

# Android Studio

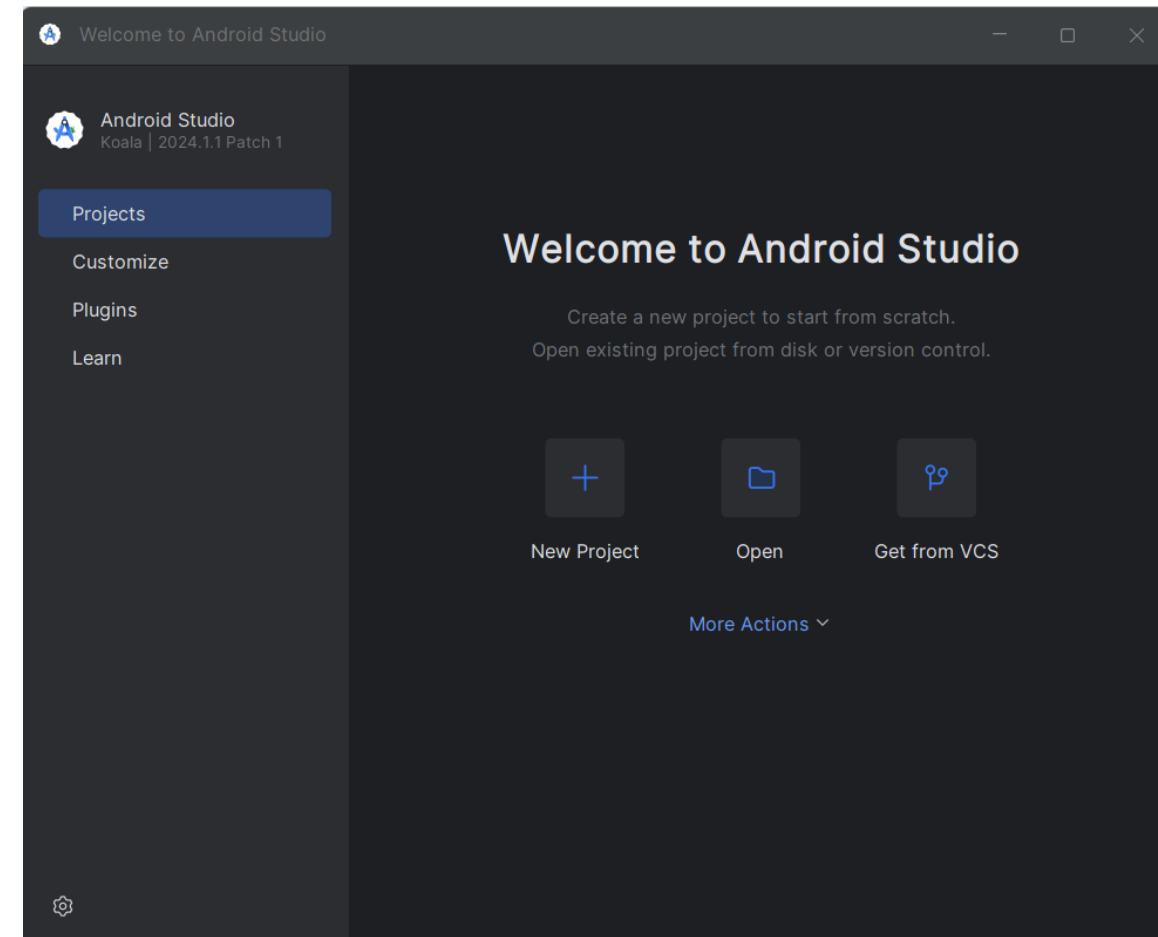


# IDE - Android Studio

- Embora os aplicativos Android possam ser desenvolvidos usando qualquer ambiente de programação, o **IDE oficial e melhor para programação Android é o Android Studio**.
- Este é um fork do aplicativo **IntelliJ IDEA da JetBrains** — um IDE Java personalizado para desenvolvimento Android.
- O pacote **Android Studio** inclui o **Android SDK (Standard Development Kit)**: as ferramentas e bibliotecas necessárias para o desenvolvimento do Android. incluindo:
  - **adb**, o “**Android Device Bridge**”, que é uma **conexão entre seu computador e o dispositivo (físico ou virtual)**. Esta ferramenta é usada para saída do console!
  - **emulador**, que executa o **emulador do Android**: uma máquina virtual de um dispositivo Android.

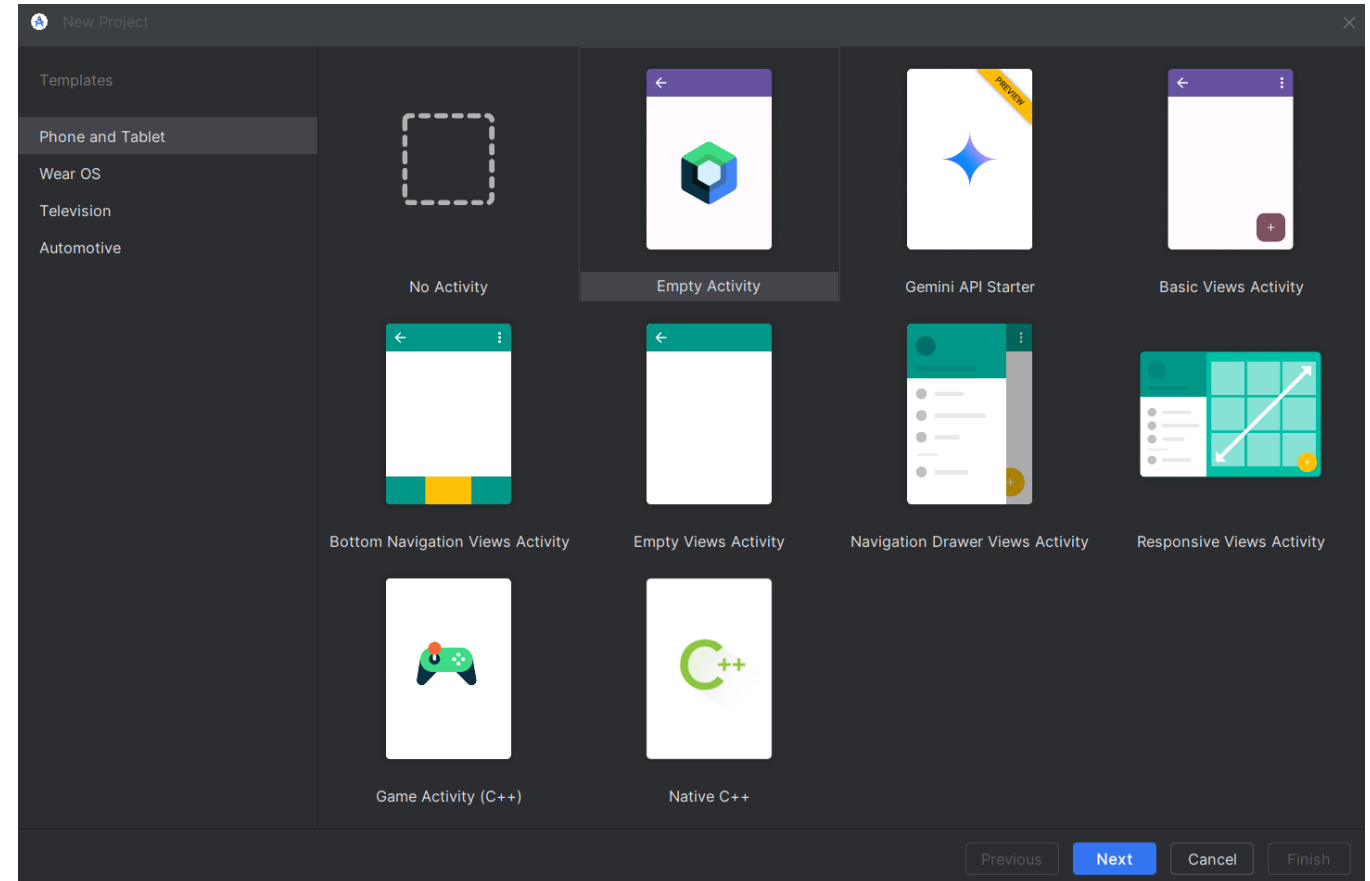
# Criando um Projeto

- Abra o Android Studio
- Escolha **Projects/New Projects**
- Aparecerão diversos templates, usados em diversas tipos de aplicações



