



Introdução a Programação Java para Google Android

INTRODUÇÃO

- **Introdução aos conceitos de Mobilidade com Android**
- **Introdução ao Android e a Linguagem de Programação Java**
- **Utilizando o Android Studio**
- **Estrutura do sistema operacional Android**
- **Desenvolvimento do primeiro aplicativo**
- **Instalando a app no smartphone**
- **Definição de Strings (Internacionalização)**
- Interface Gráfica e Associação a Eventos
- Utilização da classe Activity
- Utilização de Intents
- Back Stack
- Introdução as Views
- Utilização dos componentes Button, TextView, EditText e ImageView
- Utilização dos componentes Checkbox, RadioGroup, Spinner, ListView e Dialog

ÍNDICE

- INTRODUÇÃO
- PRINCIPAIS COMPONENTES
- AMBIENTE DE DESENVOLVIMENTO
- CRIANDO E EXECUTANDO UM PROJETO ANDROID
- UMA PRIMEIRA APLICAÇÃO
- RECURSOS EXTERNOS / INTERNACIONALIZAÇÃO

POR QUE ANDROID?

É uma plataforma em expansão no mercado de desenvolvimento mobile;

Oferece todos os recursos para desenvolvimento gratuitamente;



Tem como base a linguagem Java;

Dispõe de uma série de recursos muito maior do que a plataforma JME;

Powered by Google

SISTEMAS OPERACIONAIS

Sistema Operacional	Market Share Jul-14 e Mai-15		Linguagem Nativa
Android	44,6%	81,5%	Java
iOS	44,1%	14,8%	Objetive-C
Windows Phone	2,5%	2,7%	C#
Symbian	2,5%	----	C++
BlackBerry	1,2%	0,5%	Java, C++, HTML5
Other OS	7,2%	0,6%	HTML5

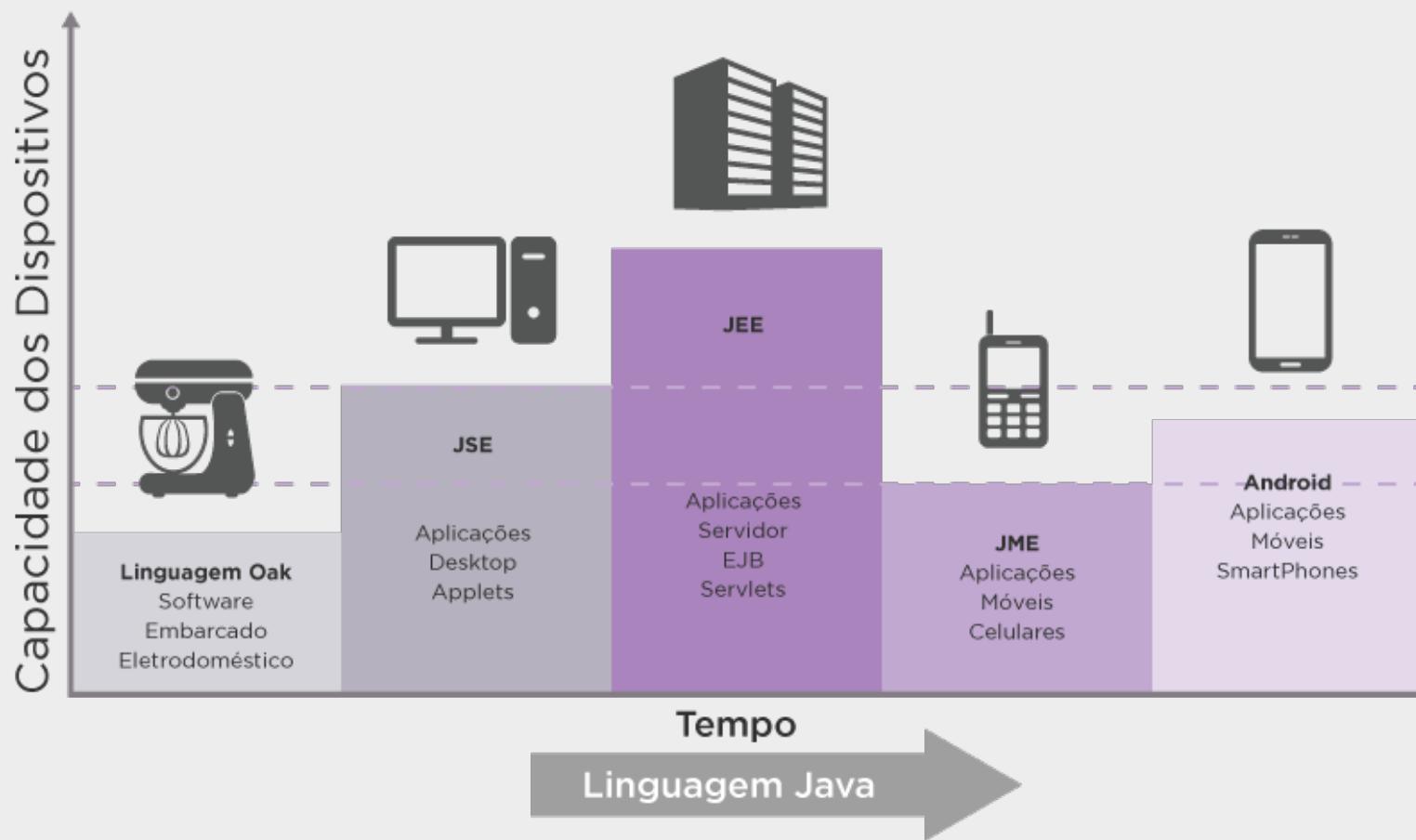
Fontes:

<http://www.loiane.com/2014/02/curso-phonegap-cordova-aula-01-introducao-ao-phonegap/>

<http://www.netmarketshare.com/>

http://www.teleco.com.br/sist_operacional.asp

CAPACIDADE DOS DISPOSITIVOS



Android é uma plataforma aberta (open source) de desenvolvimento de aplicações para dispositivos móveis;

Tal plataforma inclui um sistema operacional (Linux), bibliotecas, uma máquina virtual (Dalvik), frameworks e aplicações já prontas;

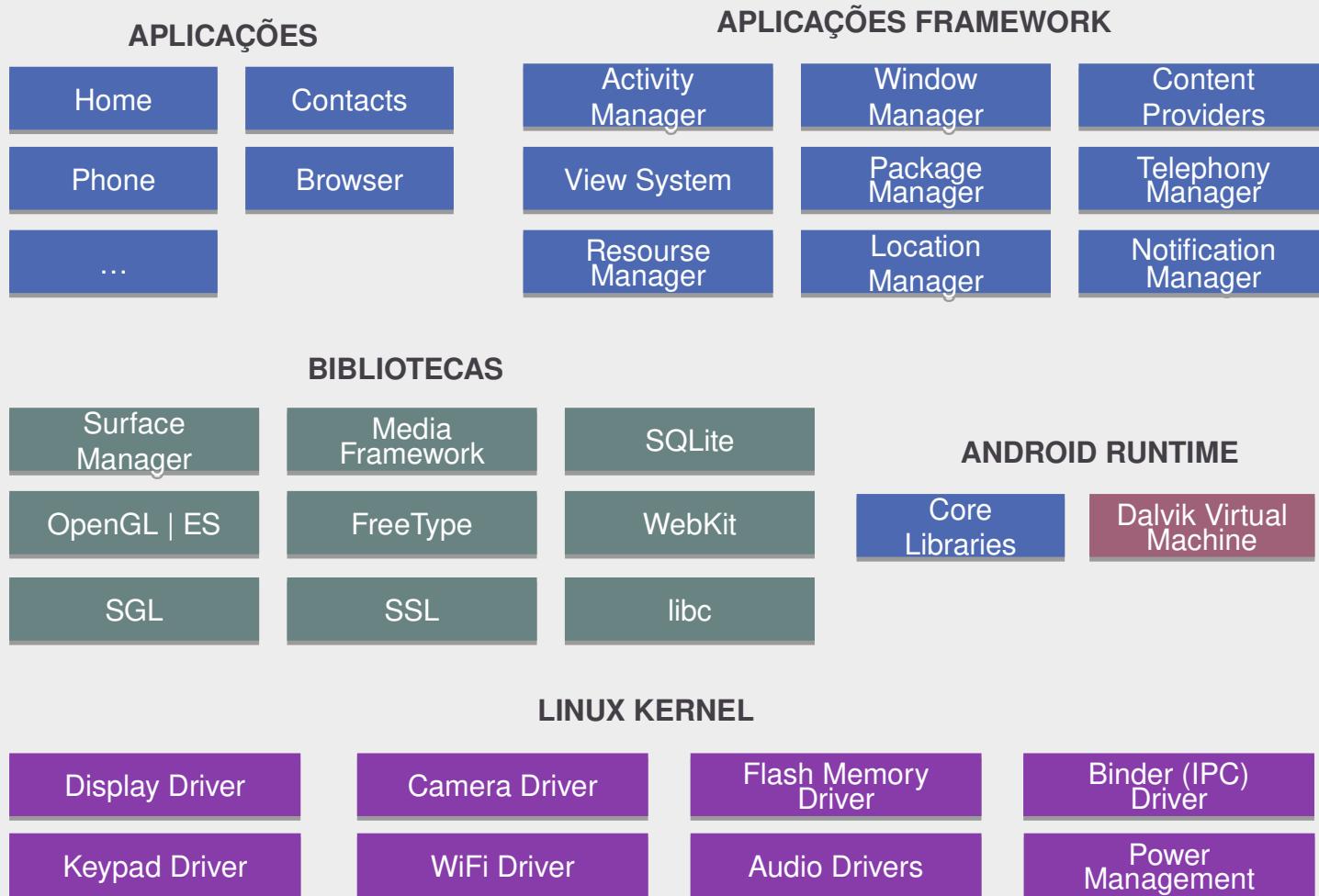
A linguagem utilizada para desenvolvimento de aplicações é o **Java**;

O site oficial para desenvolvedores Android é:
<http://developer.android.com/index.html>

Aplicações desenvolvidas podem ser publicadas ou obtidas do site:
<https://market.android.com/>

PRINCIPAIS COMPONENTES

FIAP



Fonte: <http://developer.android.com/guide/basics/what-is-android.html>



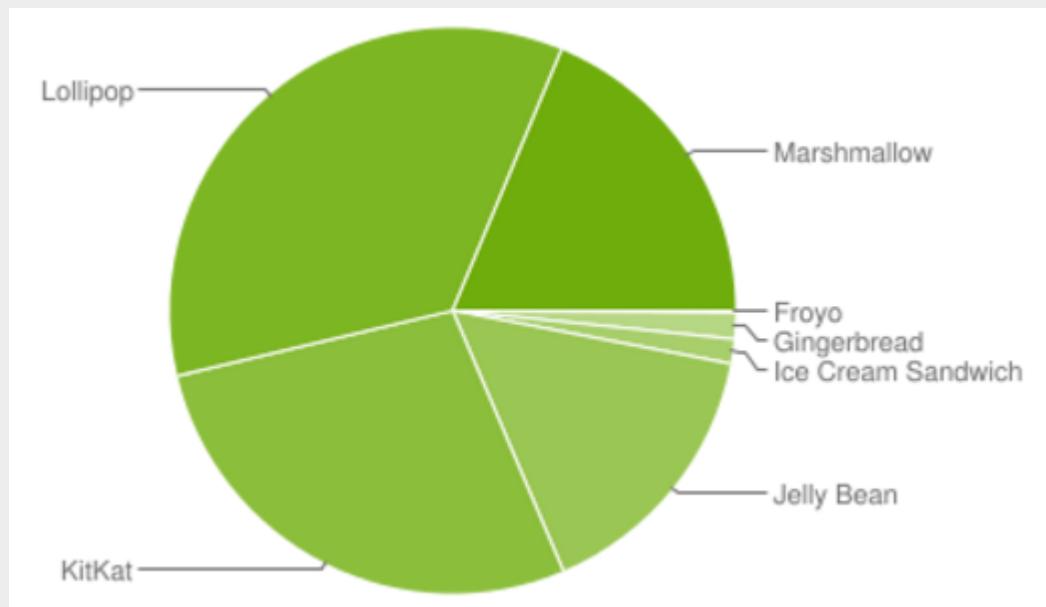
- Para desenvolver aplicações Android é necessário um Software Development Kit (SDK);
- O SDK Android é composto pelo conjunto de APIs, ferramentas de desenvolvimento, emuladores de dispositivos, documentação e código fonte de aplicações exemplo.
- Um SDK pode ser obtido no endereço:
<http://developer.android.com/sdk/index.html>

The image shows the official logo for Android Studio, which consists of a green circular icon with a white stylized 'A' shape containing a gear, followed by the text "Android Studio". Below the logo, there is a bulleted list of components:

- Android Studio IDE
- Android SDK tools
- Android 5.0 (Lollipop) Platform
- Android 5.0 emulator system image with Google APIs

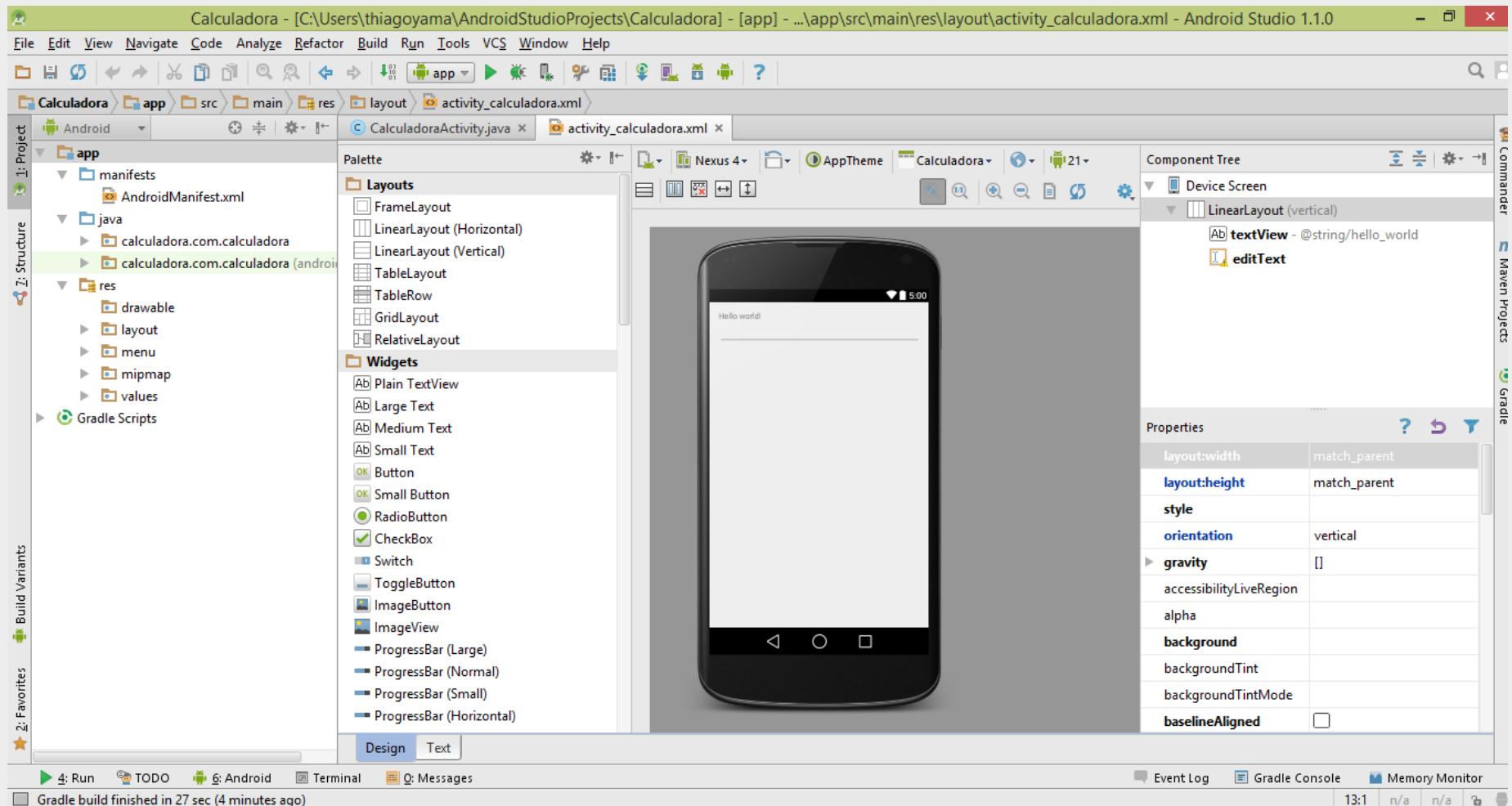
VERSÕES ANDROID

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.4%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%



AMBIENTE DE DESENVOLVIMENTO

FIAP



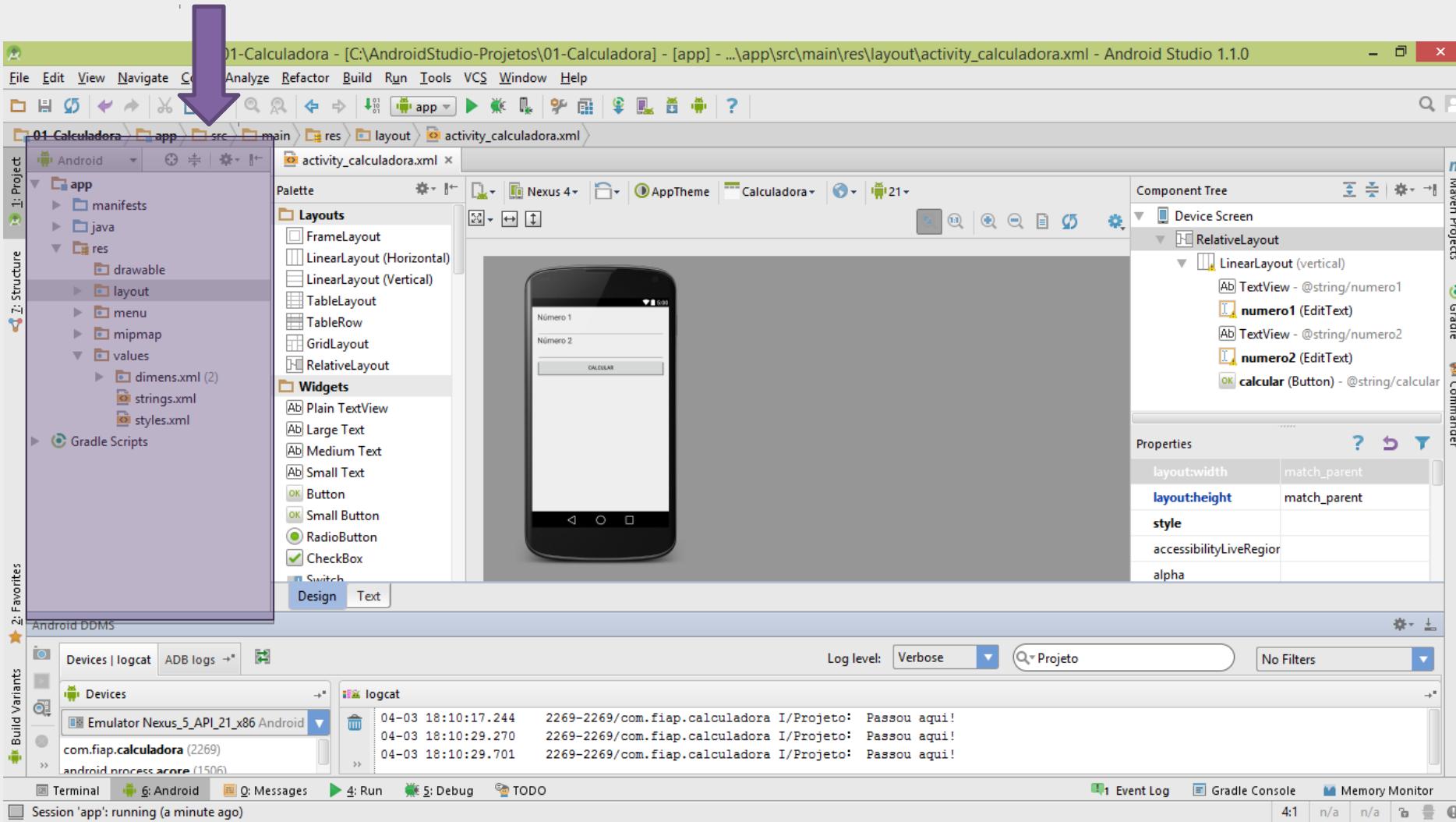


Ao executar a aplicação ela é visualizada em um emulador de dispositivo móvel (AVD) que pode representar modelos distintos de aparelhos reais.

