

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	3
3 Linguagem do Android	5
3.1 Linguagem	5
3.2 Entendendo os arquivos XML	6
4 Criando seu primeiro aplicativo	7
5 Design	17
5.1 Listas (ListView)	17
5.2 Listas Compostas	20
5.3 Listas expandíveis (ExpandableListView)	23
5.4 Grades (GridView) e imagens ImageView	28
5.5 Fragmentos	33
5.6 Abas (Tabs)	36
5.7 Arrastar (SwipeView) com abas	37
5.8 Menu lateral	37
A Especificação blá, blá, blá	39

Lista de Figuras

1.1	Departamento de Computação	1
4.1	Primeira janela de criação de novo aplicativo	8
4.2	Segunda janela de criação de novo aplicativo	8
4.3	Selecionando o Hello world	9
4.4	<code>activity</code> com os elementos colocados na tela	10
4.5	Criando uma nova <code>activity</code>	13
4.6	Primeira tela do primeiro aplicativo	16
4.7	Primeira tela após escrever texto na caixa de texto	16
4.8	Segunda tela mostrando a mensagem enviada	16
5.1	Esquema de uma lista	17
5.2	Detalhes de um elemento da lista	17
5.3	Lista simples	20
5.4	Lista Composta	22
5.5	Exemplo de lista expandível	28
5.6	Esquema de um <code>Grid view</code>	28
5.7	Demonstração de um <code>Grid view</code>	31
5.8	Esquema da interface com abas	37

Lista de Tabelas

Lista de Algoritmos

1

- de configuração de versão do SDK no `AndroidManifest.xml` 9 2
- dos *intent-filters* no `AndroidManifest.xml` 9 3
- da caixa de texto 10 4
- do botão 11 5
- de strings com as duas strings adicionadas 11 6
- método à classe MainActivity 11 7
- de import de classe Android 11 8
- uma `Intent` 12 9
- o conteúdo da caixa de texto e enviando para outra `activity` 12 10
- como chave para um extra 12 11
- extras passados através do `Intent` 13 12
- `onCreate()` recebendo um `Intent` e mostrando a mensagem 14 13
- Linear* no `activity_main.xml` 18 14
- XML de um `ListView` 18 15
- populada com elementos 18 16
- de uma `activity` com lista clicável 19 17
- do arquivo `item.xml` 21 18
- da lista customizada 22 19
- expandível no `activity_main.xml` 23 20
- `list_item_parent.xml` 23 21
- `list_item_child.xml` 24 22
- `Parent` 24 23
- `CustomAdapter` 26 24
- a lista expandível na `activity` 27 25
- do `Grid View` 29 26
- `ImageAdapter` 30 27
- com `grade` 31 28

`full_image.xml`31 29
`FullImageActivity`32 30
com grade, melhorado33 31
`BasicFragment`34 32
da `activity` com um fragmento35 33
da `activity` com o `FrameLayout`35 34
com adição dinâmica de fragmento36

Lista de Abreviaturas

Resumo

Esse material didático oferece ao aluno de graduação 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



Figura 1.1: Departamento de Computação

**** Não sei o que escrever na introdução que eu não tenha escrito no resumo ****

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.

¹<http://developer.android.com/sdk/>

Linguagem do Android

3.1 Linguagem

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

A diferença entre o Java convencional e o Android então se dá na organização e configuração feita através de arquivos XML específicos do Android. Alguns XML servem para configurar o aplicativo, layout de cada tela, outro dá suporte a strings para facilitar o suporte a múltiplos idiomas. Felizmente o conjunto Eclipse com ADT já cuida disso automaticamente ou então facilita através de interfaces gráficas para os programadores, então 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 melhor estilo *drag and drop*.

3.2 Entendendo os arquivos XML

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

Dentro da pasta `res/` de recursos, encontram-se outros arquivos, estes que são referentes a disposição do layout e a valores de strings. A pasta `layout/` junto com as pastas `drawable-*/` são para dispor o layout, cada *drawable* é para comportar artefatos para cada tamanho diferente de tela, enquanto que a *layout* é a disposição geral do layout, nesse arquivo é que se coloca os itens 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 se escreve os valores das strings do aplicativo, sempre que quiser referenciar alguma string, essa deverá estar expressa nesse arquivo. Fica fácil dessa forma fazer o aplicativo suportar múltiplos idiomas através da tradução desse conteúdo para outros idiomas.

Tudo isso será trabalhado a partir de exemplos mais pra frente no material.

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). Um aplicativo simples do tipo "Hello World".

