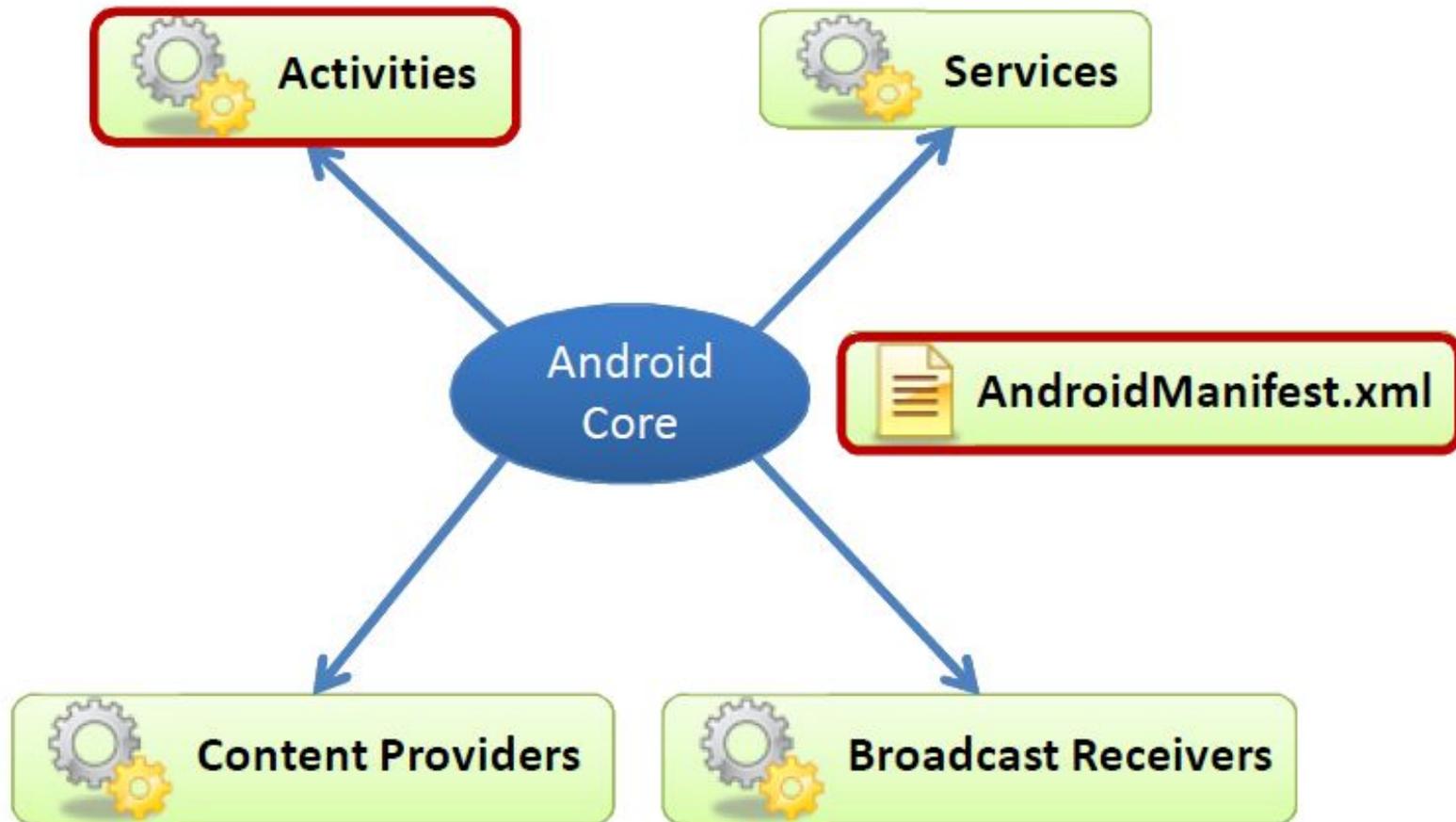


# DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS

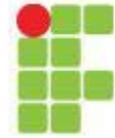
PROF<sup>a</sup>. M.Sc. JULIANA H Q BENACCHIO