AMBIENTE DE DESENVOLVIMENTO

FIAP

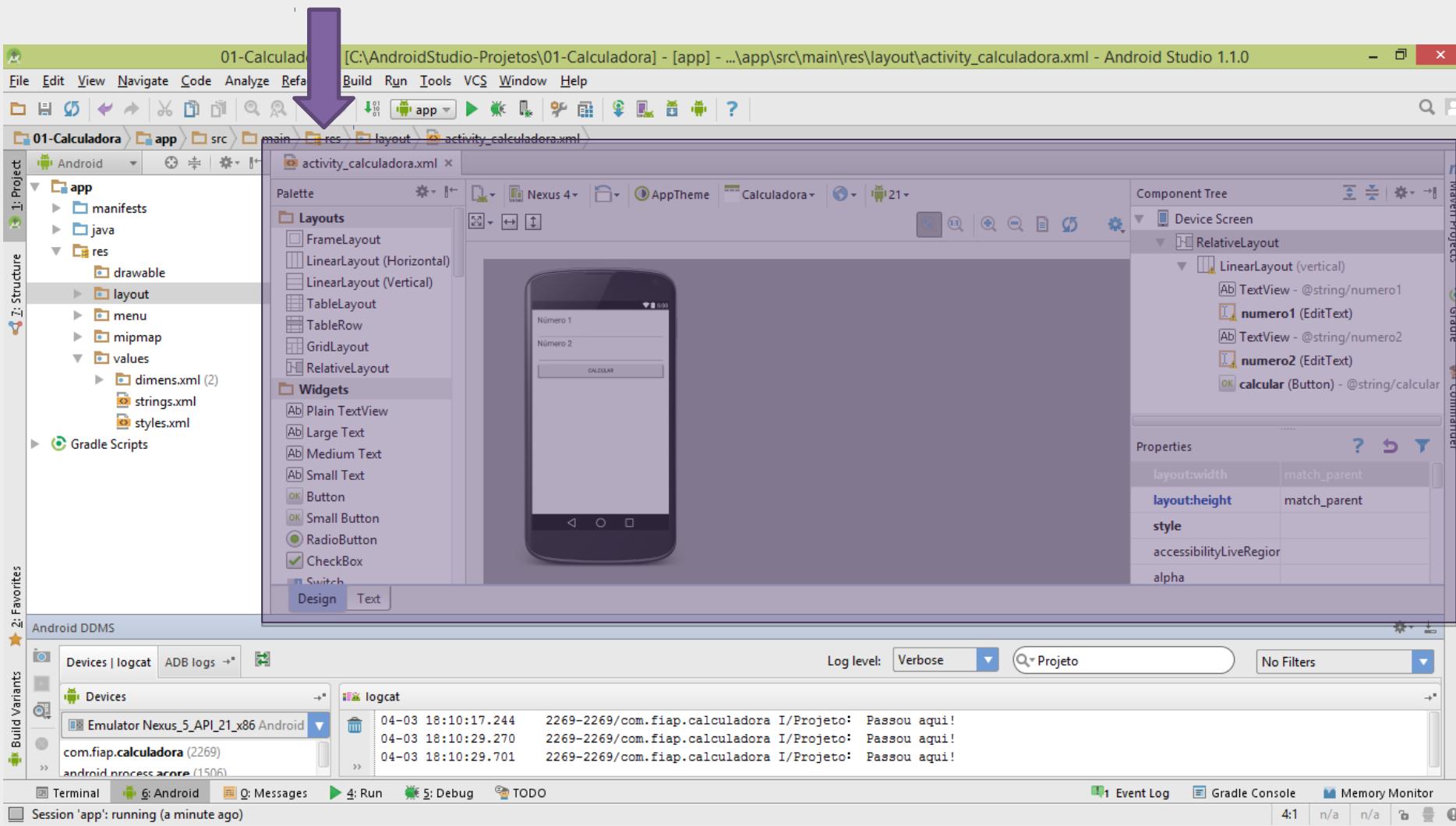
Projeto e seus respectivos arquivos



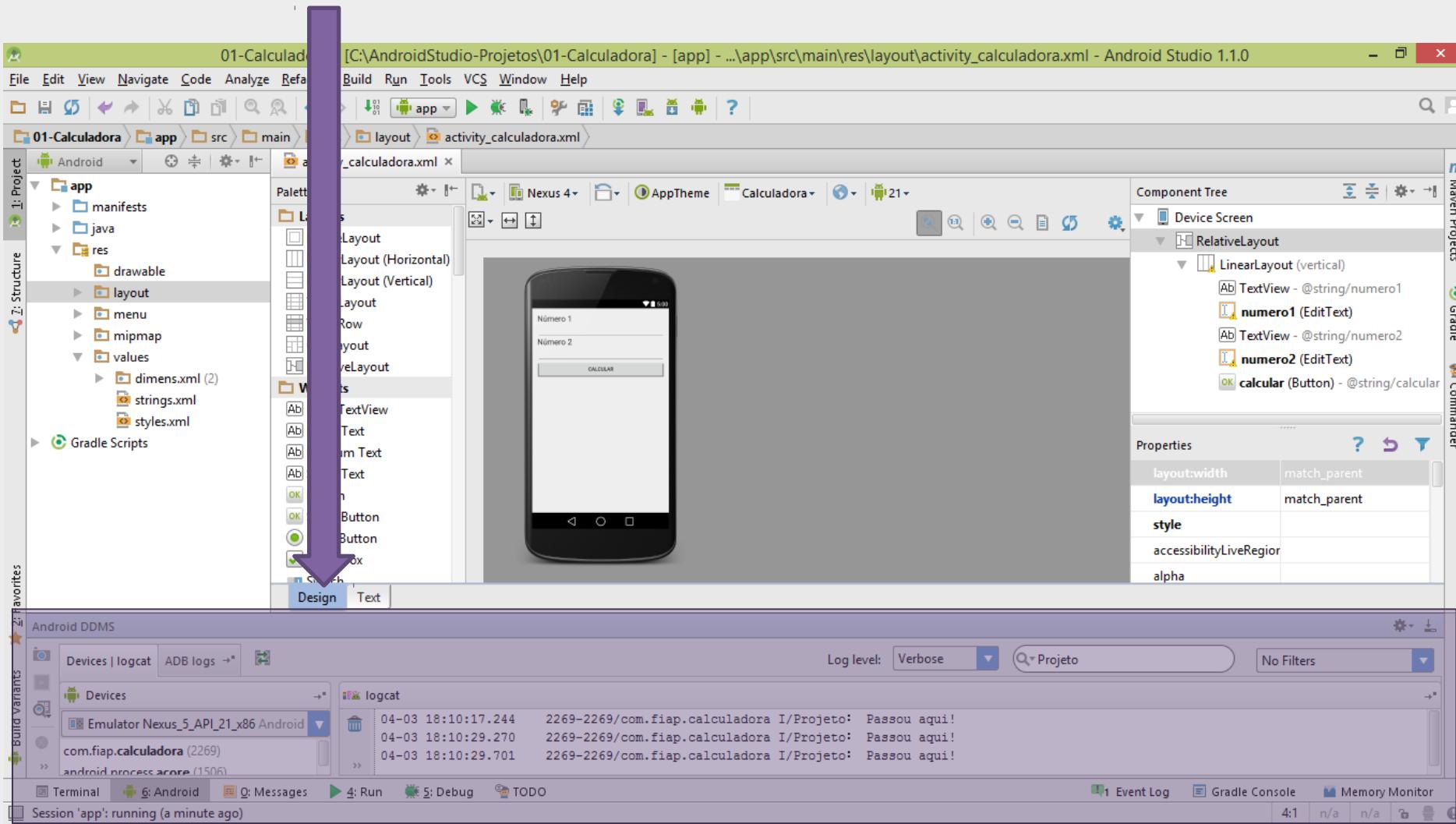
AMBIENTE DE DESENVOLVIMENTO

FIAP

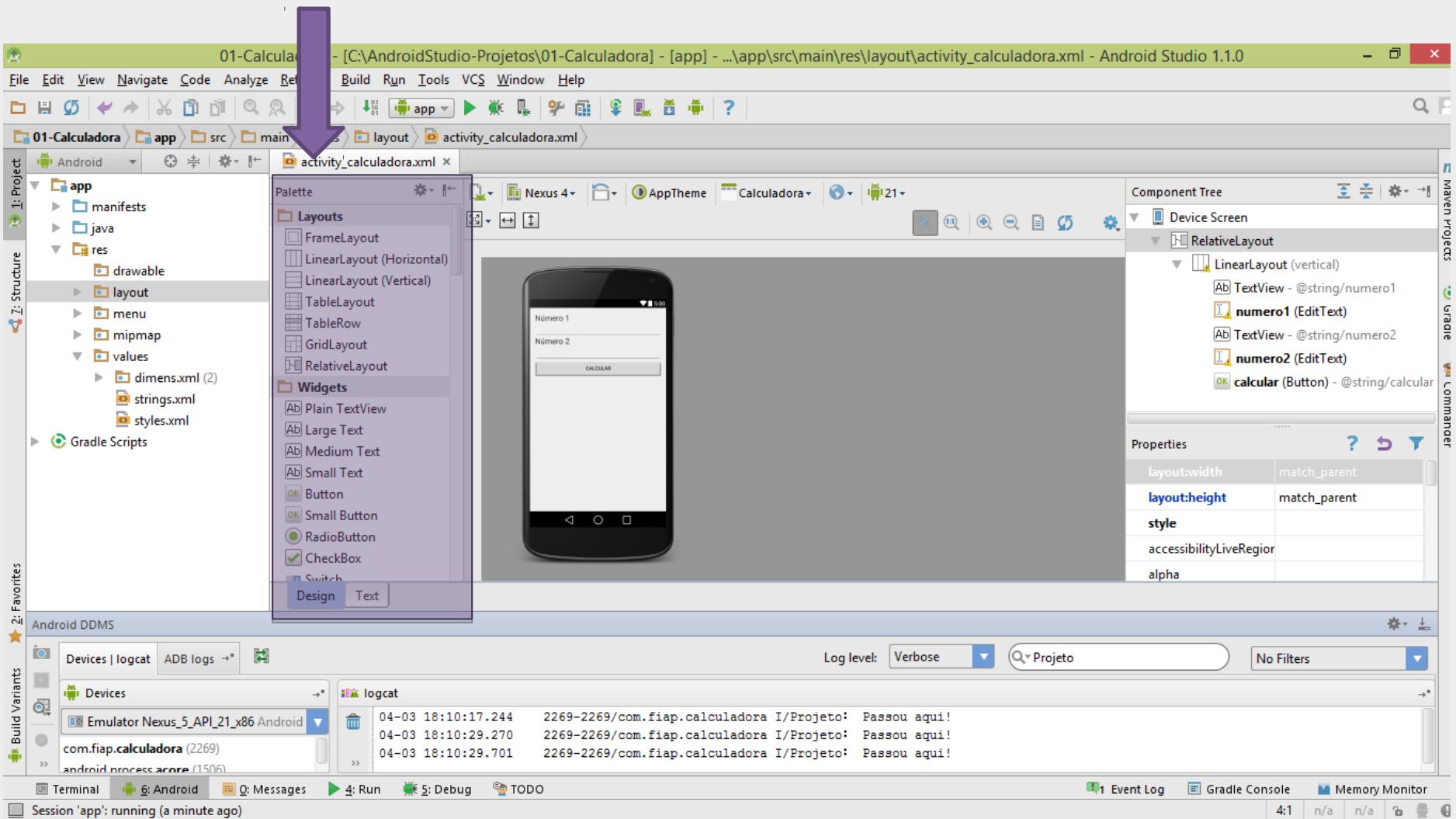
Editor de fontes e layout das aplicações



Mensagens de logs



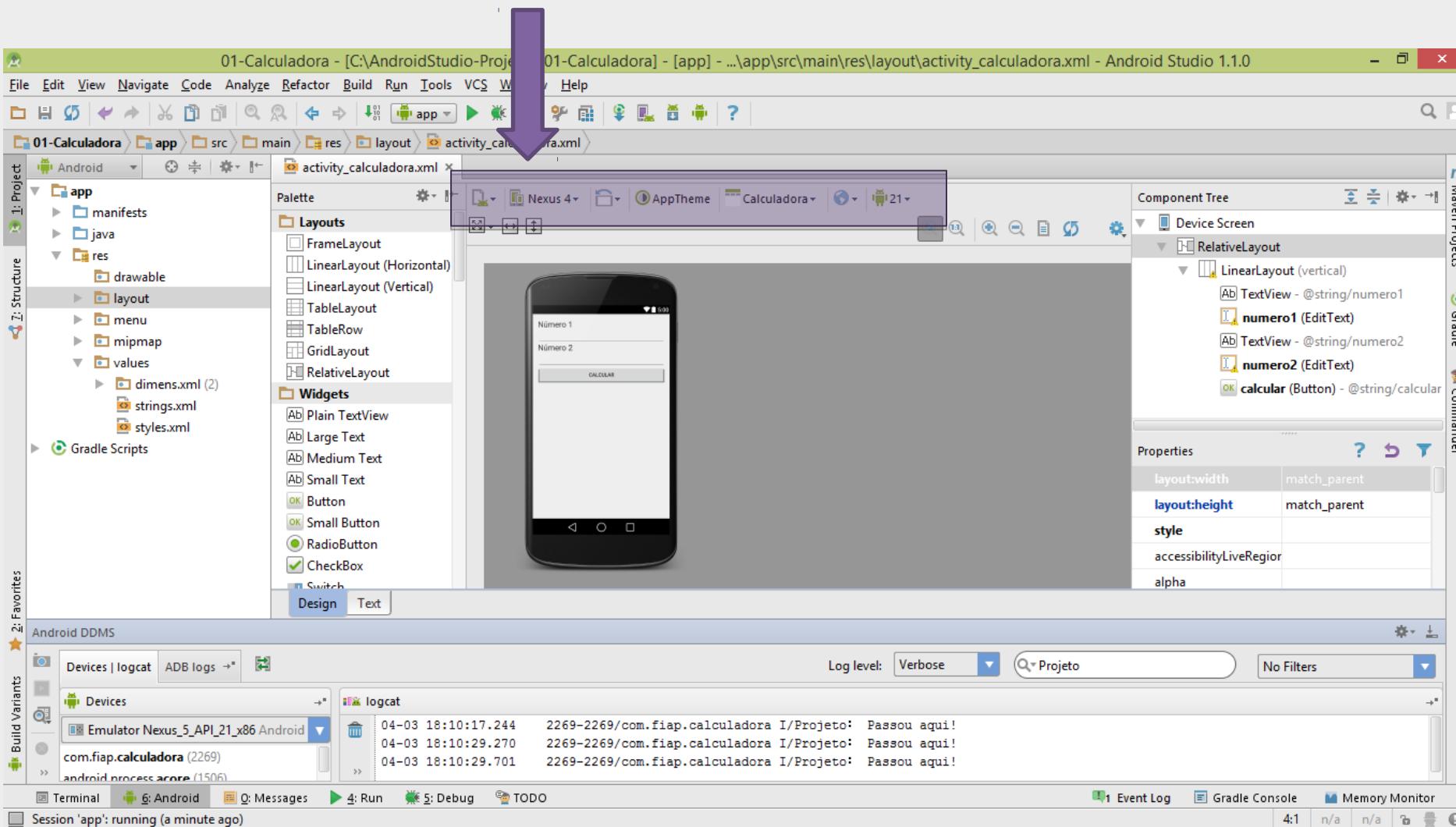
Componentes de Interface (Textos, Botões, Caixas de Seleção..)



AMBIENTE DE DESENVOLVIMENTO

FIAP

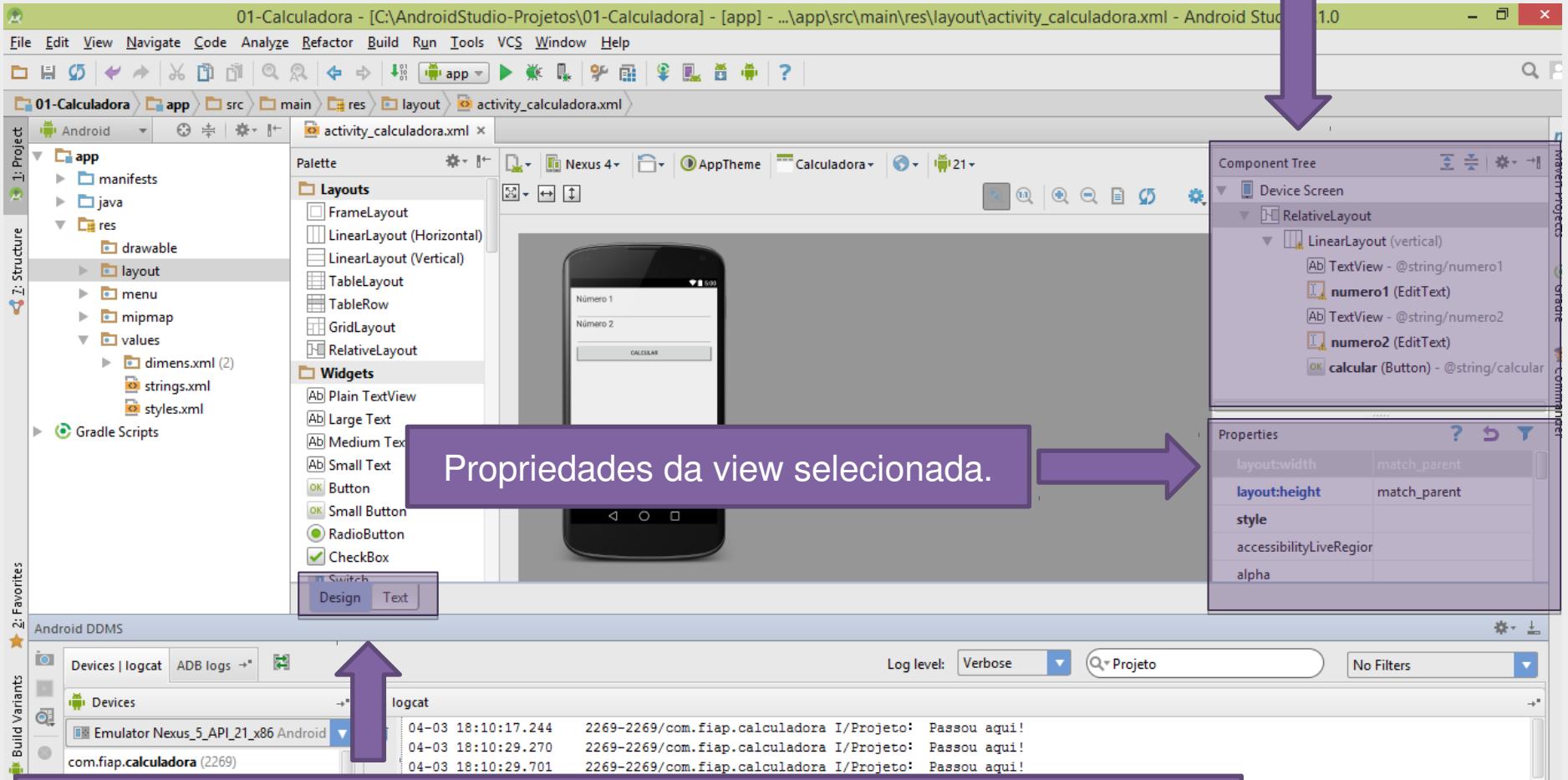
Modelo do dispositivo para a pré-visualização da interface gráfica do app, pode-se configurar a resolução do dispositivo, visualização (retrato ou paisagem), tema e nível da API.



AMBIENTE DE DESENVOLVIMENTO

FIAP

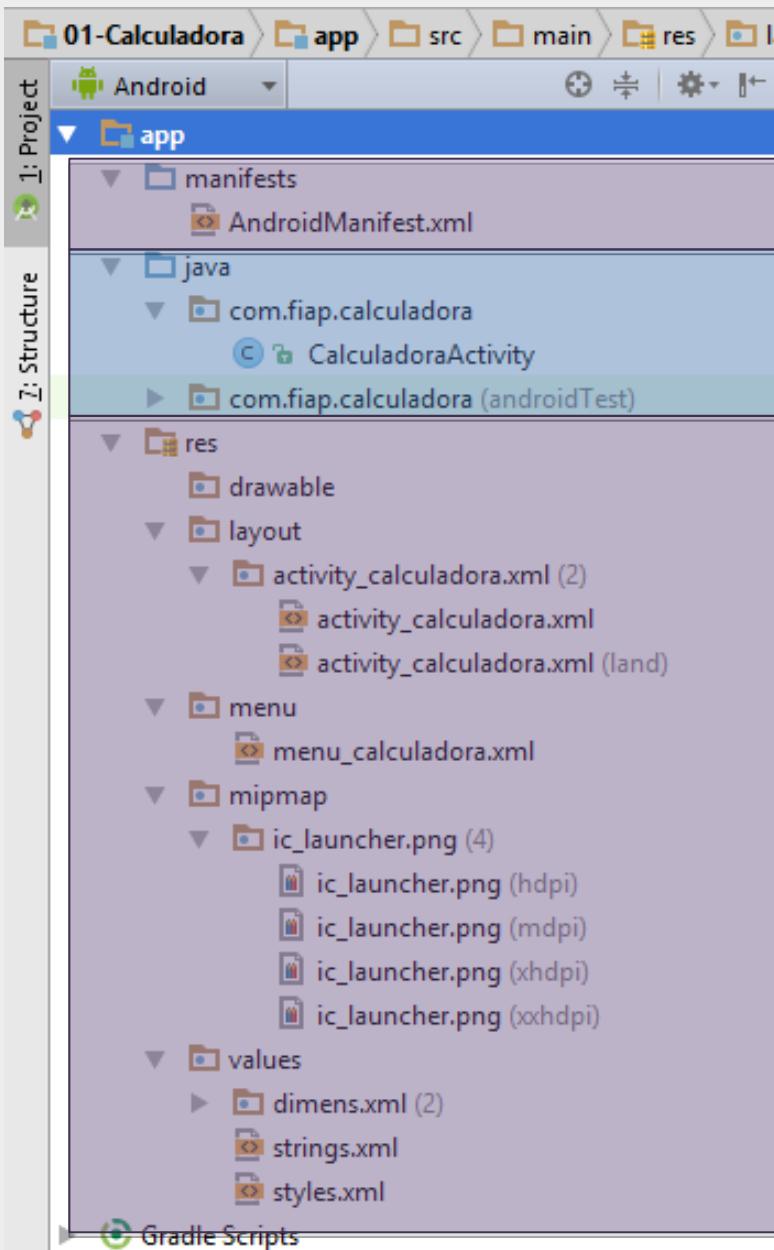
Árvore de views da interface gráfica.