Iniciaremos criando um novo projeto no Eclipse: *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 para qual seu aplicativo está sendo desenvolvido
- *Compile With*: Versão para qual seu aplicativo será compilado
- *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>

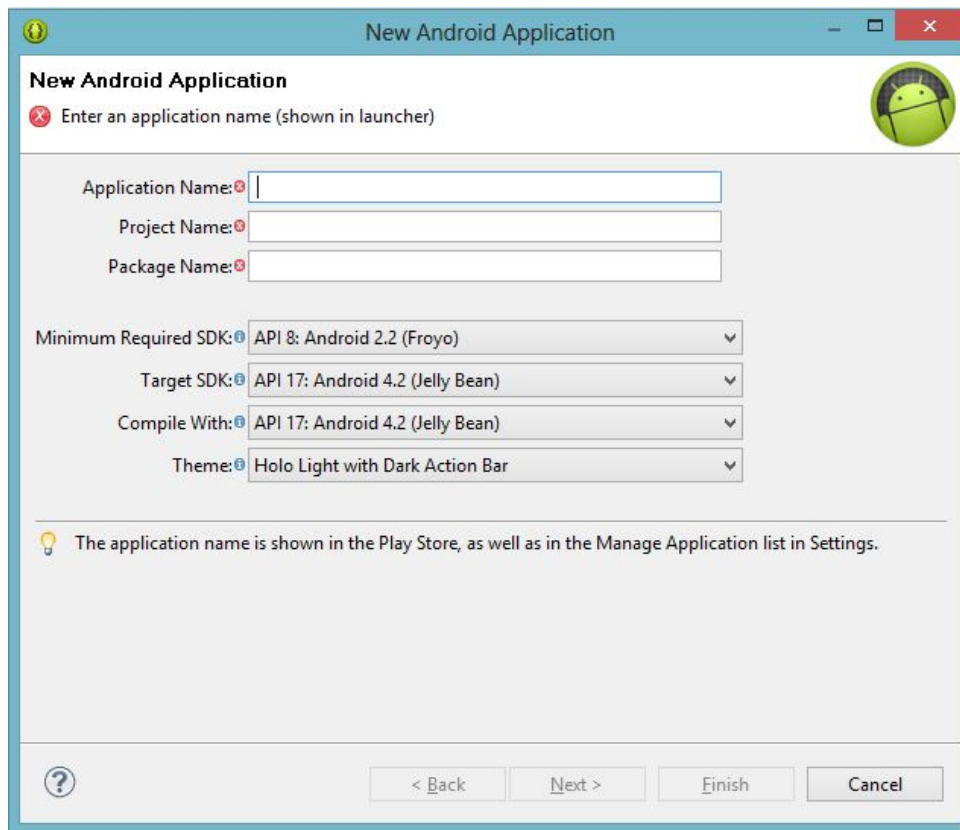


Figura 4.1: Primeira janela de criação de novo aplicativo

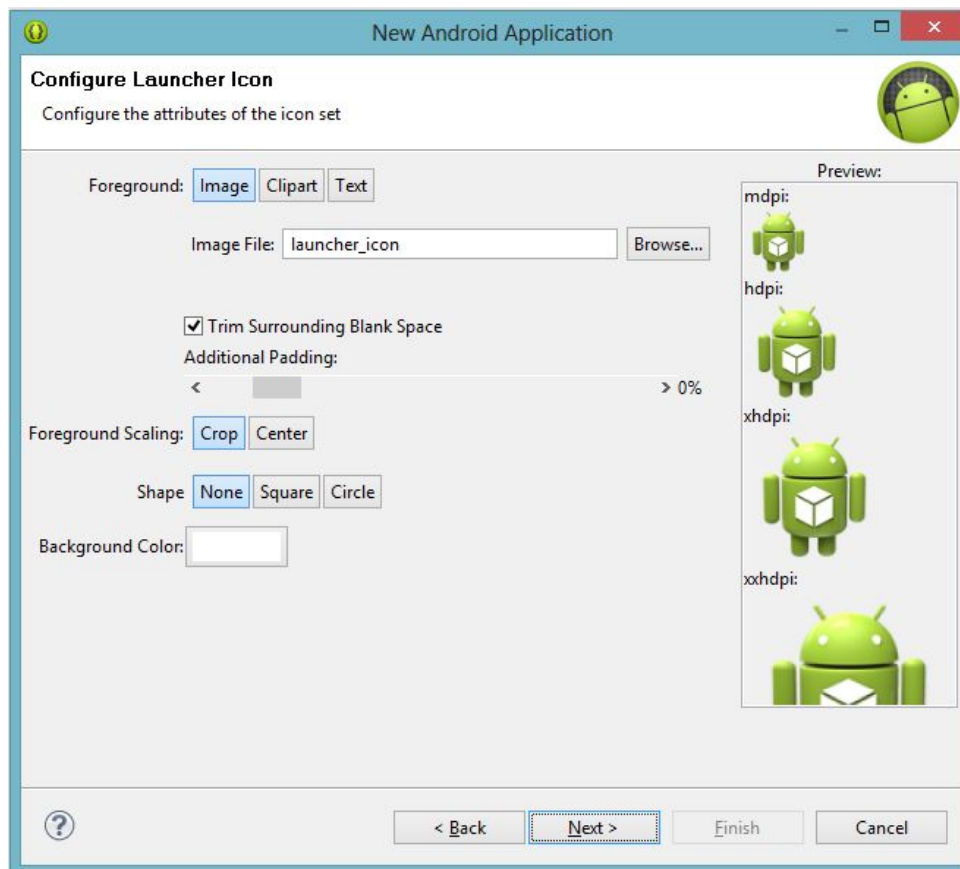


Figura 4.2: Segunda janela de criação de novo aplicativo

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

Listing 1: Exemplo de configuração de versão do SDK no `AndroidManifest.xml`

Primeiro vamos criar um layout para o aplicativo usando a interface gráfica do Eclipse, primeiro abra o arquivo `res/layout/activity_main.xml`, segundo o `AndroidManifest`, é essa `activity` que será aberta quando o aplicativo for iniciado, configurado através do *intent-filter*. Cada `activity` é uma tela do aplicativo e é responsável primariamente por mostrar as informações, o Android segue o modelo *MVC (Model-View-Control)*, e as `activity` são as *Views* do aplicativo.

