

Iniciando no Desenvolvimento Android



Um Guia básico
para Novos
Desenvolvedores

Iniciando a exploração

O desenvolvimento para a plataforma Android tem se tornado uma das habilidades mais requisitadas e empolgantes no campo da tecnologia.

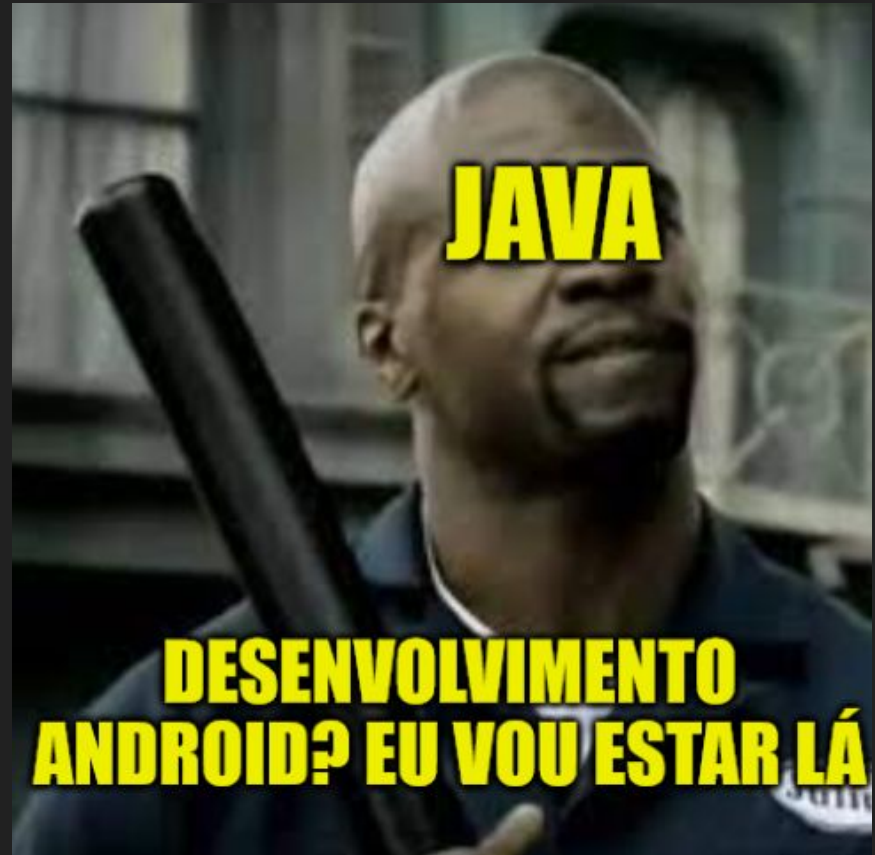
Com bilhões de dispositivos Android ativos em todo o mundo, desde smartphones e tablets até TVs e dispositivos vestíveis, a demanda por aplicativos inovadores e funcionais nunca foi tão alta.

Nesse ebook vamos explorar alguns tópicos sobre o desenvolvimento de programas para sistemas Android.



1. Introdução ao Android:

O Android é um sistema operacional baseado no kernel Linux, desenvolvido principalmente pela Google para dispositivos móveis. Ele fornece uma plataforma robusta e flexível para desenvolver aplicativos usando a linguagem de programação Java ou Kotlin.



