Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

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.

CAPÍTULO

1

Introdução

Figura 1.1: Departamento de Computação

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

CAPÍTULO

2

Configuração do Ambiente

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

CAPÍTULO

3

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 especificos do Android. Alguns XML servem para configurar o aplicativo, layout de cada tela, outro dá suporte a strings para facilitar o suporte a multiplos 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 e recomendável a utilizacão 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 dispoção 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.

CAPÍTULO

4

Criando seu primeiro aplicativo

Para exemplificar a criação de um aplicativo, seguiremos o exemplo dado pelo próprio manual do Google sobre o Android (Ver original[1]). 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 Java[2].

- *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

[1] Original em: http://developer.android.com/training/basics/firstapp/creating-project.html
[2] Convenção sobre nome dos pacotes: http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html

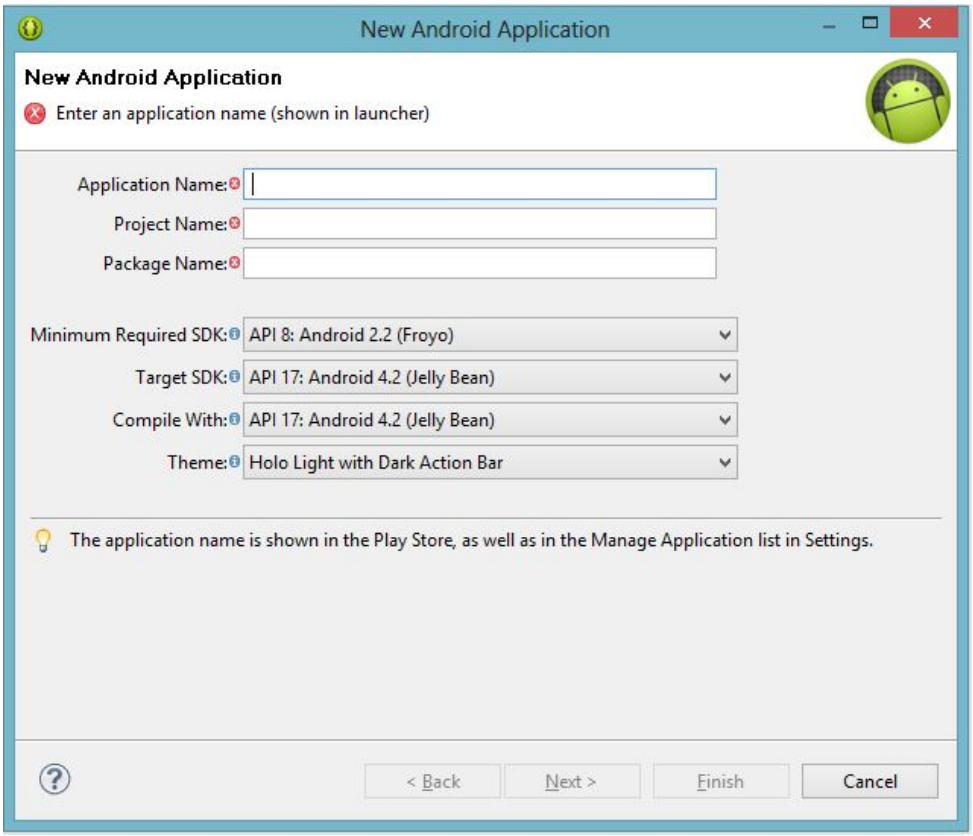

Figura 4.1: Primeira janela de criação de novo aplicativo

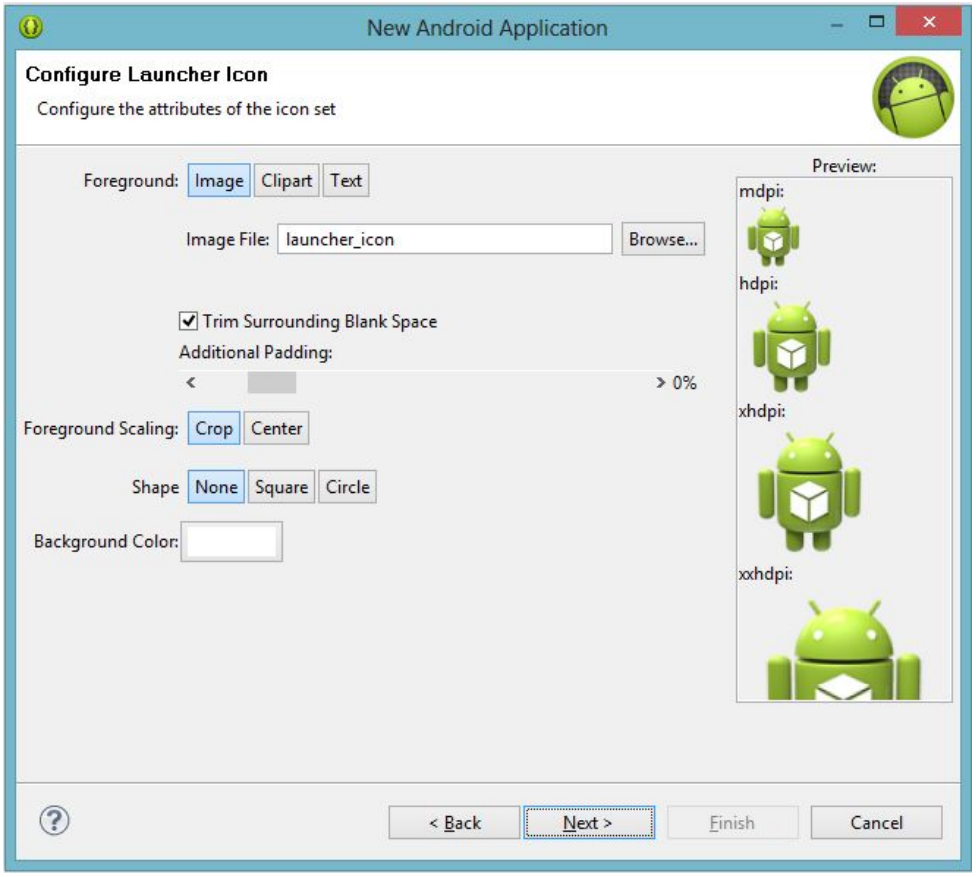

Figura 4.2: Segunda janela de criação de novo aplicativo

```
<uses-sdk
    android:minSdkVersion="8"
    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 responsáve primariamente por mostrar as informações, o Android segue o modelo *MVC (Model-View-Control)*, e as `activity` são as *Views* do aplicativo.

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

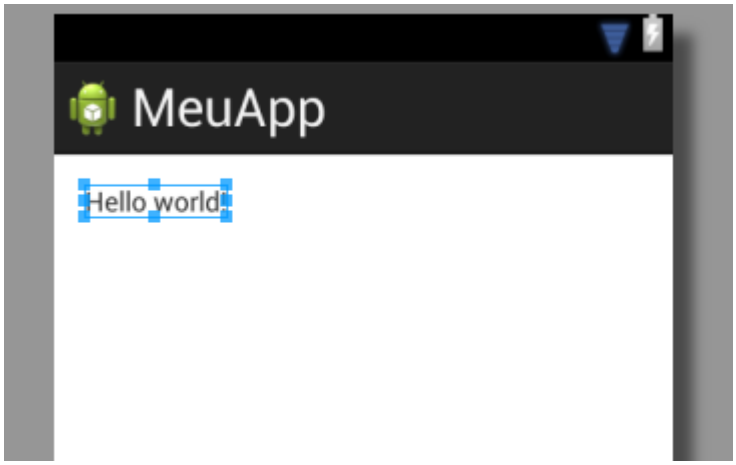

Figura 4.3: Selecionando o Hello world

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

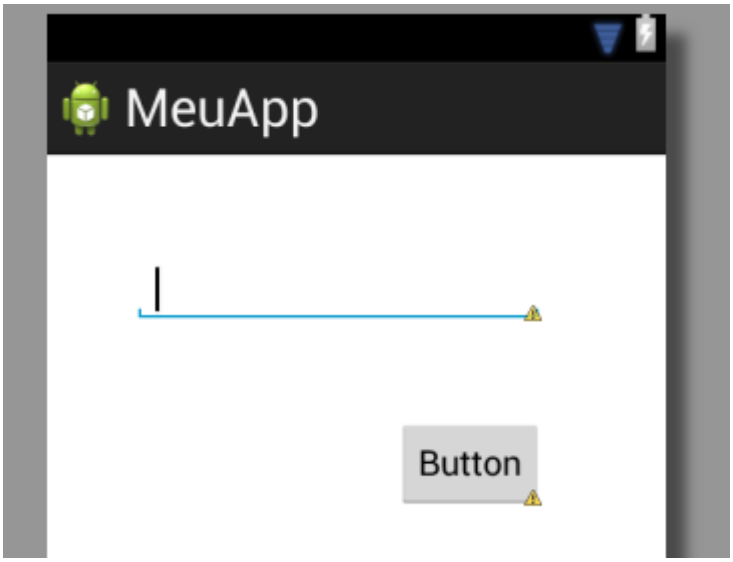

Figura 4.4: `activity` com os elementos colocados na tela

Ao clicar duas vezes no elemento no modo visual, você sera 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.