```
1 <intent-filter>  
2     <action android:name="android.intent.action.MAIN" />  
3     <category android:name="android.intent.category.LAUNCHER" />  
4 </intent-filter>
```

Listing 2: Configuração dos *intent-filters* no `AndroidManifest.xml`

Não entraremos em detalhes sobre os *intent-filters* agora. Selecione o "Hello world" e delete ele da sua `activity`

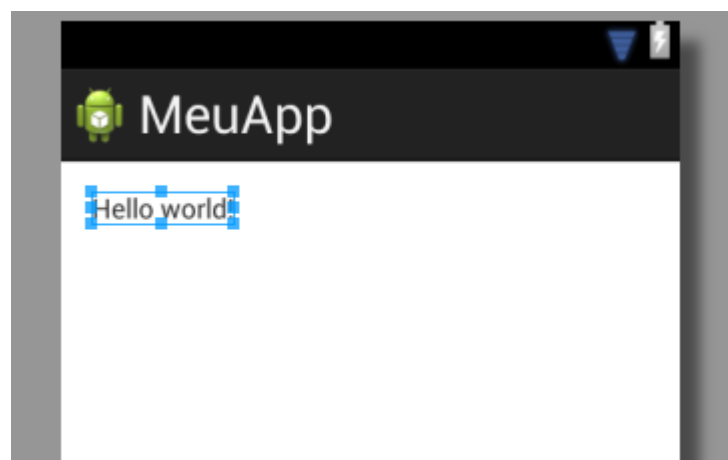


Figura 4.3: Selecionando o Hello world

Agora arraste um *Text Field -> Plain Text* e um *Form Widgets -> Button* para sua `activity`

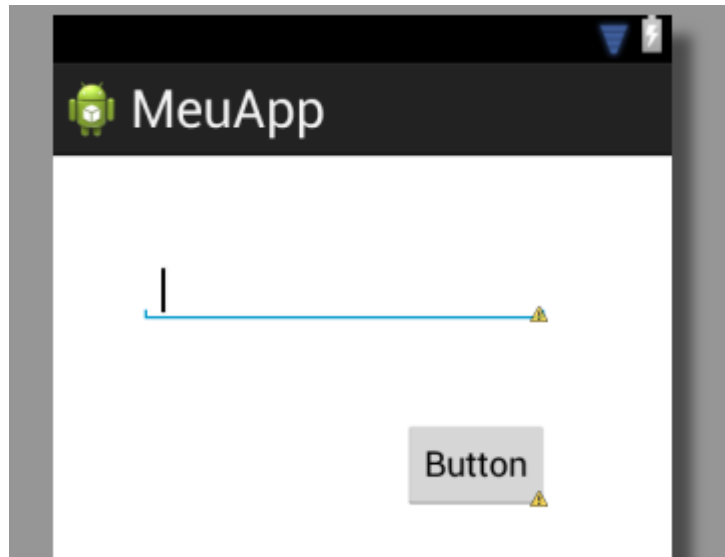


Figura 4.4: activity com os elementos colocados na tela

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

```
1 <EditText
2   android:id="@+id/caixaTextoNome"
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/nome">
11
12  <requestFocus />
13 </EditText>
```

