

FIAP

Criação de uma Interface Gráfica Hybrid Mobile App Development



Prof. Vinny Albuquerque
<profvinny.albuquerque@fiap.com.br>

- Uma Atividade (Activity) fornece a janela na qual sua aplicação desenha a interface com o usuário (User Interface – UI).
- Cada aplicação tem uma ou mais atividades
- A primeira Atividade é frequentemente designada por MainActivity e é fornecida pelo modelo do projeto.

Toda Activity possui um arquivo XML para representar sua interface gráfica.

The screenshot shows the Android Studio interface with the following details:

- Project Bar:** Shows "Android" selected, with tabs for "activity_main.xml" and "MainActivity.kt".
- Project Tree:** Under "app", there are "manifests", "java", "java (generated)", "res" (expanded), "drawable", "layout" (expanded), "activity_main.xml", "mipmap", "values", and "res (generated)".
- MainActivity.kt:** The code is as follows:

```
1 package br.com.dominiodaaplicacao.helloworld
2
3 import ...
5
6 class MainActivity : AppCompatActivity() {
7     override fun onCreate(savedInstanceState: Bundle?) {
8         super.onCreate(savedInstanceState)
9         setContentView(R.layout.activity_main)
10    }
11 }
```

Two red arrows point from the "activity_main.xml" file in the Project tree to the "activity_main" reference in the "setContentView" line of the code, highlighting the connection between the XML layout and the Java code.

Abrindo o arquivo `activity_main.xml` no modo texto, vamos apagar todo o conteúdo existente e inserir o seguinte conteúdo:

The screenshot shows the Android Studio interface with the following details:

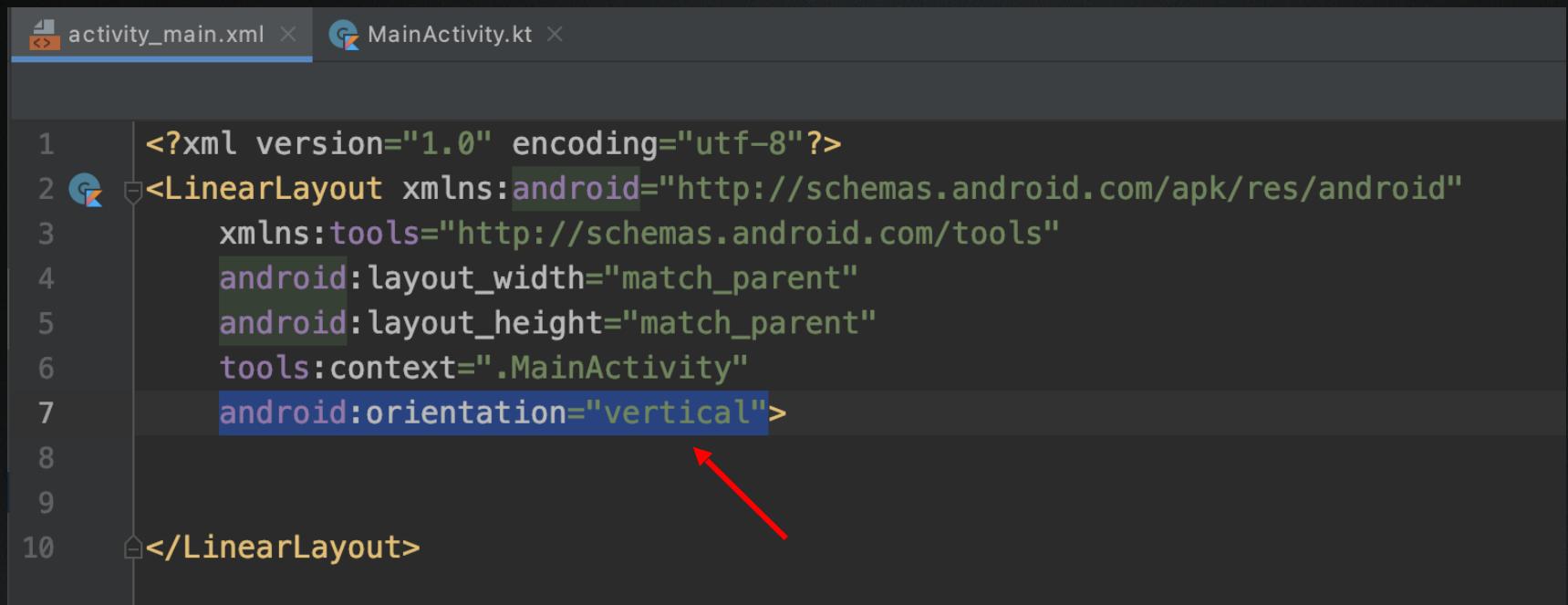
- Project Bar:** Shows "Android" selected.
- Toolbars:** Standard Android Studio toolbars for search, zoom, and settings.
- Project Tree:** Shows the project structure:
 - 1_Project
 - app
 - manifests
 - java
 - br.com.fiap.exemplo
 - MainActivity
 - br.com.fiap.exemplo (androidTest)
 - br.com.fiap.exemplo (test)
 - res
 - drawable
 - layout
 - activity_main.xml
 - mipmap
 - values
 - Gradle Scripts
- Code Editor:** The file `activity_main.xml` is open in the editor. The code is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<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">