2. Android Studio

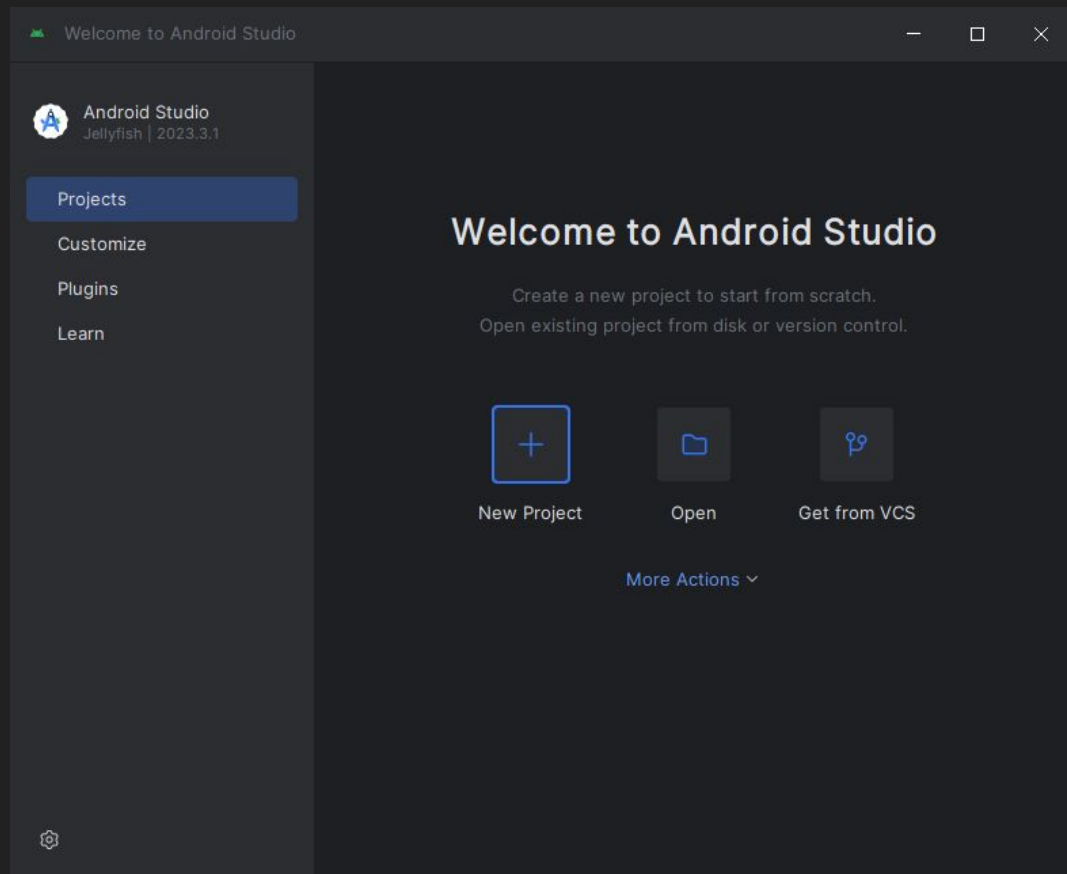
O Android Studio, o ambiente oficial de desenvolvimento, oferece uma variedade de ferramentas poderosas para criar, depurar e otimizar aplicativos Android de forma eficiente.



3. Criando um projeto:

Para criar um novo projeto Android no Android Studio, você pode seguir estes passos:

