Sumário

Lista de Figuras

Lista de Tabelas

Listings

Lista de Abreviaturas

Resumo

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

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

CAPÍTULO

1

Introdução

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

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

Atualmente[1] 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]

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

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

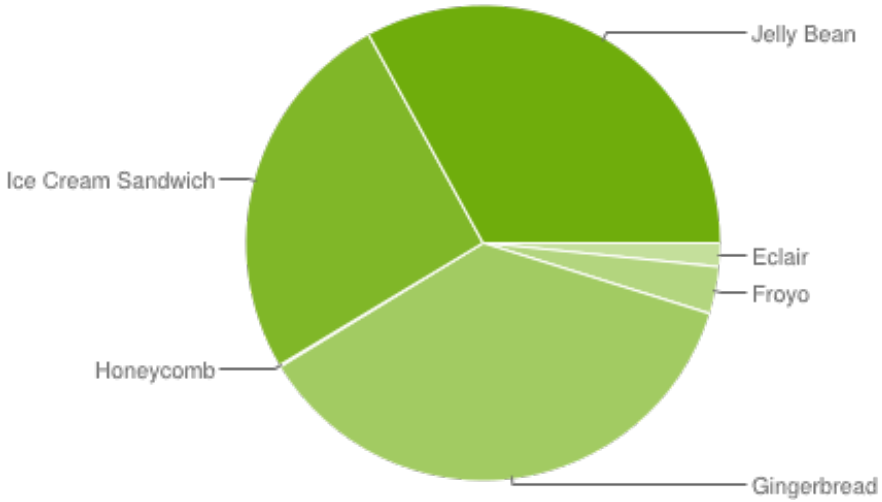

Figura 1.1: Distribuição das versões do Android

[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:

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

Para esse material, algumas convenções serão seguidas:

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

CAPÍTULO

2

Configuração do Ambiente

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

CAPÍTULO

3

Linguagem do Android

3.1 Linguagem

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

O Android tem algumas particulariedades 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 serie de facilidades alcançadas por meio de interfaces gráficas para os programadores. Por esse motivo, 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 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 *activites* 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 dispoção 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.

[1] Documentação 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 *(activites)*;
  - `drawable/`: Todas as imagens, separados por tamanho de tela; e
  - `menu/`: *layout* dos menus do aplicativo.

CAPÍTULO

4

Criando seu primeiro aplicativo

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

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

Na janela que apareceu você deve colocar o nome do aplicativo, do projeto e do pacote. O nome do pacote deve seguir a convenção do Java[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 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.