# Aplicações no Android



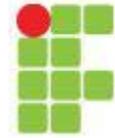
# Arquivo `AndroidManifest.xml`



- Arquivo que declara os componentes do aplicativo e outras informações
- Todo aplicativo possui um arquivo `AndroidManifest.xml`
- O formato do arquivo é XML



# Arquivo AndroidManifest.xml



INSTITUTO FEDERAL  
PARANÁ

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nb_ifpr_foz_juliana.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <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>
    </application>
</manifest>
```



# Permissões de Acesso

- Para realizar determinados tipos de operações, a aplicação deve solicitar permissão para tal
- As permissões são configuradas no `AndroidManifest.xml`

```
<manifest>
  <uses permission
    android:name="android.permission.RECEIVE_SMS"
  />
</manifest>
```



# Permissões de Acesso

- As permissões são verificadas apenas no momento em que a aplicação é instalada
- Algumas permissões comuns do pacote `android.permission`

Permissão	Descrição
<code>RECEIVE_SMS</code>	Recebimento de SMS
<code>VIBRATE</code>	Ativação da vibração do telefone
<code>READ_CONTACTS</code>	Leitura dos contatos cadastrados no dispositivo
<code>INTERNET</code>	Acesso à internet
<code>CALL_PHONE</code>	Realização de chamadas telefônicas
<code>CAMERA</code>	Acesso à câmera



# Activities

- Uma **activity** normalmente representa uma tela da aplicação
- Representada por uma classe que herda de **`android.app.Activity`**

```
public class MainActivity extends Activity {  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```



# Activities e Views

- **Views** são os componentes que se agrupam para montar a interface gráfica
- As **activities** e as **views** têm uma relação próxima
  - **Activities** representam a tela
  - **Views** representam o que é renderizado na tela



# Activities e Views

- O método `setContentView(layoutResID: int)` da **activity** recebe como argumento um número inteiro indicando o recurso de algum arquivo XML de layout.

```
setContentView(R.layout.activity_main);
```



# Activities e Views

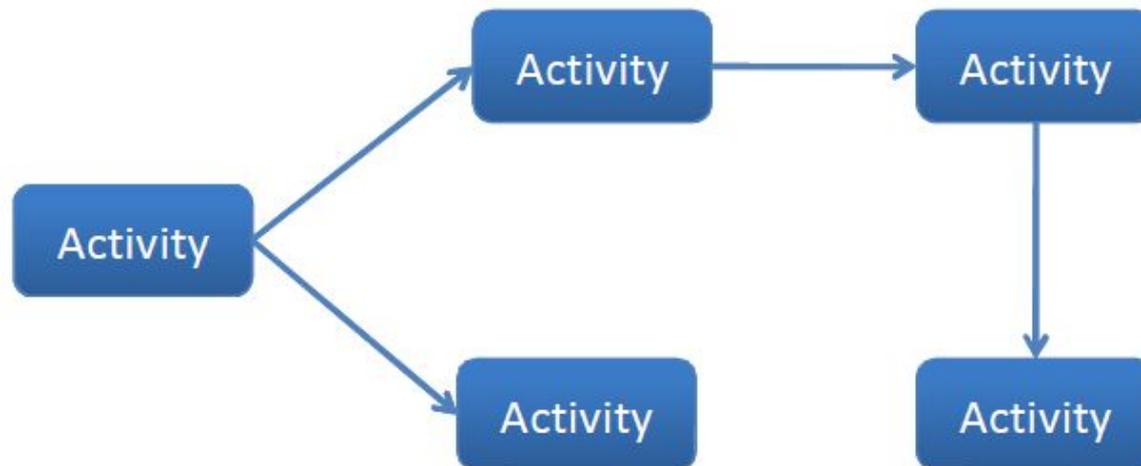
- O método `setContentView (view: android.view.View)` da **activity** é utilizado para determinar qual **view** será renderizada. Este método é utilizado diretamente no código-fonte Java.

```
TextView view = new TextView(this);  
view.setText("Hello World!");  
setContentView(view);
```