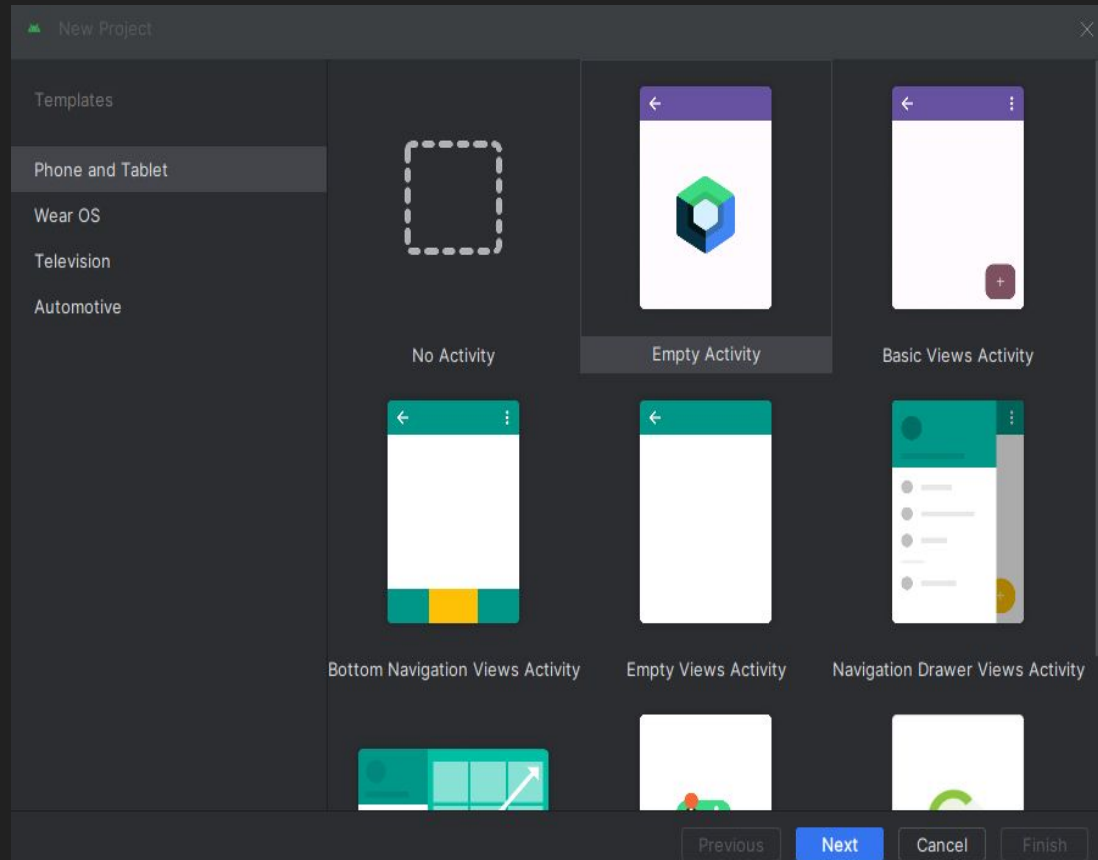
1. Abra o Android Studio.
2. Na tela inicial, clique em "New Project".



Uma "activity" é um componente fundamental de uma aplicação Android.

Basicamente, uma activity é uma janela onde você pode colocar seus elementos de interface (botões, textos, imagens, etc.)

A maioria das aplicações Android começa com uma activity principal que é lançada quando o usuário clica no ícone do aplicativo.



Dando nome ao projeto

Será solicitado a configurar alguns detalhes básicos do seu projeto, como o nome do aplicativo, o nome do pacote (package name), a localização do projeto, a linguagem de programação (Java ou Kotlin) e a versão mínima do SDK Android que você pretende suportar.

Convenção Reversa do Domínio (Reverse Domain Name): A prática mais comum é usar a convenção reversa do nome de domínio da empresa ou desenvolvedor. Isso ajuda a garantir a exclusividade e a identificação clara do proprietário do aplicativo. Por exemplo, se o seu domínio for **example.com**, o nome do pacote poderia ser **com.example.meuapp**.

Ao lado temos um exemplo.

No campo “Language”, podemos escolher Java ou Kotlin.

A linguagem Kotlin tem se tornado cada vez mais popular e é altamente recomendada para o desenvolvimento Android, mas Java ainda mantém uma presença significativa no mercado.

No Activity

Creates a new empty project

Name: My Application

Package name: com.example.myapplication

Save location: C:\Users\mathe\AndroidStudioProjects\MyApplication

Language: Java

Minimum SDK: API 24 ("Nougat"; Android 7.0)

i Your app will run on approximately 97,4% of devices.
[Help me choose](#)

Build configuration language *?* Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

O Android Studio então criará o projeto para você e abrirá a estrutura do projeto no editor. Aguarde alguns momentos enquanto o Android Studio configura o projeto e sincroniza as dependências.

Após esses passos, você terá um projeto Android básico criado no Android Studio e estará pronto para começar a desenvolver seu aplicativo. Você verá a estrutura do projeto no painel esquerdo do Android Studio, onde poderá acessar e editar os arquivos de código, recursos, layouts e muito mais.

4. Aprendendo os Fundamentos:

Assim que o ambiente de desenvolvimento estiver configurado, é hora de começar a aprender os fundamentos do desenvolvimento Android. Isso inclui entender a estrutura de um projeto Android.

Um projeto Android típico segue uma estrutura de diretórios e arquivos organizada de acordo com as diretrizes do Android Studio e do sistema Android.

No desenvolvimento Android, o XML é frequentemente usado para criar a interface do usuário (UI), enquanto os arquivos Java são usados para definir a lógica da aplicação.