[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

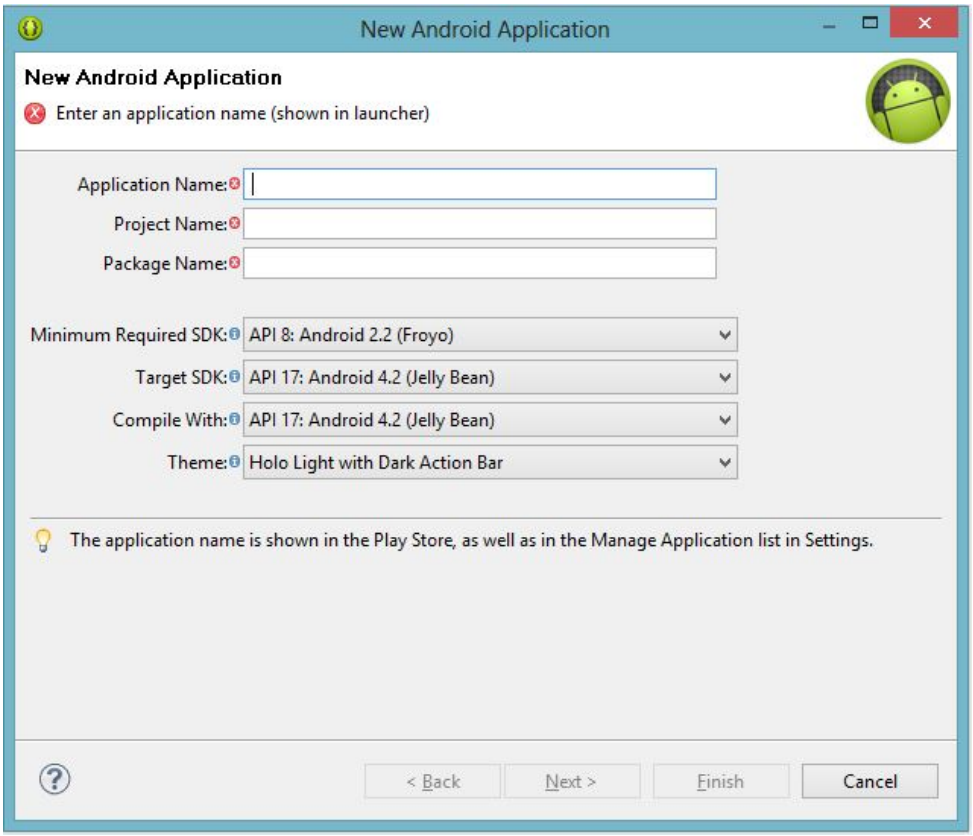

Figura 4.1: Primeira janela de criação de novo aplicativo

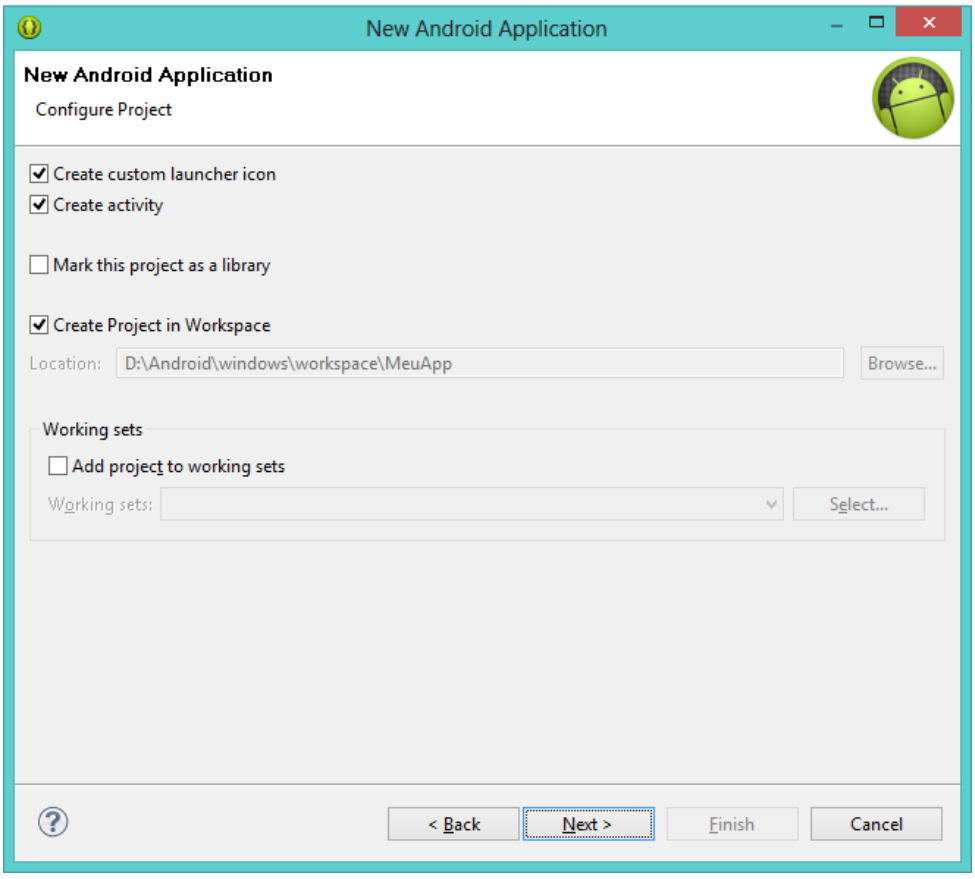

Figura 4.2: Segunda janela de criação de novo aplicativo

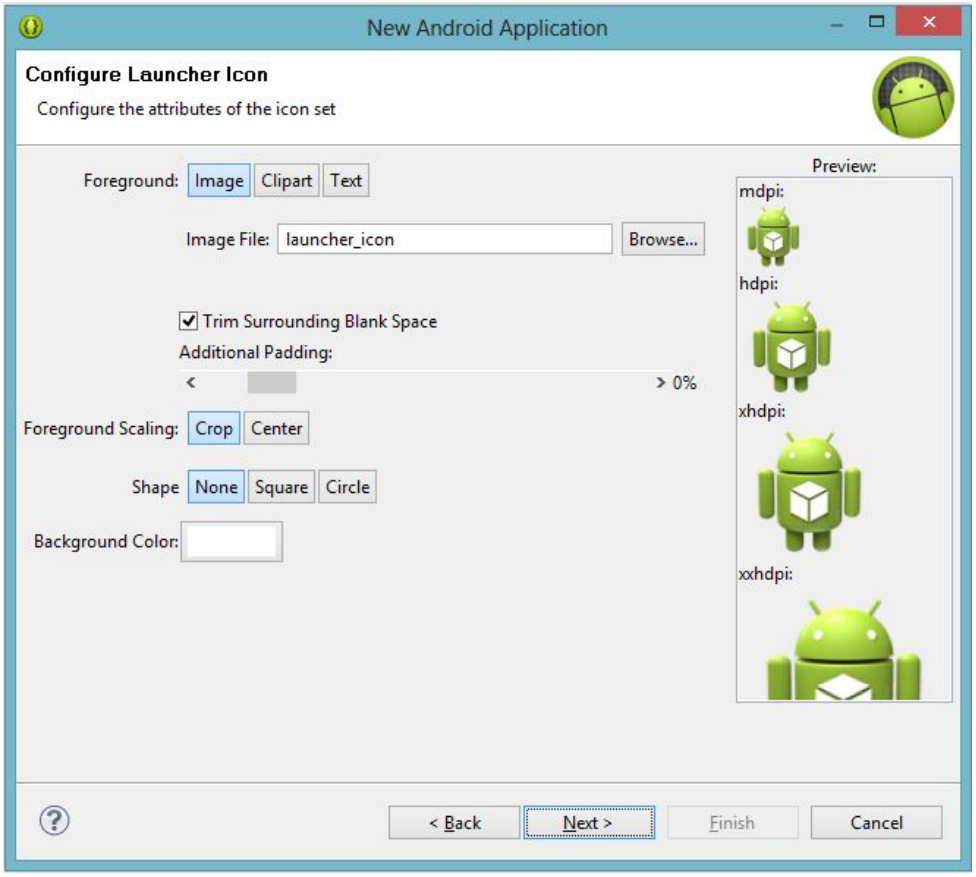

Figura 4.3: Terceira janela de criação de novo aplicativo

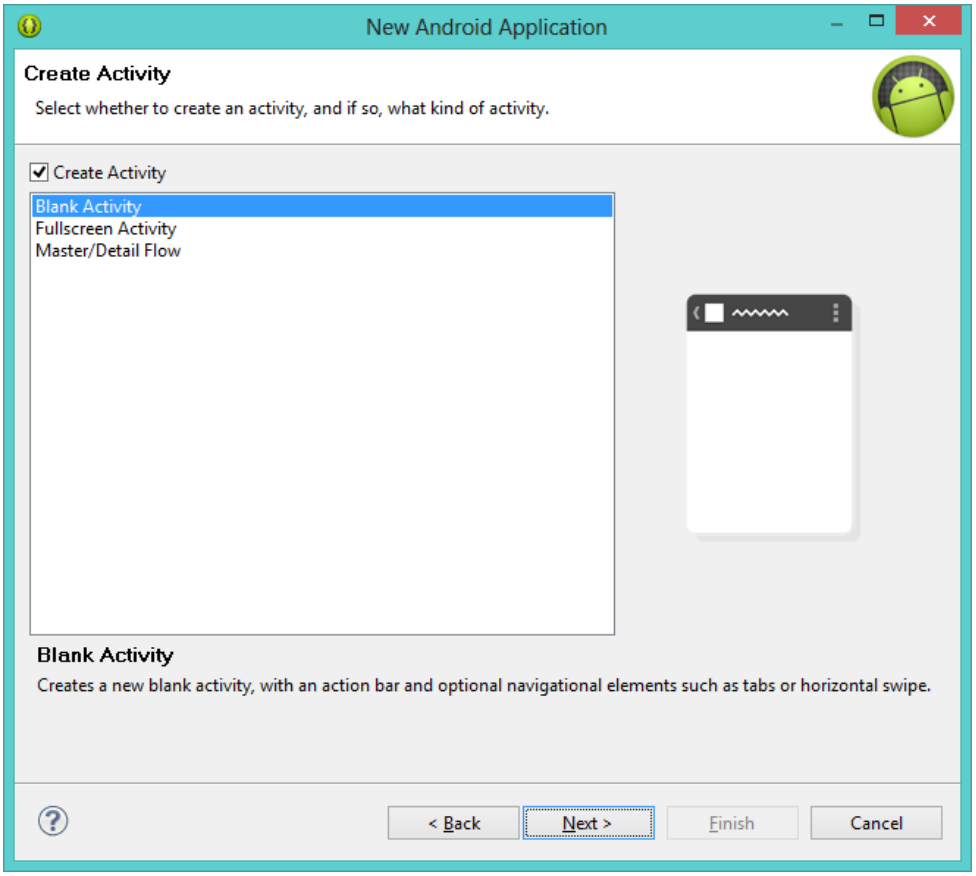

Figura 4.4: Quarta janela de criação de novo aplicativo

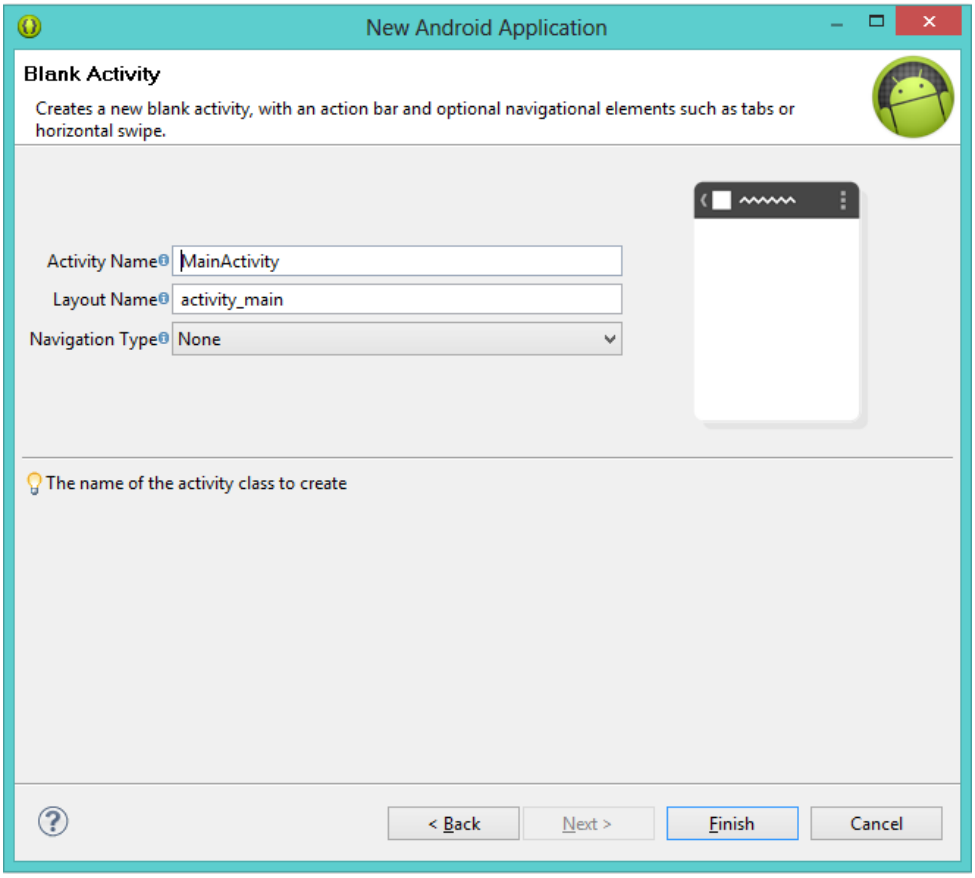

Figura 4.5: Quinta janela de criação de novo aplicativo

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 e 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,