Forma de visualização da interface: forma gráfica ou texto (código fonte)

ESTRUTURA DO PROJETO

FIAP



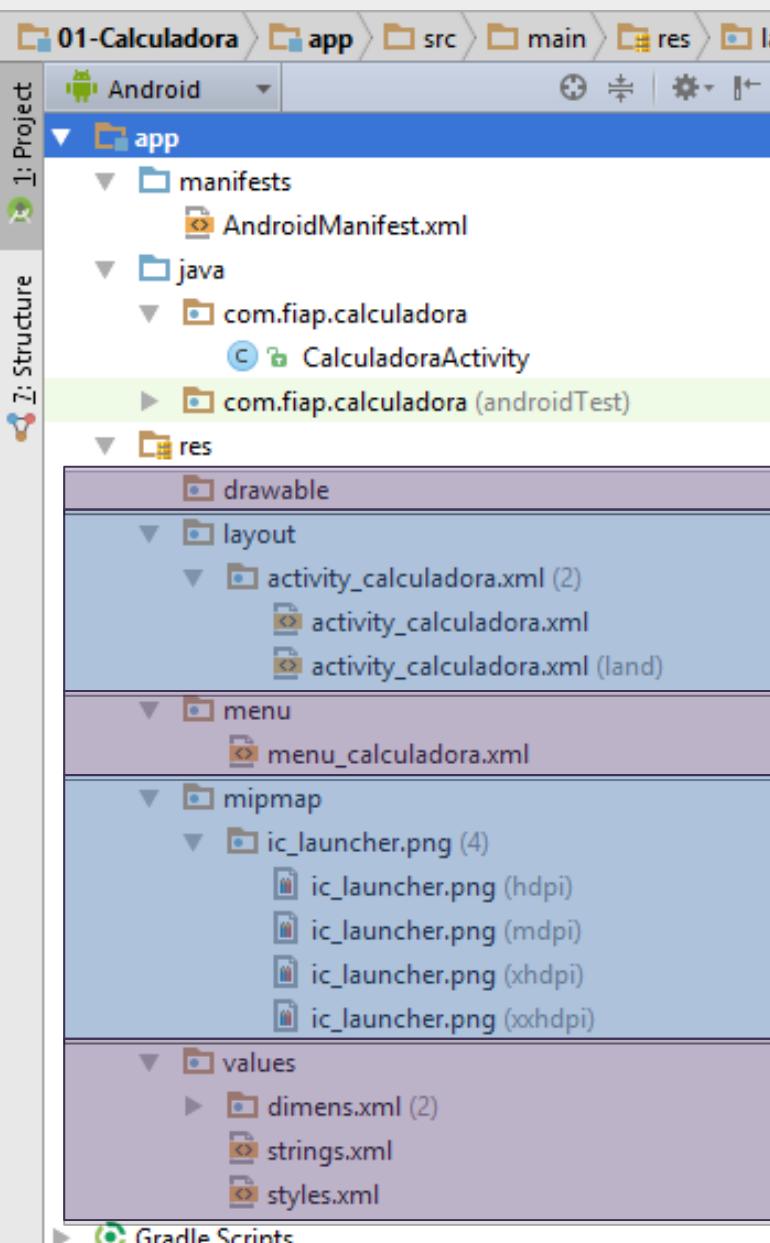
Android Manifest, arquivo de configuração do app (permissões, tema, ícone, telas, etc..)

Pasta Java:
Código fonte Java do app.
Existe um pacote principal, porém outros podem ser criados.
O pacote (AndroidTest) armazena as classes de Teste do app.

Pasta res:
Armazena os recursos utilizados pelo app (imagens, layouts de tela, itens de menu, folhas de estilo, textos e etc..)

ESTRUTURA DO PROJETO

FIAP



Pasta para imagens, que é dividida em vários níveis de resolução (dpi). Importar imagem: new → Asset Image

Recursos das views que compõe as telas do app.

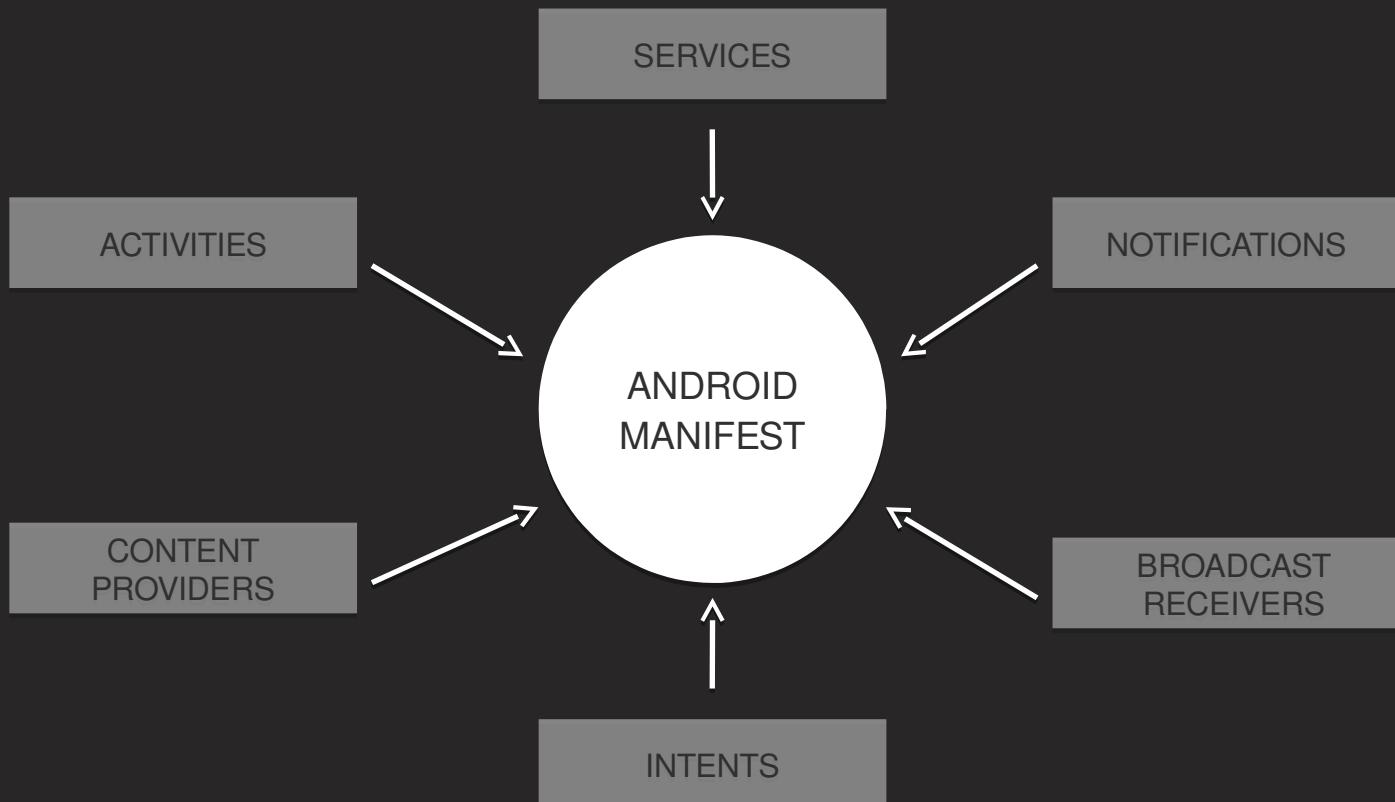
Recursos de menu do app.

Armazena imagens para o ícone do app.

Recursos como dimensões, textos e estilos.

PRINCIPAIS COMPONENTES DA APLICAÇÃO

FIAP



Android Manifest é um arquivo XML (*AndroidManifest.xml*) que define e integra os componentes de uma aplicação vistos acima.

Activities - representam a camada visual da aplicação onde estão contidos os controles de interface (botões, caixas de texto, etc) chamados de Views;

Services - componentes não visuais que executam processamento em background;

Notifications - permitem disparar determinadas ações que notificam o usuário sobre a ocorrência de um evento - como o recebimento de uma mensagem de texto;

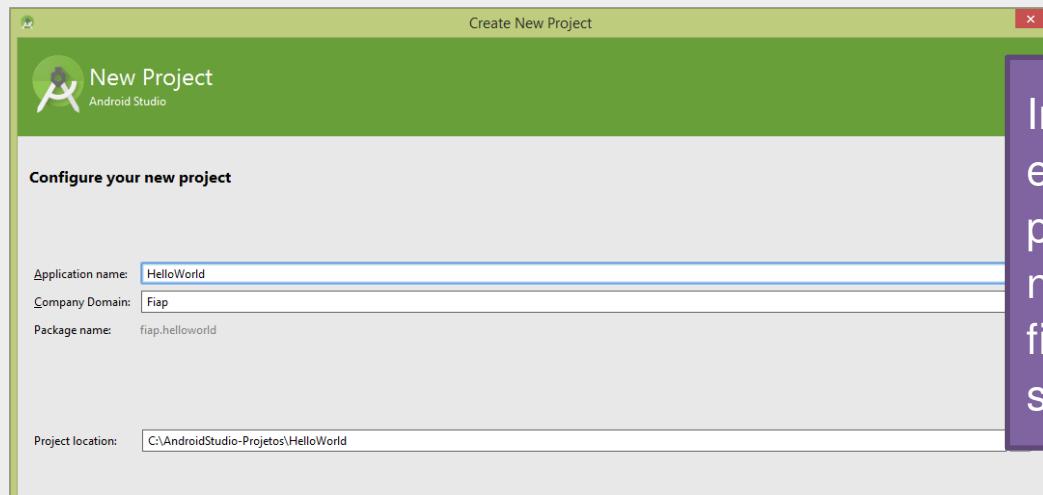
Intents - são mensagens que podem ser trocadas entre aplicações ou entre diversas Activities da mesma aplicação;

Broadcast Receivers - são os receptores das Intents. Uma vez que um Intent é enviado algum Broadcast Receiver deverá ser acionado;

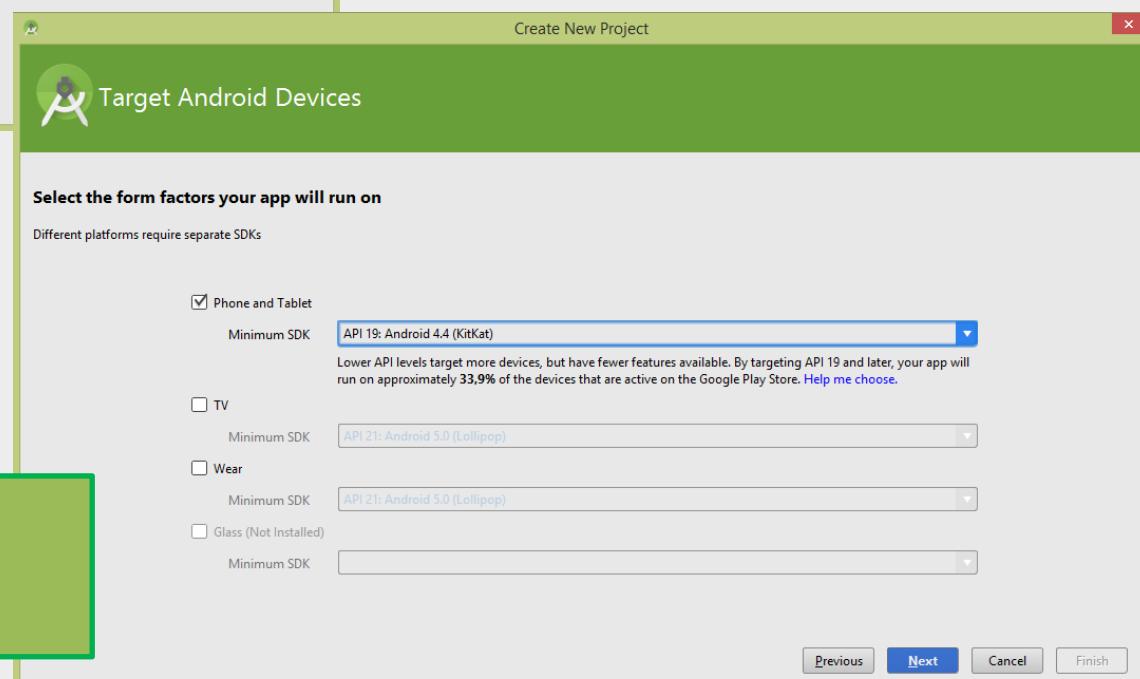
Content Providers -utilizados para armazenamento de informações que podem ser compartilhadas entre aplicações distintas.

PRIMEIRA APLICAÇÃO

FIAP



Informar o nome do app, o nome da empresa (que será utilizado para criar o pacote de classe), é possível alterar o nome do pacote através do link Edit. Por fim, selecione o diretório onde o projeto será criado.



Selecione o tipo de dispositivo e a versão do Android (API).

PRIMEIRA APLICAÇÃO

FIAP

The screenshot shows the App Inventor 'Create New Project' interface. On the left, a sidebar titled 'Add an activity to Mobile' lists several activity templates: 'Add No Activity', 'Blank Activity' (selected), 'Blank Activity with Fragment', 'Fullscreen Activity', 'Google Maps Activity', and 'Custom Activity' (represented by a puzzle piece icon). Below these are two more complex templates involving fragments and lists. On the right, a main window titled 'Customize the Activity' is open, showing settings for creating a new blank activity with an action bar. The 'Activity Name' field contains 'HelloActivity'. The 'Layout Name' field contains 'activity_hello'. The 'Title' field contains 'HelloActivity'. The 'Menu Resource Name' field contains 'menu_hello'. A purple callout box points to the 'Blank Activity' template in the sidebar with the text: 'Selecione um template para a tela inicial do app. Neste caso, uma tela em branco (Empty Activity)'. A green callout box at the bottom left points to the 'HelloActivity' field in the customization dialog with the text: 'Informar o nome da classe principal, os demais campos serão preenchidos automaticamente.'

Informar o nome da classe principal,
os demais campos serão preenchidos
automaticamente.

Selecione um template para a tela inicial
do app. Neste caso, uma tela em branco
(Empty Activity)

Create New Project

Add an activity to Mobile

Add No Activity

Blank Activity

Blank Activity with Fragment

Fullscreen Activity

Google Maps Activity

Custom Activity

Creates a new blank activity with an action bar.

Activity Name: HelloActivity

Layout Name: activity_hello

Title: HelloActivity

Menu Resource Name: menu_hello

Blank Activity

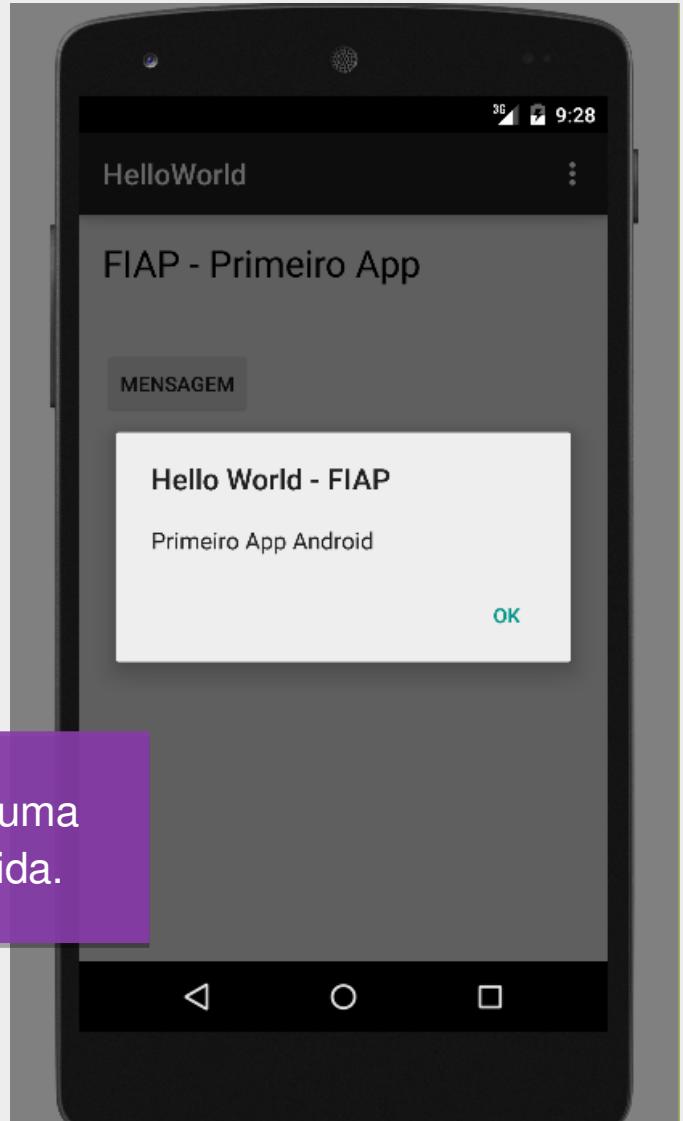
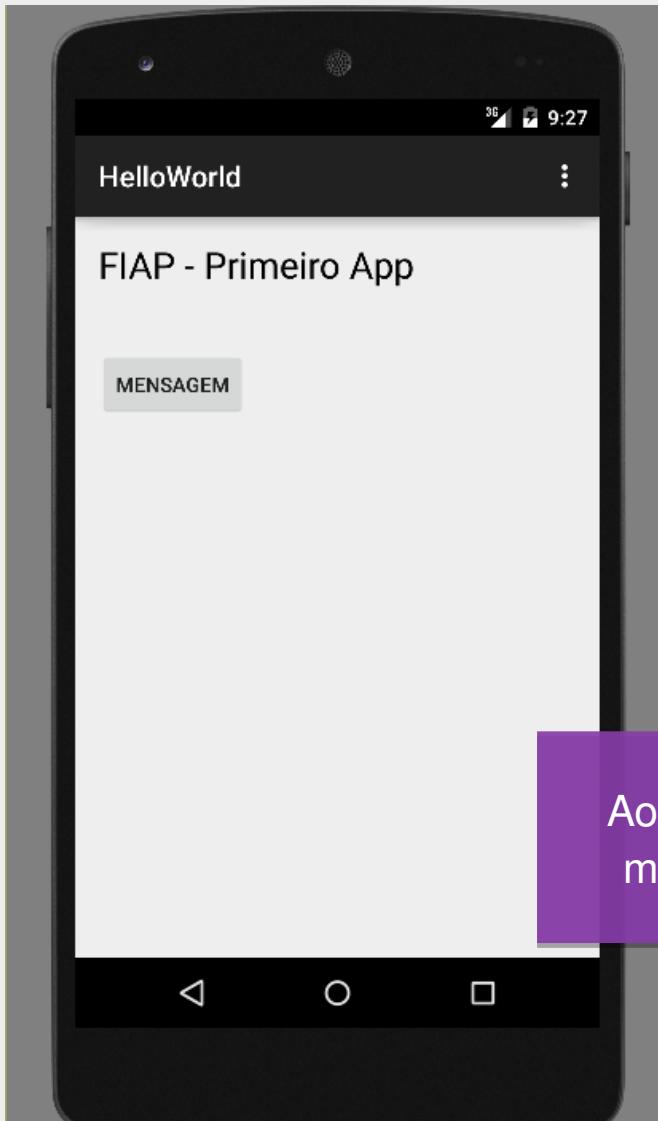
The name of the activity class to create

Previous Next Cancel Finish

PRIMEIRA APLICAÇÃO

FIAP

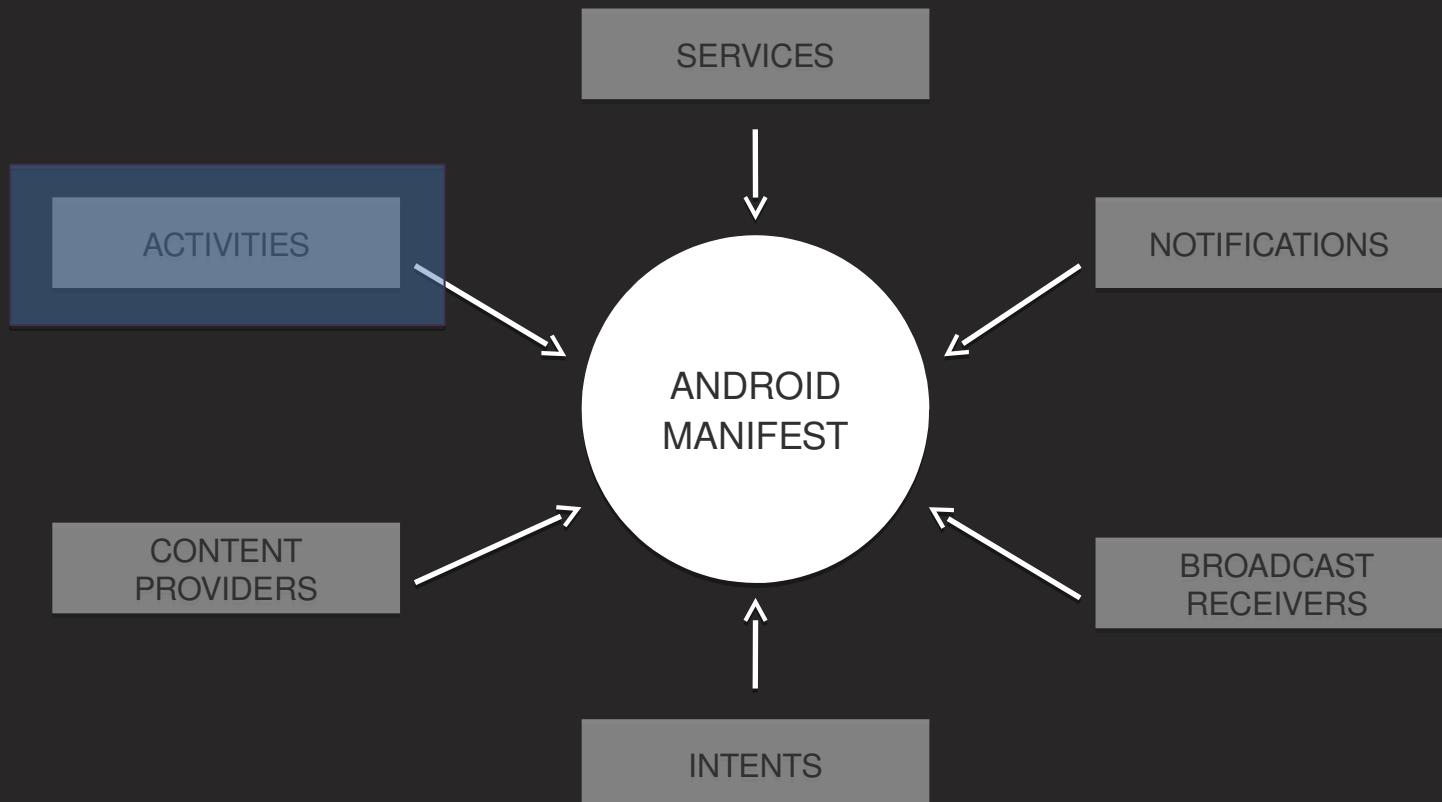
A aplicação inicial:



Ao clicar no botão uma
mensagem é exibida.