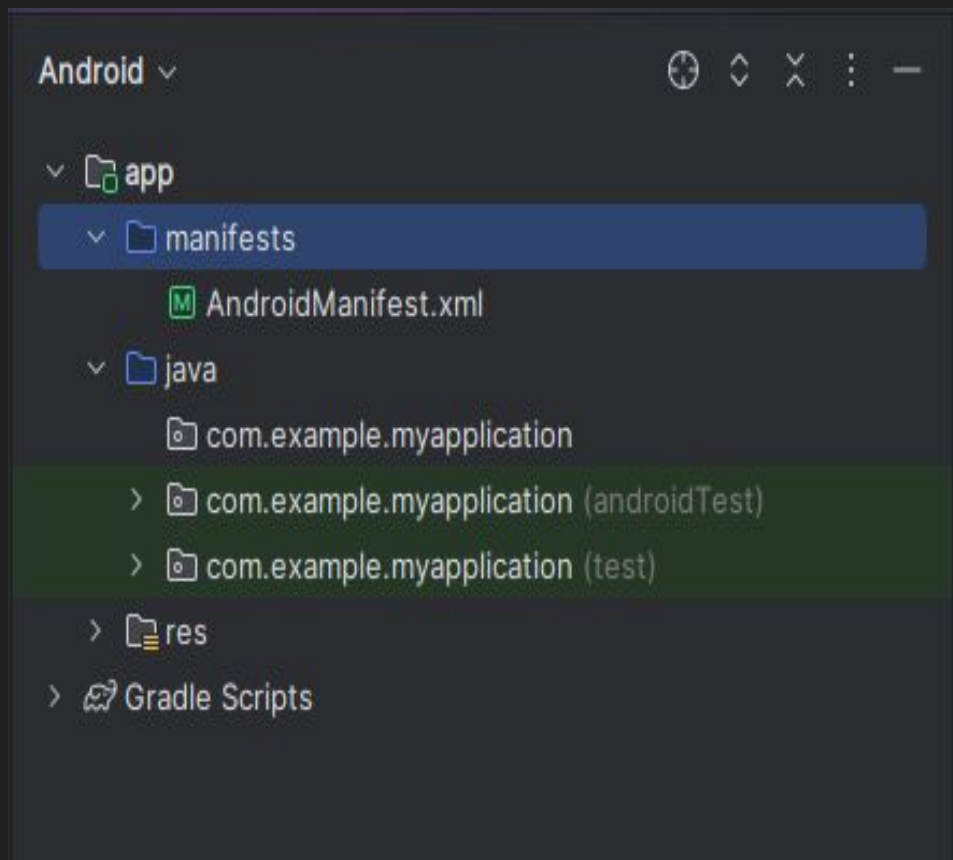
Essa separação de responsabilidades entre XML e Java é uma prática comum no desenvolvimento Android, seguindo o padrão de arquitetura Modelo-Visão-Controlador (MVC) ou outros padrões de arquitetura, como Modelo-Visão-Apresentação (MVP) ou Modelo-Visão-ViewModel (MVVM). Isso ajuda a manter o código mais organizado, modular e fácil de dar manutenção.

Alterando a visualização

Clique em project -> Android para filtrar os arquivos exibidos

A tela deve ser similar a essa:

O arquivo mais importante é o `AndroidManifest.xml`, onde você define configurações globais do aplicativo.



5. Estrutura de um projeto android

app: Este é o diretório principal do módulo do aplicativo. Ele contém todos os arquivos relacionados ao seu aplicativo, incluindo código-fonte, recursos e configurações.

manifests: Este diretório contém o arquivo AndroidManifest.xml, que descreve os detalhes do seu aplicativo, como permissões, componentes da aplicação (atividades, serviços, receptores de transmissão) e outras configurações importantes.

java: Este diretório contém o código-fonte Java do seu aplicativo. Normalmente, ele segue a estrutura de pacotes Java, onde os pacotes representam a estrutura do seu aplicativo.

res: Este diretório contém recursos (resources) usados pelo seu aplicativo, como layouts XML, strings, imagens, estilos, valores de dimensão, etc. É dividido em subdiretórios como:

drawable: Contém recursos de imagens.

layout: Contém arquivos XML que definem o layout das interfaces de usuário.