# Invocando Activities

- Uma aplicação normalmente é composta por um conjunto de **activities**



A comunicação entre activities é feita através de uma **intent**

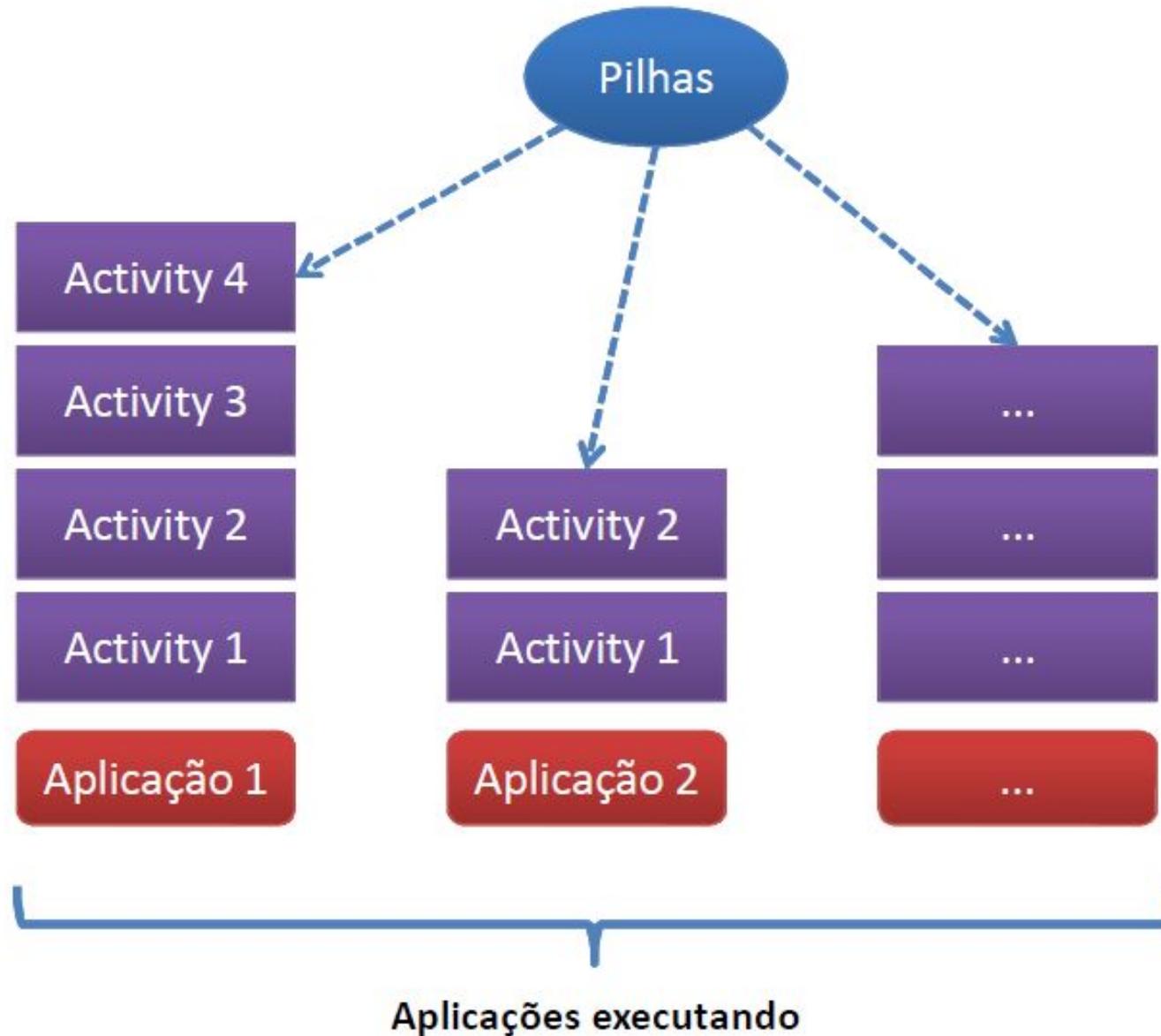


# As Activities na Memória

- As activities que fazem parte da sua aplicação são organizadas numa pilha
- A activity que está no topo da pilha é a activity mostrada na tela
- Toda vez que uma nova activity é invocada, ela passa a ficar no topo da pilha
- Quando o botão voltar é pressionado, a activity atual é removida da pilha e dá espaço para a activity que ocupa o topo da pilha ficar ativa