PRINCIPAIS COMPONENTES DA APLICAÇÃO

FIAP



Representa uma tela onde os usuários podem interagir;

Um *app* pode conter mais de uma Activity;

O desenho da tela pode ser especificado em um arquivo xml que está dentro da pasta layout;

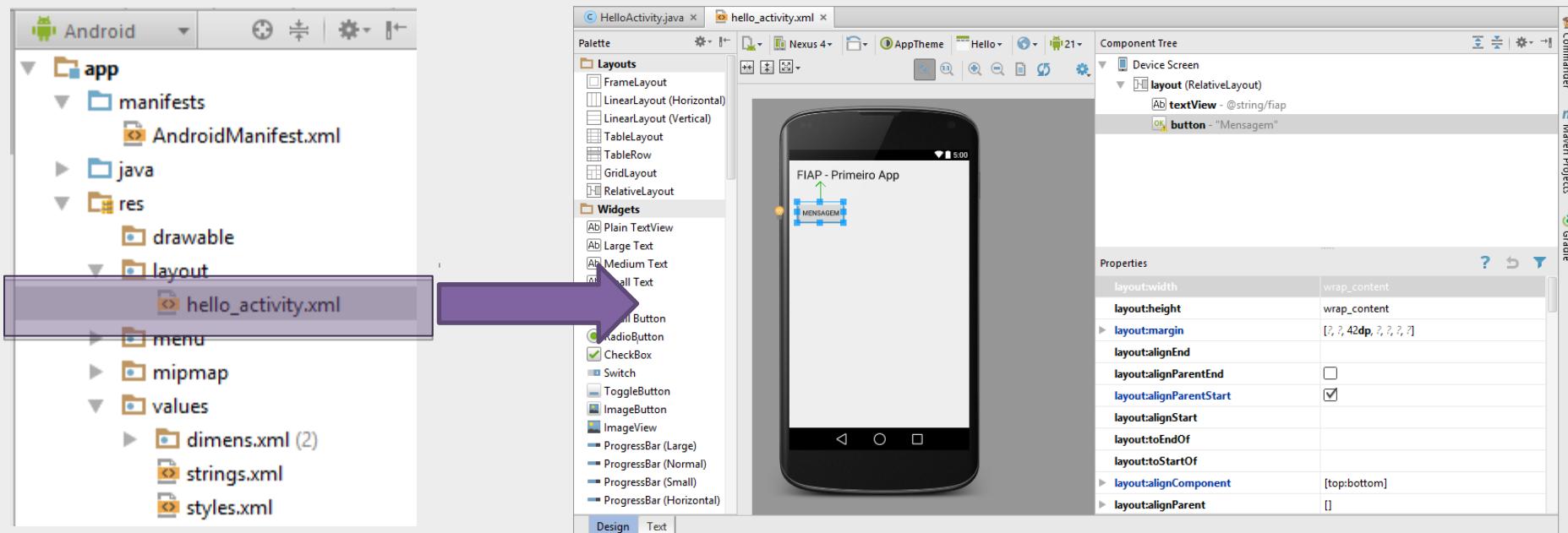
Uma Activity deve ser a principal, que será a primeira a ser executada;

Uma Activity pode chamar outras activities, que estão dentro do próprio *app* ou de outras aplicações.

PRIMEIRA APLICAÇÃO

FIAP

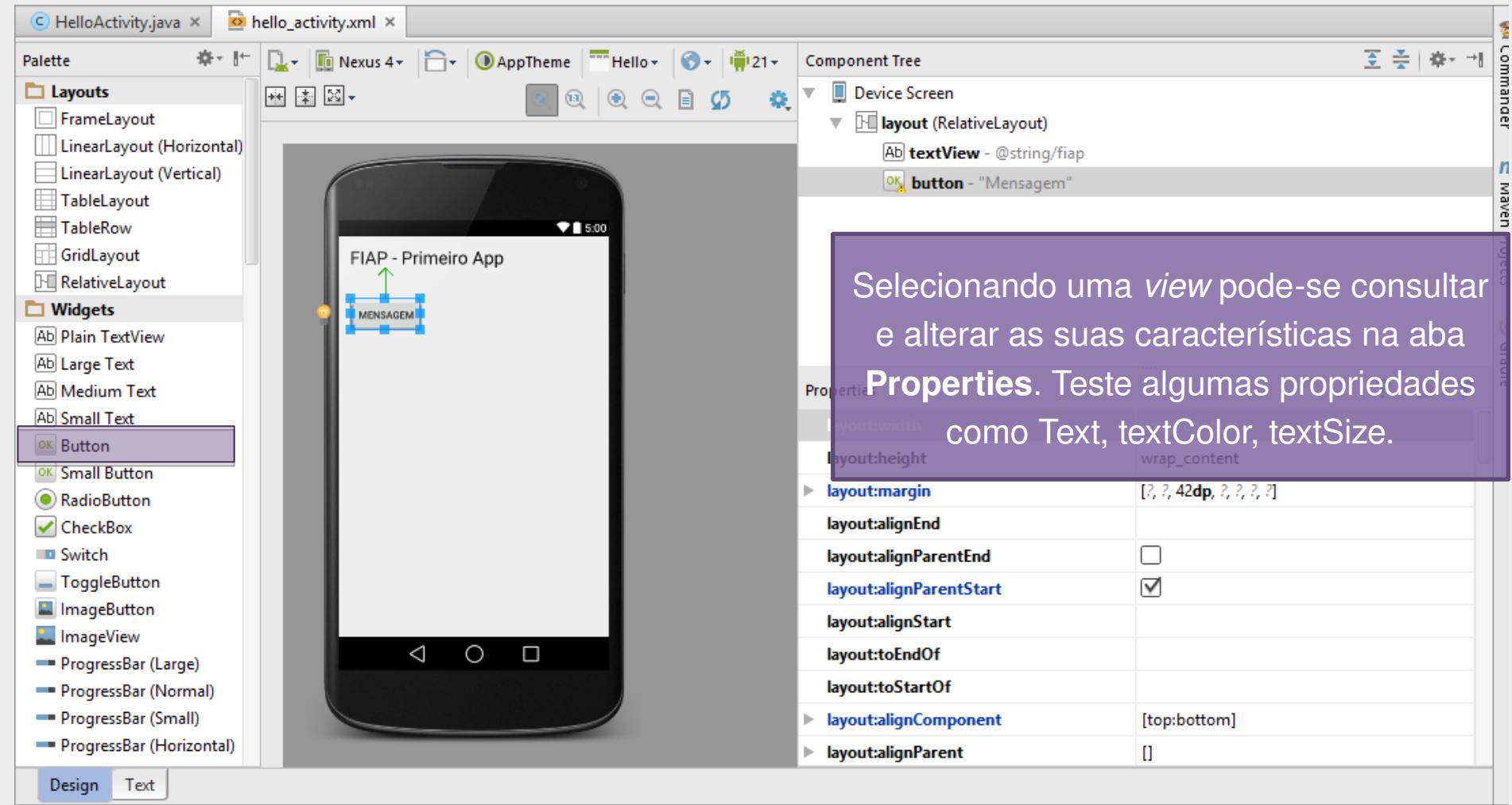
Criar a interface editando o arquivo **activity_main.xml**.



PRIMEIRA APLICAÇÃO

FIAP

Editar as propriedades das views.



PRIMEIRA APLICAÇÃO

FIAP

Codificação.

Uma Activity pode ser vista como uma janela da aplicação.

```
public class HelloActivity extends ActionBarActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.hello_activity);  
    }  
  
    public void exibir(View v){  
        }  
    }  
}
```

Neste ponto o layout é criado

Método que será acionado no clique do botão

PRIMEIRA APLICAÇÃO

FIAP

Passo 2: Editar as propriedades das views.

The screenshot shows the Android Studio interface. On the left, there's a preview of a smartphone displaying the application with the title "FIAP - Primeiro App" and a text view containing "MENSAGEM". On the right, the "layout (RelativeLayout)" is selected in the hierarchy. The properties panel at the bottom shows the "onClick" property set to "exibir". A purple callout box with a downward arrow points to the "onClick" dropdown menu, which lists three options: "<unset>", "setContentView (android.support.v7.app.ActionBarActivity)", and "exibir (fiap.helloworld.HelloActivity)". The third option is highlighted in blue. A purple text box contains the instruction: "Defina o método que será chamado quando o botão for clicado, através da propriedade onClick."

Device Screen

layout (RelativeLayout)

textView - @+id/tvMensagem

button - "MENSAGEM"

Properties

nestedScrollingEnabled

onClick

- <unset>
- setContentView (android.support.v7.app.ActionBarActivity)
- exibir (fiap.helloworld.HelloActivity) **selected**
- setListFooter (android.preference.PreferenceActivity)

shadowColor

singleLine

stateListAnimator

text Mensagem

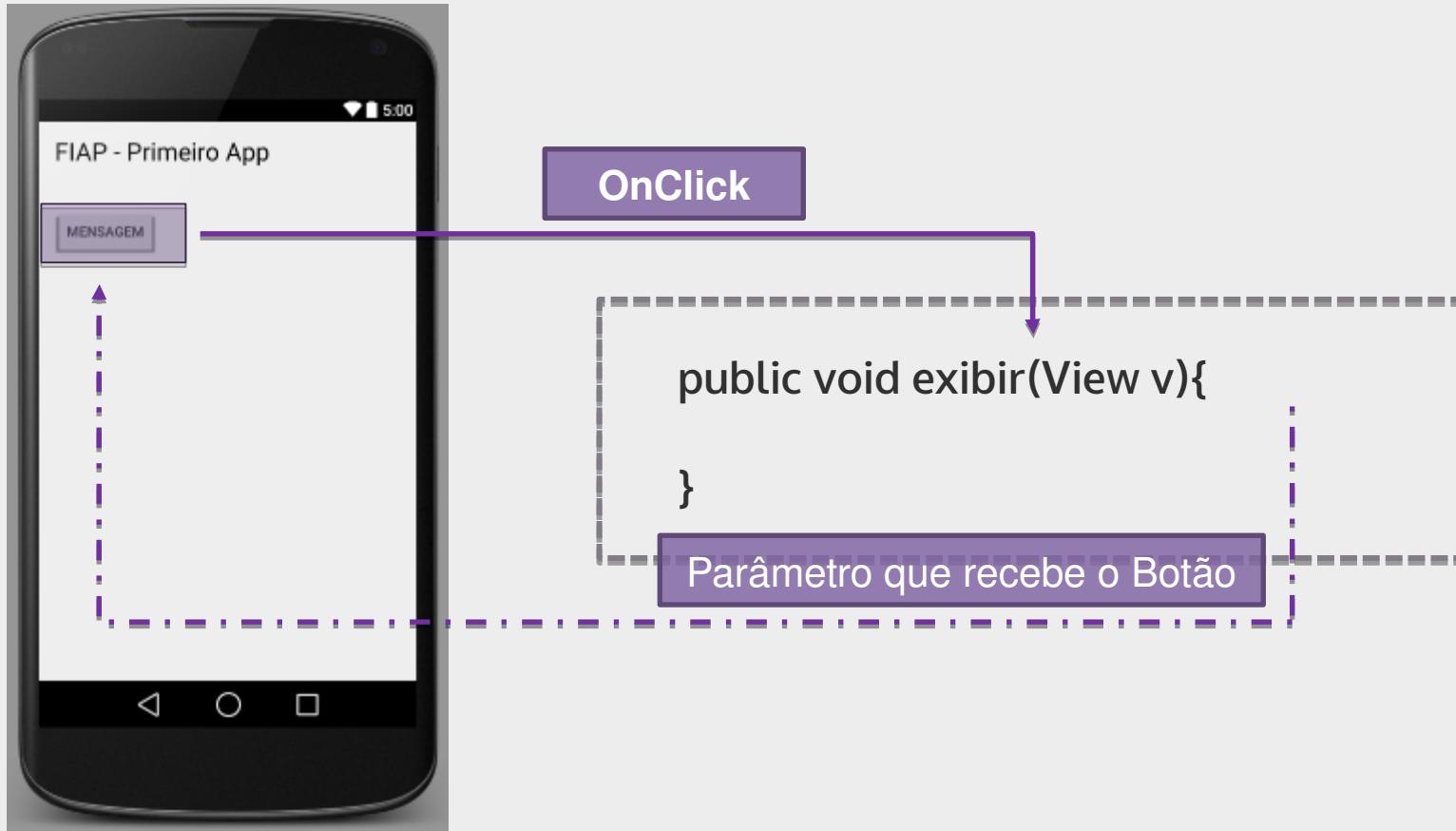
textAlignment

Defina o método que será chamado quando o botão for clicado, através da propriedade onClick.

PRIMEIRA APLICAÇÃO

FIAP

Método onClick:



Os métodos que implementam eventos de click devem ser públicos, sem retorno (void) e receber um parâmetro do tipo View, para receber o elemento da tela que gerou o evento.

Codificação.

```
public class HelloActivity extends ActionBarActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.hello_activity);  
    }  
  
    public void exibir(View v){  
        AlertDialog.Builder alert = new AlertDialog.Builder(this);  
        alert.setTitle("Hello World - FIAP");  
        alert.setMessage("Primeiro App Android");  
        alert.setPositiveButton("OK",null);  
        alert.show();  
    }  
}
```

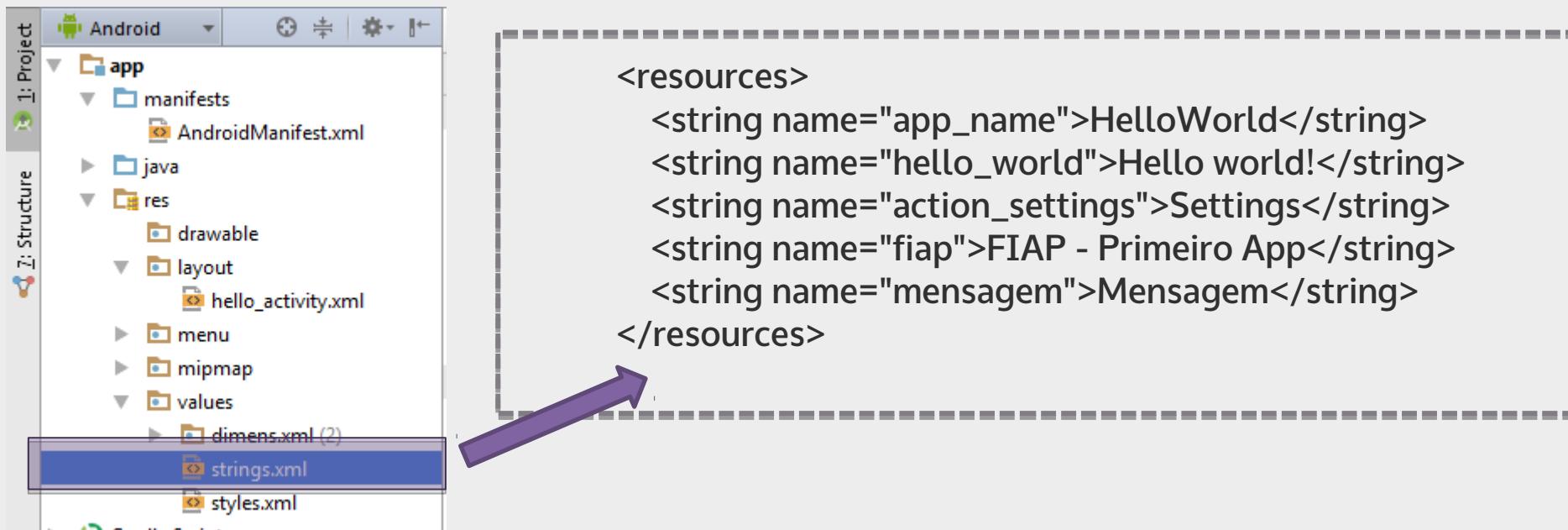
Finalmente, cria-se um AlertDialog para que a mensagem seja exibida no momento em que o método *show()* é acionado.

CRIANDO TEXTOS EXTERNOS

FIAP

No arquivo **strings.xml** todos os textos de uma aplicação podem ser criados (rótulos, mensagens de erro, textos explicativos, etc...)

O formato de **strings.xml** é apresentado abaixo:

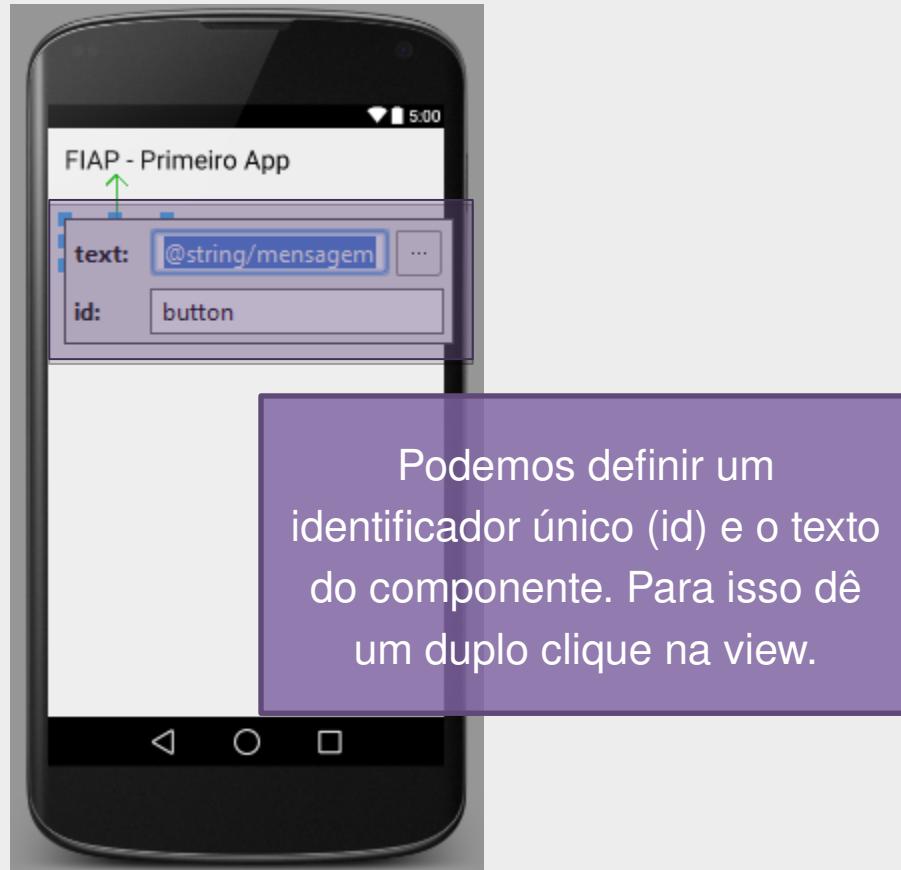


Na tag string pode-se criar um texto que será futuramente referenciado na aplicação pelo valor do atributo **name**. Exemplo: **R.string.mensagem**

Para referenciar um texto a partir de uma propriedade de uma view é só acrescentar o prefixo **@**. Exemplo, na propriedade Text de um Button: **@string/btnMensagem**

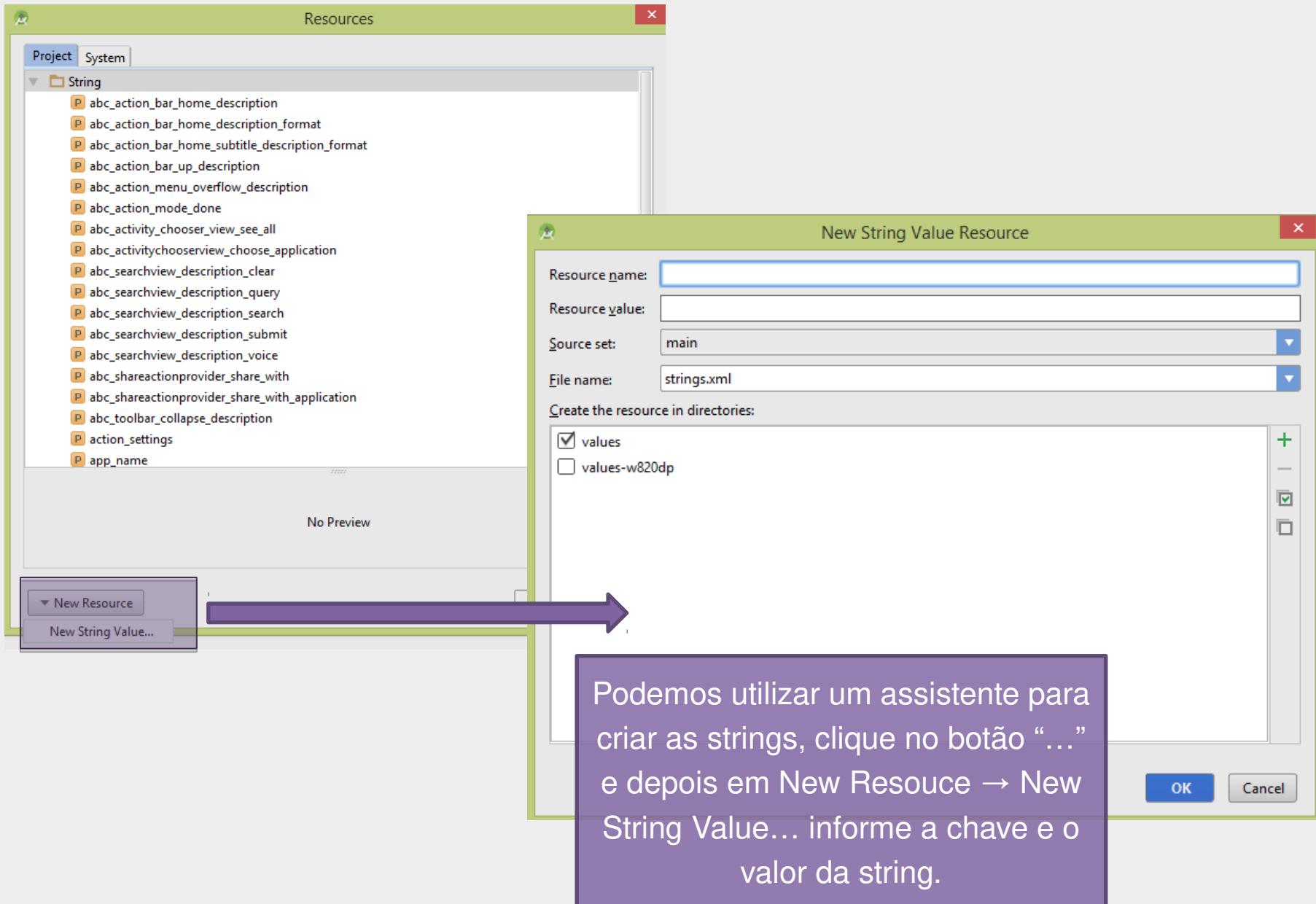
CRIANDO TEXTOS EXTERNOS

FIAP



CRIANDO TEXTOS EXTERNOS

FIAP

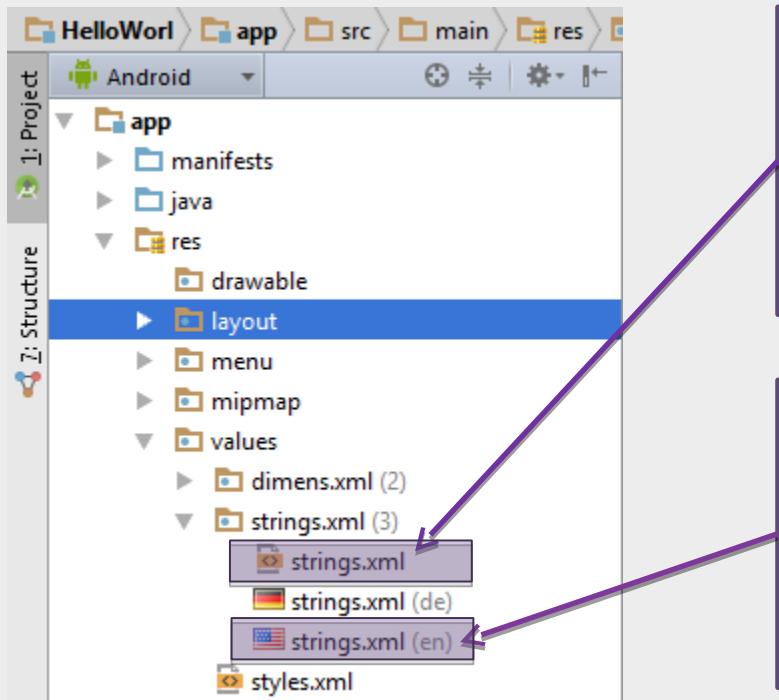


CRIANDO TEXTOS EXTERNOS

FIAP

Os textos exibidos nas aplicações podem ser automaticamente selecionados pelo dispositivo conforme o seu idioma;

Basta criar diversos arquivos strings.xml cada um dentro de uma pasta da língua específica apenas alterando o nome para values-en (Inglês), values-fr (Francês), values-de(Alemão), etc...