values: Contém arquivos XML para valores constantes, como cores, strings, dimensões, estilos, etc.

assets: Este diretório é usado para armazenar arquivos de dados que serão acessados por seu aplicativo usando a API AssetManager. Esses arquivos não são processados pelo Android SDK, mas são incluídos no APK final.

gradle: Este diretório contém os scripts Gradle utilizados para construir, testar e executar seu aplicativo. O arquivo mais importante aqui é o `build.gradle`, que configura as dependências, plugins e outras configurações do projeto.

build: Este diretório é criado pelo sistema de construção Gradle e contém todos os arquivos gerados durante o processo de construção do projeto, incluindo o APK final

6. Desenvolvendo Aplicações Simples:

Uma vez que você tenha uma compreensão sólida dos fundamentos, é hora de começar a desenvolver suas próprias aplicações Android

Dica: Lembre-se de focar na experiência do usuário, seguindo as diretrizes de design do Material Design da Google e testando regularmente seu aplicativo em diferentes dispositivos e resoluções de tela

<https://io.google/2022/products/material-design/intl/pt/>

7. Conceitos Básicos de XML

O **XML** (Extensible Markup Language) é amplamente utilizado no desenvolvimento Android para definir a estrutura e a aparência das interfaces de usuário. No XML, você pode definir layouts de tela, widgets (componentes de interface do usuário), cores, estilos e outros recursos visuais

Os arquivos XML estão localizados na pasta “res/layout” do projeto android.

Se esse diretório não estiver presente, você pode adicioná-lo manualmente clicando com o botão direito sobre a pasta res → New → Android Resource Directory.

No menu que aparecer, em “Resource type”, selecione “layout” e clique em OK

Nomenclatura de Arquivos de Layout

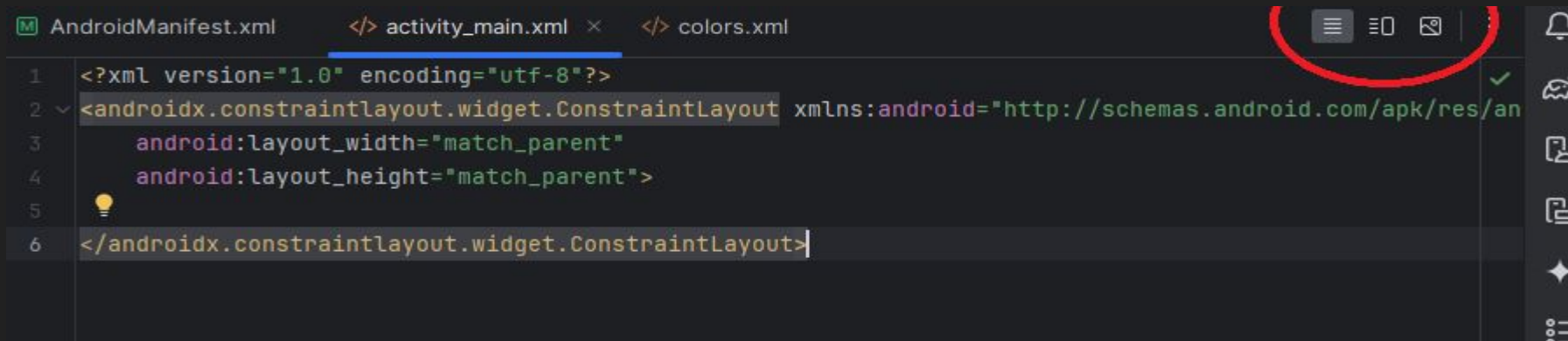
Use letras minúsculas e underscores (`_`) para separar palavras. Por exemplo, “activity_main.xml”, “fragment_detail.xml”.

Use nomes descritivos que refletem a função do layout.

Por exemplo, layout para activities podem ser nomeados como “activity_<name>.xml” e para fragmentos como “fragment_<name>.xml”.

Ao criar um arquivo XML, podemos optar pela visualização “Design” ou “Code”.