[3] Documentação 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 *activites* através de um menu na *Action Bar*.

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

Algoritmo 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.

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

Algoritmo 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*

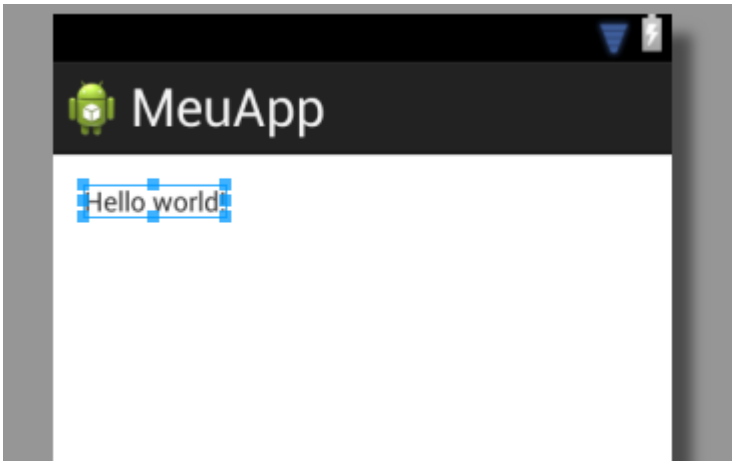

Figura 4.6: Selecionando o Hello world

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

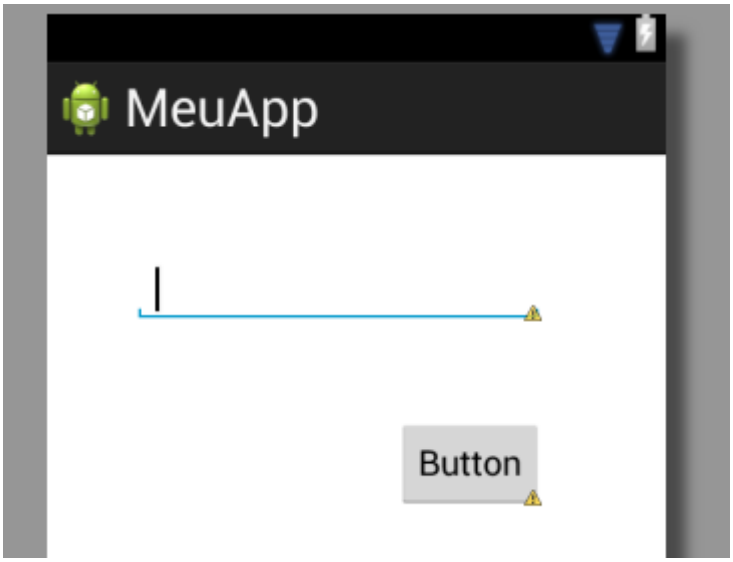

Figura 4.7: *activity* com os elementos colocados na tela

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>
```

Algoritmo 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" />
```

Algoritmo 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>
```

Algoritmo 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
}
```

Algoritmo 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;
```

Algoritmo 7: Exemplo de import de classe Android

Primeiro, crie um novo `Intent`[5], 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*.

[5]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 }
```

Algoritmo 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 }
```

Algoritmo 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 }
```

Algoritmo 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*.