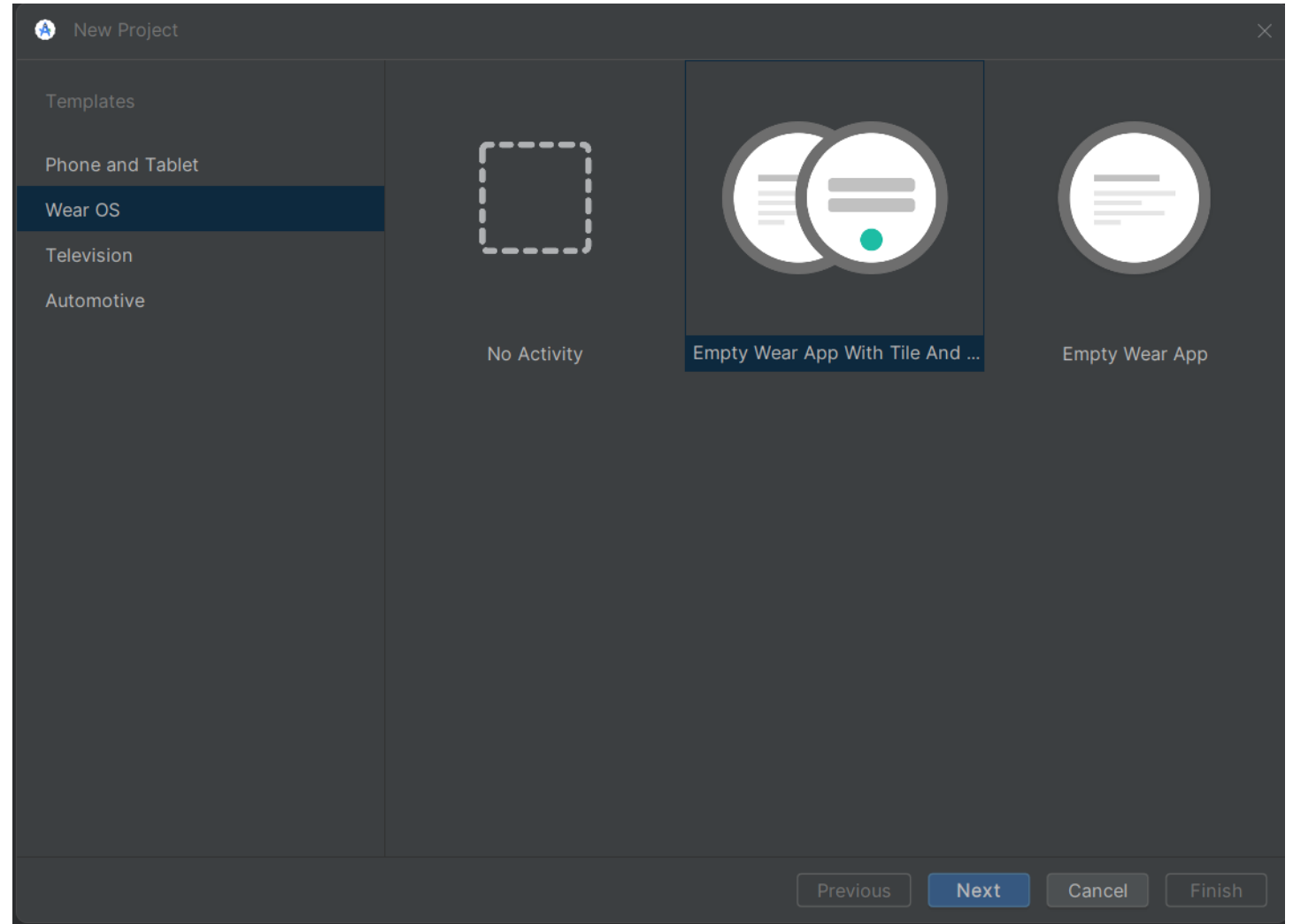


# Phone e Tablets



# Wearables

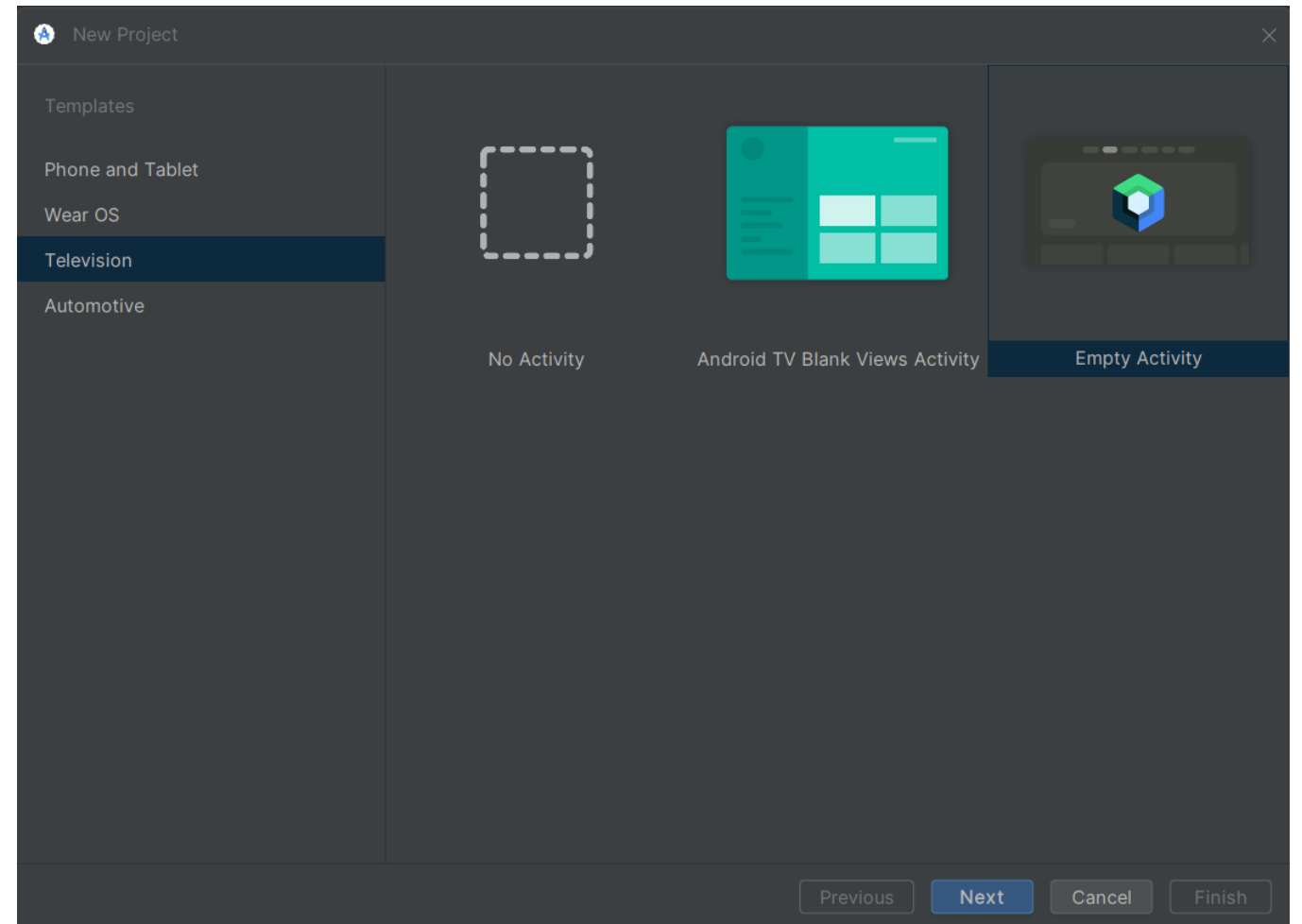
---