```
<Button
  android:id="@+id/botaoEnviar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignRight="@+id/caixaTextoNome"
  android:layout_below="@+id/caixaTextoNome"
  android:layout_marginTop="48dp"
  android:text="@string/botao_enviar"
  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.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MeuApp</string>
    <string name="action_settings">Settings</string>
    <string name="botao_enviar">Enviar</string>
    <string name="nome">Nome</string>
</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

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

Listing 6: Adicionando método à classe MainActivity

- Isso vai requer você importe a classe View, você pode apertar `Ctrl+Shit+O` no Eclipse para importar classes que estejam faltando

```
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`.

[3]Documentação 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*.

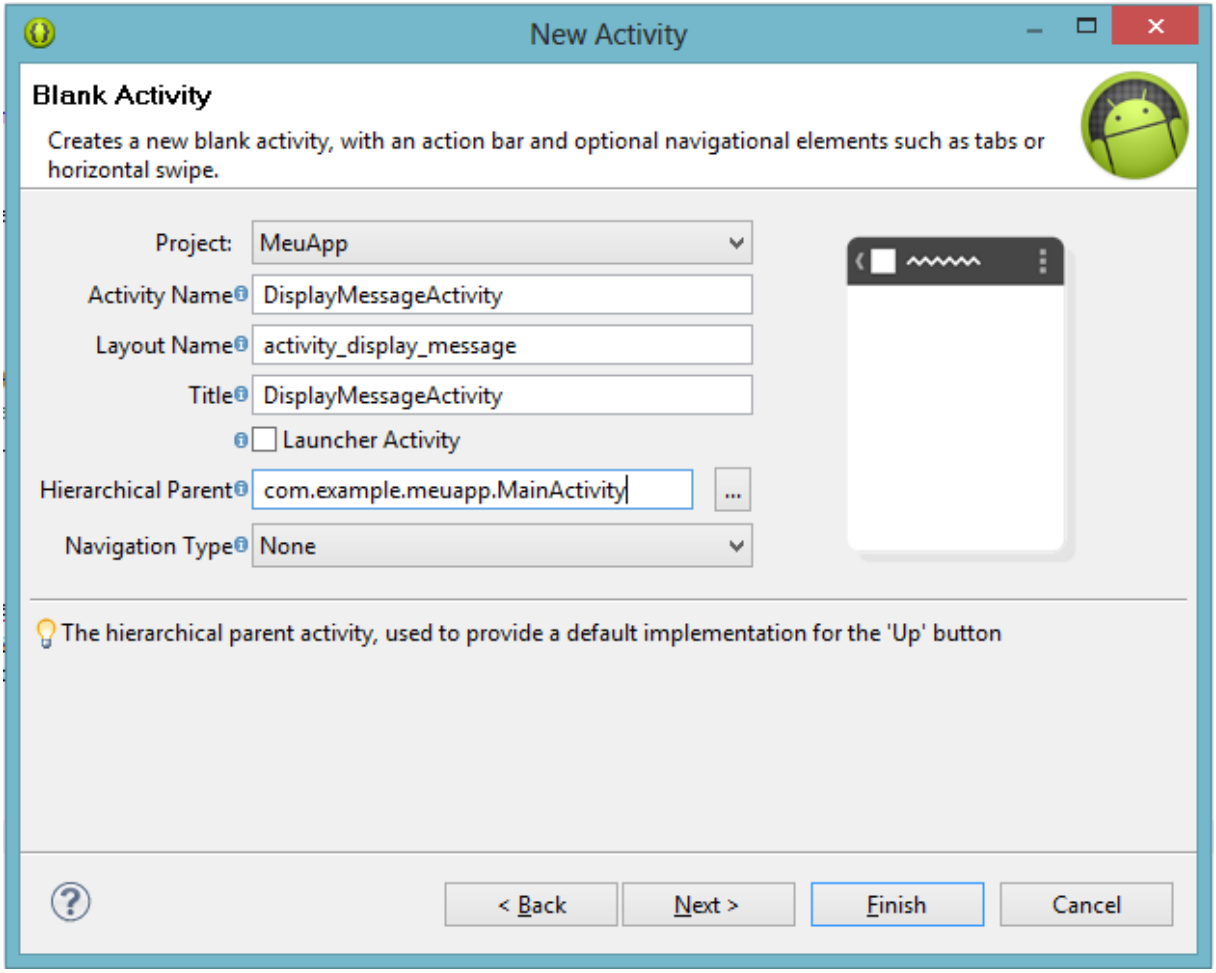

Figura 4.5: Criando uma nova `activity`

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

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

  // Show the Up button in the action bar.
  setupActionBar();

  //Obtem o conteudo da Intent
  Intent intent = getIntent();
  String mensagem = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

  //Cria o TextView
  TextView textView = new TextView(this);
  textView.setTextSize(40);
  textView.setText("Hello " + mensagem);

  //Estabelece o text view como o layout da atividade
  setContentView(textView);
}
```