Listing 3: Código da caixa de texto

Primeiro, modifique a *id* para um mais intuitivo, por exemplo você pode chama-lo de *caixaTextoNome*. Todo *id* deve ser precedido de *@+id/* como definido pelo Android. Os outros atributos são para definir o tamanho, alinhamento e margem, todos relativos ao elemento pai desse elemento que no caso é o próprio layout. 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, essa *hint* é uma referência a string chamada *nome* que está definida no arquivo *res/values/strings.xml*. No código acima a *hint* já foi colocada na linha 10.

Depois modifique o código do botão, trocando o *id* e as referências ao *id* da caixa de texto, também edite o *android:text* para fazer uma referência a uma string no arquivo de strings. Por último adicione um atributo *android:onClick* que é o método que será executado quando o botão for pressionado.

```

1 <Button
2     android:id="@+id/botaoEnviar"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content"
5     android:layout_alignRight="@+id/caixaTextoNome"
6     android:layout_below="@+id/caixaTextoNome"
7     android:layout_marginTop="48dp"
8     android:text="@string/botao_enviar"
9     android:onClick="enviarMensagem" />

```

Listing 4: Código do botão

Adicione as strings ao arquivo de strings, essas strings podem ser referenciadas a qualquer momento tanto na construção do layout como na codificação do aplicativo.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="app_name">MeuApp</string>
4     <string name="action_settings">Settings</string>
5     <string name="botao_enviar">Enviar</string>
6     <string name="nome">Nome</string>
7 </resources>

```

Listing 5: Arquivo de strings com as duas strings adicionadas

Agora abra a classe `MainActivity` localizada na pasta `src/` do seu projeto e adicione um novo método

```

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

```

Listing 6: Adicionando método à classe `MainActivity`

- 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;

```

Listing 7: Exemplo de import de classe `Android`

Primeiro, crie um novo `Intent`³, um `Intent` é um objeto que providencia uma ligação entre dois componentes separados (tais como duas *activities*). O `Intent` representa uma "intenção do aplicativo de fazer algo", eles podem ser usados para uma variedade de tarefas, mas são principalmente usados para iniciar uma nova *activity*.

³Documentação `Intent`: <http://developer.android.com/reference/android/content/Intent.html>

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

Listing 8: Adicionando uma Intent

Agora você precisa 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.

```
1 public void enviarMensagem(View view) {  
2     // Fazer alguma coisa em resposta ao clique do botao  
3     Intent intent = new Intent(this, DisplayMessageActivity.class);  
4     EditText caixaTexto = (EditText) findViewById(R.id.caixaTextoNome);  
5     String mensagem = caixaTexto.getText().toString();  
6     intent.putExtra(EXTRA_MESSAGE, mensagem);  
7     startActivity(intent);  
8 }
```

Listing 9: Obtendo o conteúdo da caixa de texto e enviando para outra *activity*

O código na linha 3 está obtendo a referência a caixa de texto usando o método `findViewById()` e o argumento passado é o *id* da caixa de texto, observe que você chama com `R.id.caixaTextoNome` isso é feito dessa forma porque o android compila automaticamente uma classe *R* (*R* de *Resources*) que contém as *id*'s e *strings* criadas nos arquivos XML. Em seguida usando o método `getText()` da caixa de texto, obtem-se a string que foi escrita lá.