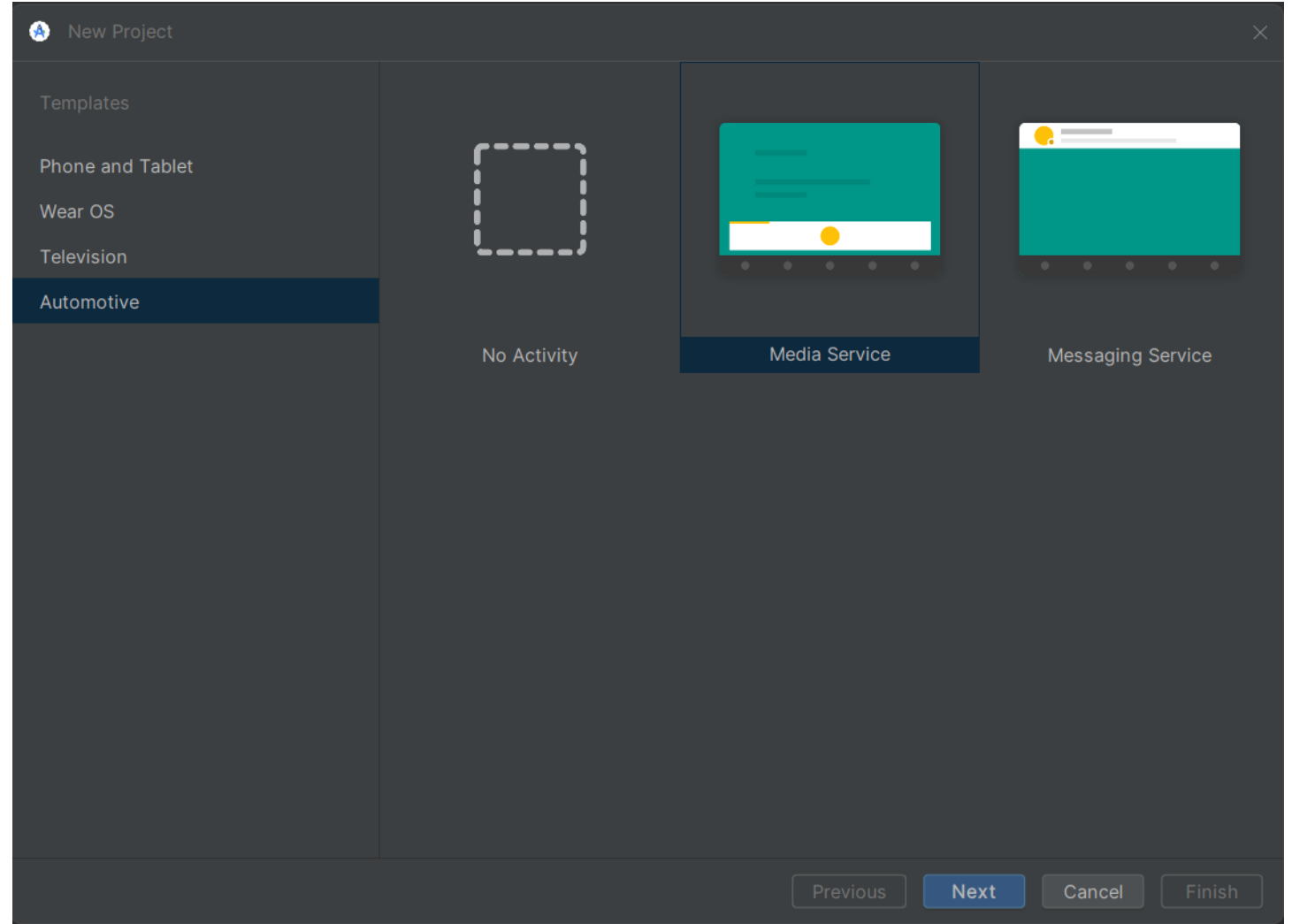
# Television

---

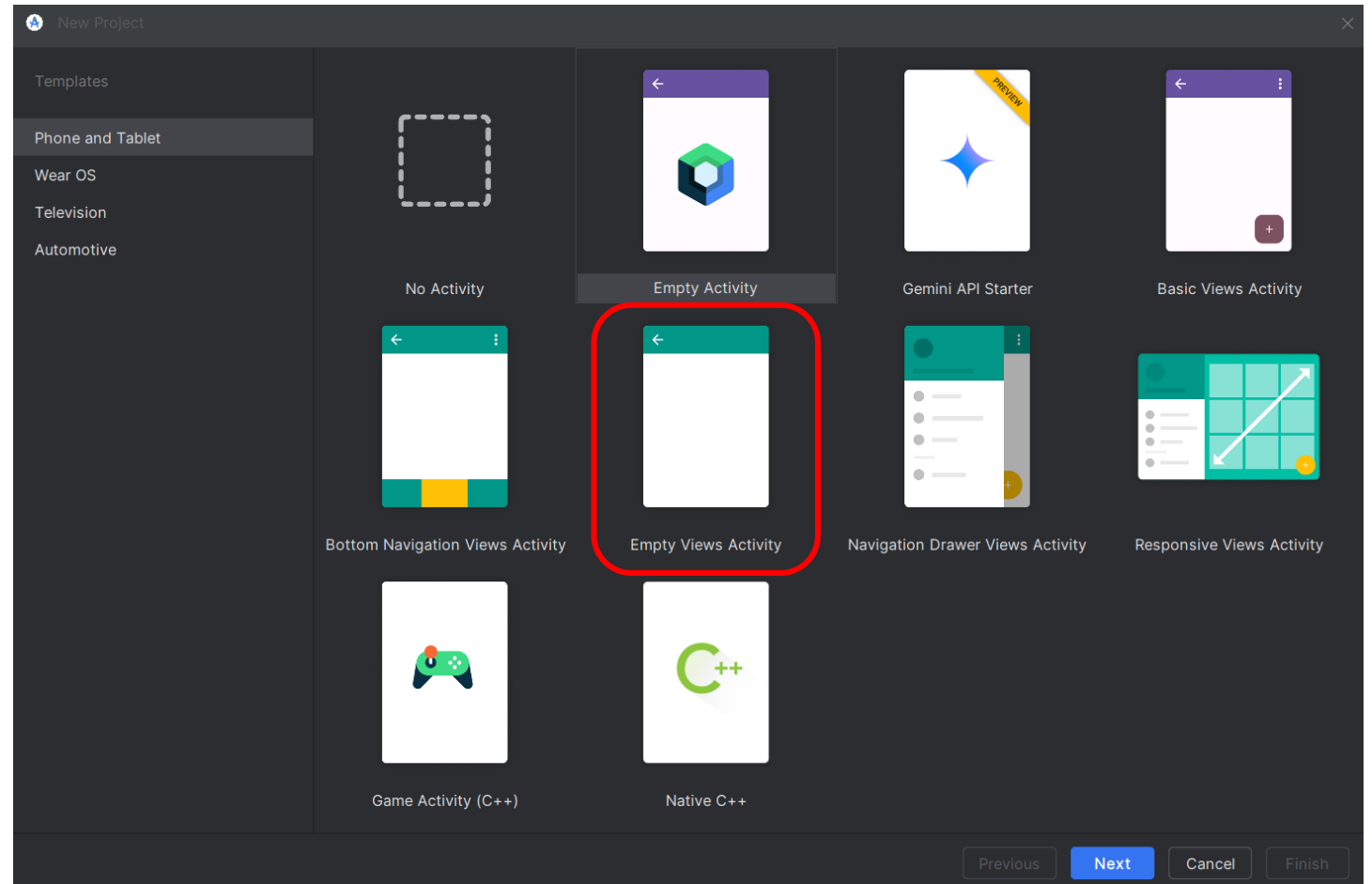


# Automotive

---



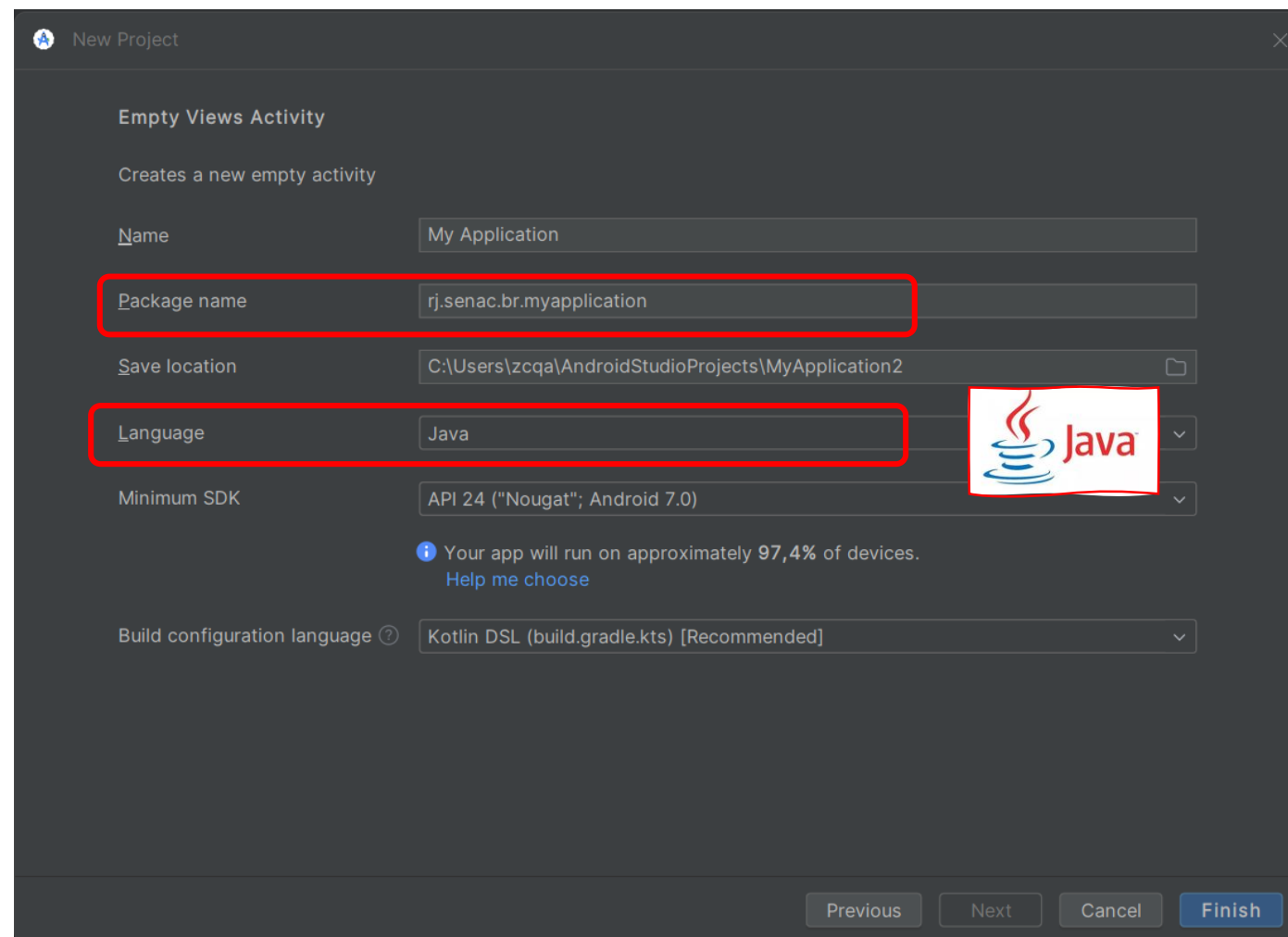