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[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 Andoird 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 Device[8]), 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

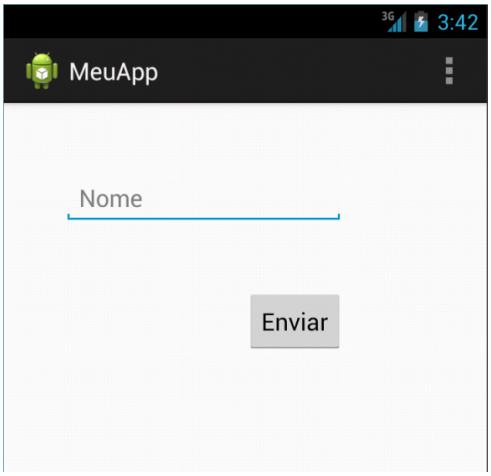

Figura 4.6: Primeira tela do primeiro aplicativo

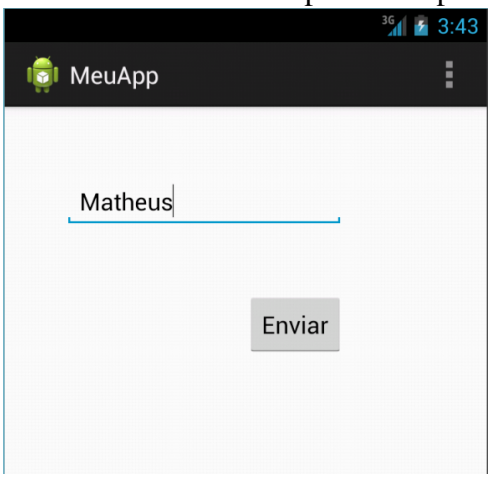

Figura 4.7: Primeira tela após escrever texto na caixa de texto

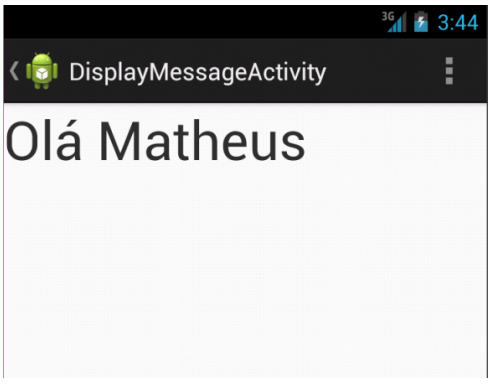

Figura 4.8: Segunda tela mostrando a mensagem enviada

CAPÍTULO

5

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.

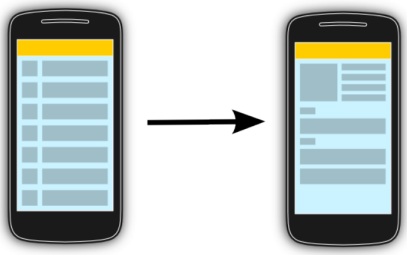

Figura 5.2: Detalhes de um elemento da lista

[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.

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

```
<ListView
  android:id="@+id/listView1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >
</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.

```
<string-array name="listString">
  <item>Menu 1</item>
  <item>Menu 2</item>
  <item>Menu 3</item>
  <item>Menu 4</item>
</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 extende 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 `ListViews`. 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"`.