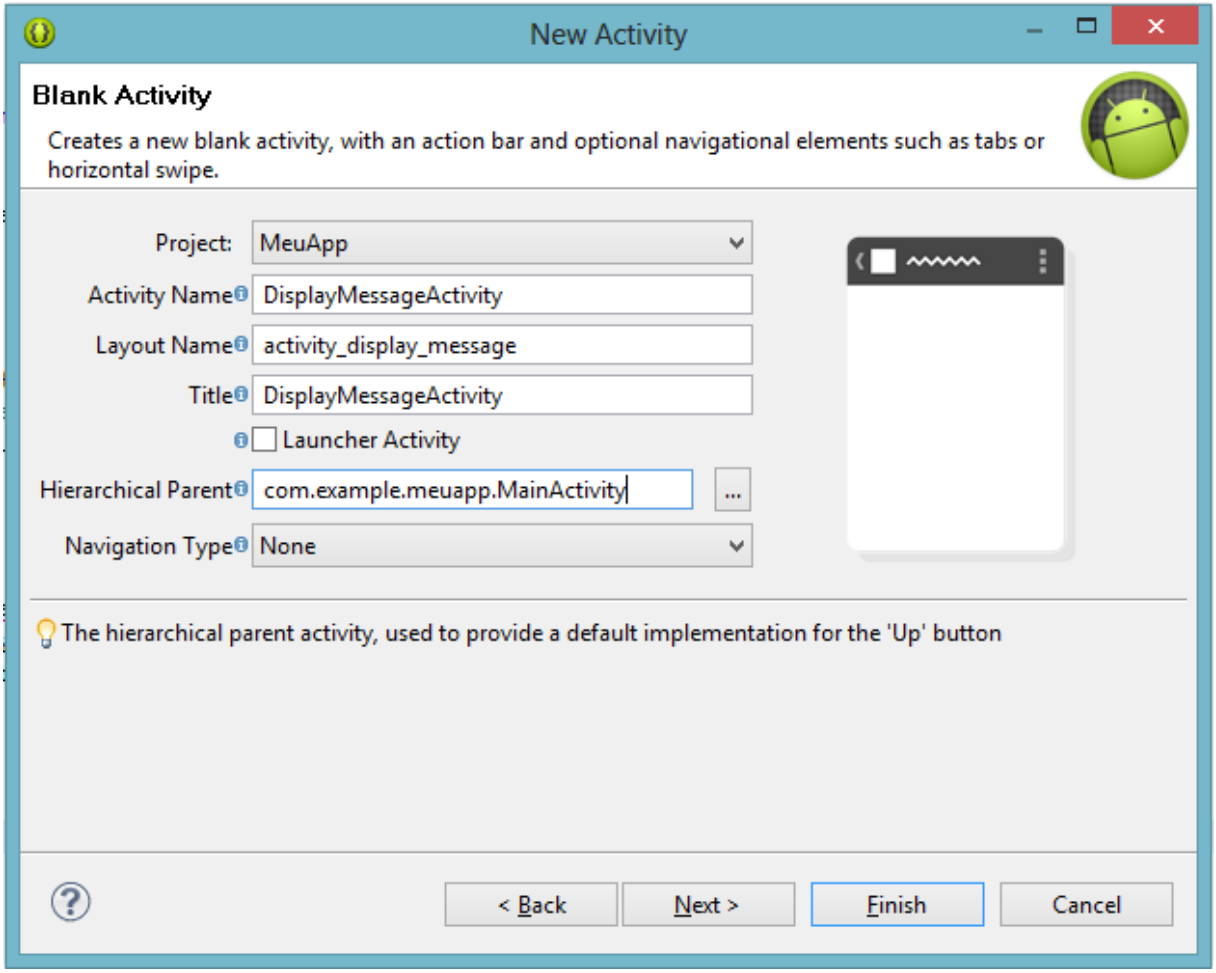

Figura 4.8: Criando uma nova *activity*

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()`[6].

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()`[7].

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

Algoritmo 11: Obtendo extras passados através do `Intent`

Agora para mostrar a mensagem na tela, você precisa criar um `TextView`[8]