# Escolha o template Phone e Tablets



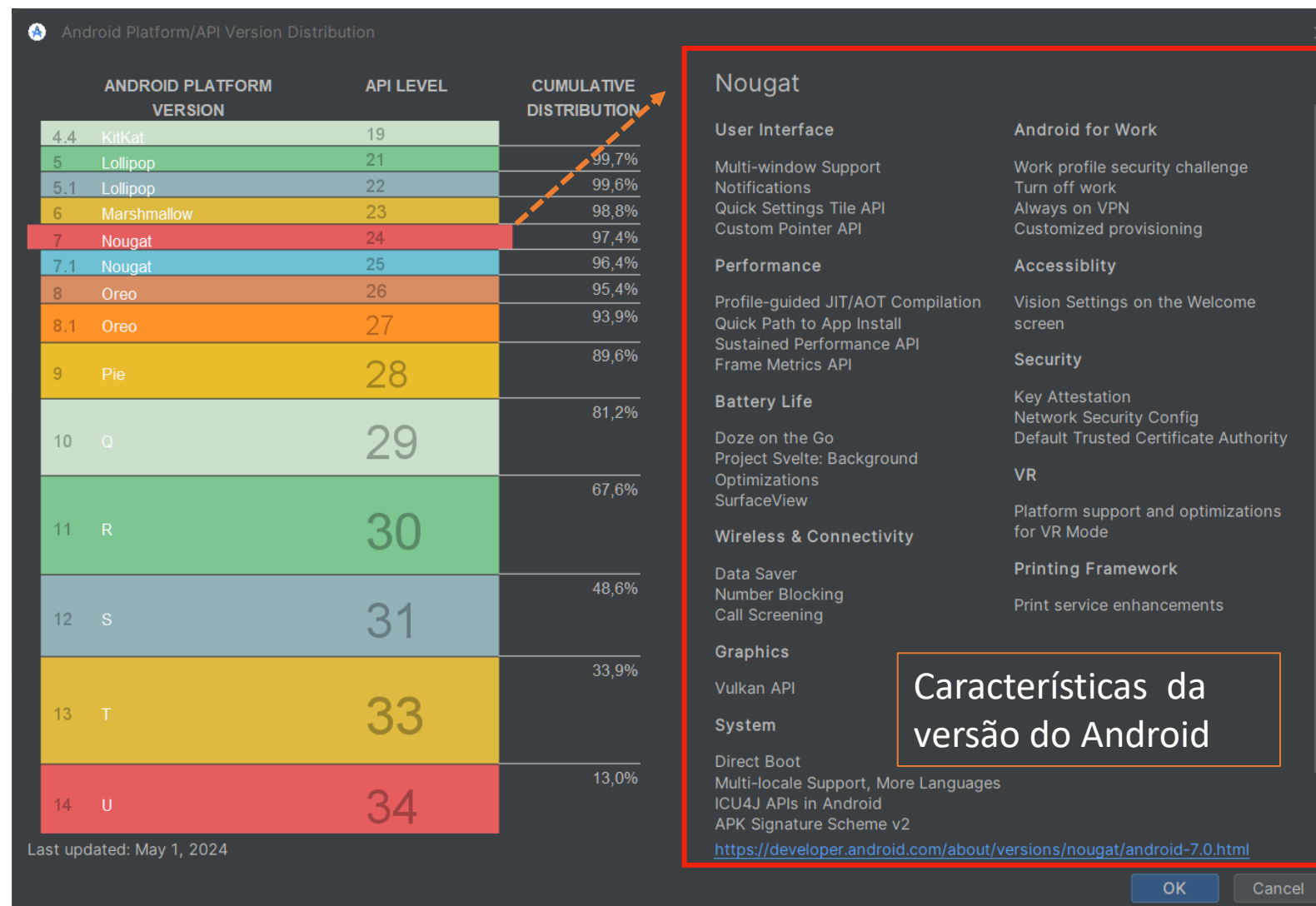
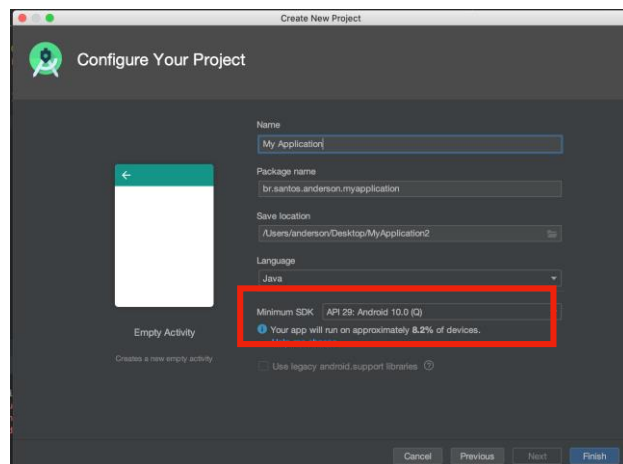


