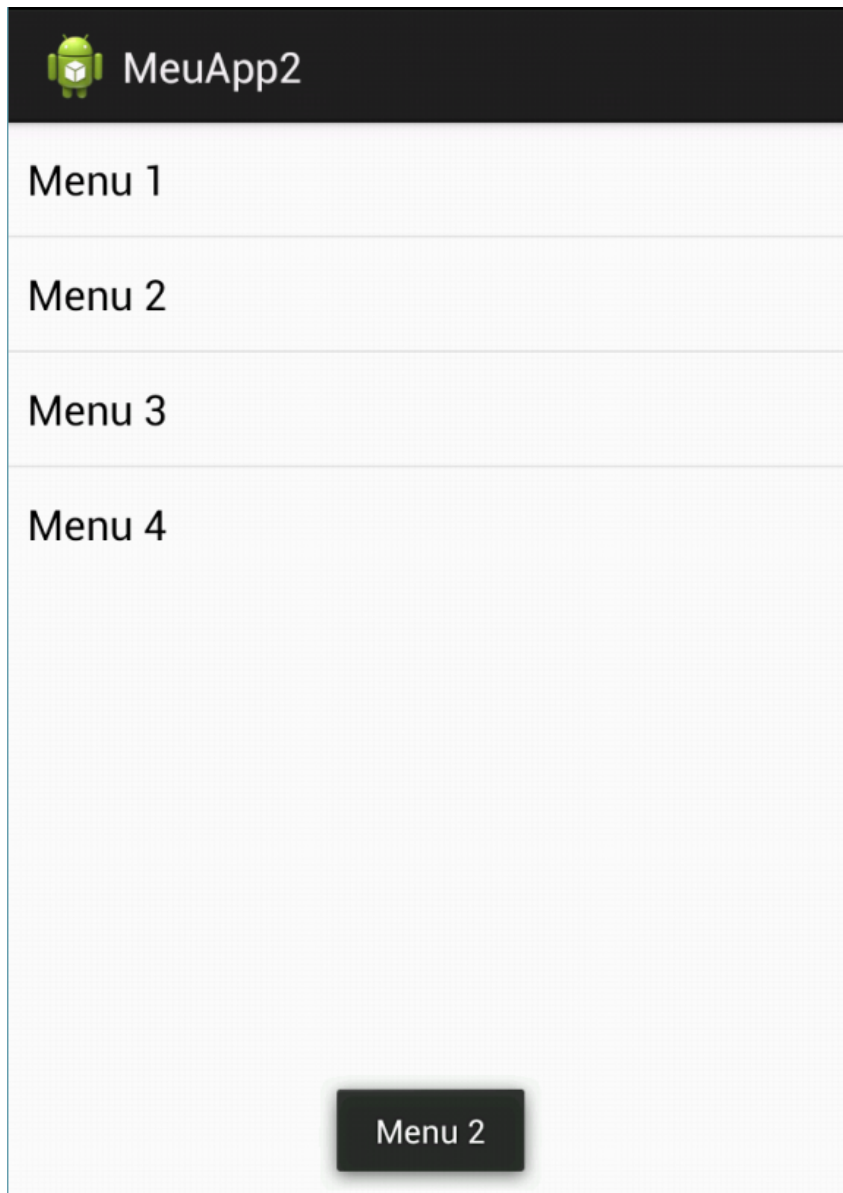


Figura 5.4: Lista simples

Obs.: O `Toast` é esse pequeno retângulo preto com uma mensagem, ele aparece e desaparece rapidamente apenas para mostrar uma mensagem ao usuário.