</LinearLayout>
```

Criação de uma Interface Gráfica

O LinearLayout é um gerenciador de layout simples e organiza os componentes dentro dele de forma linear, seja na orientação horizontal ou na vertical. Para especificar a orientação, vamos inserir o seguinte atributo: android:orientation="vertical"



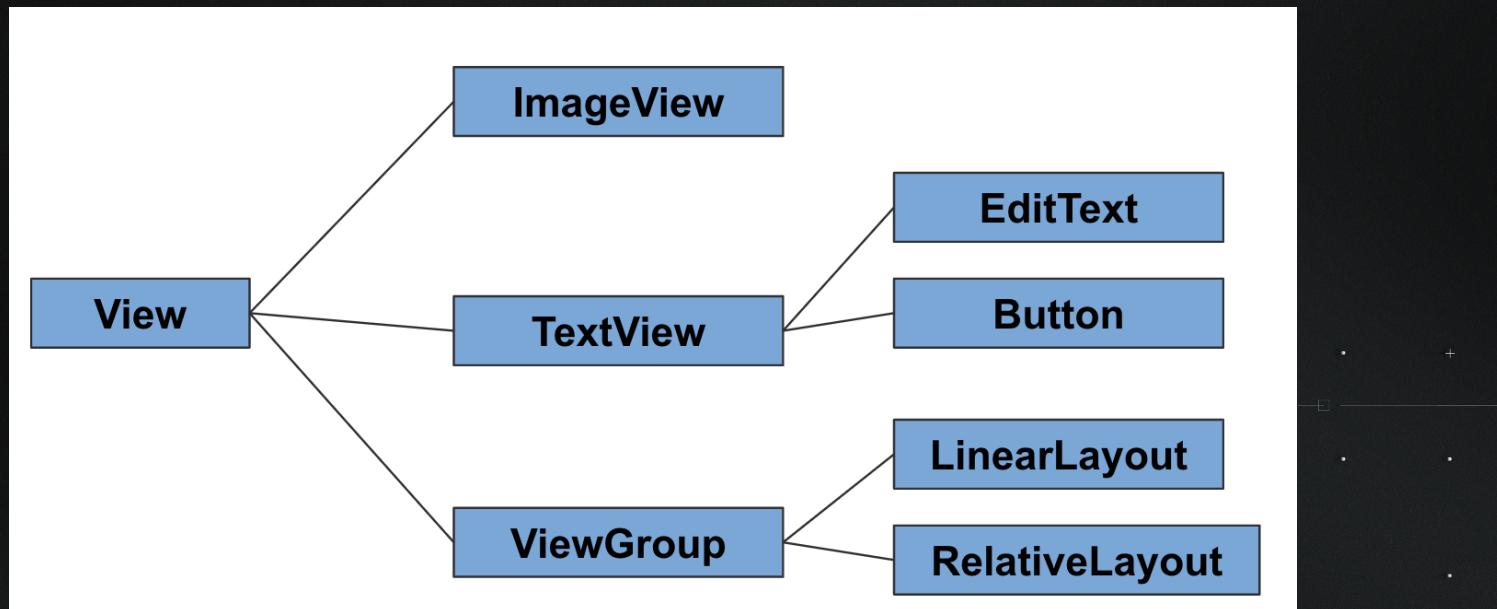
```
activity_main.xml × MainActivity.kt ×