# A Pilha de Activities

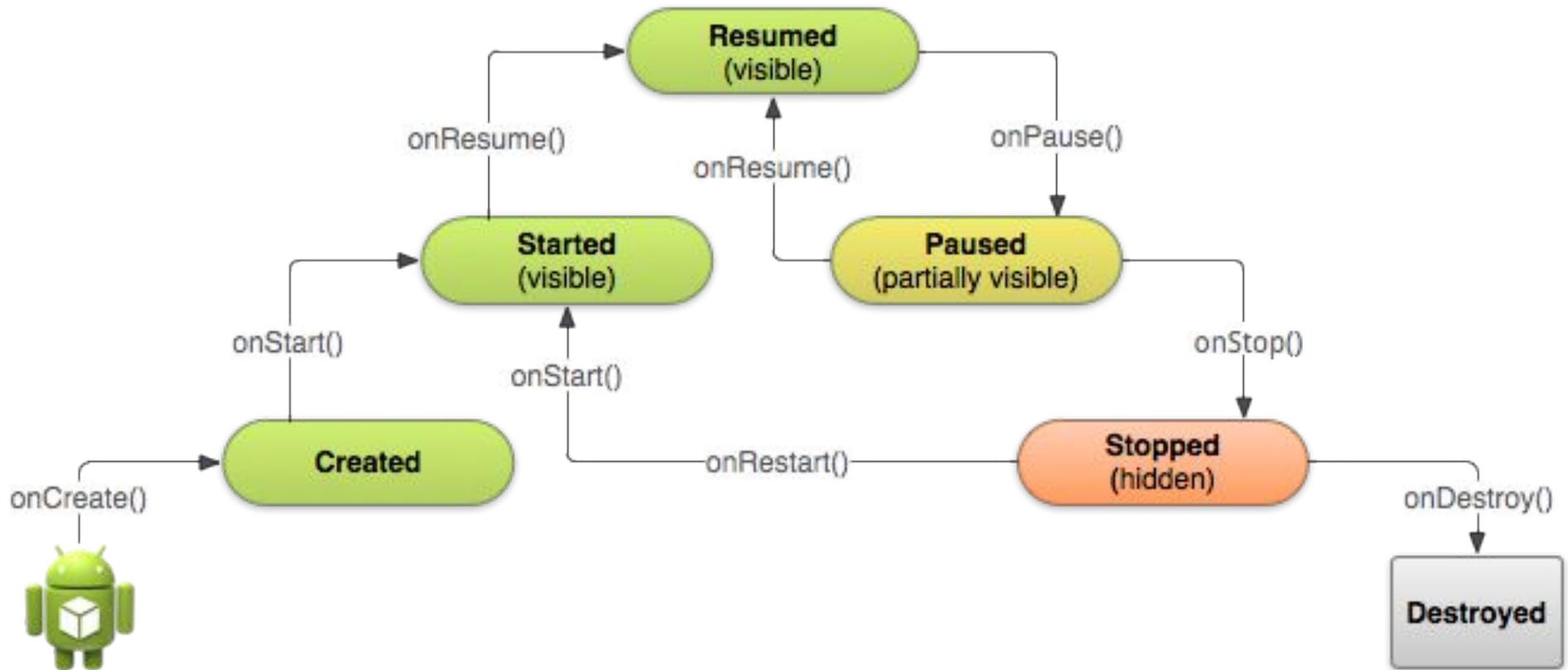


# Ciclo de vida de uma Activity

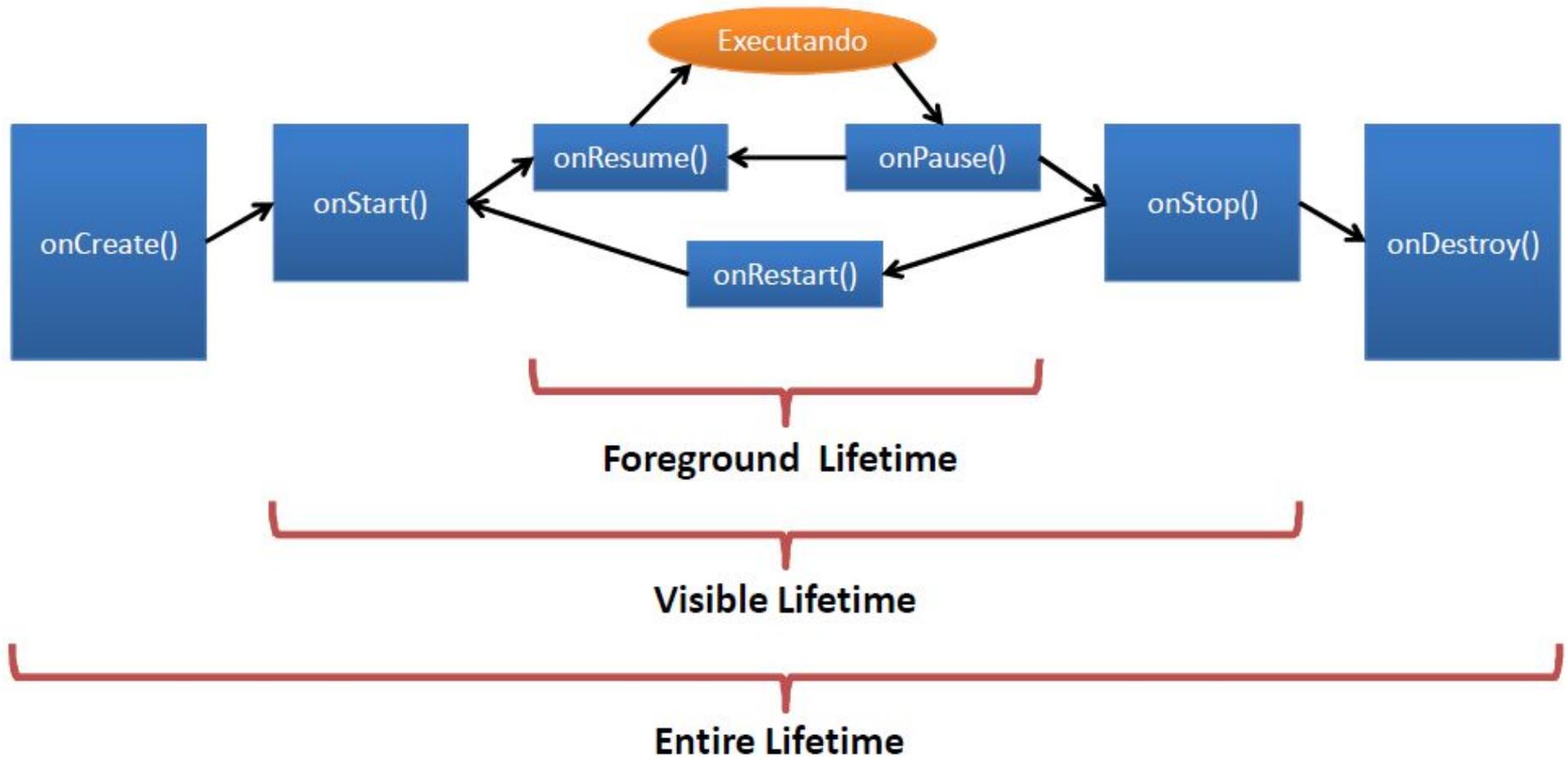
- Métodos de *callback* invocados pelo Android durante o ciclo de vida da activity
  - `onCreate()`
  - `onStart()`
  - `onResume()`
  - `onPause()`
  - `onStop()`
  - `onRestart()`
  - `onDestroy()`