[6] http://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)
[7] http://developer.android.com/reference/android/app/Activity.html#getIntent()
[8] 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);
}
```

Algoritmo 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[9]

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*

[9] 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[10]), 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.

[10] http://developer.android.com/tools/devices/index.html

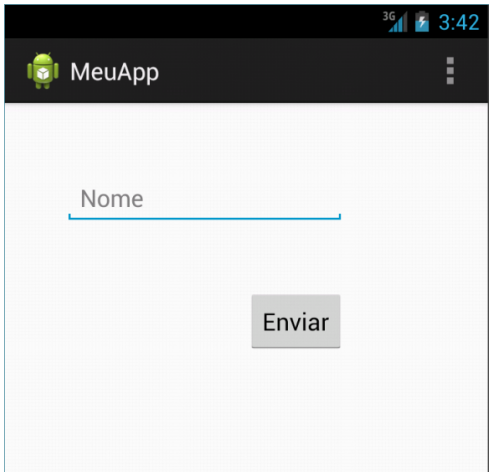

Figura 4.9: Primeira tela do primeiro aplicativo

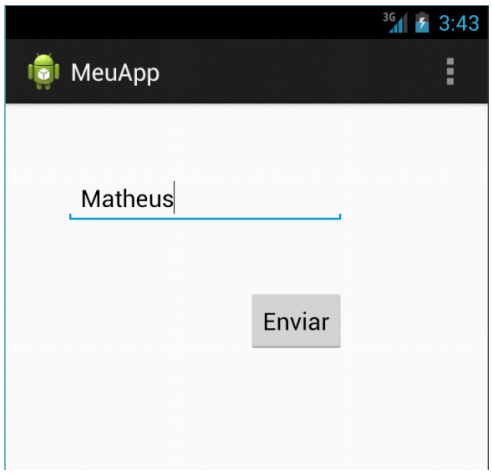

Figura 4.10: Primeira tela após escrever texto na caixa de texto

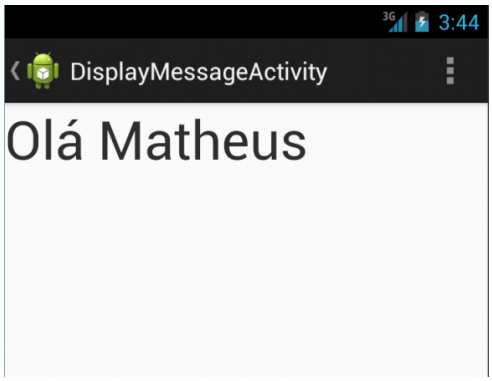

Figura 4.11: Segunda tela mostrando a mensagem enviada

CAPÍTULO

5

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 configuracão inicial. O sistema ao criar uma nova instância de uma *activity*, cada método muda o estado da *activity* um degrau pra cima na pirâmide.

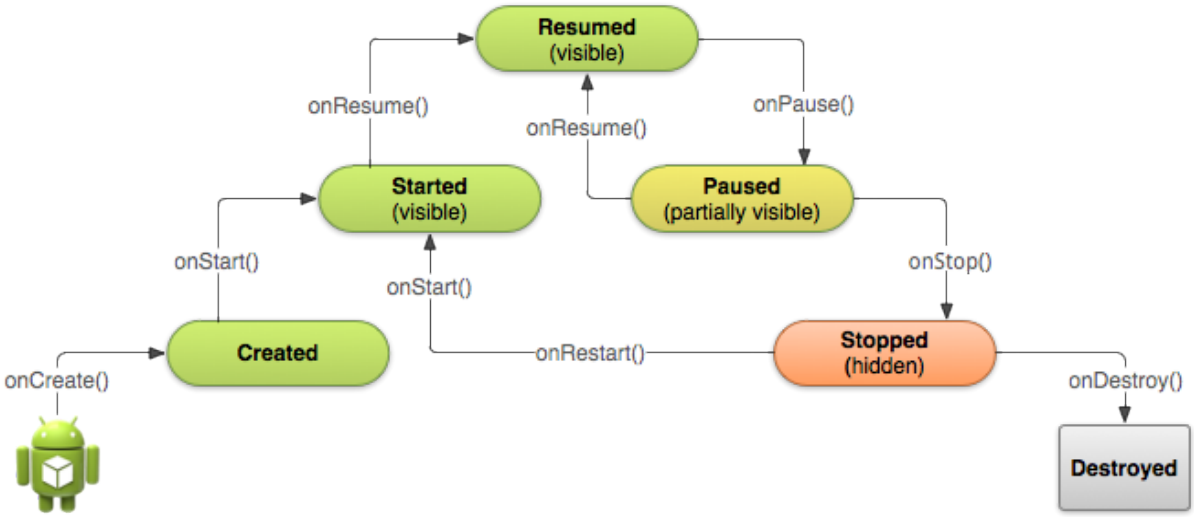

Figura 5.1: Ciclo de vida de uma *activity*

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 consegur 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 raíz 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:

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Algoritmo 13: Exemplo de *Launcher activity*

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

> **Dica:** Quando você cria um projeto Android no Eclipse, por padrão é incluida 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.

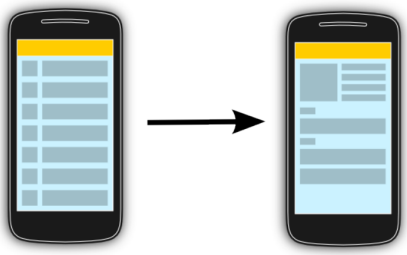

Figura 5.3: Detalhes de um elemento da lista

[2] 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>
```

Algoritmo 14: 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>
```

Algoritmo 15: 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>
```

Algoritmo 16: `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 17: 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"`.