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context=".MainActivity"
7     android:orientation="vertical">
8
9
10 </LinearLayout>
```

Todo componente gráfico no Android herda diretamente ou indiretamente a classe View. Outra observação importante é que toda tag XML de um componente no Android possui uma classe de mesmo nome.



Veja abaixo um pequeno pedaço da Hierarquia de classes no Android para os componentes gráficos:



Vamos inserir uma imagem em nosso layout. Para isso usaremos o ImageView.

The screenshot shows the Android Studio interface with two tabs: "activity_main.xml" and "MainActivity.kt". The "activity_main.xml" tab is active, displaying the XML code for a layout. The code defines a Linear Layout containing an ImageView with specific dimensions. The XML code is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context=".MainActivity"
7     android:orientation="vertical">
8
9     <ImageView
10        android:layout_width="200dp"
11        android:layout_height="200dp"/>
12
13
14 </LinearLayout>
```

Para especificar uma imagem no ImageView, basta colocar no atributo android:src="" o nome de uma imagem presente dentro da pasta res/drawable (sem a extensão do arquivo) conforme o exemplo abaixo:

The screenshot shows the Android Studio project structure on the left and the XML code editor on the right. In the project structure, the 'drawable' folder contains 'ic_launcher_background.xml', 'ic_launcher_foreground.xml (v24)', and 'p1.png'. The 'layout' folder contains 'activity_main.xml'. In the XML code editor, the 'activity_main.xml' file is open, showing the following code:

```
7     android:orientation="vertical">
8
9         <ImageView
10            android:layout_width="200dp"
11            android:layout_height="200dp"
12            android:src="@drawable/p1" />
13
14
```

Para inserir imagens dentro da pasta Drawable, basta copiar a imagem (CTRL + C) de algum lugar de seu computador, clicar em cima da pasta Drawable e colar (CTRL + V).

OBS: Arrastar a imagem não funciona!

Para centralizar a nossa imagem na horizontal de nossa LinearLayout, usaremos o atributo `android:layout_gravity="center_horizontal"`:

```
<ImageView  
    android:layout_gravity="center_horizontal"  
    android:layout_width="200dp"  
    android:layout_height="200dp"  
    android:src="@drawable/p1"/>
```

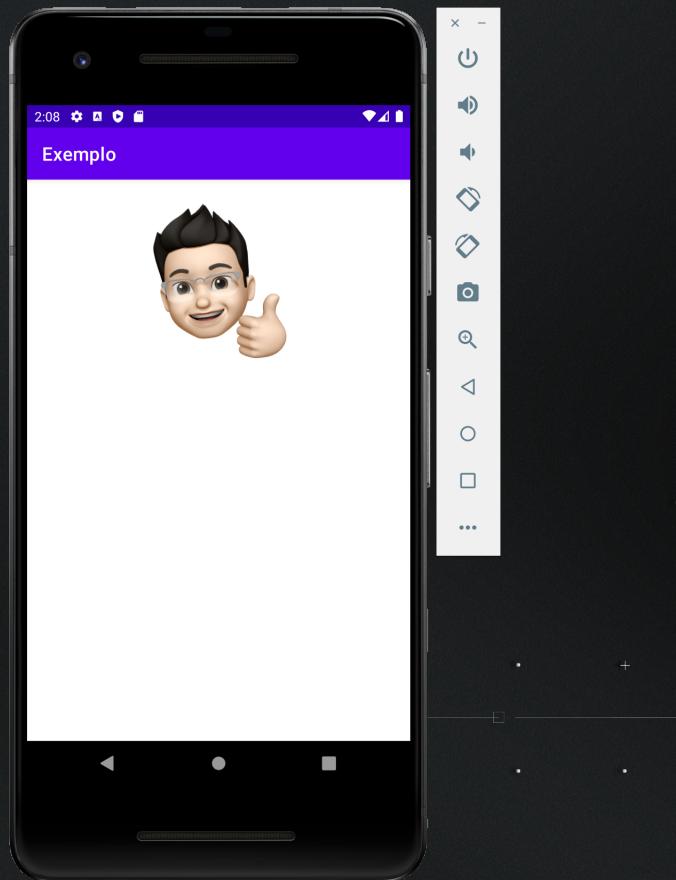
Para colocar um "respiro" entre os componentes e os cantos da tela, vamos inserir um padding em nosso LinearLayout conforme a imagem abaixo:

```
<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"  
    android:orientation="vertical"  
    android:padding="16dp">
```

Criação de uma Interface Gráfica

O resultado de nossas modificações até o momento:

(Obs: A imagem pode variar de acordo com a inserida por você!)



Criação de uma Interface Gráfica

Vamos inserir agora, logo abaixo da imagem que colocamos anteriormente os seguintes componentes: `TextView`, `EditText` e um `Button`.