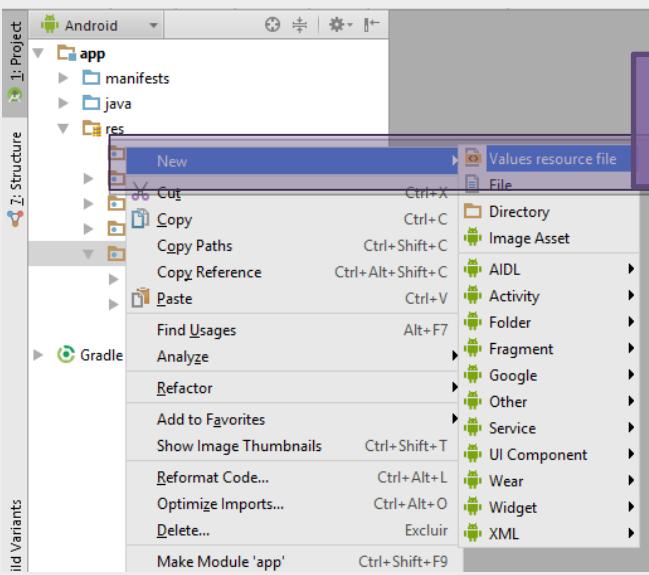


```
<resources>
    <string name="mensagem">Boa Noite</string>
    <string name="btnMensagem">Mostre!</string>
</resources>
```

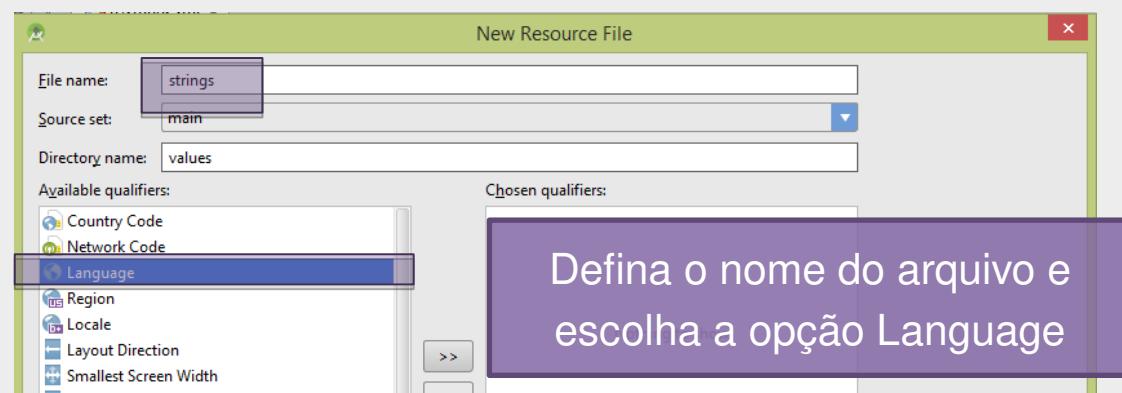
```
<resources>
    <string name="mensagem">Good night!</string>
    <string name="btnMensagem">Show me!</string>
</resources>
```

CRIANDO TEXTOS EXTERNOS

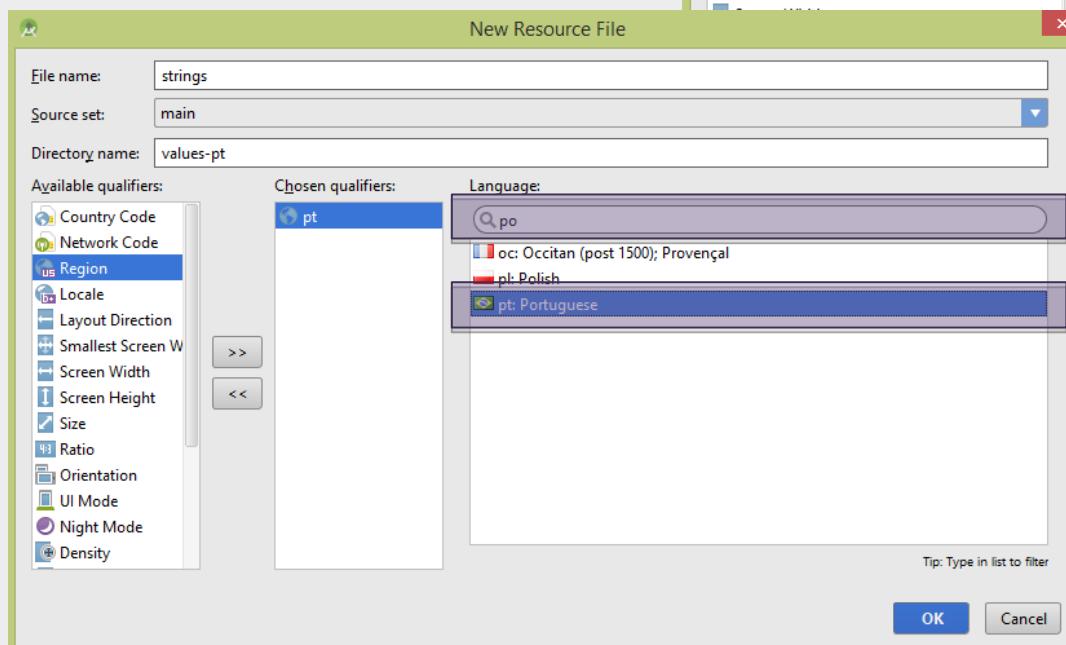
FIAP



Clique com o botão direito do mouse e escolha
New → Values resource file.



Defina o nome do arquivo e
escolha a opção Language



Procure pela linguagem
desejada, para efetuar uma
busca, basta selecionar a lista e
digitar o valor procurado

Para a aplicação que exibe a mensagem implementada anteriormente, crie os textos utilizados na forma de recursos externos (em português, **francês** e **alemão**):

Mensagem = {**Message**, **Nachricht**}

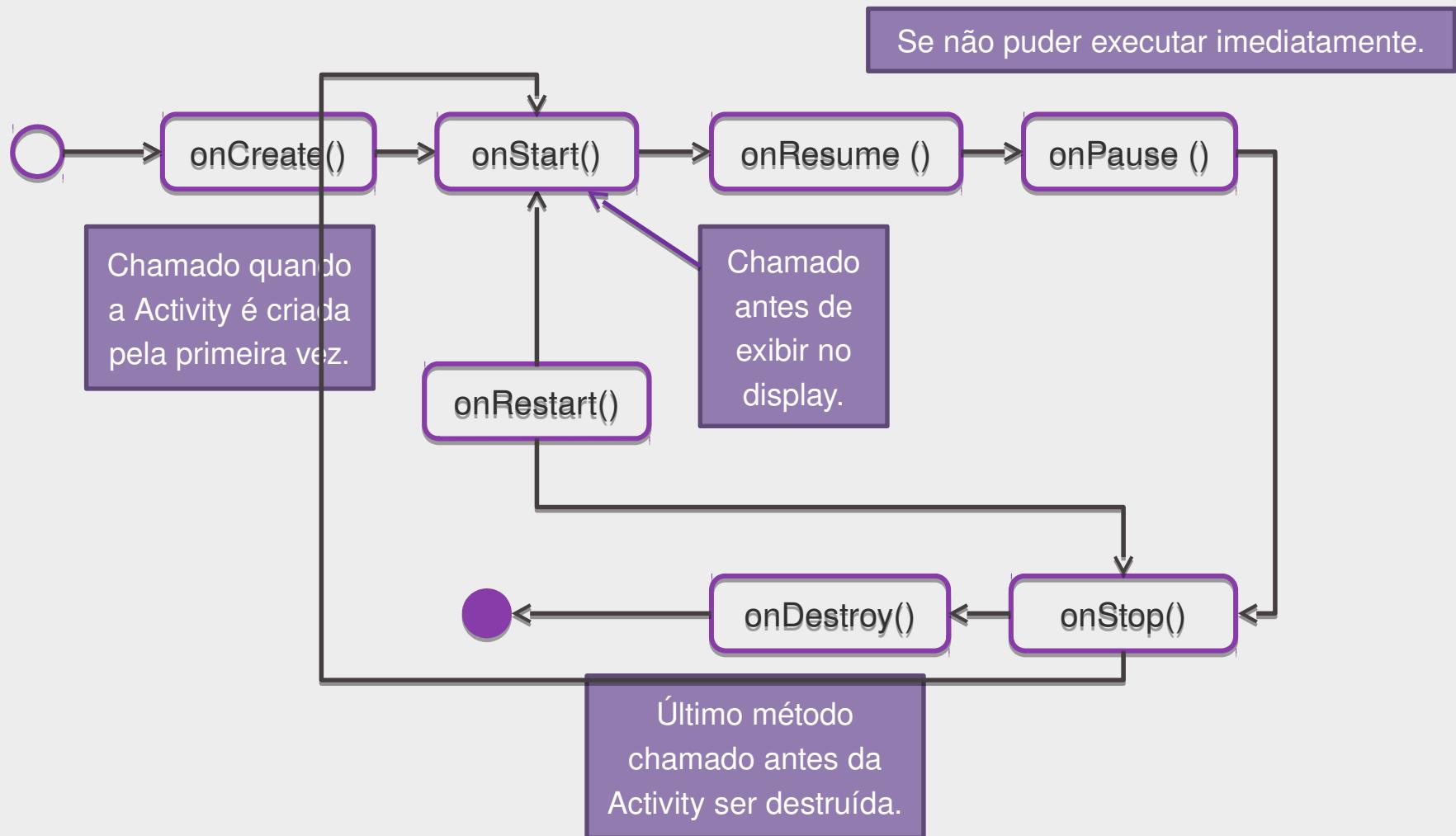
Boa Noite = {**Bonne Nuit**, **Gute Abend**}

Primeira Aplicação Android = {**D'abord l'application Android**, **Erste Android-Anwendung**}

Tradução by Google Translator...

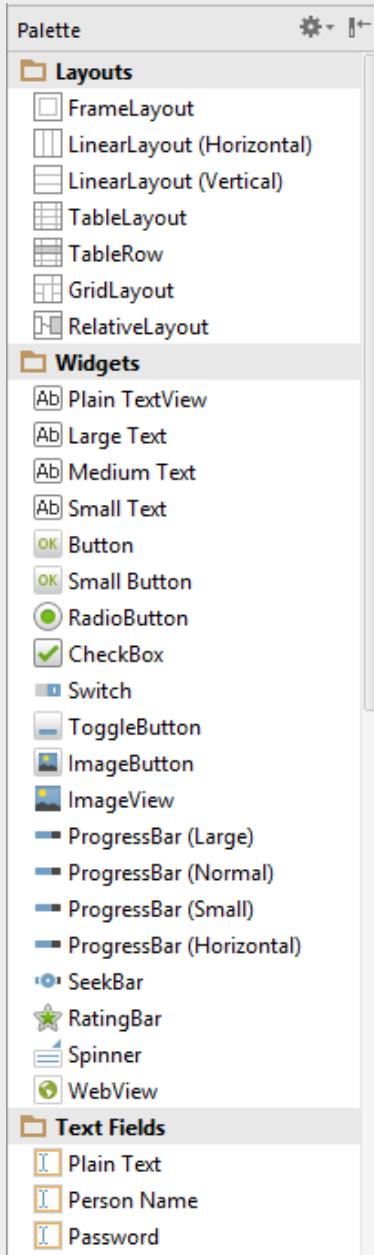
Aproveite e efetue uma pesquisa nas propriedades para descobrir como trocar a imagem de fundo da aplicação associando o ícone **ic_launcher**.

Como a **Activity** é o elemento central de nossa aplicação, é necessário conhecer o seu ciclo de vida:



INTRODUÇÃO

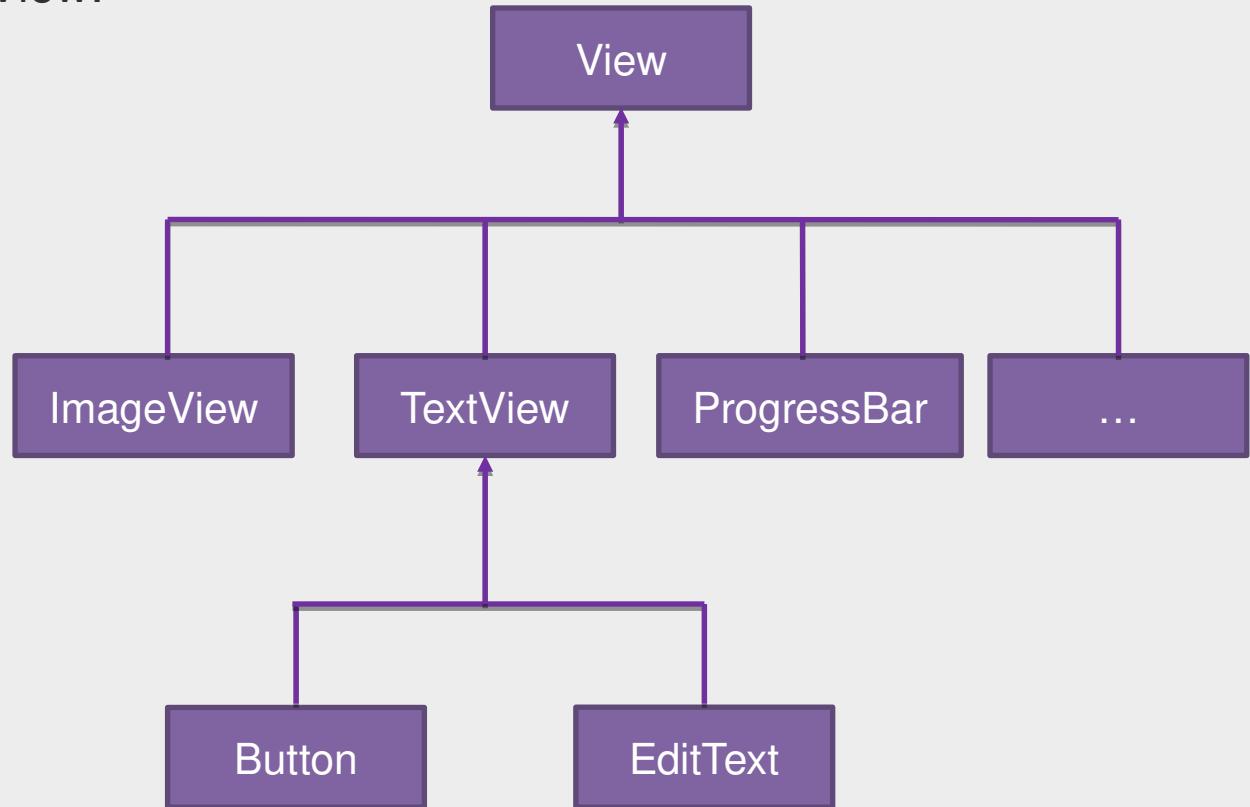
- Introdução aos conceitos de Mobilidade com Android
- Introdução ao Android e a Linguagem de Programação Java
- Utilizando o Android Studio
- Estrutura do sistema operacional Android
- Desenvolvimento do primeiro aplicativo
- Instalando a app no smartphone
- Definição de Strings (Internacionalização)
- **Interface Gráfica e Associação a Eventos**
- **Utilização da classe Activity**
- **Utilização de Intents**
- **Back Stack**
- **Introdução as Views**
- **Utilização dos componentes Button, TextView, EditText e ImageView**
- **Utilização dos componentes Checkbox, RadioGroup, Spinner, ListView e Dialog**



Views são elementos da interface gráfica que compõem a parte visual de uma Activity;

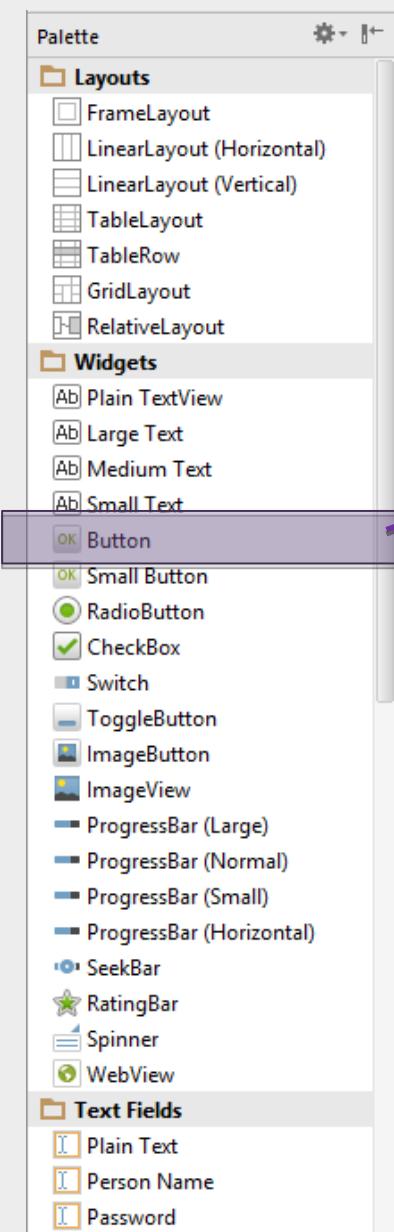
São as caixas de textos, botões, imagens e etc...

Todas as views são subclasses diretas ou indiretas da classe View:



VIEWS

FIAP



```
package fiap.helloworld;

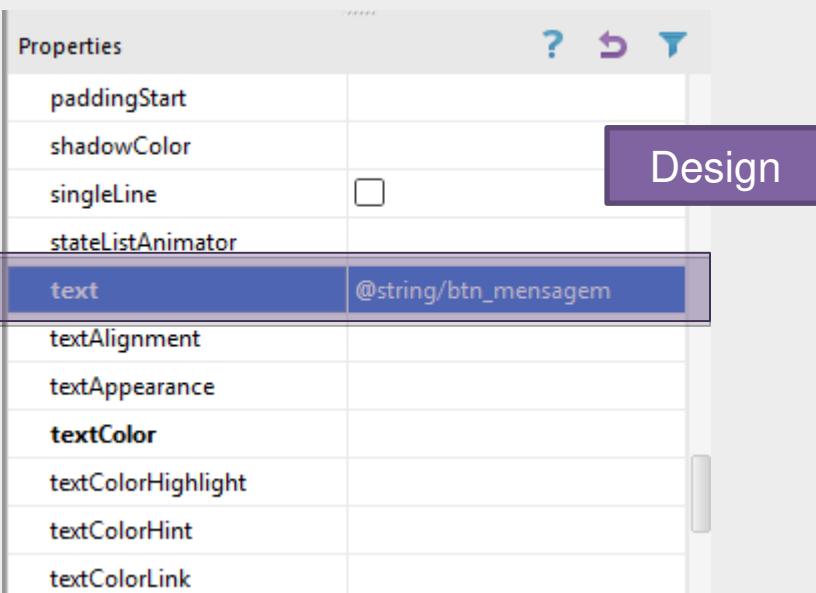
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.widget.Button;

public class HelloActivity extends ActionBarActivity {
}
```

- As *views* podem ter suas características alteradas por meio de suas propriedades;
- Existem propriedades que são comuns a todas as views e algumas que são específicas;
- Na maioria dos casos, para alterar uma propriedade dentro do código basta utilizar métodos get e set seguidos pelo nome da propriedade (ex: getId(), setText(), etc...).