5.4 Listas Compostas

É possível compor um item da lista colocando mais elementos no mesmo além de um texto. Para isso você precisa criar um novo arquivo XML que irá definir a customização de cada linha de uma `ListView`, defina um arquivo chamado `item.xml`.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3   xmlns:android="http://schemas.android.com/apk/res/android"
4   android:layout_width="wrap_content"
5   android:layout_height="wrap_content"
6   android:orientation="horizontal" >
7
8   <ImageView
9     android:id="@+id/userIcon"
10    android:layout_width="wrap_content"
11    android:layout_height="wrap_content"
12    android:layout_margin="8dp" >
13  </ImageView>
14
15  <LinearLayout
16    android:layout_width="fill_parent"
17    android:layout_height="wrap_content"
18    android:layout_marginBottom="5dp"
19    android:layout_marginTop="5dp"
20    android:orientation="vertical"
21    android:paddingLeft="0px"
22    android:paddingRight="5dp" >
23
24    <RelativeLayout
25      android:layout_width="fill_parent"
26      android:layout_height="wrap_content" >
27
28      <TextView
29        android:id="@+id/username"
30        android:layout_width="wrap_content"
31        android:layout_height="wrap_content"
32        android:layout_alignParentLeft="true"
33        android:textColor="#FFF38585"
34        android:textSize="15sp" >
35      </TextView>
36    </RelativeLayout>
37
38    <TextView
39      android:id="@+id/usertext"
40      android:layout_width="wrap_content"
41      android:layout_height="wrap_content"
42      android:layout_marginTop="4dp"
43      android:textColor="#FF444444"
44      android:textSize="13sp" >
45    </TextView>
46  </LinearLayout>
47
48 </LinearLayout>
```

Agora você precisa usar um adaptador para mostrar esse layout customizado em cada linha da lista, uma maneira fácil é usando a classe `SimpleAdapter`.

```
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 18: Código da lista customizada

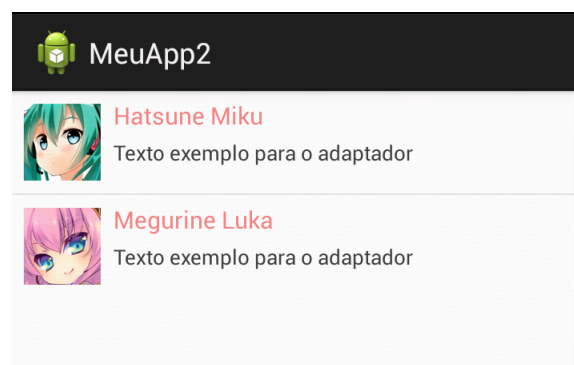


Figura 5.5: Lista Composta

5.5 Listas expandíveis (ExpandableListView)

Listas expandíveis são úteis para agrupar conjuntos de itens semelhantes, funcionam da mesma maneira que as listas comuns e podem ser customizadas. Comece colocando sua lista na *activity* desejada.

```

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     <ExpandableListView
7         android:id="@+id/listaExpandivel"
8         android:layout_width="match_parent"
9         android:layout_height="wrap_content"
10        android:transcriptMode="alwaysScroll"
11        android:listSelector="@android:color/transparent">
12    </ExpandableListView>
13
14 </LinearLayout>

```

Algoritmo 19: Lista expandível no activity_main.xml

O atributo `android:transcriptMode="alwaysScroll"` vai fazer com que a lista sempre role até o final quando você expande ou contrai um grupo.

Agora crie 2 novos arquivos XML, um chamado `list_item_parent.xml` e o outro chamado `list-item-child.xml`

```

1 <LinearLayout
2     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     android:id="@+id/list_item">
7
8     <TextView
9         android:layout_width="0dp"
10        android:layout_height="wrap_content"
11        android:id="@+id/list_item_text_view"
12        android:textSize="20sp"
13        android:padding="10dp"
14        android:layout_weight="1"
15        android:layout_marginLeft="35dp" />
16
17 </LinearLayout>

```

Algoritmo 20: Layout list_item_parent.xml

```

1 <LinearLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:id="@+id/list_item_child"
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 21: Layout list_item_child.xml

Agora crie uma classe para o pai dos grupos chamada *Parent*, que será responsável apenas por conter os dados do pai e os objetos filhos.

```

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 22: Classe Parent

Crie uma nova classe, CustomAdapter que será o adaptador da lista expandível para os dados, para esse exemplo estaremos adaptando apenas para o uso de *strings*. Essa classe deve estender a classe BaseExpandableListAdapter

```
1 public class CustomAdapter extends BaseExpandableListAdapter {
2     private LayoutInflater inflater;
3     private ArrayList<Parent> parent;
4
5     public CustomAdapter(Context context, ArrayList<Parent> parent){
6         this.parent = parent;
7         inflater = LayoutInflater.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 23: Classe CustomAdapter

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.