# Ciclo de vida de uma Activity



# Ciclo de vida de uma Activity



- Exemplos de recursos (***resources***)
  - Textos
  - Imagens
  - Sons
  - Layouts de tela
- O ideal é não referenciar os ***resources*** diretamente no código
  - Repetição de código
  - Difícil de manter



# Localização dos *Resources*

- Os *resources* ficam localizados dentro do diretório `/res`
- Este diretório contém subdiretórios de acordo com tipo de recurso que representam
  - `/res/values-xxx`
  - `/res/layout`
  - `/res/drawable-xxx`
- Dentro dos subdiretórios ficam os arquivos de *resources*



# String Resources

- Representam textos (strings) como recursos
- Estão localizados no diretório `/res/values`
- A representação dos *resources* é feita em XML

```
<resources>  
  <string name="app_name">My Application</string>  
  <string name="hello_world">Hello world!</string>  
  <string name="action_settings">Settings</string>  
</resources>
```

Nome

Valor



# Color Resources

- Semelhantes às *string resources*
- Também localizados em `/res/values`

```
<resources>  
    <color name="red">#FF0000</color>  
    <color name="green">#00FF00</color>  
</resources>
```



# A Classe R

- A classe R é gerada automaticamente ao compilar o projeto e permite que a aplicação acesse qualquer recurso como arquivos e imagens utilizando as constantes desta classe.
- Essa classe nunca deve ser alterada manualmente.
- O arquivo `R.java` é gerado na pasta `app/build/generated/source/r` do módulo



# Usando a Classe R

`/res/values/strings.xml`

```
<resources>  
  <string name="button_text">Pressione Aqui</string>  
</resources>
```

`MainActivity.java`

```
Button b = new Button(this);  
String text = getString(R.string.button_text);  
b.setText(text);  
setContentView(b);
```

`Classe R`



# Implementação da Classe R

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * apt tool from the resource data it found. It
 * should not be modified by hand.
 */

package ifpr.foz.android;

public final class R {
    public static final class attr {
    }
    public static final class string {
        public static final int button_text=0x7f050003;
        public static final int hello_world=0x7f050002;
    }
}
```

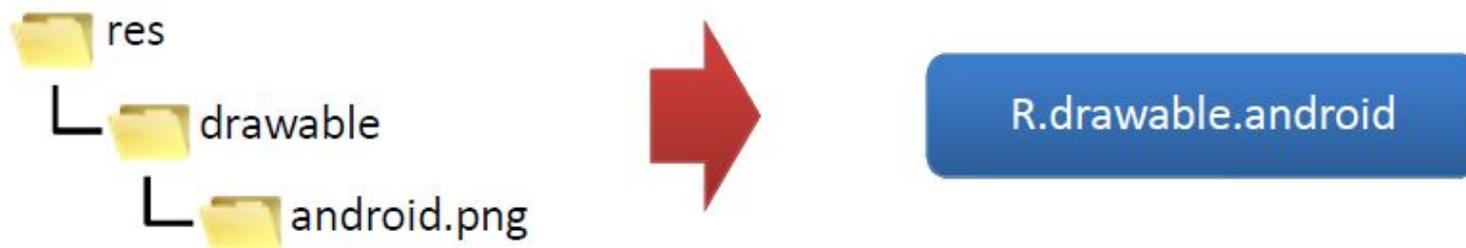
A classe não  
deve ser  
editada

Inner classes e constantes  
representam os resources



# Drawable Resources

- **Resources** que representam imagens
- Localizados no diretório `/res/drawable`
  - Extensões `.png`, `.jpg` e `.gif` são suportadas



MyActivity.java

```
ImageView img = new ImageView(this);
img.setImageResource(R.drawable.android);
setContentView(img);
```



# Layout Resources

- Permitem representar o *layout* das telas da aplicação em formato XML
- A composição da interface fica separada do código-fonte da aplicação
- Utilizar arquivos de *layout* é a forma recomendada para criar a interface gráfica da aplicação
- Os *layouts* ficam localizados em `/res/layout`



# Arquivo de *Layout*

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

  <TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
  />

  <EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
  />
</LinearLayout>
```