PROPRIEDADES DAS VIEWS

FIAP



Text

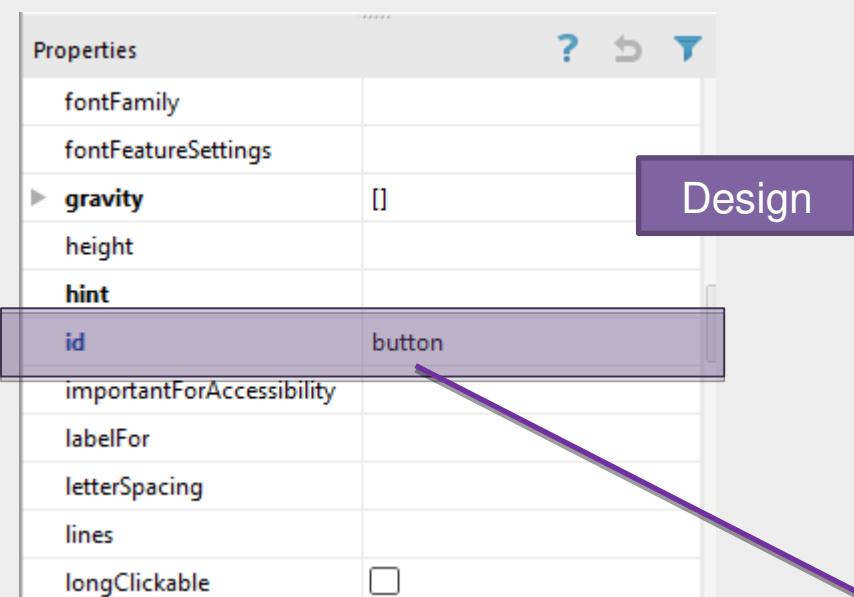
```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/btn_mensagem"  
    android:id="@+id/button"  
    android:layout_marginTop="42dp"  
    android:layout_below="@+id/textView"  
    android:layout_alignParentStart="true"  
    android:onClick="exibir" />
```

Java

```
public class HelloActivity extends ActionBarActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.hello_activity);  
        Button button = (Button) findViewById(R.id.button);  
  
        button.setText(R.string.btn_mensagem);  
        String texto = button.getText().toString();  
    }  
}
```

PROPRIEDADES DAS VIEWS

FIAP



```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/btn_mensagem"  
    android:id="@+id/button"  
    android:layout_marginTop="42dp"  
    android:layout_below="@+id/textView"  
    android:layout_alignParentStart="true"  
    android:onClick="exibir" />
```

Text

```
public class HelloActivity extends ActionBarActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

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

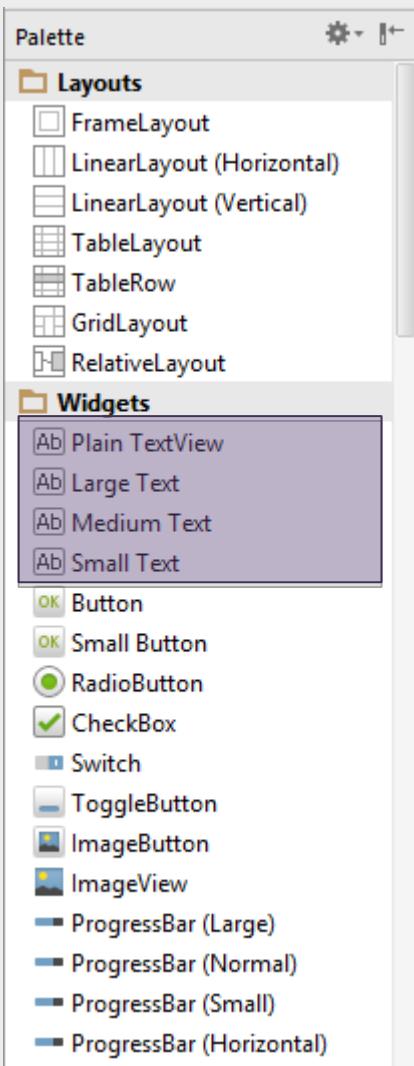
```
        Button button = (Button) findViewById(R.id.button);
```

```
        button.setText(R.string.btn_mensagem);
```

```
        String texto = button.getText().toString();
```

Java

Para recuperar uma view na classe Java, devemos utilizar o método **findViewById**, passando o **id** da view.



Permite a visualização de textos, sem a edição;

Para definir o texto a ser exibido, basta alterar a propriedade `text`;

A classe `TextView` possui os métodos `getText()` e `setText()`;

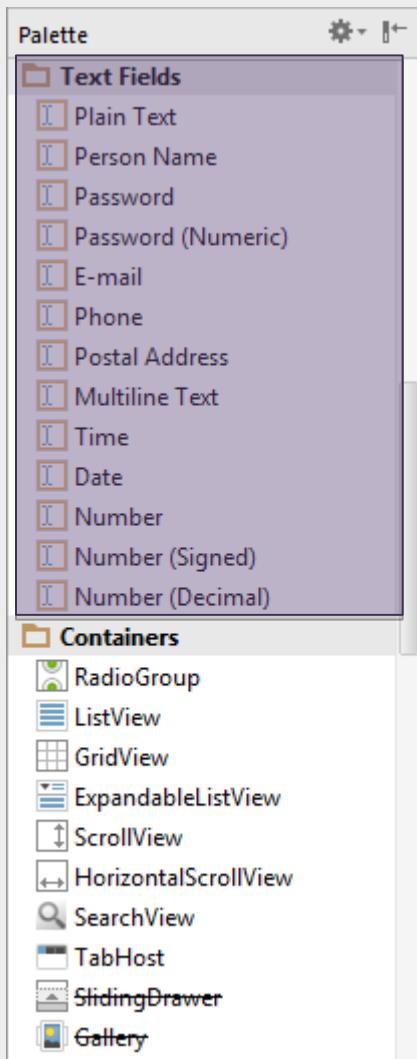
Como parâmetro para o `setText` pode-se passar uma **String** ou uma referência ao **Strings.xml** por meio da referencia **R.strings**

```
TextView msg = (TextView) findViewById(R.id.mensagem);
```

```
msg.setText("Mensagem!");
```

Ou:

```
msg.setText(R.string.mensagem_boas_vindas);
```



Permite que o usuário insira informações texto: nome, e-mail, número, número de telephone e etc..

A classe EditText possui os métodos `getText()` e `setText()`;

Para recuperar o valor inserido pelo usuário, utilizamos o método:
Editable getText();

Para recuperar a String do **Editable** basta chamar o método
`toString();`

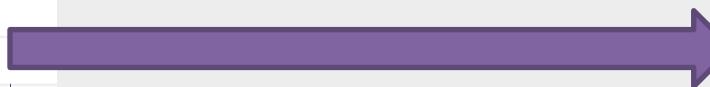
```
EditText txtNome = (EditText) findViewById(R.id.txt_nome);
```

```
String nome = txtNome.getText().toString();
```

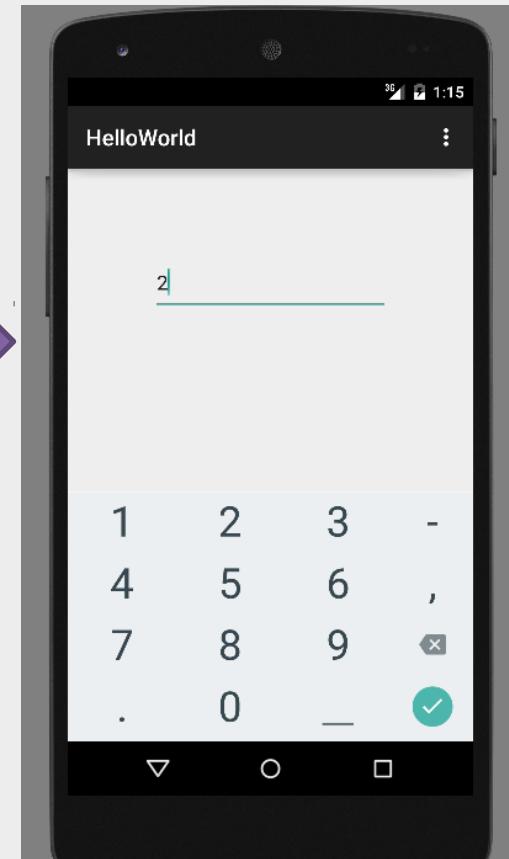
Properties	
inputType	[number]
none	<input type="checkbox"/>
text	<input type="checkbox"/>
textCapCharacters	<input type="checkbox"/>
textCapWords	<input type="checkbox"/>
textCapSentences	<input type="checkbox"/>
textAutoCorrect	<input type="checkbox"/>
textAutoComplete	<input type="checkbox"/>
textMultiLine	<input type="checkbox"/>
textImeMultiLine	<input type="checkbox"/>
textNoSuggestions	<input type="checkbox"/>
textUri	<input type="checkbox"/>
textEmailAddress	<input type="checkbox"/>
textEmailSubject	<input type="checkbox"/>

Podemos determinar o tipo de valor que o **EditText** aceita. Dessa forma, um teclado será exibido conforme o tipo de dado definido.

Para isso, basta utilizar a propriedade **inputType**.



inputType=“number”, o teclado número será exibido e somente número serão aceitos no campo.



VIEWS – BUTTON E IMAGEBUTTON

FIAP

Existem três formas de exibir um Botão: com imagem e texto, somente texto e somente imagem.



O texto fica na propriedade **android:text**;

Para imagem e texto, defina a propriedade **android:drawableLeft** ou **android:drawableRight**; (existem ainda top e bottom – acima e abaixo)

Para exibir somente uma imagem, utilize a view **ImageButton**.

O evento do clique pode ser tratado através da propriedade **android:onClick**

XML

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Botão"  
    android:id="@+id/button"  
    android:onClick="exibir" />
```

Java

```
public void exibir(View v){  
}
```

Também é possível tratar eventos (não somente o click) por meio de um listerner, neste caso **OnClickListener**:

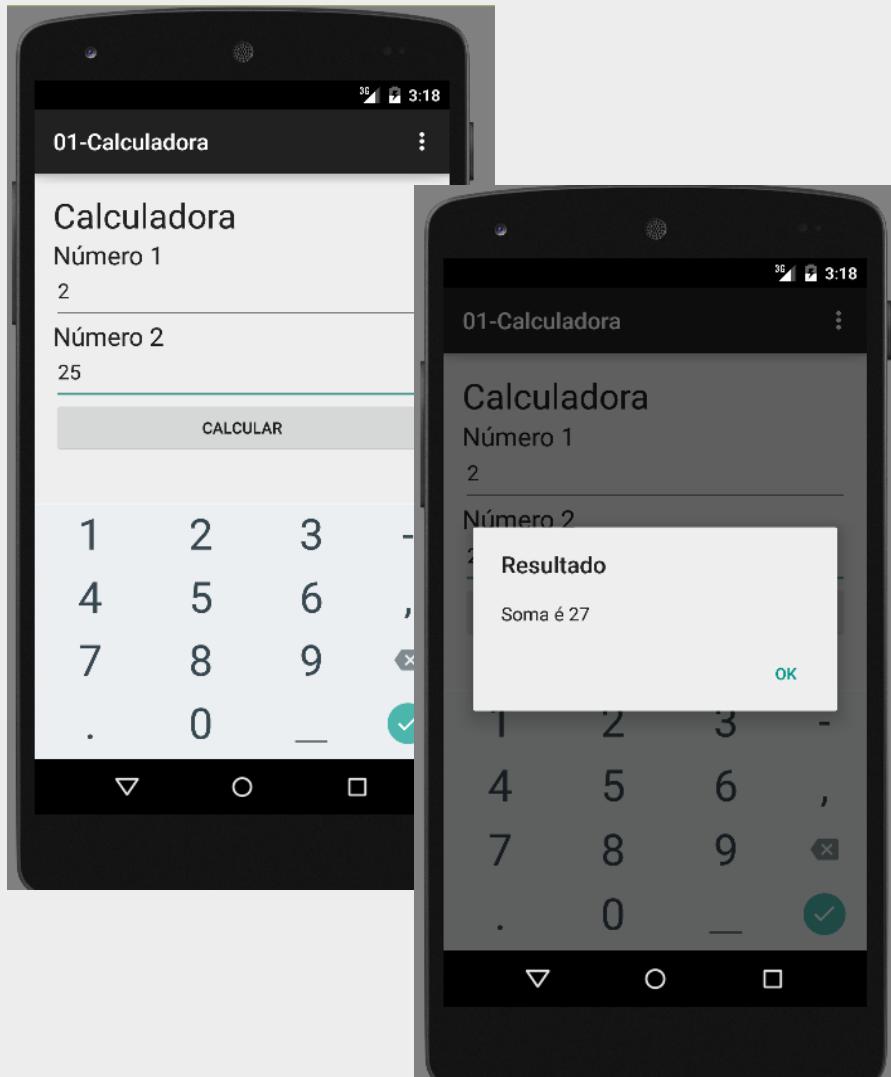
```
public class DadoActivity extends ActionBarActivity {

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

        Button botao = (Button) findViewById(R.id.button);
        botao.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Evento do clique do botão....
            }
        });
    }
}
```

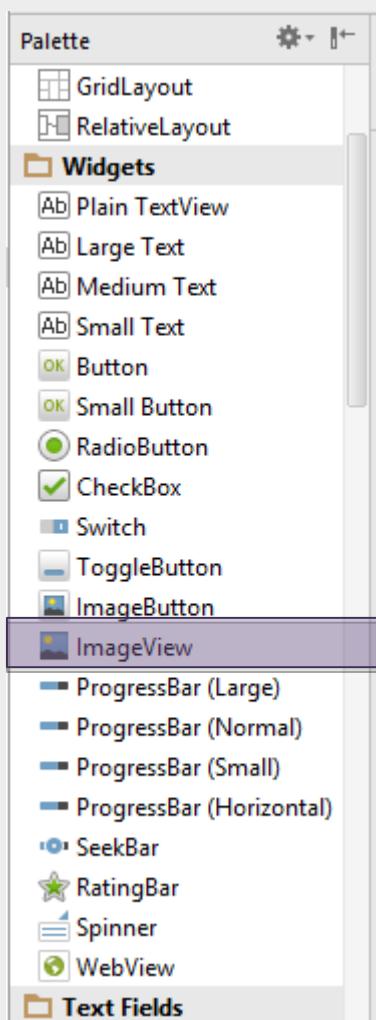
Classe anônima que implementa
a interface **OnClickListener**

Crie uma aplicação que permita efetuar a soma de dois valores inteiros conforme abaixo:



Uma dica: para que o botão calcular ocupe todo o espaço horizontal do display, altere a propriedade **Layout width** para **fill parent**.

OBS: utilizar o recurso externo para a definição dos textos apresentados pela aplicação.



Exibe uma imagem (png, jpg e gif);

As imagens devem ser armazenadas na pasta drawable ou mipmap;

As imagens são armazenadas em pastas diferentes, de acordo com a resolução (hdpi, mdpi, xhdpi, xxhdpi, etc..)

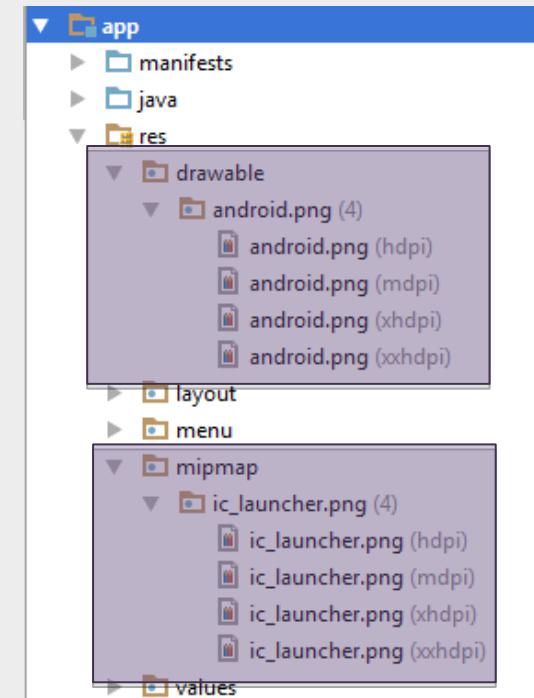
Uma imagem pode ser referenciada através do R.drawable ou R.mipmap;

Exemplo:

**@drawable/android
R.drawable.android**

**@mipmap/ic_launcher
R.mipmap.ic_launcher**

(observe que a extensão é suprimida)



VIEWS – IMAGEVIEW

Importando uma imagem

FIAP

The screenshot shows the Android Studio interface with the Project and Structure toolbars on the left. A context menu is open over the 'res' folder in the Project tree, with 'Image Asset' highlighted. To the right is the 'Asset Studio' window for creating a new image asset.

Clique com o botão direito do mouse sobre a pasta **res e escolha:**
New → Image Asset

Escolha o tipo:
Launcher Icons (mipmap);
Action Bar and Tab Icon ou
Notification Icon (drawable)

Seleciona o arquivo da imagem

Informe o nome do recurso

Também é possível copiar a imagem diretamente para o diretório desejado.

The 'Asset Studio' window displays the following settings for a Launcher Icons asset:

- Asset Type: Launcher Icons
- Foreground: Image (radio button selected)
- Image file: `lates\gradle-projects\NewAndroidModule\root\res\mipmap-xhdpi\ic_launcher.png`
- Additional padding: Trim surrounding blank space
- Foreground scaling: Crop (radio button selected)
- Shape: None (radio button selected)
- Background color:
- Resource name: `ic_launcher`

Preview sections for MDPI, HDPI, XHDPI, and XXHDPI are visible on the right.

VIEWS – IMAGEVIEW

Exibindo uma imagem

The screenshot shows the Android Studio interface. On the left, the 'Palette' panel is open, displaying various UI components under the 'Widgets' category, with 'ImageView' selected. In the center, a smartphone-shaped preview shows an ImageView with its properties window open. The 'src' field is set to a placeholder icon, and the 'id' field is set to 'imageView'. A tooltip above the properties window says 'Shift+Enter'. A purple arrow points from the 'ImageView' entry in the palette to the selected 'ImageView' in the preview. Another purple arrow points from the 'src' field in the properties window to the 'Resources' dialog on the right. The 'Resources' dialog shows a tree view of resources, with 'android' selected. At the bottom, there's a search bar containing 'android' and a green Android icon. A purple box contains the text: 'Dê um duplo clique sobre o ImageView e escolha o botão “...”' (Double-click on the ImageView and choose the "... button"). Below it, another purple box contains the text: 'Procure pela imagem e clique em Ok Dica: digite o nome para realizar a busca' (Search for the image and click OK. Hint: type the name to perform the search).

Dê um duplo clique sobre o ImageView e escolha o botão “...”