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 }
```

Listing 10: 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 dessa maneira, depois clique *Finish*.

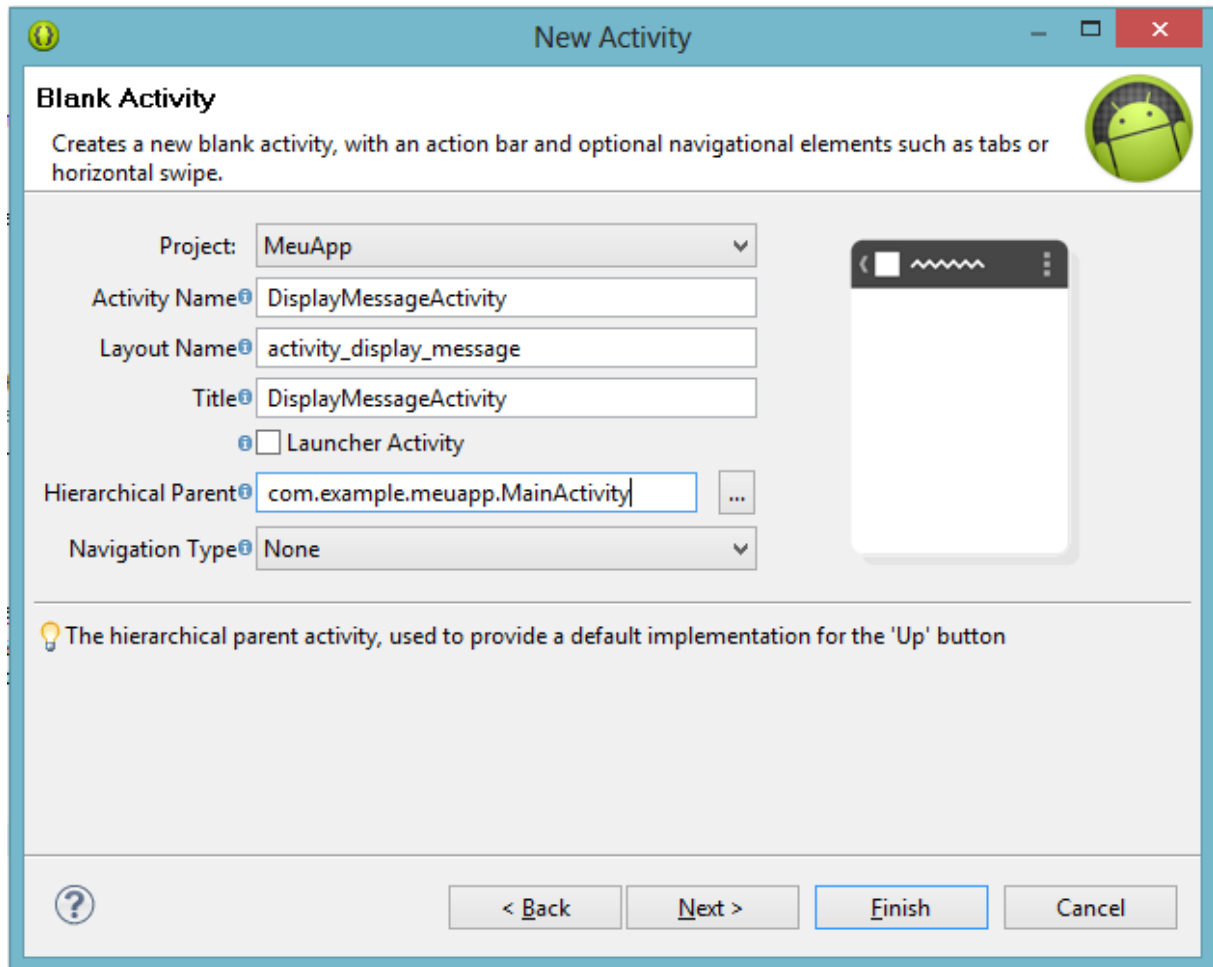


Figura 4.5: Criando uma nova activity

A classe já vem com alguns métodos implementados, alguns não serão necessários para esse aplicativo, mas mantenha eles na classe. Todas as classes que são subclasses de `Activity` precisam implementar o método `onCreate()`⁴.

Geralmente você precisaria adicionar uma nova string com o nome da classe no XML, e adicionar a activity no `AndroidManifest.xml` porém como estamos trabalhando com o Eclipse, você não precisa se preocupar que ele faz isso sozinho.

Agora, precisamos extrair os dados enviados a essa activity através do intent, você pode pegar o intent que começou a activity chamando o método `getIntent()`⁵.

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

Listing 11: Obtendo extras passados através do Intent

Agora para mostrar a mensagem na tela, você precisa criar um `TextView`⁶

⁴[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))

⁵[http://developer.android.com/reference/android/app/Activity.html#getIntent\(\)](http://developer.android.com/reference/android/app/Activity.html#getIntent())

⁶<http://developer.android.com/reference/android/widget/TextView.html>