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Informe seu nome: "/>  
  
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"/>  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Clique-me"/>
```

Vamos inserir um evento para o clique do botão. Após inserir o nome do evento no atributo onClick, basta pressionar ALT + ENTER com o cursor do mouse piscando em cima do nome do evento para que o Android Studio crie o método necessário para nós em nossa Activity:

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Clique-me"  
    android:onClick="cliqueBotao"/>
```

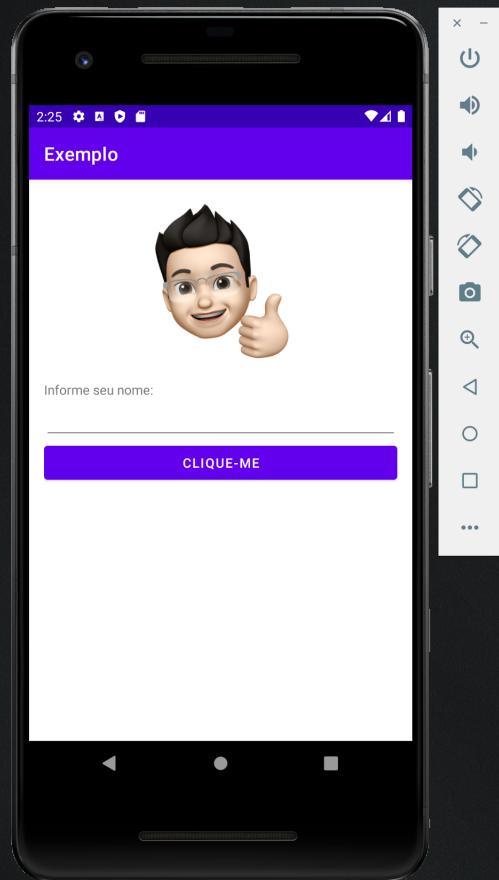
Na classe
Activity

```
fun cliqueBotao(view: View) {  
}  
}
```

Criação de uma Interface Gráfica

FIAP

Resultado



Na classe em Kotlin, pegamos o texto de nosso EditText. Para isso precisamos que no XML de nossa Layout, o EditText esteja identificado através de um ID.

```
<EditText
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/txtNome"/>
```

Antes de manipular os atributos do id informado devemos modificar o nosso arquivo build.gradle (Module:) e adicionar o seguinte import na classe de Activity.

build.gradle

```
plugins {  
    id 'com.android.application'  
    id 'kotlin-android'  
    id 'kotlin-android-extensions'  
}
```

Activity

```
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.view.View  
import kotlinx.android.synthetic.main.activity_main.*
```

Com o id informado, agora podemos manipular seus atributos em nossa classe de Activity.

No método para o botão criado anteriormente vamos exibir um Toast dando um olá para o nome informado:

```
fun cliqueBotao(view: View) {  
    val msg = "Olá ${txtNome.text}"  
    Toast.makeText(context, msg, Toast.LENGTH_LONG).show()  
}
```

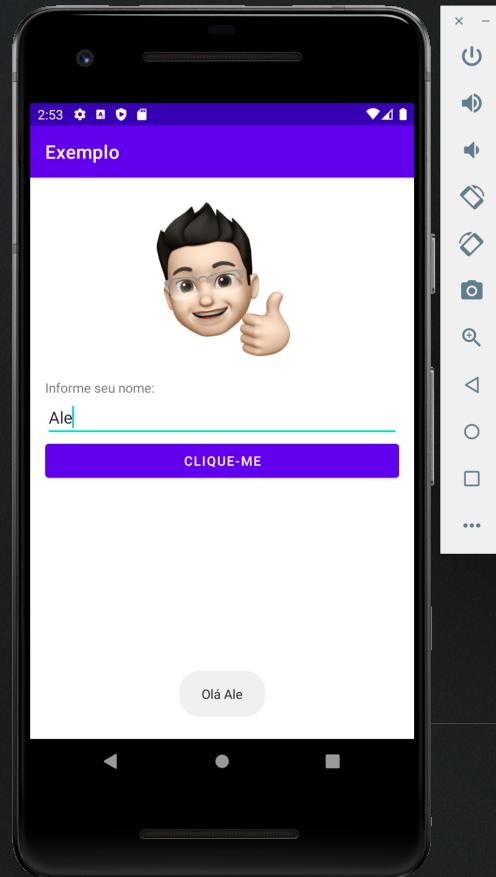
Criação de uma Interface Gráfica

O resultado final de nossa aplicação ao executar em um dispositivo será:

OBS: O tempo de exibição do Toast poderá ser definido apenas de duas formas, utilizando as seguintes constantes em sua criação:

`Toast.LENGTH_SHORT` ⇒ para um tempo de exibição mais curto.