# Criando um projeto...

- Vamos usar o **JAVA** neste curso, apesar de diversas outras atualmente disponíveis
- O “**Domínio da empresa**” deve ser um domínio exclusivo para você. Para este curso, você deve incluir **rj.senac.br**
- Em seguida é necessário escolher **o nível mínimo do SDK** que deseja oferecer suporte, ou seja, qual é a versão mais antiga do Android em que seu aplicativo poderá ser executado
- Tecle “**Finish**” e aguarde. O Android Studio levará alguns minutos para criar seu projeto e configurar tudo. (Fique de olho na barra de status inferior para esperar que tudo seja concluído).

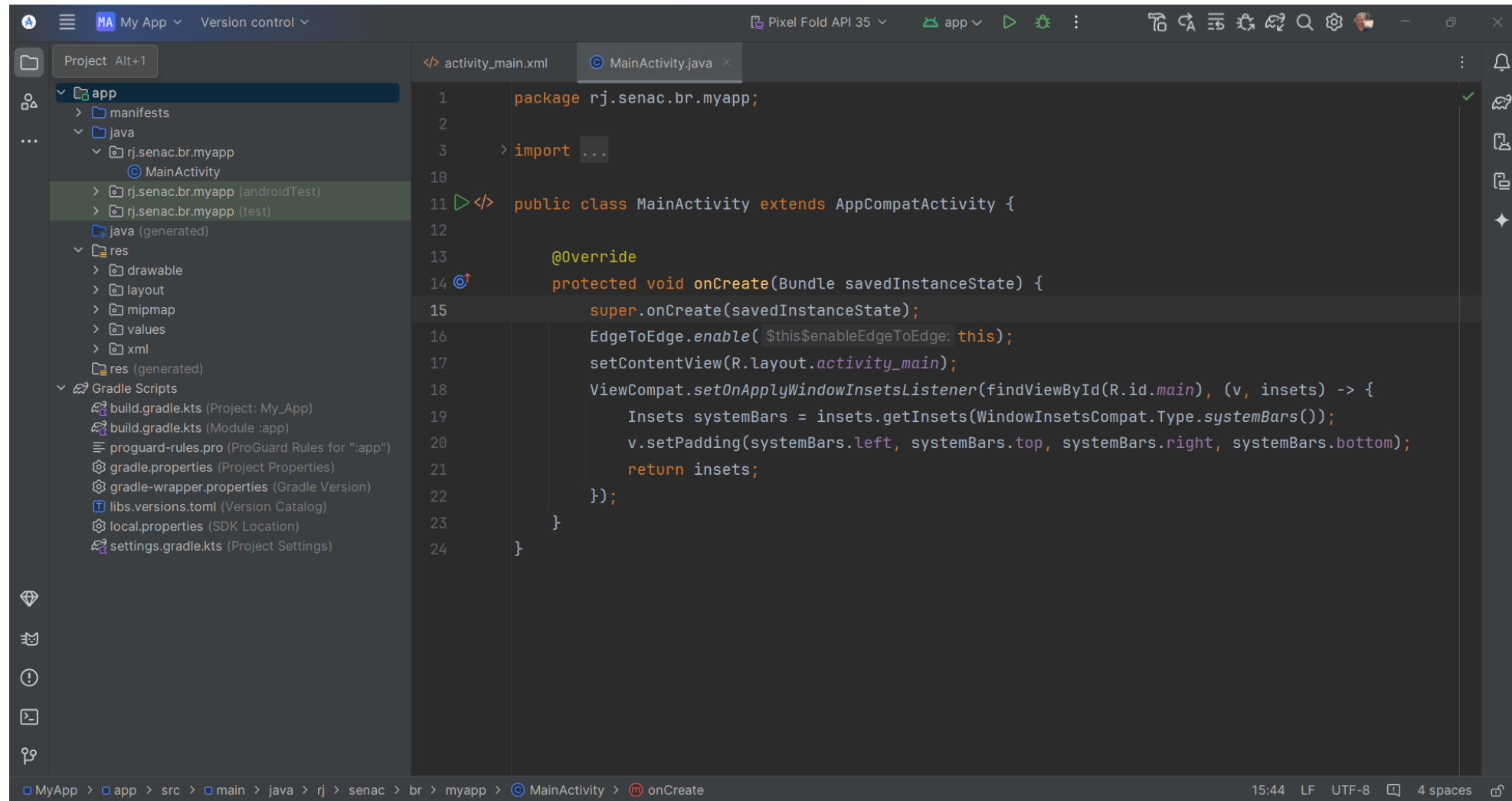


# Criando um projeto... Compatibilidade



Características da versão do Android

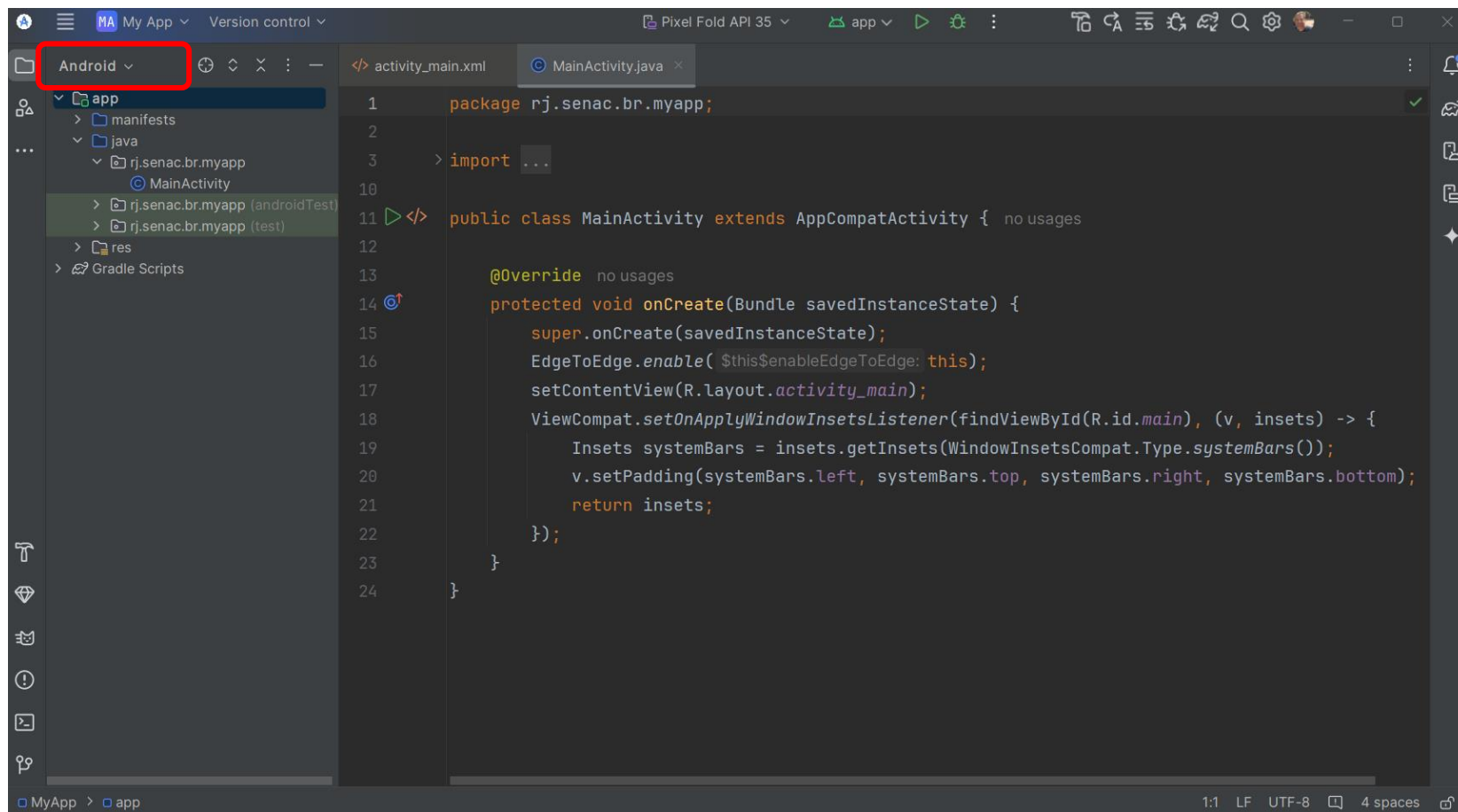
# Tela do Projeto no Android Studio



# Entendendo o IDE

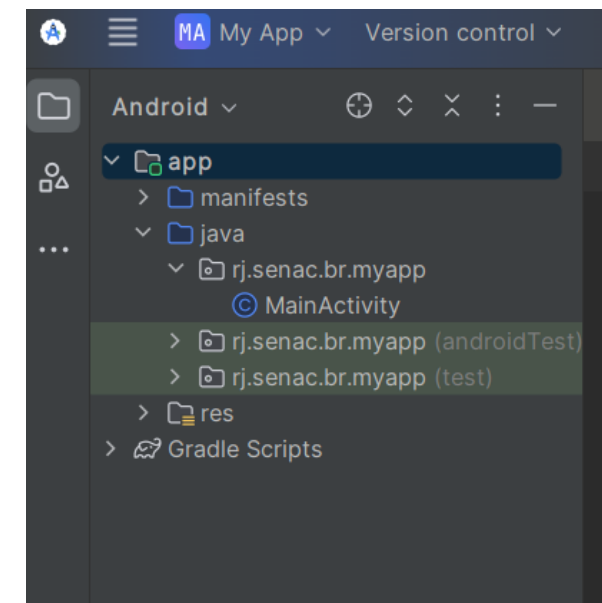
---

- O **Android Studio** criará um monte de arquivos de projeto por padrão — quase todos são usados para alguma coisa.
- Por padrão, ele mostrará seu projeto usando a **visualização do Android**, que organiza os arquivos tematicamente.

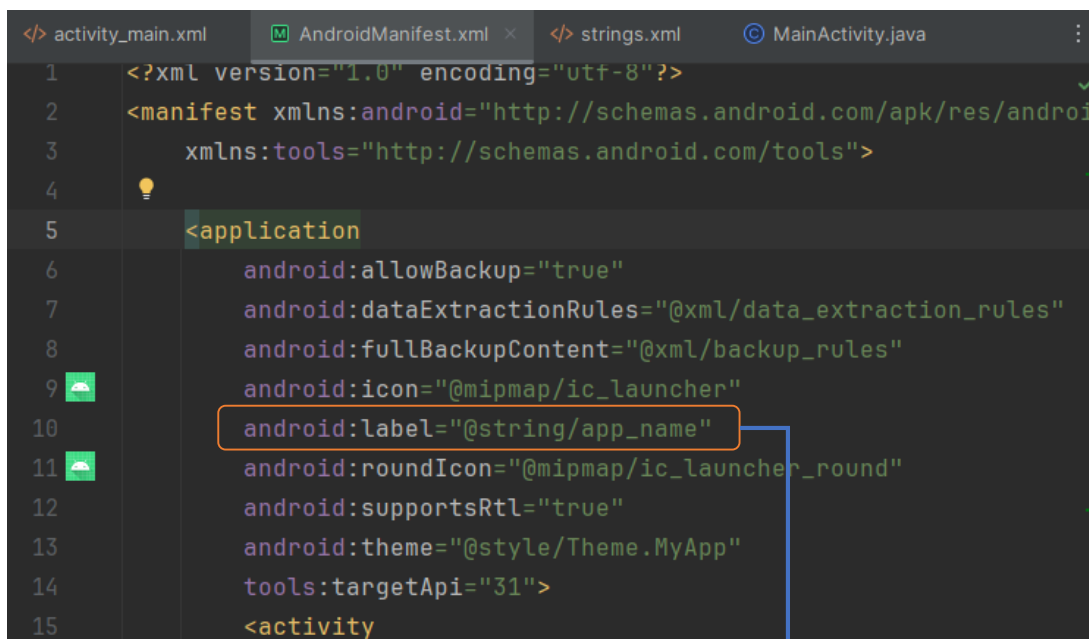


# Entendendo o IDE

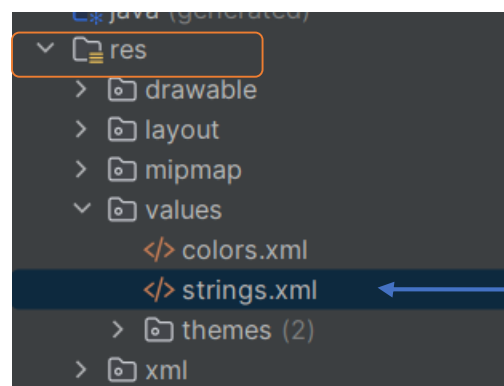
- Na visualização do Android, os arquivos são organizados da seguinte forma:
- **app/** contém o **código-fonte do nosso aplicativo**
  - **manifests/** contém os arquivos do Android Manifest, que são como **um arquivo de "config" para o aplicativo**
  - **java/** contém o **código-fonte Java** para seu projeto. É aqui que a "lógica" do aplicativo vai.
  - **res/** contém arquivos de **recursos XML usados no aplicativo**. É aqui que colocaremos **as informações de layout/aparência**
- **Gradle Scripts** contém scripts para a ferramenta de construção Gradle, que é usada para ajudar a compilar o código-fonte para instalação em um dispositivo.