```
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 }
```

Listing 12: Método `onCreate()` recebendo um `Intent` e mostrando a mensagem

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 emulador.

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)⁷
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*

⁷<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*
 - *ARM EABI v7a System Image* ou
 - *Intel x86 Atom System Image*
3. Verifique também se em *Tools*, os pacotes *Android SDK Tools* e *Android SDK Platform-tools* estão instalado
4. Agora é necessário criar um AVD (Android Virtual Device⁸), no Eclipse vá em *Window -> Android Virtual Device Manager*
5. No AVD Manager clique em *New*
6. Complete as informações do AVD, dê um nome, plataforma, espaço de armazenamento, quantidade de memória RAM
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.

⁸<http://developer.android.com/tools/devices/index.html>

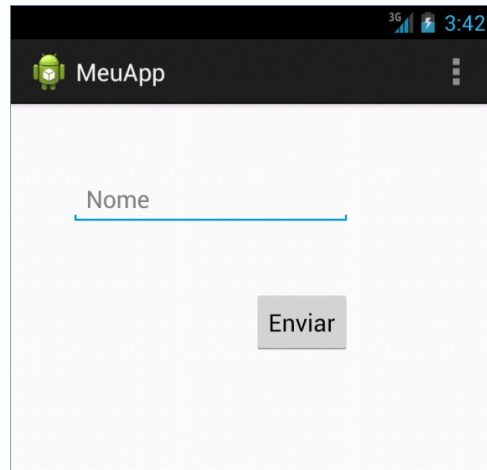


Figura 4.6: Primeira tela do primeiro aplicativo

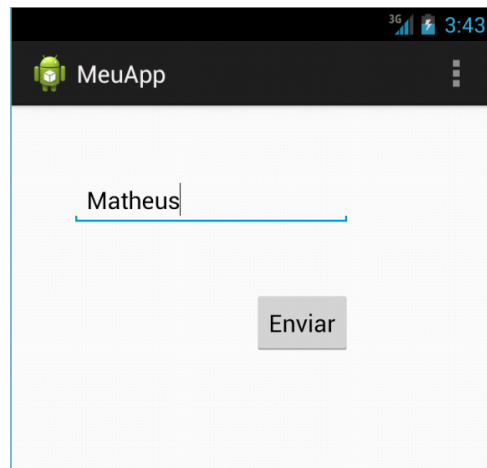


Figura 4.7: Primeira tela após escrever texto na caixa de texto

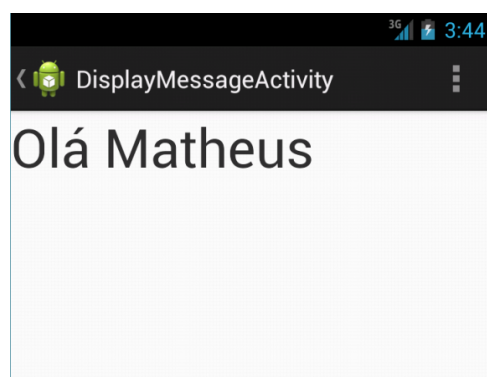


Figura 4.8: Segunda tela mostrando a mensagem enviada

Design

5.1 Listas (ListView)

¹ 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 5.1: Esquema de uma lista

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

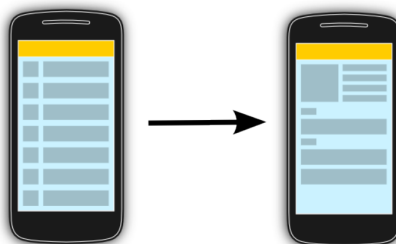


Figura 5.2: Detalhes de um elemento da lista

¹ Documentação `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>
```

Listing 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>
```

Listing 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>
```