```
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 24: Construindo a lista expandível na *activity*

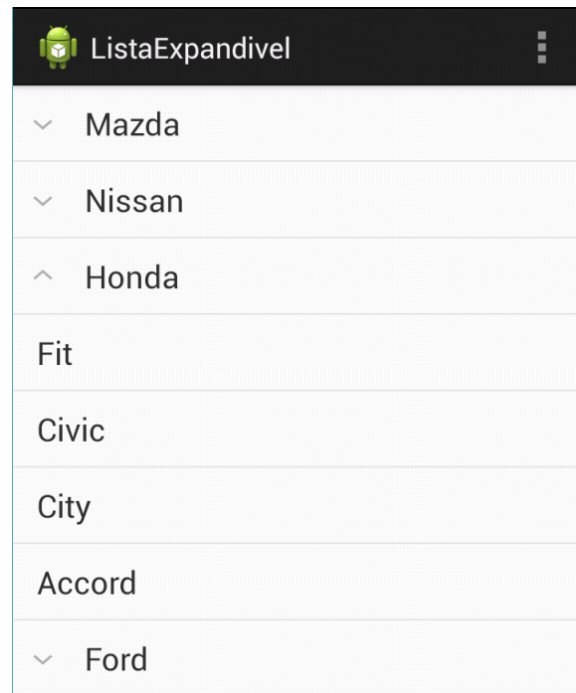


Figura 5.6: Exemplo de lista expandível

5.6 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.

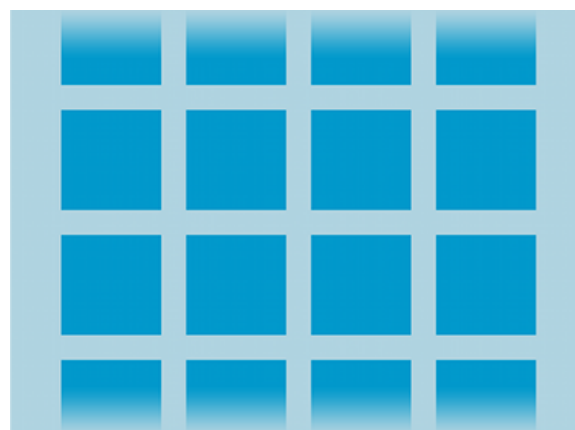


Figura 5.7: Esquema de um Grid view

Comece por simplesmente colocando um layout grid view em sua activity.

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

Algoritmo 25: Layout do Grid View

Crie uma nova classe que vai ser o adaptador para mostrar a imagem na grade, lembre-se que você pode adaptar qualquer layout para uma grade. Chame essa classe de `ImageAdapter`

```
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, ViewGroup parent) {
37        ImageView imageView = new ImageView(mContext);
38        imageView.setImageResource(thumbIds[position]);
39        imageView.setLayoutParams(new GridView.LayoutParams(200,200));
40        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
41        return imageView;
42    }
43 }
```

Algoritmo 26: Classe `ImageAdapter`

Agora basta criar na grade na 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 27: *activity* com grade

Exemplo rodando:

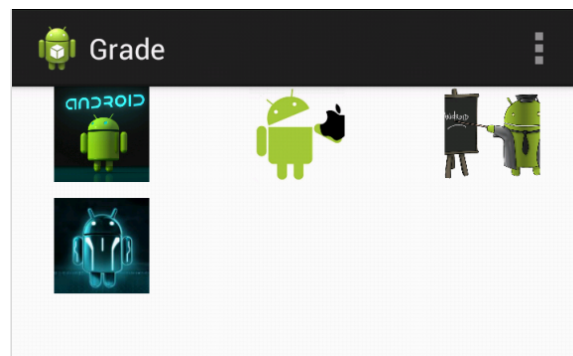


Figura 5.8: Demonstração de um Grid view

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 a nova *activity*. Crie um novo XML e chame de `full_image.xml`, nele teremos apenas um `ImageView`.

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical" >
5
6     <ImageView
7         android:id="@+id/full_image_view"
8         android:layout_width="fill_parent"
9         android:layout_height="fill_parent" >
10
11     </ImageView>
12
13     <TextView
14         android:id="@+id/myImageViewText"
15         android:layout_width="fill_parent"
16         android:layout_height="40dp"
17         android:layout_alignLeft="@+id/full_image_view"
18         android:layout_alignRight="@+id/full_image_view"
19         android:layout_alignTop="@+id/full_image_view"
20         android:gravity="center"
21         android:background="#55555555"
22         android:textSize="16sp"
23         android:textColor="#FFFFFF" />
24
25 </LinearLayout>
```

Algoritmo 28: Layout full_image.xml

Em seguida, crie uma nova classe chamada FullImageActivity, essa é *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
3     public void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.full_image);
6
7         //Obtem os dados do intent
8         Intent intent = getIntent();
9
10        //Seleciona o id da imagem
11        int id = intent.getExtras().getInt("id");
12        ImageAdapter imageAdapter = new ImageAdapter(this);
13
14        //Configura o ImageView para mostrar a imagem correspondente
15        ImageView imageView = (ImageView) findViewById(R.id.full_image_view);
16        imageView.setImageResource(imageAdapter.thumbIds[id]);
17
18        //Configura o TextView para mostrar uma descricao da imagem
19        TextView textView = (TextView) findViewById(R.id.myImageViewText);
20        textView.setText("Image id: " + id);
21    }
22 }
```