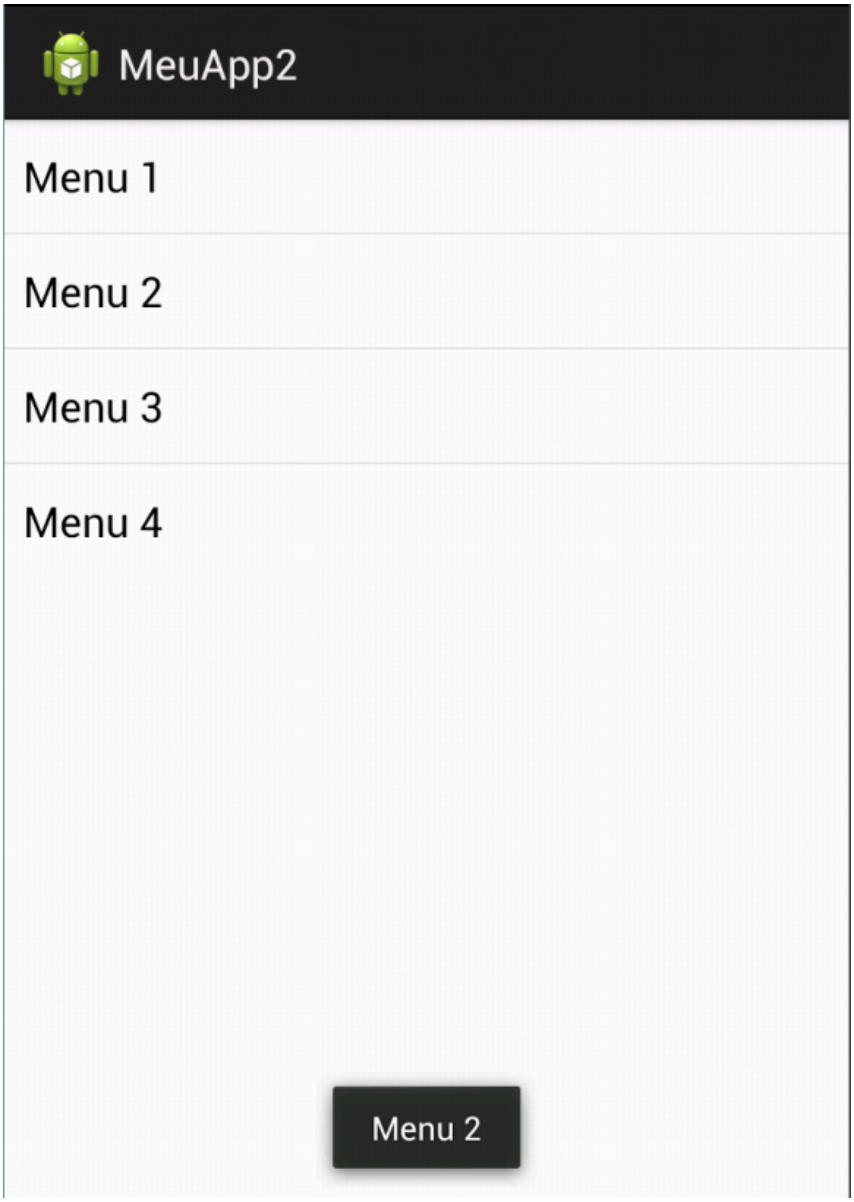

Figura 5.4: Lista simples

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`.

```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>
```

Algoritmo 18: 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   }
```

Algoritmo 19: Código da lista customizada

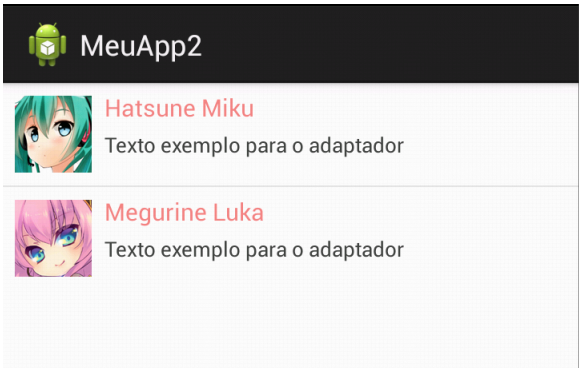

Figura 5.5: Lista Composta

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.

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

    <ExpandableListView
        android:id="@+id/listaExpandivel"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:transcriptMode="alwaysScroll"
        android:listSelector="@android:color/transparent">
    </ExpandableListView>

</LinearLayout>
```

Algoritmo 20: 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`

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:id="@+id/list_item">

  <TextView
  android:layout_width="0dp"
  android:layout_height="wrap_content"
  android:id="@+id/list_item_text_view"
  android:textSize="20sp"
  android:padding="10dp"
  android:layout_weight="1"
  android:layout_marginLeft="35dp" />

</LinearLayout>
```

Algoritmo 21: Layout `list_item_parent.xml`

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:id="@+id/list_item_child"
  android:gravity="center_vertical">

    <TextView
    android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/list_item_text_child"
        android:textSize="20sp"
        android:padding="10dp"
        android:layout_marginLeft="5dp"/>

</LinearLayout>
```

Algoritmo 22: 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.

```
public class Parent {
    private String mTitle;
    private ArrayList<String> mArrayChildren;

    public String getTitle() {
        return mTitle;
    }

    public void setTitle(String mTitle) {
        this.mTitle = mTitle;
    }

    public ArrayList<String> getArrayChildren() {
        return mArrayChildren;
    }

    public void setArrayChildren(ArrayList<String> mArrayChildren) {
        this.mArrayChildren = mArrayChildren;
    }
}
```

Algoritmo 23: 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 extender 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 24: 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 25: Construindo a lista expandível na *activity*

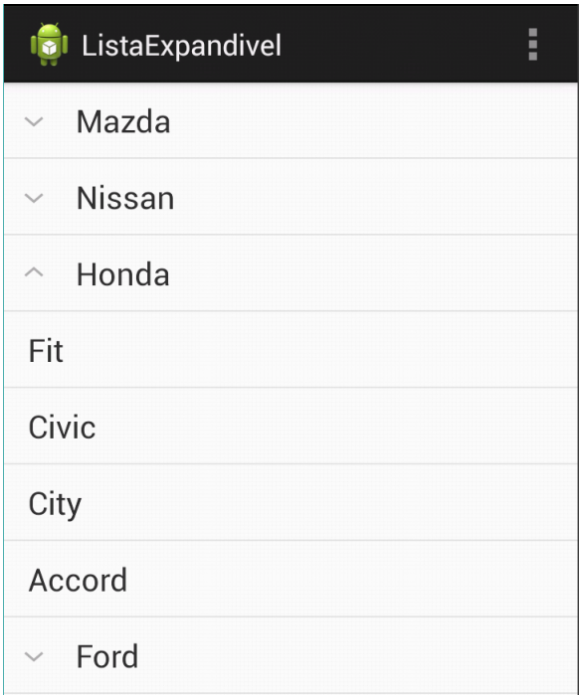

Figura 5.6: Exemplo de lista expandível

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.