Listing 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    ...
```

Listing 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"`.

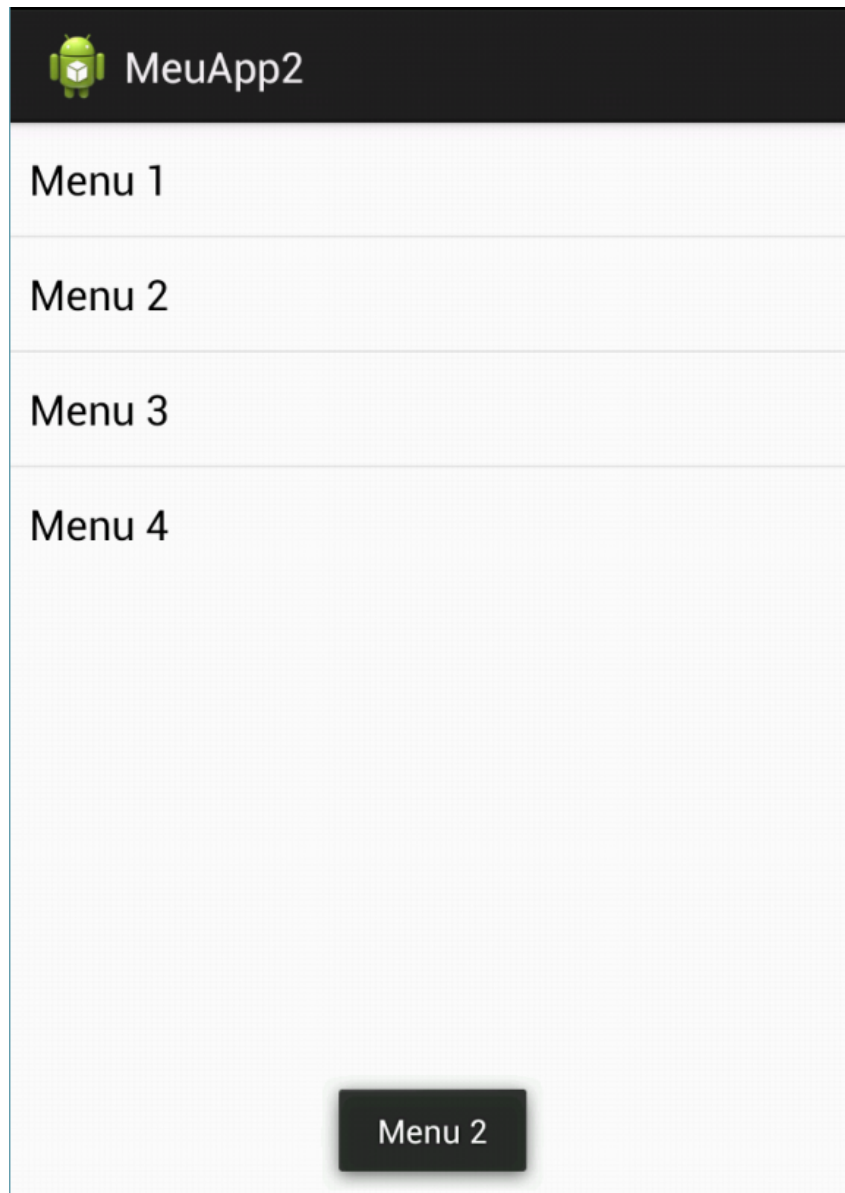


Figura 5.3: Lista simples

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

5.2 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>
```

Listing 17: Código do arquivo `item.xml`

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    }

```

Listing 18: Código da lista customizada

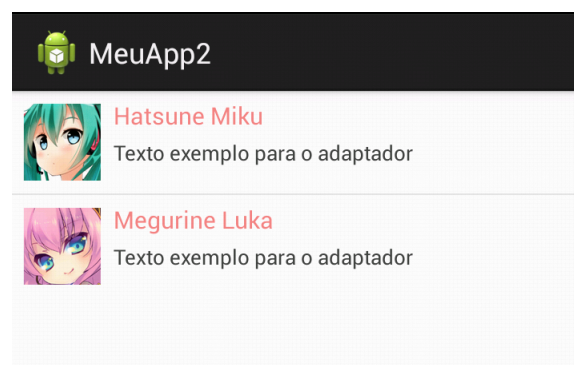


Figura 5.4: Lista Composta

5.3 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>
```

Listing 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>
```

Listing 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>
```

Listing 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 }
```

Listing 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  }

```

Listing 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 }
```

Listing 24: Construindo a lista expandível na activity

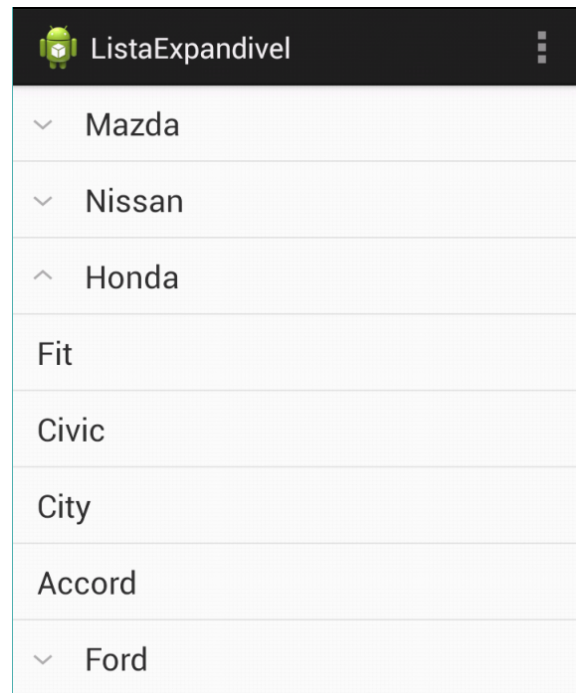


Figura 5.5: Exemplo de lista expandível

5.4 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 ideia 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.

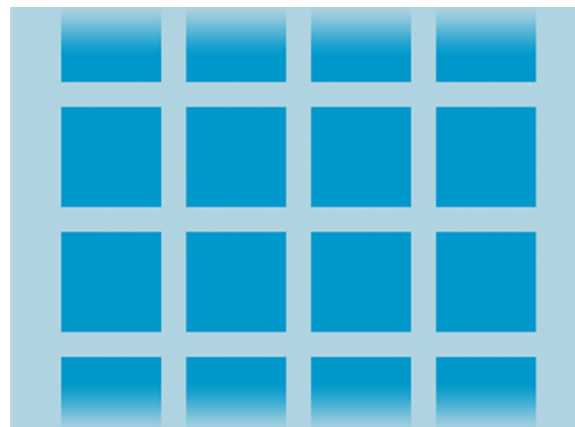


Figura 5.6: Esquema de um Grid view

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>
```