Algoritmo 29: Classe FullImageActivity

Finalmente, você deve modificar a *activity* da sua grade para suportar o novo comportamento, para isso é necessário configurar o *gridview* para tomar algumas ações quando um elemento da tela for clicado. Observe o código que deve ser adicionado da sua *activity*.

```

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      // Instance of ImageAdapter Class
7      gridView.setAdapter(new ImageAdapter(this));
8
9      //Parte 2
10     //Cria um listener para o evento de clique em um elemento da grade
11     gridView.setOnItemClickListener(new OnItemClickListener() {
12
13         @Override
14         public void onItemClick(AdapterView<?> parent, View view, int pos,
15             long id) {
16
17             //Envia o id da imagem para o FullImageActivity
18             Intent intent = new Intent(getApplicationContext(), FullImageActivity.class);
19             intent.putExtra("id", pos);
20             startActivity(intent);
21         }
22     });
23 }

```

Algoritmo 30: *activity* com grade, melhorado

Na linha 14 você pode observar a criação de um *listener* para os itens clicados da grade, o método `onItemClick()` permite obter algumas informações sobre o item clicado e você pode usar essas informações como quiser, nesse caso é colocado dentro de um `intent` o *id* do item clicado como sendo sua posição nessa grade. Finalmente o método `startActivity` é chamado e passando o `intent` criado como parâmetro.

5.7 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 31: 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
12 </fragment>
13 </LinearLayout>

```

Algoritmo 32: 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-element³](#)

Novamente, caso esteja desenvolvendo para APIs menores que 11, você vai precisar fazer a *activity* estender 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 33: Layout da *activity* com o `FrameLayout`

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

³<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 obtem
12        //qualquer outra view usando o FragmentManager
13        Fragment fragment = fm.findFragmentById(R.id.fragment_content);
14
15        if(fragment == null){
16            //Comeca uma transacao de fragmentos
17            FragmentTransaction ft = fm.beginTransaction();
18            //Adiciona o fragmento
19            ft.add(R.id.fragment_content, new BasicFragment());
20            //"Commita" a transacao
21            ft.commit();
22        }
23    }
24    ...
```

Algoritmo 34: *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.

5.8 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`.

5.8.1 Usando `TabHost` e `TabWidget`

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

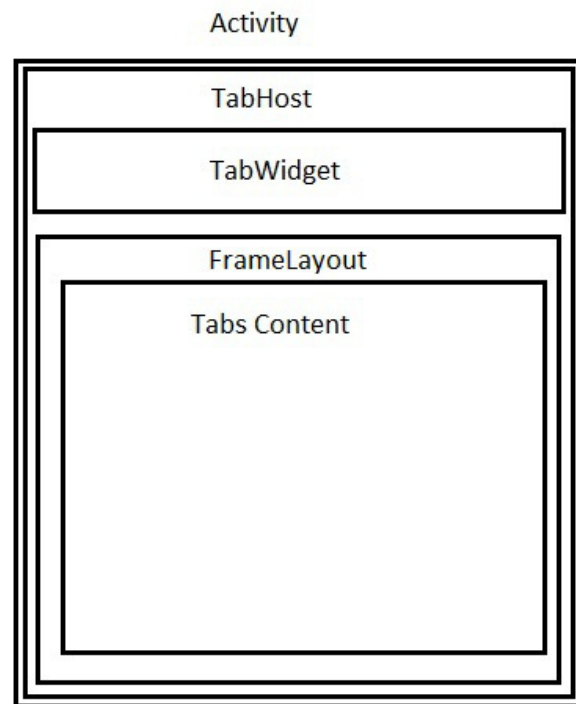


Figura 5.9: Esquema da interface com abas

5.9 Arrastar (SwipeView) com abas

5.10 Menu lateral

Especificação blá, blá, blá

Isto é um apêndice...