Procure pela imagem e clique em Ok
Dica: digite o nome para realizar a busca

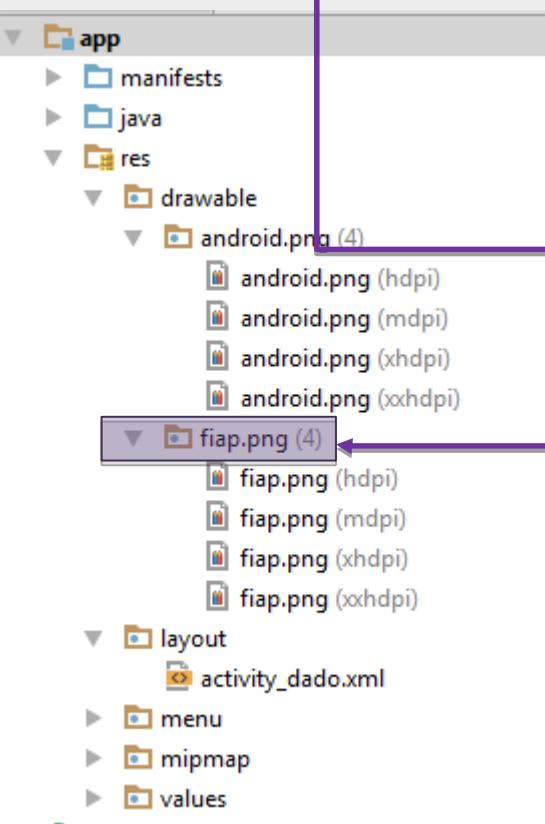
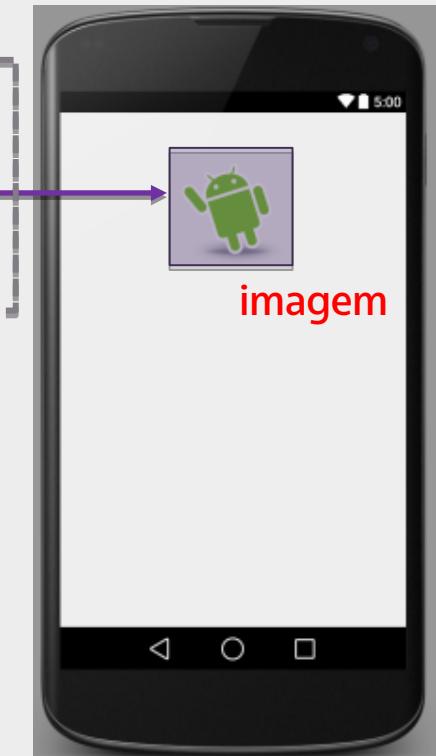
VIEWS – IMAGEVIEW

FIAP

Podemos alterar a imagem em tempo de execução através da classe **ImageView** e método **setImageResource**:

```
ImageView img = (ImageView) findViewById(R.id.imagem);
```

```
img.setImageResource(R.drawable.fiap)
```



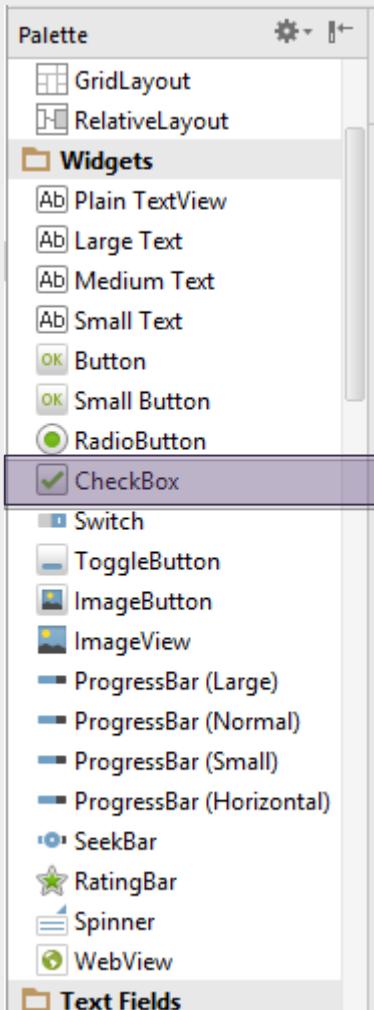
Crie um simples jogo de dados entre o computador e o jogador que efetue o sorteio dos dados e exiba o vencedor da partida em um TextView na tela;



Utilize as imagens já fornecidas;

Para sortear um número entre 0 e 5 utilize o exemplo abaixo (onde n conterá o valor sorteado):

```
Random r = new Random();  
int n = r.nextInt(5);
```

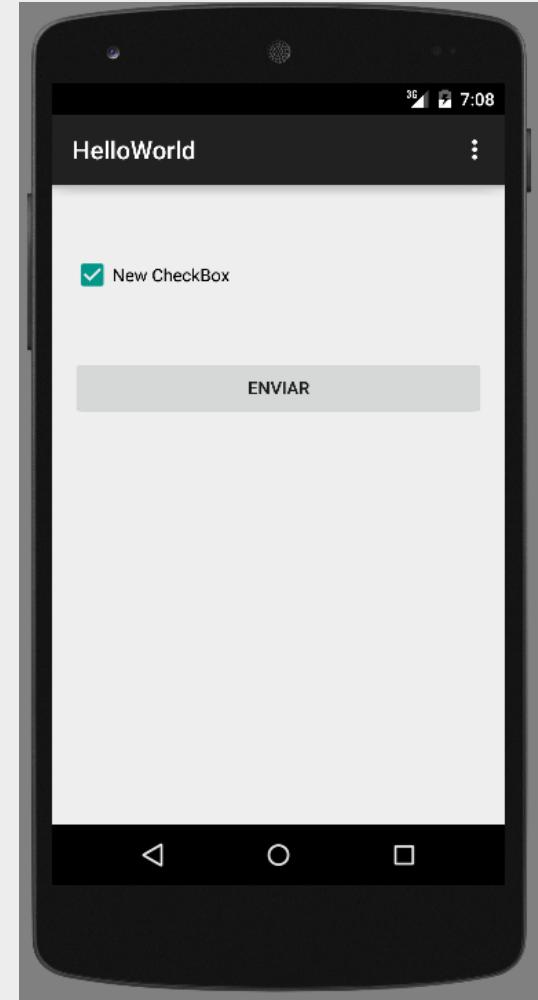


Caixa de seleção onde o usuário pode selecionar os itens individualmente;

Armazena verdadeiro ou falso;

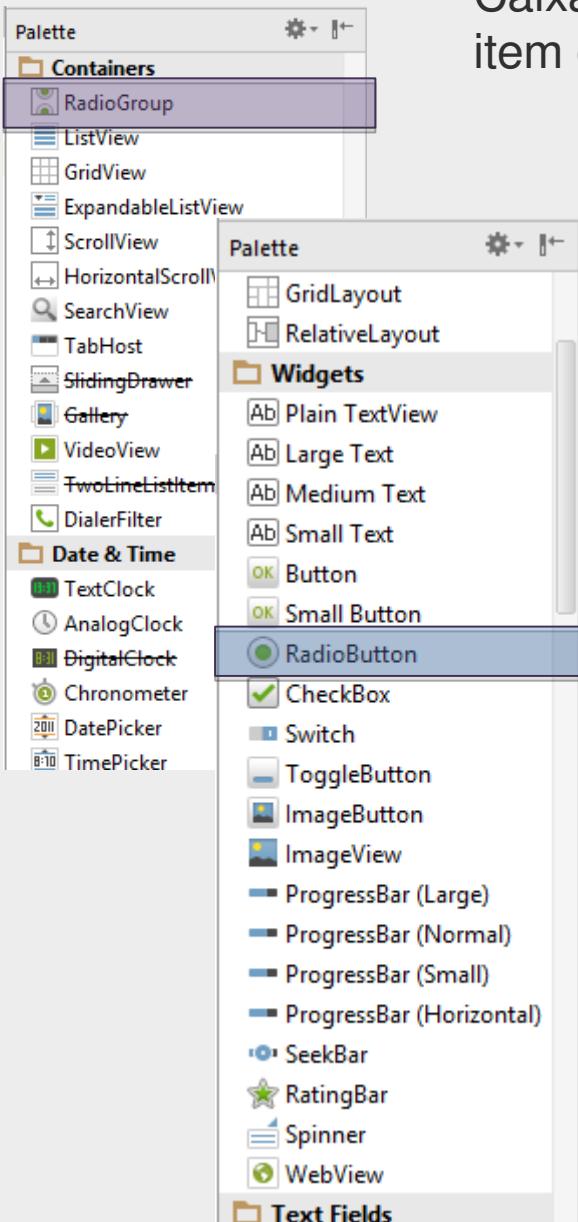
Principais métodos:

- **isChecked()**: verifica se o checkbox está selecionado (true / false);
- **setChecked(boolean p1)**: seleciona ou não o checkbox (true / false);



IEWS – RADIobutton

FIAP

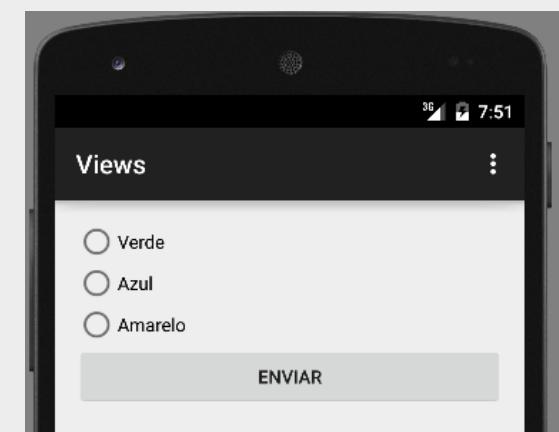


Caixa de seleção onde o usuário pode selecionar somente um item do grupo;

As opções para seleção, os **RadioButton** devem ser agrupadas em um **RadioGroup**;

Para descobrir qual o item selecionado basta chamar o método do RadioGroup **getCheckedRadioButtonId()**, que retorna o id do item selecionado;

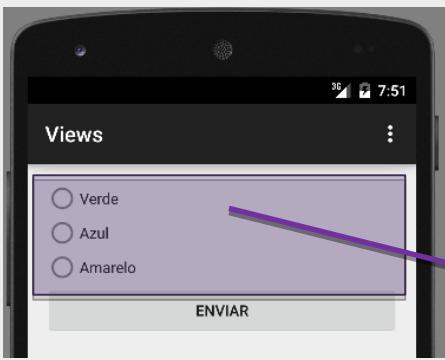
Pode-se associar eventos de clique a cada um dos RadioButtons e também ao RadioGroup;



VIEWS – RADIobutton

FIAP

Por exemplo, para verificar qual a cor selecionada nos itens acima, supondo que os ids dos itens sejam radAzul, radAmarelo e radVerde:



Recupera o id do
radio selecionado

Recupera o radio
selecionado
através do id

Recupera o texto
do radio
selecionado

```
RadioGroup rg = (RadioGroup) findViewById(R.id.radioGroupCores);
```

```
int idSelecionado = rg.getCheckedRadioButtonId();
```

```
RadioButton radCor = (RadioButton) findViewById(idSelecionado);
```

```
String nomeCor = radCor.getText();
```

```
Log.i("ProjetoViews", "Cor Selecionada: " + nomeCor);
```

Exibe a cor
selecionada no log.

VIEWS – RADIobutton

FIAP

Podemos implementar o método **onCheckedChanged** da interface **OnCheckedChangeListener** e configurar o **RadioGroup** para executá-lo quando um radio for selecionado:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.hello_activity);  
  
    RadioGroup rg = (RadioGroup) findViewById(R.id.radioCor);  
    //Configura o listener no RadioGroup  
    rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {  
        @Override  
        public void onCheckedChanged(RadioGroup group, int checkedId) {  
            RadioButton radCor = (RadioButton) findViewById(checkedId);  
            Log.i("ProjetoViews", "Cor selecionada: " + radCor.getText());  
            Toast.makeText(ViewsActivity.this,radCor.getText(),Toast.LENGTH_LONG).show();  
        }  
    });  
}
```

RadioGroup que gerou o evento

Id do radio selecionado

Também é possível adicionar e remover elementos dinamicamente (via código) de um RadioGroup;

Para tanto, existem os métodos da classe **RadioGroup**:

addView (View P1) → adiciona um **RadioButton** ao **RadioGroup**;

removeViewAt (int P1) → remove um item da posição P1 do **RadioGroup**;

removeAllViews() → remove todos os **RadioButtons** de um **RadioGroup**.

```
radCores = (RadioGroup) findViewById(R.id.radioGroupCores);
```

```
//Cria um novo elemento
```

```
RadioButton radBranco = new RadioButton(this);
```

```
radBranco.setId(1);
```

```
radBranco.setText("Branco");
```

```
// Adiciona um novo elemento
```

```
radCores.addView(radBranco);
```

```
// Remove o primeiro elemento (índice 0)
```

```
radCores.removeViewAt(0);
```



Contexto, se estiver dentro
de uma classe anônima,
utilize o “Nome da
Activity”.this

Crie uma aplicação que permita calcular o preço de uma pizza baseado no sabor e preferência de borda recheada.



Os preços são os seguintes:

- Mussarela: R\$ 10,00
- Calabresa: R\$ 15,00
- Portuguesa: R\$ 20,00

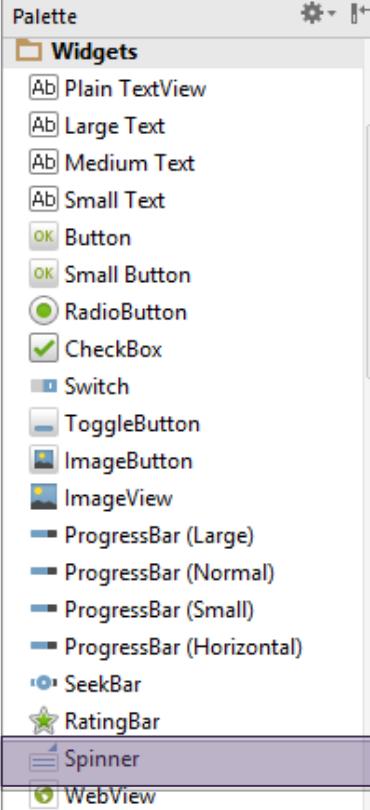
A opção de borda recheada acrescenta R\$5,00 ao pedido.

Desafio: suponha que a pizza de mussarela não permita borda recheada. Assim, esta opção deve ser automaticamente desabilitada quando o usuário selecionar Mussarela como sabor.

Dica: utilizar o método **setEnabled(boolean)** para habilitar ou desabilitar uma view.

IEWS – SPINNER

FIAP



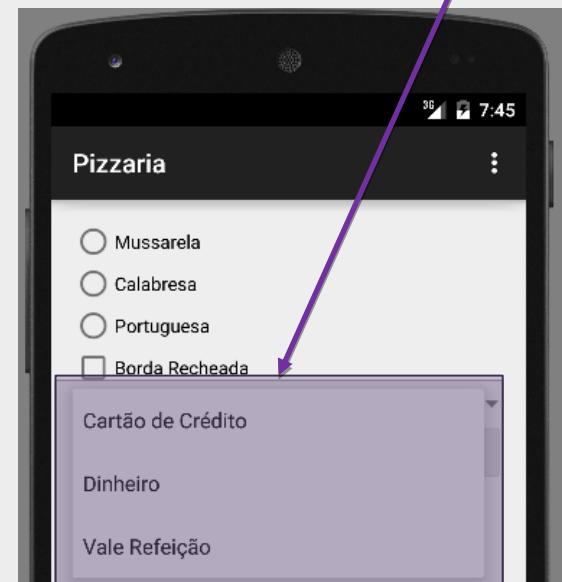
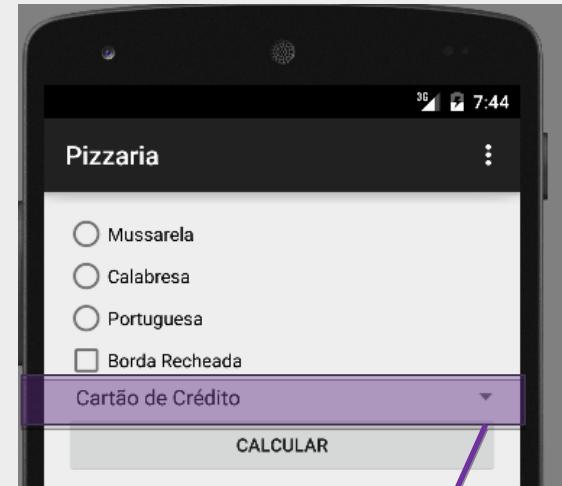
Cria uma lista de opções para seleção;
Podemos definir os valores de forma estática ou dinâmica;

Os itens da lista serão um array de strings e
podem ser definidos estaticamente no arquivo
strings.xml, dentro da pasta **res/values**:

```
<string-array name="formas_pagamento">
    <item>Cartão de Crédito</item>
    <item>Dinheiro</item>
    <item>Vale Refeição</item>
</string-array>
```

Depois, precisamos associar o array criado ao **Spinner** por meio da
propriedade **android:entries**

```
android:entries="@array/formas_pagamento"
```



VIEWS – SPINNER – ARRAY ADAPTER

FIAP

Podemos carregar o **Spinner** dinamicamente através de código java, para isso, será necessário um **ArrayAdapter**.

Precisamos de uma lista de objetos, a classe desse objeto deve sobrescrever o método **toString()**, pois o retorno desse método será exibido no **Spinner**.

```
public class Curso {  
  
    private int codigo;  
  
    private String nome;  
  
    private int numeroHora;  
  
    @Override  
    public String toString() {  
        return numeroHora + "h - " + nome;  
    }  
  
    //Gets e Sets  
    //Construtores  
}
```

Classe que armazena os valores para exibir no Spinner

Método sobreescrito da classe Object.

Podemos criar uma classe que retorna a lista de “*Cursos*”, essa classe pode acessar um Banco de Dados, Web Services, etc..

Neste caso, a própria classe vai gerar a lista:

```
public class CursoBO {  
  
    public List<Curso> list(){  
        List<Curso> lista = new ArrayList<>();  
        lista.add(new Curso(1,"Android",16));  
        lista.add(new Curso(2,"IOS",20));  
        lista.add(new Curso(3,"Windows Phone",16));  
  
        return lista;  
    }  
  
}
```



VIEWS – SPINNER – ARRAY ADAPTER

FIAP

Na Activity, no método **onCreate** podemos popular o Spinner:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.curso_activity);  
  
    //Recupera o spinner  
    Spinner spinner = (Spinner) findViewById(R.id.spinner);  
  
    //Instancia a classe de negócio para recuperar a lista de cursos  
    CursoBO bo = new CursoBO();  
    List<Curso> lista = bo.list();  
  
    //Cria um adapter com a lista de curso  
    ArrayAdapter<Curso> adapter =  
        new ArrayAdapter<Curso>(this, android.R.layout.simple_spinner_item, lista);  
  
    //Associa o adapter ao spinner  
    spinner.setAdapter(adapter);  
  
    //Também é possível associar novos elementos através do adapter  
    adapter.add(new Curso(4,"Java",20));  
}
```

Alguns métodos da view Spinner:

- **getCount()** → retorna o número total de elementos contidos no Spinner;
- **getSelectedItemPosition()** → retorna o índice do elemento selecionado (inicia em 0);
- **getItemAtPosition (int P1)** → retorna o item na posição indicada em P1;
- **getSelectedItem()** → retorna o item selecionado (Object);

Exemplo:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
```

```
Curso curso = (Curso) spinner.getSelectedItem();
```

```
Curso curso2 = (Curso) spinner.getItemAtPosition(1);
```

- Também é possível executar uma ação assim que um elemento do Spinner é selecionado;
- Para isso basta implementar a interface **OnItemSelectedListener** e seus métodos **onItemSelected** e **onNothingSelected** que são acionados quando um item é selecionado ou quando nenhum item é escolhido no *Spinner*;
- Finalmente basta associar ao Spinner a classe que tratará as ações por meio do método **setOnItemSelectedListener**;

No método **onCreate** da activity, vamos associar o evento ao Spinner:

Parâmetros do método **onItemSelected**:

- **parent** → lista de elementos exibidos no spinner;
- **view** → spinner que gerou o evento;
- **position** → índice do item selecionado;
- **id** → id do item selecionado;

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);

//Associa o evento do listerner ao Spinner utilizando uma classe anônima
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener {

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        Curso curso = (Curso) parent.getItemAtPosition(position);
        Toast.makeText(HelloActivity.this, curso.getNome(),Toast.LENGTH_LONG).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }
});
```

VIEWS – AUTOCOMPLETETEXTVIEW

FIAP



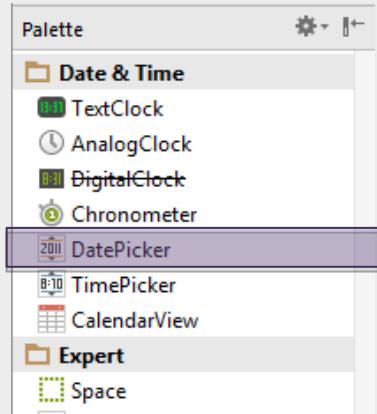
Cria uma caixa de textos com valores pré-definidos que são exibidos na medida em que o dado é digitado:

Os itens a serem pré-carregados seguem o mesmo procedimento do Spinner visto anteriormente.

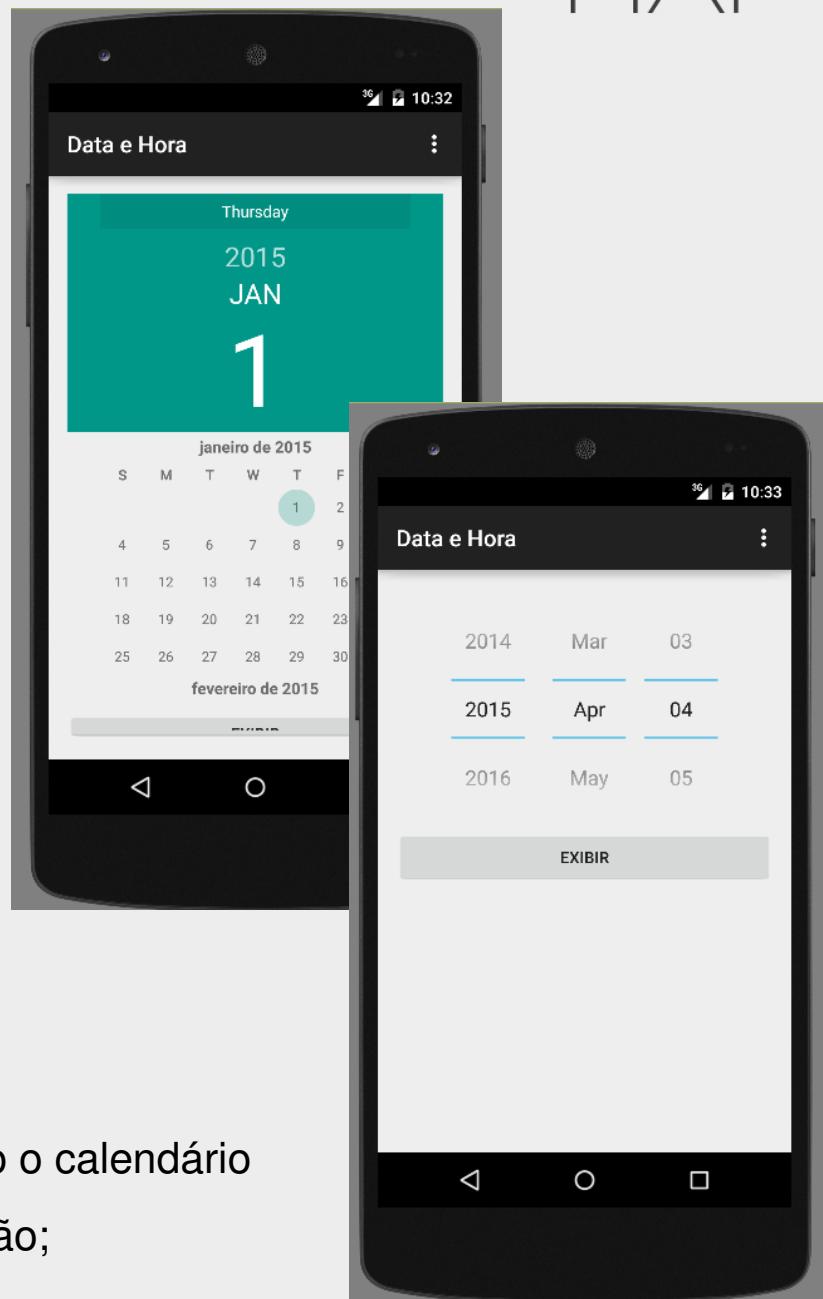
```
AutoCompleteTextView complete =  
    (AutoCompleteTextView) findViewById(R.id.autoCompleteTe  
  
CursoBO bo = new CursoBO();  
List<Curso> lista = bo.list();  
  
ArrayAdapter<Curso> adapter =  
    new ArrayAdapter<Curso>(this,  
        android.R.layout.simple_list_item_1,lista);  
  
complete.setAdapter(adapter);});
```

VIEWS – DATEPICKER

Permite a seleção de data:



```
<DatePicker  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/datePicker"  
    android:calendarViewShown="false"  
    android:datePickerMode="spinner" />
```



Principais métodos:

- **getDayOfMonth()** → retorna o dia do mês;
- **getMonth()** → retorna o mês (Janeiro é 0);
- **getYear()** → retorna o ano;
- **updateDate(int ano, int mês, int dia)** → atualiza a data;

```
DatePicker picker = (DatePicker) findViewById(R.id.datePicker);
```

```
Calendar calendar = new GregorianCalendar(  
    picker.getYear(),  
    picker.getMonth(),  
    picker.getDayOfMonth());
```

Para converter a data selecionada em um objeto basta utilizar o método construtor da classe **GregorianCalendar**...