# manifests/ - Manifesto



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApp"
        tools:targetApi="31">
        <activity
```



- O **manifesto** atua como um **arquivo de "configuração" para o aplicativo**, especificando detalhes de nível de aplicativo, como **nome, ícone e permissões do aplicativo**.
- Por exemplo, você pode **alterar o nome exibido do aplicativo** modificando o atributo **android:label** do elemento **<application>**. Por padrão, o rótulo é uma **referência** a outro recurso encontrado no arquivo **res/values/strings.xml**, que contém definições para "constantes" de string. O ideal é que **todas as strings voltadas para o usuário** — incluindo coisas como texto de botão — sejam definidas como essas constantes.
- Geralmente é preciso fazer pelo menos uma alteração no manifesto para cada aplicativo (por exemplo, ajustar o nome de exibição).

# AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.anderson.logincompleto">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <service
            android:name=".ServiceLogin"
            android:enabled="true"
            android:exported="true"></service>

        <activity android:name=".Cadastro" />
        <activity android:name=".Status" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# res/ - XML Resources

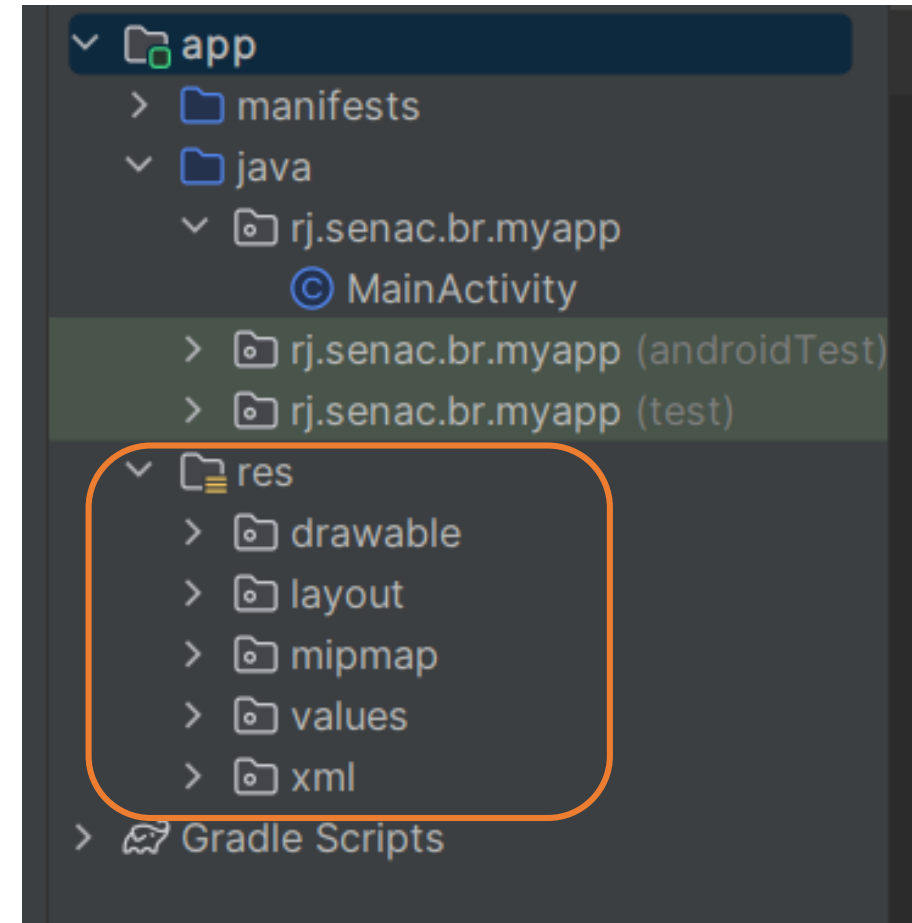
- A pasta **res/** contém **arquivos de recursos**.
- **Arquivos de recursos** são usados para **definir a interface do usuário** e outros ativos de mídia (imagens, etc.) para o aplicativo.
- Usar arquivos separados para definir a **interface do aplicativo** do que aqueles usados para a **lógica do aplicativo** (o código Java) ajuda a **manter a aparência e o comportamento separados**.
  - Comparando **om a programação da web**: os **recursos** contêm o conteúdo **HTML/CSS**, enquanto o **código Java** conterá o que normalmente seria escrito em **JavaScript**.
- A grande maioria dos arquivos de recursos são **especificados em XML**, que tem exatamente a **mesma sintaxe do HTML**, mas você pode criar suas próprias tags com quaisquer valores semânticos que desejar. Exceto que usaremos as tags que o Android criou e forneceu.
- Portanto, **definir uma interface de aplicativo Android** será muito parecido com **definir uma página da web**, mas com um novo conjunto de elementos.



# res/ - XML Resources

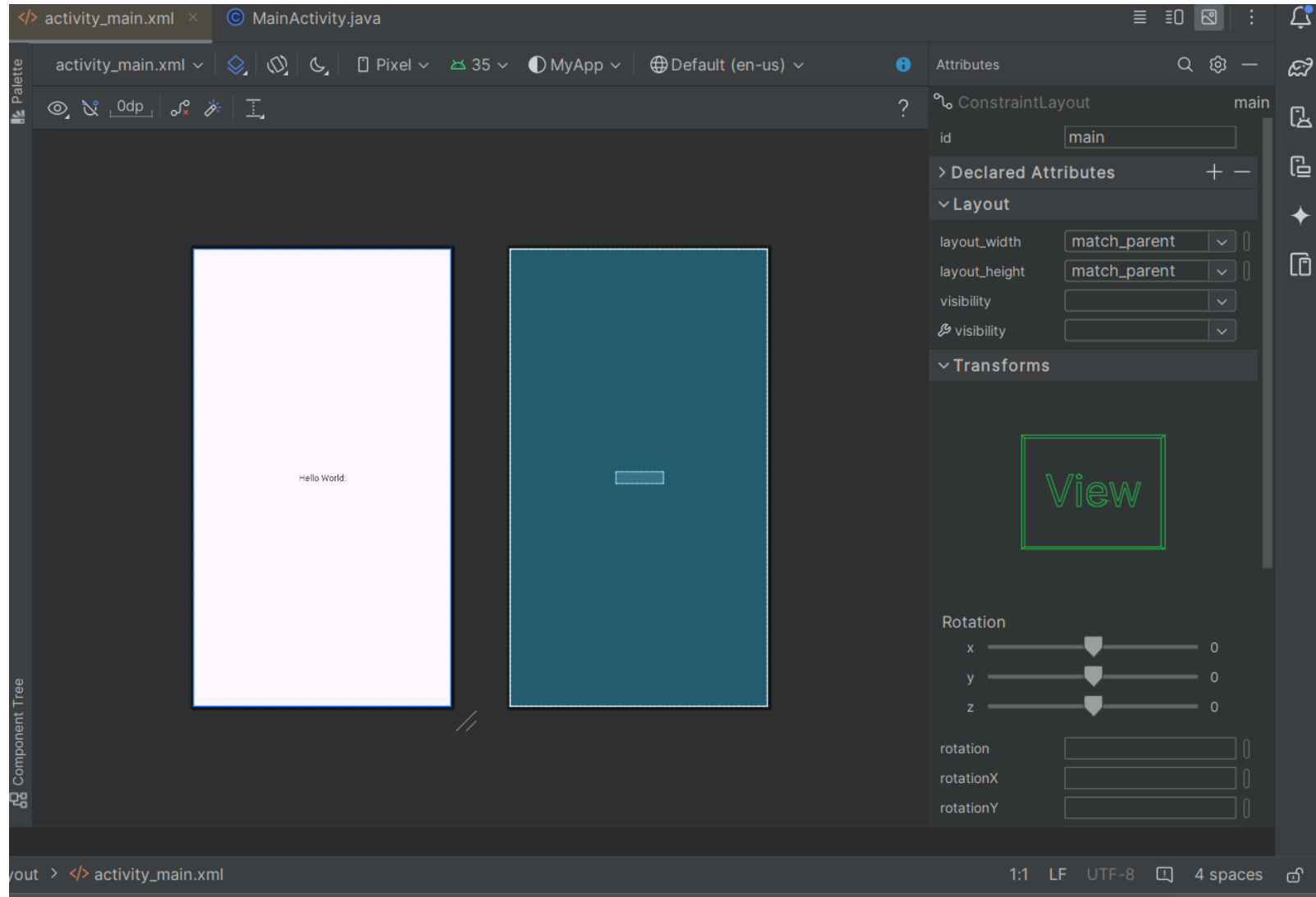
---

- Há um grande número de diferentes tipos de recursos, que são organizados em diferentes pastas:
- **res/drawable/**: contém gráficos (PNG, JPEG, etc.) que serão “desenhados” na tela
- **res/layout/**: contém arquivos de **layout XML** da interface do usuário para o conteúdo do aplicativo
- **res/mipmap/**: contém arquivos de **ícones** do iniciador em diferentes resoluções para oferecer suporte a diferentes dispositivos
- **res/values/**: contém definições XML para **constantes** gerais



# activity\_main.xml – Visão Gráfica

- Se você abrir um arquivo de layout (por exemplo, **activity\_main.xml**) no Android Studio, por padrão ele será exibido em uma visualização “Design”.
- Essa visualização permite que você use um sistema gráfico para dispor seu aplicativo, semelhante ao que você faria com um slide do PowerPoint.



# activity\_main.xml – Visão Gráfica

- É possível mudar a visualização

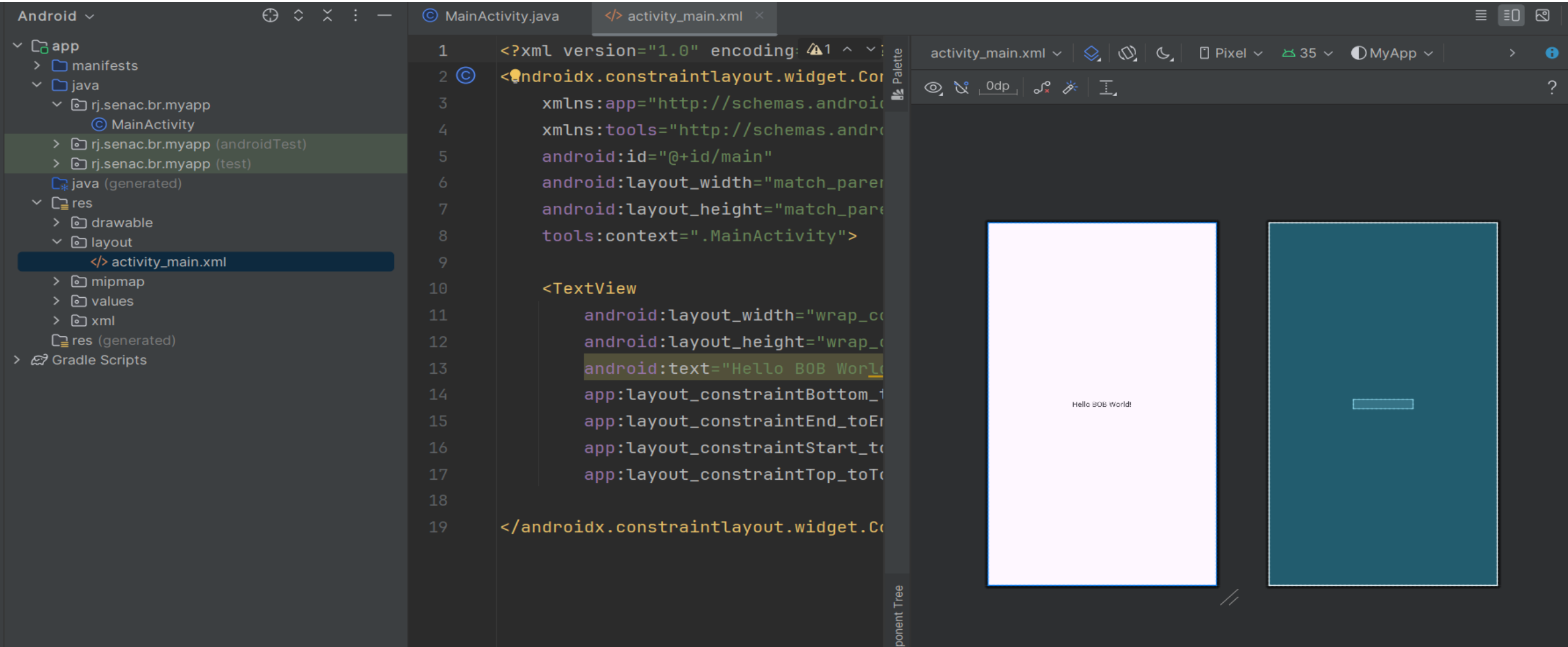
Modo exibição



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello BOB World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintEnd_toEndOf="parent"
16         app:layout_constraintStart_toStartOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

# activity\_main.xml – Visão Gráfica

- É possível mudar a visualização



# activity\_main.xml – Visão Código

- Na **visualização do código**, você pode ver o **XML: tags, atributos, valores**. Os elementos são aninhados uns dentro dos outros.
- O código XML fornecido **define um layout** (um `<android.support.constraint.ConstraintLayout>`) para organizar as coisas, e dentro dele há um `<TextView>` (uma View representando algum texto).
- Observe que a maioria dos atributos dos elementos são **namespaced**, por exemplo, com um prefixo **android:**, para evitar quaisquer conflitos potenciais.
- O atributo **android:text** do `<TextView>` contém algum **texto**. Você pode alterá-lo e executar o aplicativo novamente para vê-lo atualizar!