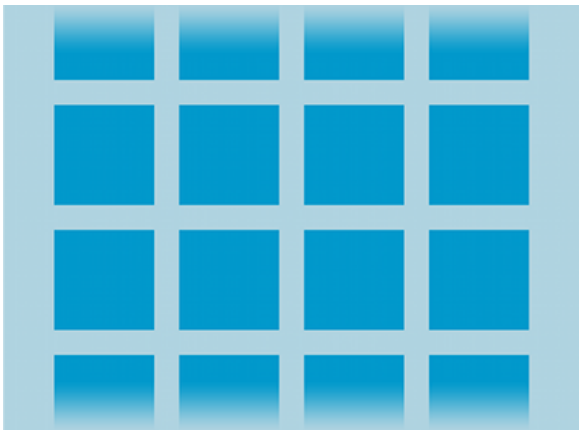

Figura 5.7: Esquema de um `Grid view`

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

```
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:columnWidth="90dp"
    android:horizontalSpacing="10dp"
    android:verticalSpacing="10dp"
    android:gravity="center"
    android:stretchMode="columnWidth" >
</GridView>
```

Algoritmo 26: 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 27: 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 28: *activity* com grade

Exemplo rodando:

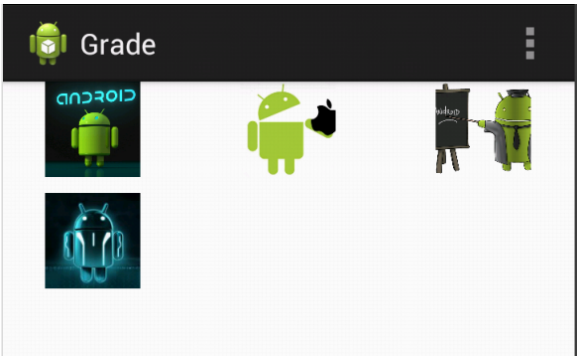

Figura 5.8: Demonstração de um `Grid view`

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`.

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

    <ImageView
      android:id="@+id/full_image_view"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >

     </ImageView>

    <TextView
        android:id="@+id/myImageViewText"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_alignLeft="@+id/full_image_view"
        android:layout_alignRight="@+id/full_image_view"
        android:layout_alignTop="@+id/full_image_view"
        android:gravity="center"
        android:background="#55555555"
        android:textSize="16sp"
        android:textColor="#FFFFFF" />

</LinearLayout>
```

Algoritmo 29: 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 30: 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 31: *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 32: 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 inlcuir o fragmento no XML da *activity* como você faria com qualquer `view`.

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

    <fragment
        android:id="@+id/fragment_content"
        android:name="com.example.fragmento.BasicFragment"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
  </fragment>
</LinearLayout>
```

Algoritmo 33: Layout da *activity* com um fragmento

Você pode usar o `<fragment>` quantas vezes quiser para incluir múltiplos fragmentos. Note que você precisa usar um nome qualificado em `android:name`, veja mais na documentação oficial: activity-element[3]
Novamente, caso esteja desenvolvendo para APIs menores que 11, você vai precisar fazer a *activity* extender a classe `FragmentActivity` e importar a classe de suporte:

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

public class MainActivity extends FragmentActivity
```

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

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

    <FrameLayout
        android:id="@+id/fragment_content"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />

</LinearLayout>
```

Algoritmo 34: 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

```
public class MainActivity extends FragmentActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //Como estamos usando o pacote de suporte
    //Precisamos usar o Manager desse pacote
    FragmentManager fm = getSupportFragmentManager();
    //Voce pode obter um fragmento da mesma forma que obtem
    //qualquer outra view usando o FragmentManager
    Fragment fragment = fm.findFragmentById(R.id.fragment_content);

    if(fragment == null){
      //Comeca uma transacao de fragmentos
      FragmentTransaction ft = fm.beginTransaction();
      //Adiciona o fragmento
      ft.add(R.id.fragment_content, new BasicFragment());
      //"Commita" a transacao
      ft.commit();
    }
  }
...
```

Algoritmo 35: *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:

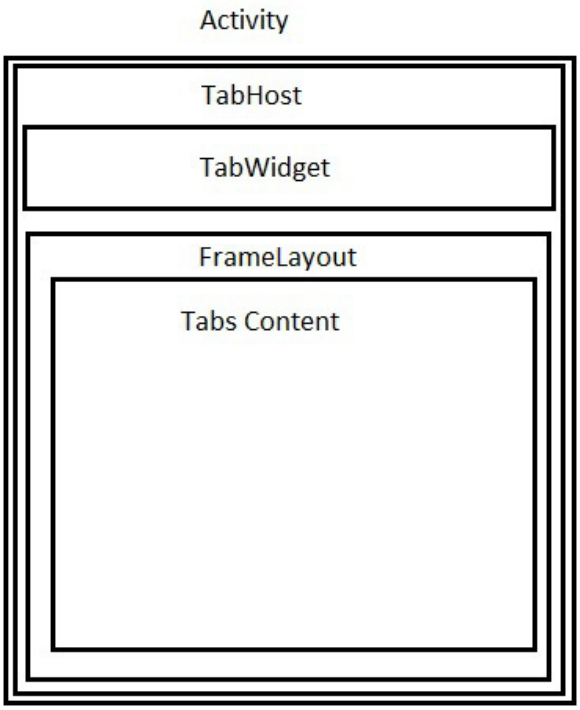

Figura 5.9: Esquema da interface com abas

5.9 Arrastar (`SwipeView`) com abas

5.10 Menu lateral

APÊNDICE

A

Especificação blá, blá, blá

Isto é um apêndice...