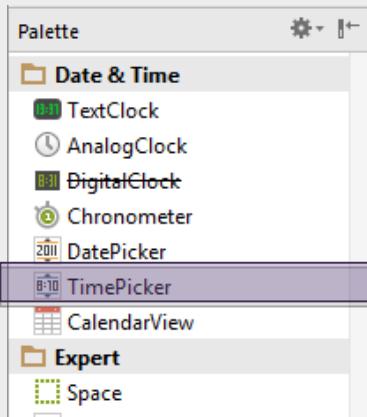
Principais métodos:

- **getDayOfMonth()** → retorna o dia do mês;
- **getMonth()** → retorna o mês (Janeiro é 0);
- **getYear()** → retorna o ano;
- **updateDate(int ano, int mês, int dia)** → atualiza a data;

```
DatePicker picker = (DatePicker) findViewById(R.id.datePicker);
```

```
Calendar calendar = new GregorianCalendar(  
    picker.getYear(),  
    picker.getMonth(),  
    picker.getDayOfMonth());
```

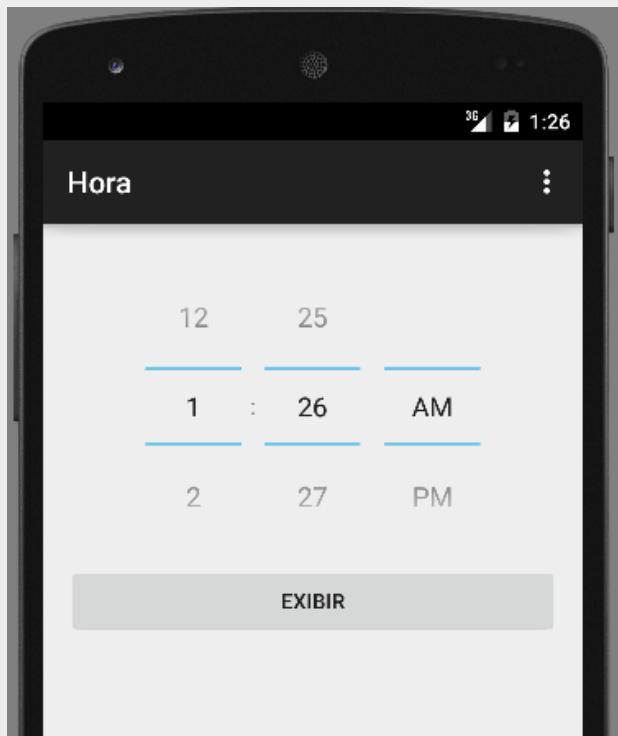
Para converter a data selecionada em um objeto basta utilizar o método construtor da classe **GregorianCalendar**...



Principais métodos:

- **getCurrentHour()** → retorna a hora;
- **getCurrentMinute()** → retorna os minutos;

Para atualizar os valores da hora e minuto, basta utilizar os respectivos métodos **set..**



```
<TimePicker  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/timePicker"  
    android:layout_gravity="right"  
    android:timePickerMode="spinner" />
```

As caixas de diálogo são um recurso interessante para comunicar ao usuário sobre eventos importantes da aplicação;

Todas as caixas de diálogo são descendentes diretos ou indiretos da classe Dialog e algumas já são padrão:

- **AlertDialog**: mensagens de alerta genéricas (conforme já visto);
- **ProgressDialog**: mostram progresso de uma ação;
- **DatePickerDialog**: permitem a seleção de uma data;
- **TimePickerDialog**: permitem a seleção de hora;

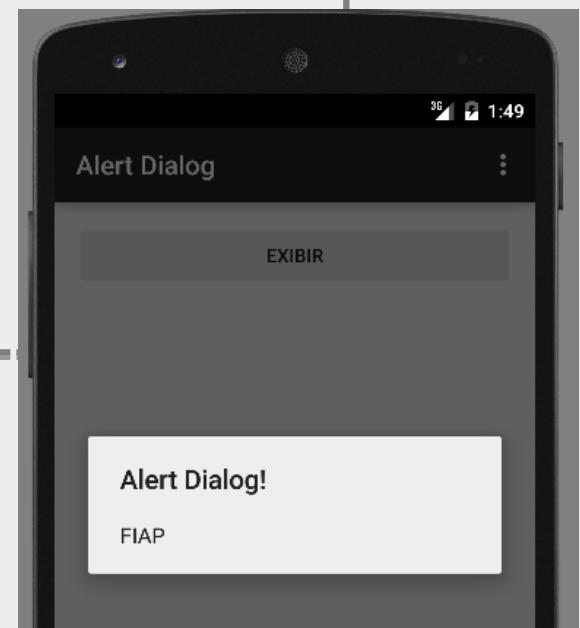
Também podem ser criadas caixas de diálogo personalizadas estendendo a classe pai **Dialog**.

Um AlertDialog deve ser criado por meio da classe **AlertDialog.Builder**:

Algumas propriedades do AlertDialog podem ser definidas utilizando-se alguns métodos do **AlertDialog.Builder**:

Exemplo:

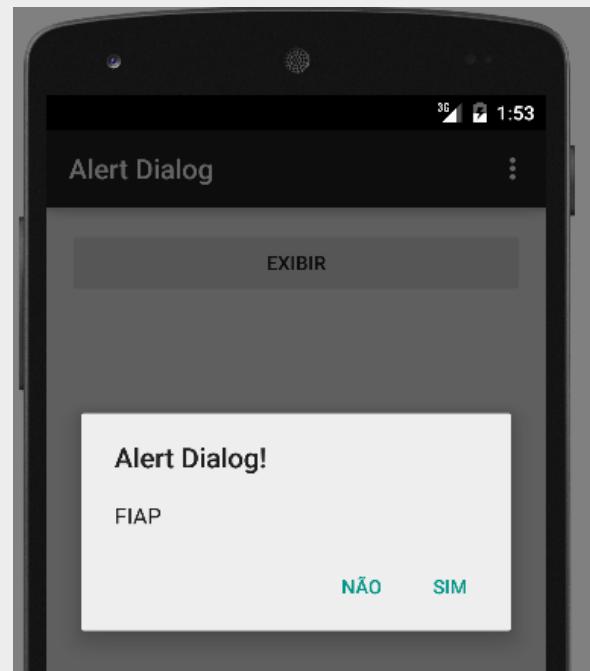
```
AlertDialog.Builder builder = new AlertDialog.Builder(this);  
  
builder.setTitle("Alert Dialog!");  
builder.setMessage("FIAP");  
builder.setCancelable(true);  
builder.show();
```



Um AlertDialog pode exibir três tipos de botões por meio dos métodos:

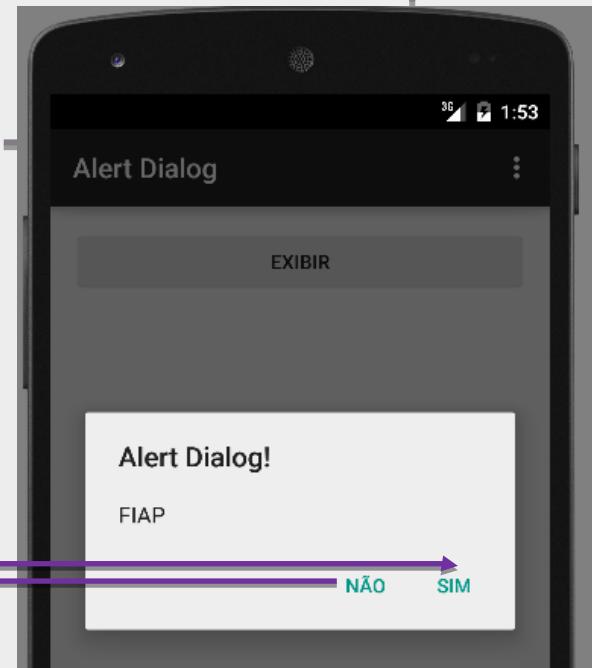
- **setPositiveButton** (por exemplo, a opção SIM, Confirmar, etc...),
- **setNegativeButton** (por exemplo, a opção NÃO, Cancelar, etc...) e
- **setNeutralButton** (por exemplo, a opção OK...);

Para capturar a ação do usuário sobre o alerta, isto é, qual botão foi clicado, é necessário implementar um listener **DialogInterface.OnClickListener** para o botão desejado.



ALERT DIALOG

```
builder.setPositiveButton("SIM", new DialogInterface.OnClickListener() {  
    @Override  
    public void onClick(DialogInterface dialog, int which) {  
        //Código para opção SIM  
    }  
});  
  
builder.setNegativeButton("NÃO", new DialogInterface.OnClickListener() {  
    @Override  
    public void onClick(DialogInterface dialog, int which) {  
        //Código para opção NÃO  
    }  
});
```



É possível criar diálogos onde o usuário pode selecionar um item de uma lista de valores pré-determinados;

Isso pode ser feito por método **setItems** que recebe como parâmetros um *array* de itens a serem exibidos e a implementação de um listener que será acionado caso um item seja selecionado;

Detalhe: a propriedade **message** (setMessage()) deve ser nula para que a lista dos itens seja exibida!

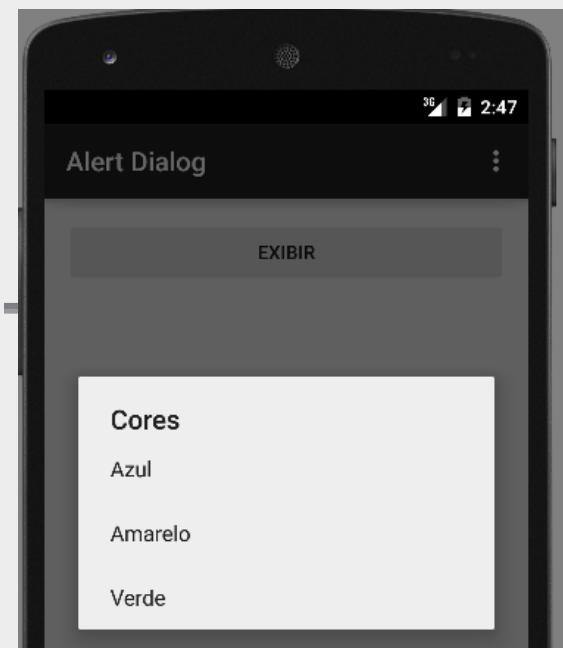
ALERT DIALOG COM ITENS

Exemplo:

```
final CharSequence[] cores = {"Azul", "Amarelo", "Verde"};  
  
AlertDialog.Builder alert = new AlertDialog.Builder(this);  
  
alert.setTitle("Cores");  
  
alert.setItems(cores, new DialogInterface.OnClickListener() {  
    @Override  
    public void onClick(DialogInterface dialog, int which) {  
        Toast.makeText(AlertActivity.this, cores[which], Toast.LENGTH_LONG).show();  
    }  
});  
  
alert.show();
```

Itens para serem exibidos

Posição do item selecionado



ALERT DIALOG COM ITENS

É possível também criar itens na forma de radio buttons trocando o método **setItems** visto anteriormente, por **setSingleChoiceItems**:

A única diferença é que o **setSingleChoiceItems** recebe um terceiro parâmetro indicando qual item deve vir selecionado (-1 para nenhum);

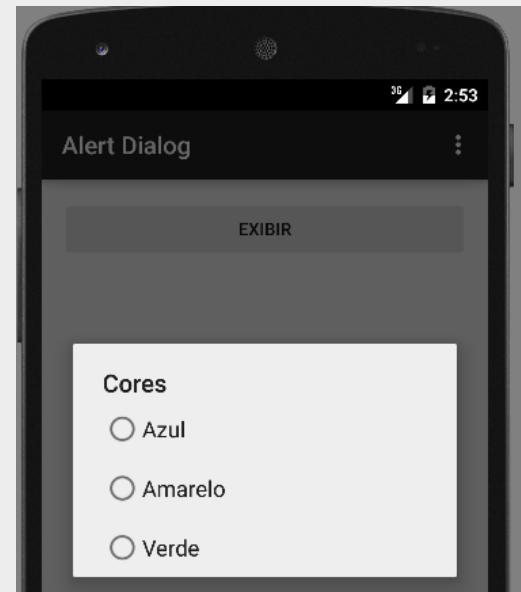
Exemplo:

```
AlertDialog.Builder alert = new AlertDialog.Builder(this);

final CharSequence[] cores = {"Azul","Amarelo","Verde"};
alert.setTitle("Cores");
alert.setSingleChoiceItems(cores, -1, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(AlertActivity.this, cores[which],Toast.LENGTH_LONG).show();
    }
});

alert.show();
```

Item pré-selecionado



ALERT DIALOG COM ITENS

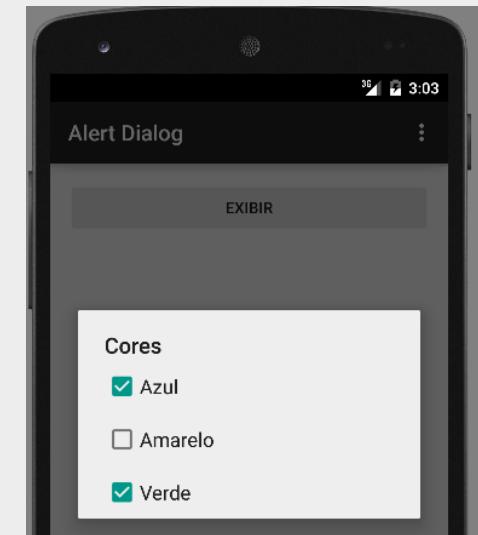
Outra possibilidade é que o usuário selecione mais de um item na lista por meio do método **setMultiChoiceItems**;

Neste método deve-se passar como parâmetros um array de booleans e a implementação da interface **OnMultiChoiceClickListener**;

Exemplo:

```
final CharSequence[] cores = {"Azul", "Amarelo", "Verde"};
final boolean[] bCores = new boolean[cores.length];

alert.setTitle("Cores");
alert.setMultiChoiceItems(cores, bCores,
    new DialogInterface.OnMultiChoiceClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which, boolean isChecked) {
            String valores = bCores[0] + "," + bCores[1] + " " + bCores[2];
            Toast.makeText(AlertActivity.this, valores, Toast.LENGTH_LONG).show();
        }
    });
alert.show();
```



Posição do item
selecionado

Valor do item
selecionado
(true/false)

PROGRESS DIALOG

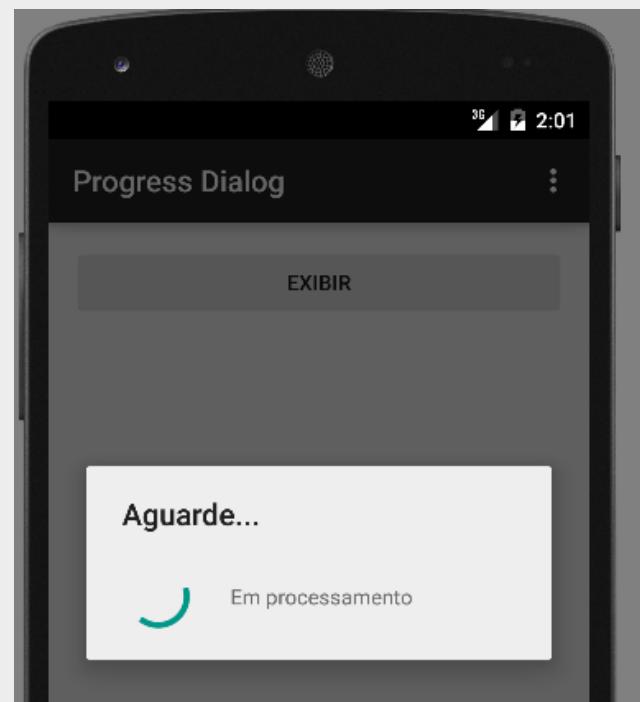
Um Progress Dialog pode representar o processamento de uma determinada tarefa;

Os métodos **show()** e **dismiss()** exibe e cancela o Dialog, respectivamente.

Exemplo:

```
ProgressDialog pd= new ProgressDialog(this);
pd.setTitle("Aguarde...");
pd.setMessage("Em processamento");
pd.show(); //Exibe o ProgressDialog

pd.dismiss(); //Cancela o ProgressDialog
```



Uma outra forma de possibilitar que a data e hora sejam informados é por meio de uma caixa de diálogos;



O método construtor da classe **DatePickerDialog** possui os seguintes parâmetros:

DatePickerDialog(P1, P2, P3, P4, P5)

- P1: Activity associada;
- P2: Classe que implemente **OnDateSetListener**;
- P3, P4 e P5: ano, mês e dia exibidos inicialmente.

A data informada pelo usuário deve ser recuperada no método **onDateSet** da interface **OnDateSetListener**;

DATEPICKER DIALOG

FIAP

Exemplo:

```
DatePickerDialog picker = new DatePickerDialog(this,  
    new DatePickerDialog.OnDateSetListener() {  
  
        @Override  
        public void onDateSet(DatePicker view, int year, int monthOfYear,  
            int dayOfMonth) {  
            //Código executado após o clique do botão OK  
        }  
    },  
    2015, 0, 1);  
  
picker.show();
```



Existe também uma caixa de diálogo específica para hora e minuto que é a **TimePickerDialog**;



Método construtor da classe **TimePickerDialog** possui os seguintes parâmetros:

TimePickerDialog(P1, P2, P3, P4, P5)

- P1: Activity associada;
- P2: Classe que implemente **OnTimeSetListener**;
- P3 e P4: Hora e minuto;
- P5: formato 24h (true ou false).

A hora / minuto informados pelo usuário deve ser recuperada no método **onTimeSet** da interface **OnTimeSetListener**;

TIMERPICKER DIALOG

FIAP

Exemplo:

```
TimePickerDialog picker = new TimePickerDialog(this,  
    new TimePickerDialog.OnTimeSetListener() {  
        @Override  
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {  
            //Código executado após o clique do botão OK  
        }  
    }, 12,12, false);  
  
picker.show();
```

24 Horas (true or false)



Você pode definir a sua própria caixa de diálogos apenas seguindo os passos abaixo:

1. Crie a interface de sua caixa em um arquivo XML (como uma interface normal, por exemplo chamada **meuDialogo.xml**) no diretório **res/layout**;
2. Instancie a caixa de diálogo por meio da classe genérica **Dialog** passando como parâmetro o contexto da sua aplicação/Activity:
Dialog dialog = new Dialog(this)
3. Associe a interface criada em 1 à sua caixa de diálogo:
dialog.setContentView(R.layout.meuDialogo);
4. Defina os valores das views que compõem a interface de sua caixa de diálogo e os respectivos listeners;
5. Recupere as views do dialog utilizando o método **findViewById** do próprio dialog:
dialog.findViewById(R.id.nome);
6. Exiba a caixa de diálogo como feito nos exemplos anteriores.

DIÁLOGOS PERSONALIZADOS

FIAP

Exemplo:

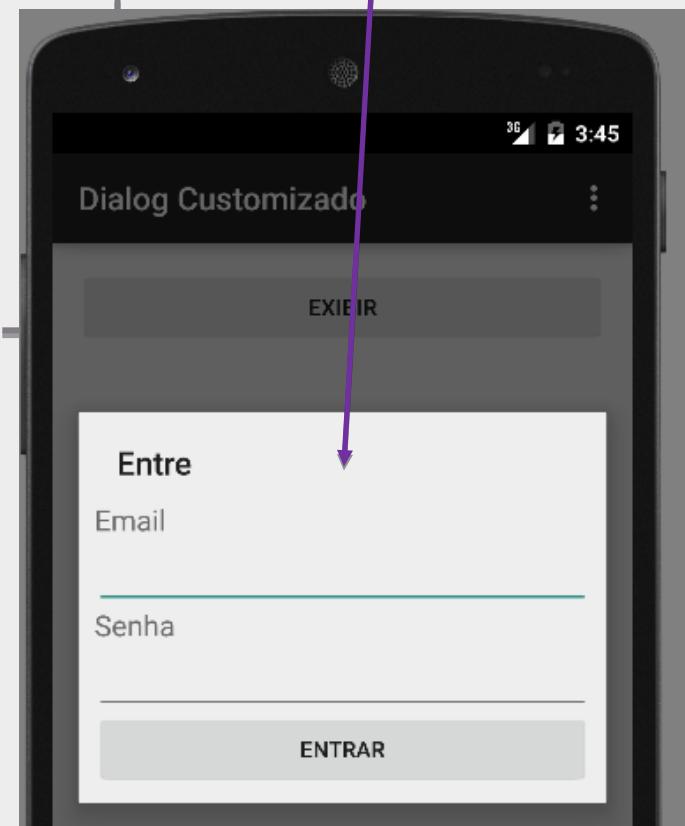
```
final Dialog dialog = new Dialog(this);
dialog.setTitle("Entre");
dialog setContentView(R.layout.dialog_customizado);

Button button = (Button) dialog.findViewById(R.id.entrar);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    }

});
dialog.show();
```

Recupera os elementos do
dialog

Layout customizado, criado
na pasta res/layout





Copyright © 2016 - Profs. Me. Leandro Rubim, Prof. Me. Thiago T. I. Yamamoto e Prof. Me. Edson Sensato

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Autor.