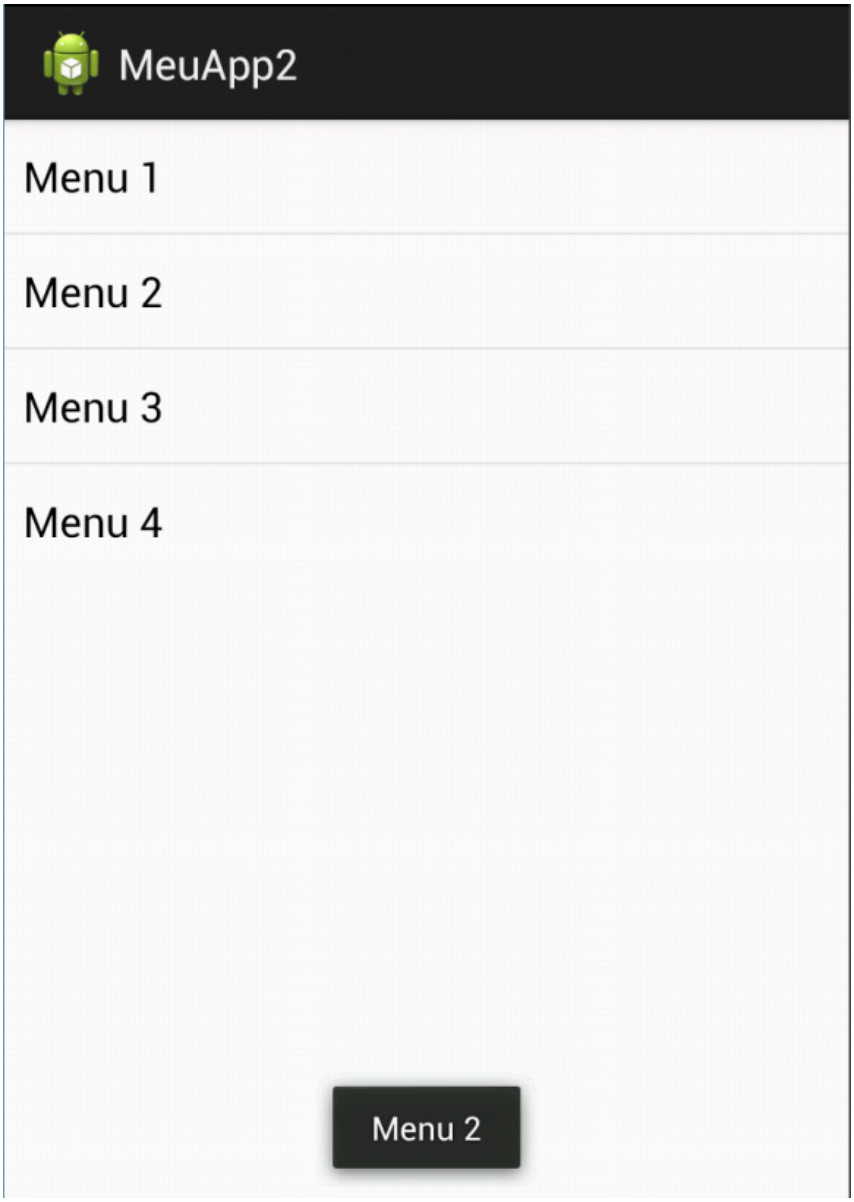

Figura 5.3: Lista simples

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`.

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

  <ImageView
   android:id="@+id/userIcon"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="8dp" >
   </ImageView>

  <LinearLayout
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_marginBottom="5dp"
   android:layout_marginTop="5dp"
   android:orientation="vertical"
   android:paddingLeft="0px"
   android:paddingRight="5dp" >

    <RelativeLayout
     android:layout_width="fill_parent"
     android:layout_height="wrap_content" >

      <TextView
       android:id="@+id/username"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_alignParentLeft="true"
       android:textColor="#FFF38585"
       android:textSize="15sp" >
       </TextView>
    </RelativeLayout>

    <TextView
     android:id="@+id/usertext"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginTop="4dp"
     android:textColor="#FF444444"
     android:textSize="13sp" >
    </TextView>
  </LinearLayout>

</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

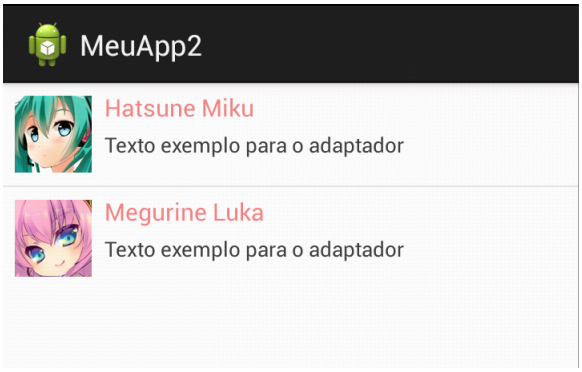

Figura 5.4: Lista Composta

5.3 Listas expandíveis (`ExpandableListView`)

5.4 Grades (`GridView`)

5.5 Abas (`TabView`)

5.6 Arrastar (`Swipe View`)

5.7 Menu lateral

APÊNDICE

$\mathcal{A}$

Especificação blá, blá, blá

Isto é um apêndice...