Listing 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.foto_1, R.drawable.foto_2,
7         R.drawable.foto_3, R.drawable.foto_4
8     };
9
10    //Construtor
11    public ImageAdapter(Context c) {
12        mContext = c;
13    }
14
15    @Override
16    //Retorna o tamanho do array
17    public int getCount() {
18        return thumbIds.length;
19    }
20
21    @Override
22    //Retorna um elemento do array
23    public Object getItem(int position) {
24        return thumbIds[position];
25    }
26
27    @Override
28    //Nao sera usado
29    public long getItemId(int position) {
30        return 0;
31    }
32
33    @Override
34    public View getView(int position, View convertView, ViewGroup parent) {
35        ImageView imageView = new ImageView(mContext);
36        imageView.setImageResource(thumbIds[position]);
37        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
38        imageView.setLayoutParams(new GridView.LayoutParams(70,70));
39        return imageView;
40    }
41 }
```

Listing 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 }

```

Listing 27: Activity com grade

Exemplo rodando:

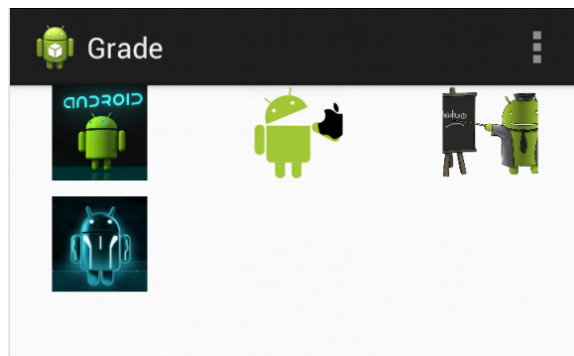


Figura 5.7: Demonstração de um Grid view

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 android:id="@+id/full_image_view"
7         android:layout_width="fill_parent"
8         android:layout_height="fill_parent"/>
9
10 </LinearLayout>

```

Listing 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        ImageView imageView = (ImageView) findViewById(R.id.full_image_view);
15        imageView.setImageResource(imageAdapter.thumbIds[id]);
16    }
17 }
```

Listing 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 }
```

Listing 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.

5.5 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 }
```

Listing 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 </fragment>
12 </LinearLayout>
```

Listing 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-element²](#)

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>
```

Listing 33: Layout da activity com o `FrameLayout`

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

²<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    ...
```

Listing 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.

5.6 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`.

5.6.1 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:

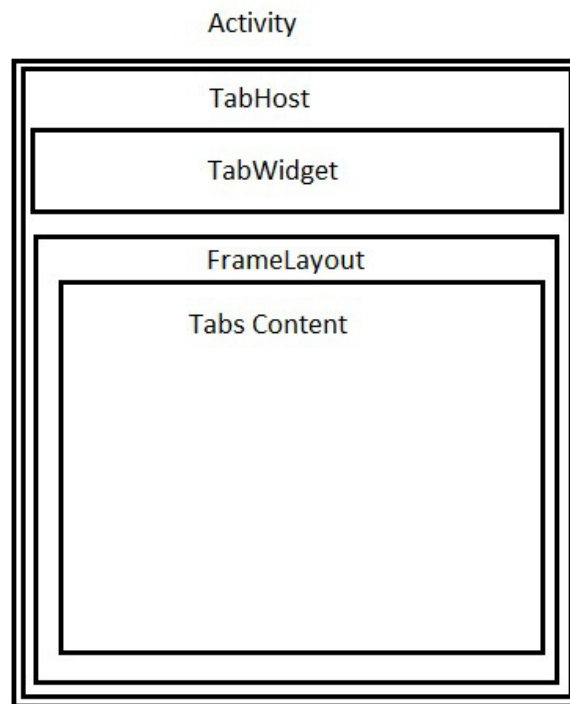


Figura 5.8: Esquema da interface com abas

5.7 Arrastar (SwipeView) com abas

5.8 Menu lateral

APÊNDICE

A

Especificação blá, blá, blá

Isto é um apêndice...