```
activity_main.xml x MainActivity.java
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:and
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:id="@+id/main"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context=".MainActivity">
9
10    <TextView
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="Hello World!"
14        app:layout_constraintBottom_toBottomOf="parent"
15        app:layout_constraintEnd_toEndOf="parent"
16        app:layout_constraintStart_toStartOf="parent"
17        app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

# Java/

- Os aplicativos **Android são escritos em Java**, com o código-fonte encontrado na pasta java/ na visualização do projeto Android (em uma estrutura de pasta aninhada com base no nome do pacote do seu aplicativo).
- O código Java lida com o controle e a lógica do programa, bem como com o armazenamento e a manipulação de dados.
- Escrever código **Android será muito parecido com escrever qualquer outro programa Java**: você cria classes, define métodos, instancia objetos e chama métodos nesses objetos.
- Mas, como você está **trabalhando dentro de uma estrutura**, há um **conjunto de códigos existentes para chamar métodos específicos**.
- Como desenvolvedor, sua tarefa será preencher o que esses métodos fazem para executar seu aplicativo específico

# Java

- Então, embora vá se usar **classes e modelos Java “normais”** no código, mais frequentemente será utilizado um **conjunto específico de classes exigido pelo framework**, dando aos aplicativos Android uma estrutura comum.
- O **componente mais básico** em um programa Android é uma **Activity**, que **representa uma única tela no aplicativo**.
- A **classe MainActivity** fornecida por padrão é um exemplo disso: a classe **estende Activity** (na verdade, ela estende uma subclasse que suporta componentes do Material Design), **permitindo que você faça suas próprias personalizações** no comportamento do aplicativo dentro do framework Android.

```
11 > </> public class MainActivity extends AppCompatActivity {  
12  
13     @Override
```



# Java

- Nesta classe, substituímos o método **onCreate()** herdado, **que é chamado pelo framework quando a Activity inicia** — esse método age um pouco **como o construtor** de uma classe .
- Chamamos o método **super** para garantir que o framework faça suas coisas e, em seguida, **setContentView()** para especificar qual deve ser o conteúdo (aparência) da Activity.

```
11 </> public class MainActivity extends AppCompatActivity { no usages
12
13     @Override no usages
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         EdgeToEdge.enable( $this$enableEdgeToEdge: this);
17         setContentView(R.layout.activity_main);
18         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
19             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
20             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
21             return insets;
22         });
23     }
24 }
```



# Java

- O **conteúdo** é passado em um valor chamado **R**.
  - **R é uma classe** que é gerada em tempo de compilação e contém constantes que são definidas pelos arquivos de "recurso" XML! Esses arquivos são convertidos em variáveis Java, sendo acessados por meio da **classe R**.
- Portanto, **R.layout.activity\_main** se refere ao **layout activity\_main** encontrado na pasta **res/layouts/**. É assim que o Android sabe qual arquivo de layout mostrar na tela.

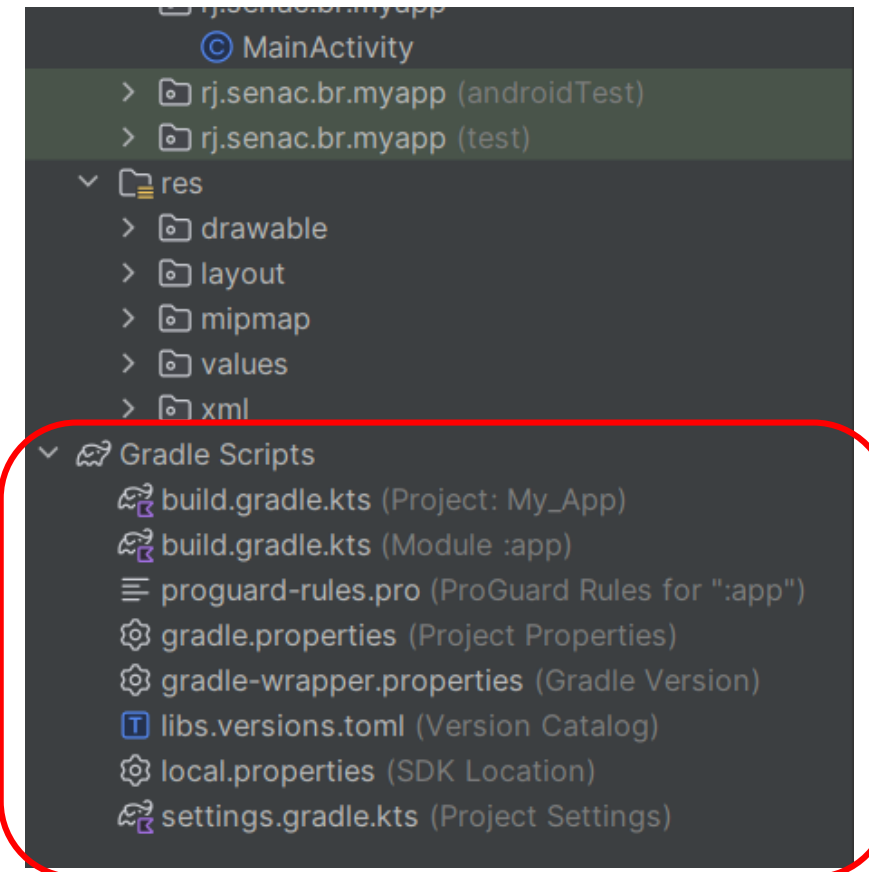
```
11 ></> public class MainActivity extends AppCompatActivity { no usages
12
13     @Override no usages
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         EdgeToEdge.enable( $this$enableEdgeToEdge: this);
17         setContentView(R.layout.activity_main);
18         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
19             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
20             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
21             return insets;
22         });
23     }
24 }
```

# O que é uma classe R?

- Os recursos do aplicativo Android são colocados em uma estrutura de diretório sob a raiz **res** (não confunda isso com o diretório de recursos do aplicativo Java), mas podem diferir em um formato.
- Ícones e layouts são colocados em seus arquivos individuais, mas as strings podem (ou não) ser armazenadas em um único arquivo e os alvos de navegação são parte de um conteúdo muito mais rico de gráficos de navegação.
- No entanto, **todos os recursos do aplicativo têm um ID de recurso exclusivo** gerado pela ferramenta aapt durante a compilação.
- **Todos os IDs de recursos são listados em uma classe Java com um nome simples — R.**
- Para cada tipo de recurso, há uma classe aninhada com o nome do tipo de recurso (por exemplo, **R.string** para recursos de **string**) e, para cada recurso desse tipo, há um inteiro estático (por exemplo, **R.string.app\_name**).

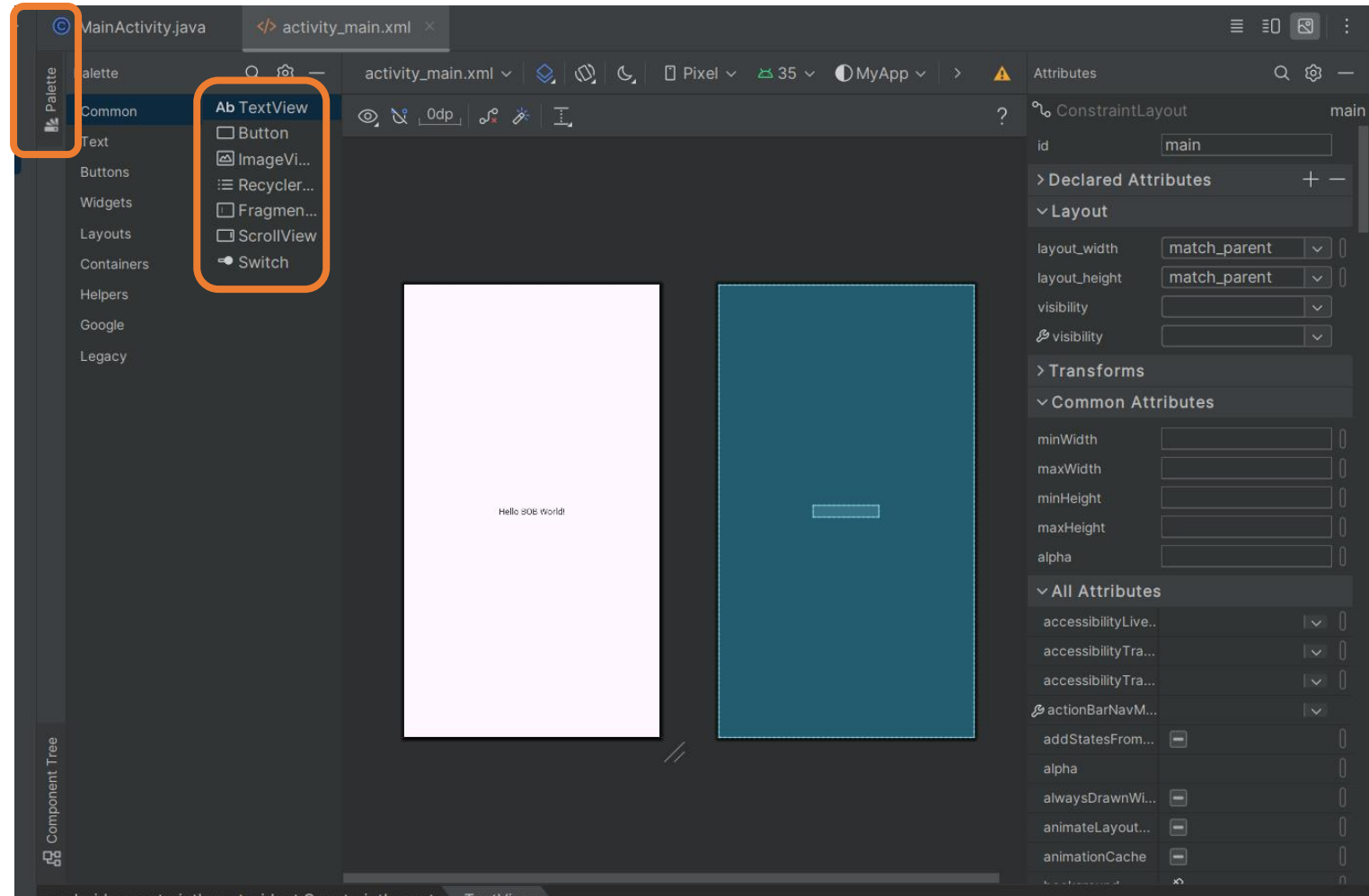
# Gradle Scripts

- Depois de escrever seu código-fonte Java e XML, para "**construir**" e **executar seu aplicativo**, é necessário:
  1. Gerar arquivos de origem Java (por exemplo, R) a partir dos arquivos de recursos XML
  2. Compilar o código Java (ou Kotlin) em bytecode JVM
  3. Empacotar código e outros ativos em um .apk
  4. Assinar criptograficamente o arquivo .apk para autorizá-lo
  5. Transferir o .apk para seu dispositivo, instalar e executá-lo!
- São muitas etapas! Felizmente, o IDE cuida disso para nós usando uma ferramenta de construção automatizada **chamada Gradle**.
- Essas ferramentas permitem que você, na verdade, especifique um único comando que executará todas essas etapas de uma vez.