Texto

Caixa de  
texto



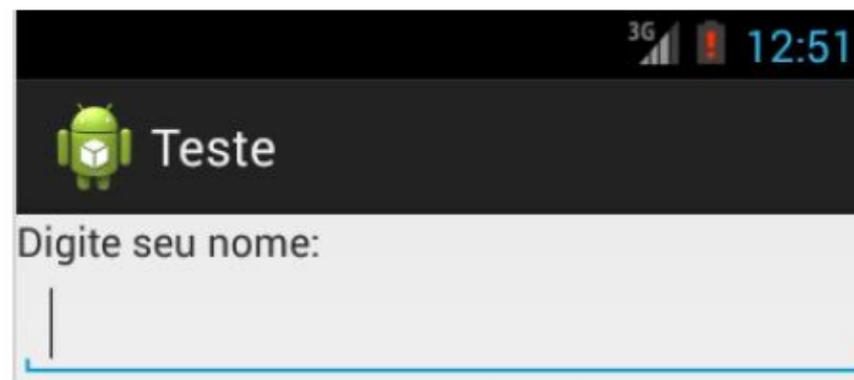
# Aplicando um *Layout*



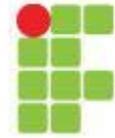
R.layout.main

MyActivity.java

```
setContentView(R.layout.main);
```



# Referenciando *Resources* no XML

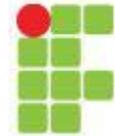


INSTITUTO FEDERAL  
PARANÁ

- A classe `R` é utilizada para referenciar ***resources*** no código
- Às vezes é necessário fazer a referência dentro de outro *resource* (definido em um XML)
- *Layout resources* são exemplos típicos
- O Android possui uma convenção para este tipo de referência



# Referenciando *Resources* no XML



`/res/values/strings.xml`

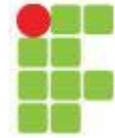
```
<resources>
  <string name="hello_world">Hello World!!</string>
</resources>
```

`/res/layout/activity_main.xml`

```
<TextView
  android:text="@string/hello_world"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```



# Referenciando *Resources* no XML



INSTITUTO FEDERAL  
PARANÁ

- Convenção para referenciar *resources*

<b>Resource</b>	<b>Dentro do Código</b>	<b>Dentro do XML</b>
String	<code>R.string.&lt;nome_res&gt;</code>	<code>@string/&lt;nome_res&gt;</code>
Color	<code>R.color.&lt;nome_res&gt;</code>	<code>@color/&lt;nome_res&gt;</code>
Drawable	<code>R.drawable.&lt;nome_res&gt;</code>	<code>@drawable/&lt;nome_res&gt;</code>
...	...	...



# Definindo *Resource* IDs

- Algumas vezes é necessário definir seus próprios IDs para *resources* para referência posterior dentro do código
- Um exemplo típico é a recuperação de dados fornecidos pelo usuário na interface gráfica



# Definindo *Resource* IDs

`/res/layout/activity_main.xml`

```
<EditText
    android:id="@+id/edtNome"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>
```

`MainActivity.java`

```
EditText edtNome = (EditText) findViewById(R.id.edtNome);
String s = edtNome.getText().toString();
```

`@+id/<algum_id>` permite criar um identificador para o componente da interface gráfica



# System Resources

- Além dos *resources* que você define, o próprio Android tem alguns previamente definidos
- Os ***system resources*** são referenciados pela classe `android.R`



# Outros tipos de Resources

- Existem outros tipos de *resources* no Android

<i>Resource</i>	Descrição
anim	Define animações
menu	Define os menus da aplicação
raw	Arquivos de qualquer formato que não são compactados
xml	Arquivos XML
...	...

- *Resources* também podem ser específicos para uma língua ou hardware
  - Durante a execução, o Android escolhe qual *resource* utilizar