`Toast.LENGTH_LONG` ⇒ para um tempo de exibição mais longo.

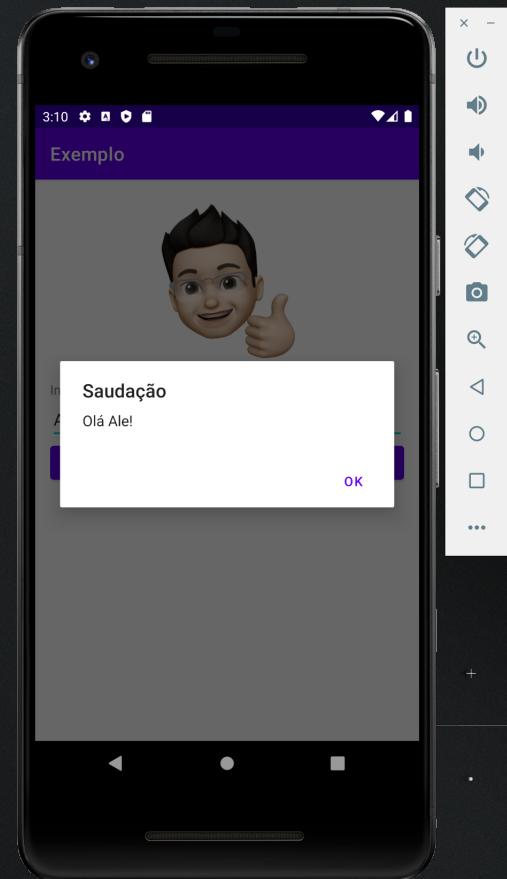


Criação de uma Interface Gráfica

FIAP

Vamos modificar o nosso exemplo para apresentar a mensagem de saudação em uma caixa de diálogo. Para isso devemos utilizar o seguinte código na classe Activity.

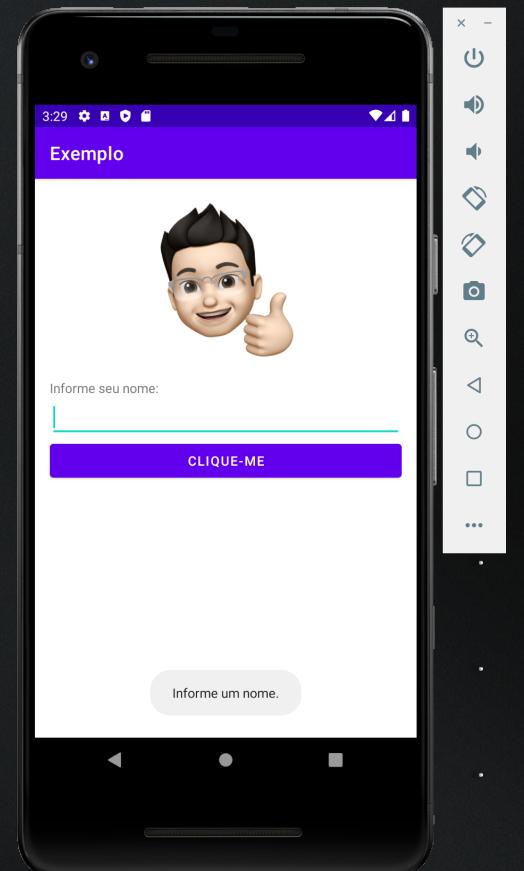
```
fun cliqueBotao(view: View) {  
    val msg = "Olá ${txtNome.text}!"  
    exibirMensagem( titulo: "Saudação", msg)  
}  
  
fun exibirMensagem(titulo: String, mensagem: String) {  
    val builder = AlertDialog.Builder( context: this)  
    builder  
        .setTitle(titulo)  
        .setMessage(mensagem)  
        .setPositiveButton( text: "OK", listener: null)  
    builder.create().show()  
}
```



Criação de uma Interface Gráfica

Vamos modificar novamente o nosso exemplo para adicionar uma validação da entrada do usuário. É necessário que ele informe algum texto para apresentar a saudação. Caso o campo esteja vazio apresentar uma mensagem em um Toast.

```
fun cliqueBotao(view: View) {
    if (txtNome.text.trim().isEmpty()) {
        Toast.makeText(context, "Informe um nome.", Toast.LENGTH_LONG).show()
    } else {
        val msg = "Olá ${txtNome.text}!"
        exibirMensagem(titulo: "Saudação", msg)
    }
}
```



Dúvidas?