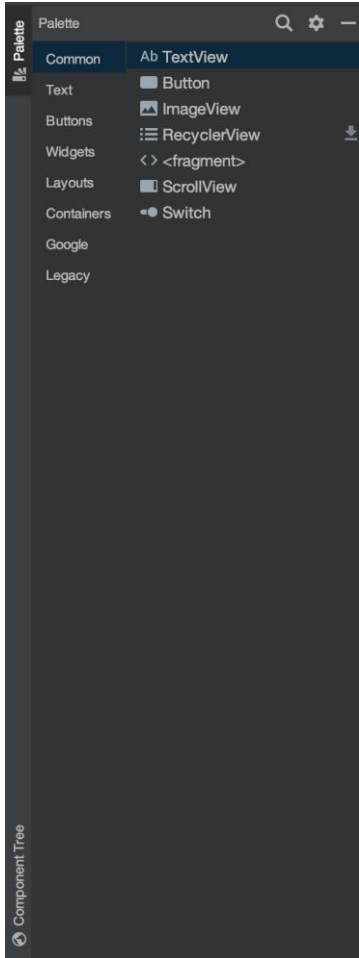


# Paleta

- Na visão palette temos os componentes gráficos para montar o layout da aplicação
- A aba “Common” apresenta os elementos mais comuns utilizados



# Programação gráfica



## Android TextView

No android, o **TextView** é um controle de interface do usuário usado para exibir **um texto**. Mais>> [Android TextView with Examples](#) .

## Android Button

No Android, o **Button** é um controle de interface do usuário usado para executar uma ação quando o usuário clica ou toca nela.

Mais>> [Botões do Android com exemplos](#) .

## RecyclerView

Versão mais avançada do ListView

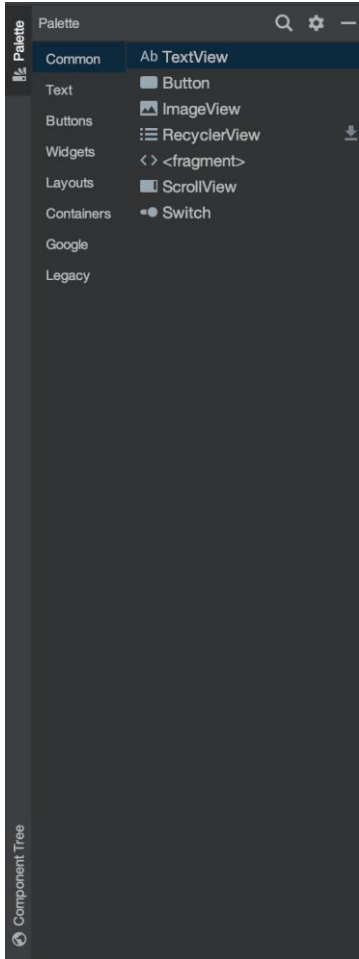
Mais >> [Como criar uma lista com RecyclerView](#)

## Fragment

Um Fragment representa o comportamento ou uma parte da interface do usuário em um FragmentActivity. É possível combinar vários fragmentos em uma única atividade para criar uma IU de vários painéis e reutilizar um fragmento em diversas atividades.

Mais >> [Fragmentos](#)

# Programação gráfica



## ScrollView

Mais>> [ScrollView e HorizontalScrollView no Android, Entendendo e Utilizando](#) .

## Switch

No Android, o Botão Alternar é um controle de interface do usuário usado para exibir os estados LIGADO (Marcado) ou DESLIGADO (Desmarcado) como um botão com um indicador luminoso.

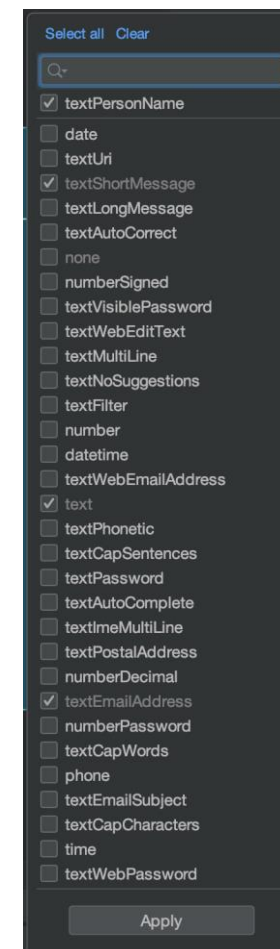
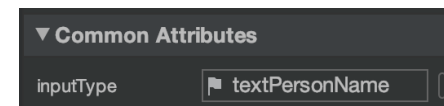
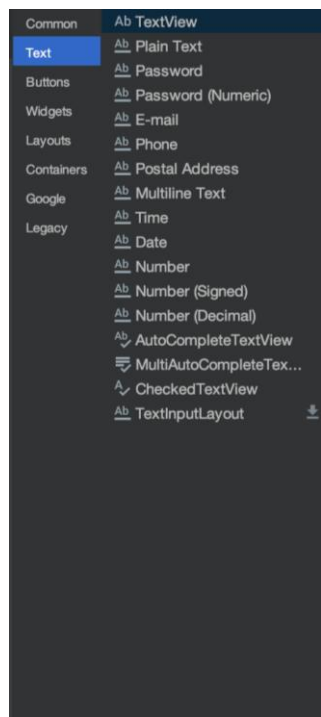
Mais>> [Botão alternância com exemplos](#) .

# Programação gráfica

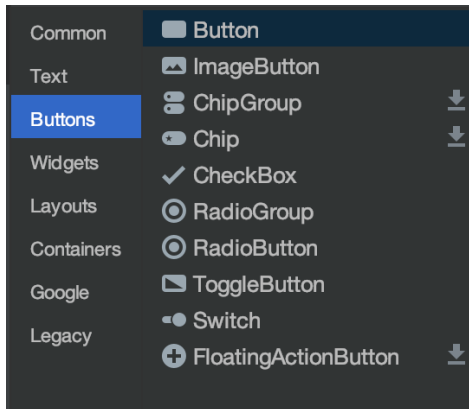
## Android EditText

No android, o **EditText** é um controle de interface do usuário que é usado para permitir que o usuário insira ou modifique o texto.

Mais>> [EditText com exemplos](#)



# Programação gráfica



## Image Button

No Android, o Image Button é um controle de interface do usuário usado para exibir um botão com uma imagem para executar uma ação quando o usuário clica ou toca nele.

Geralmente, o botão Imagem no Android é semelhante ao botão normal e executa as ações da mesma forma que o botão normal, mas a única diferença é que, no botão de imagem, adicionaremos uma imagem em vez de texto.

Mais >> [Android Image Button with Examples.](#)

## Android CheckBox

No Android, o Checkbox é um botão de dois estados que pode ser marcado ou desmarcado.

Mais >> [Android Checkbox with Examples.](#)

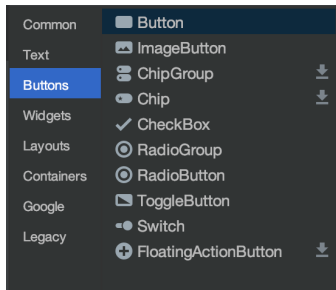
## Android Radio Button

No Android, o botão de opção é um botão de dois estados que pode ser marcado ou desmarcado e não pode ser desmarcado depois de verificado.

Mais >> [Android Radio Button with Examples.](#)



# Prog Disp Móveis



## Android Group Radio

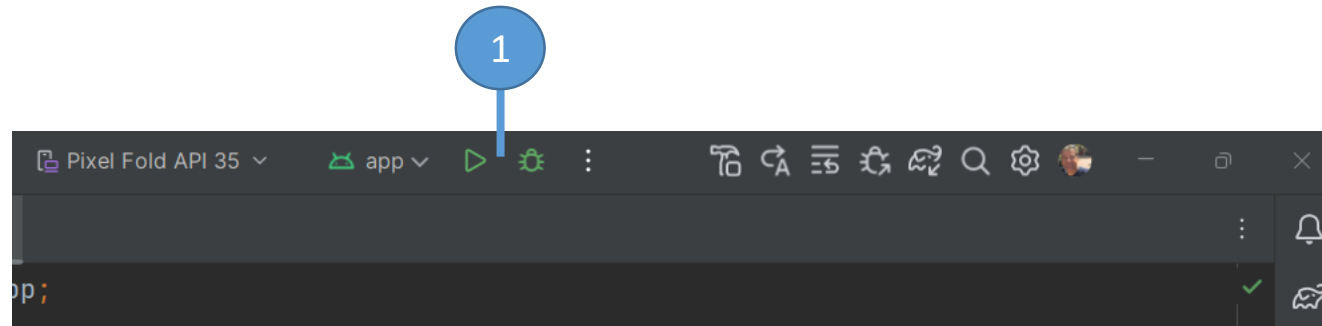
No Android, o Radio Group é usado para agrupar um ou mais botões de opção em grupos separados, com base em nossos requisitos.

No caso de agruparmos botões de opção usando o grupo de opções, por vez, apenas um item pode ser selecionado no grupo de opções.

Mais >> [Android Radio Group with Examples.](#)

# Rodando a aplicação...

- Para executar seu aplicativo clicando no botão “Play” ou “Run” na parte superior do IDE.



- Mas é preciso um dispositivo Android para executar o aplicativo...

# Rodando a aplicação...

- Felizmente, o Android Studio vem com um: **um emulador Android virtual**.
- Este modelo de máquina virtual **emula um dispositivo genérico** com hardware que você pode especificar, embora tenha algumas limitações (por exemplo, nenhum serviço de celular, nenhum bluetooth, etc.).



# Rodando a aplicação...

- Claro que também é possível executar seu aplicativo em um **dispositivo físico**.
- Esses são os melhores para desenvolvimento (eles são a **maneira mais rápida e fácil de testar código**), embora você precise de um cabo USB para poder conectar seu dispositivo ao seu computador.

**Atenção:** Você precisará ativar as **opções do desenvolvedor** para instalar aplicativos de desenvolvimento no seu dispositivo!

# Programação para Dispositivos Móveis

---

Prof. Roberto  
Harkovsky, MsC