"Design" é uma interface gráfica para arrastar e soltar componentes de UI. Esta interface é útil para designers e desenvolvedores que preferem um método visual para criar layouts. No entanto, você também pode alternar para a visualização de "Código" para escrever o XML manualmente.



```
AndroidManifest.xml  </> activity_main.xml  </> colors.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/an
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5      ⚡
6  </androidx.constraintlayout.widget.ConstraintLayout>
```

8. Primeiro projeto “Hello World”


Exemplo de activity:

O símbolo “@” indica que estamos referenciando um **recurso interno do aplicativo**.

O @ é uma maneira de vincular elementos definidos em arquivos XML (como layouts, strings, cores etc.) ao código Java/Kotlin.

res/layout/activity_main.xml:

xml

 Copiar código

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!"
        android:textSize="18sp" />


    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />

</LinearLayout>
```

A partir do Android 12, é obrigatório definir explicitamente o atributo **android:exported** para atividades, serviços e broadcast receivers que possuem filtros de intent. Isso é necessário para garantir a segurança e evitar a exportação acidental de componentes.

Para corrigir o problema, precisamos adicionar o atributo **android:exported** no **AndroidManifest.xml**. No caso de uma atividade principal que deve ser iniciada pelo launcher, você deve definir **android:exported="true"**.

xml

 Copiar código

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exemplo">

    <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">

        <activity android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```



Para criarmos um arquivo de estilo, devemos criar o arquivo **style.xml** e nele teremos a definição do estilo **AppTheme**.

Botão direito sobre a pasta **values** → **New** → **Values Resource File**

res/values/styles.xml:

xml

 Copiar código

```
<resources>

  <!-- Base application theme -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here -->

    <item name="colorPrimary">@color/purple_500</item>
    <item name="colorPrimaryDark">@color/purple_700</item>
    <item name="colorAccent">@color/teal_200</item>

  </style>
</resources>
```

Interagindo com o layout em código java

Para interagir com os elementos definidos no XML no código Java, precisamos usar o método **findViewById** para obter referências aos elementos e definir os comportamentos desejados.

```
// Obtém referências aos elementos do layout  
final TextView textView = findViewById(R.id.textView);  
Button button = findViewById(R.id.button);
```

Classe R

A classe R é uma classe automaticamente gerada pelo Android Studio (ou outro ambiente de desenvolvimento) que armazena referências a todos os recursos do seu aplicativo. Esses recursos incluem layouts, strings, imagens, IDs de views, e outros.


- **R.layout**: Contém referências a todos os arquivos de layout XML no diretório `res/layout`.
- **R.id**: Contém IDs únicos para views, definidos nos arquivos de layout XML.
- **R.string**: Contém referências a todas as strings definidas no arquivo `res/values/strings.xml`.
- **R.drawable**: Contém referências a todas as imagens no diretório `res/drawable`.

Sempre que você adiciona, remove ou modifica qualquer recurso no seu projeto (como layouts, strings, imagens, etc.), a classe R é automaticamente regenerada pelo Android Build System. Isso garante que todos os recursos do projeto estejam sempre disponíveis através de referências de código.

O código ao lado e o arquivo manifest apresentado anteriormente são os arquivos utilizados no programa “Hello World” ao final desse eBook.

No sequência iremos falar um pouco sobre algumas tag XML mais comuns no desenvolvimento Android.

java

 Copiar código

```
package com.exemplo;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.widget.Button;
import android.widget.TextView;
import android.view.View;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); // Referencia o layout XML


        // Obtém referências aos elementos do layout
        final TextView textView = findViewById(R.id.textview);
        Button button = findViewById(R.id.button);

        // Define uma ação para o botão
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Altera o texto do TextView quando o botão é clicado
                textView.setText("Button Clicked!");
            }
        });
    }
}
```


Tag LinearLayout

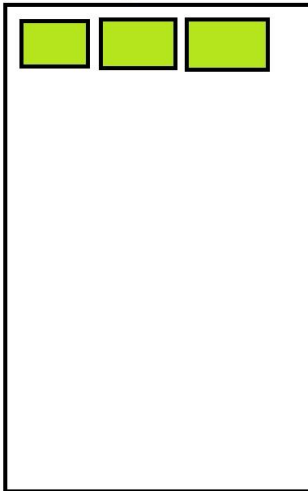
LinearLayout: Organiza as visualizações em uma única direção (vertical ou horizontal).

xml

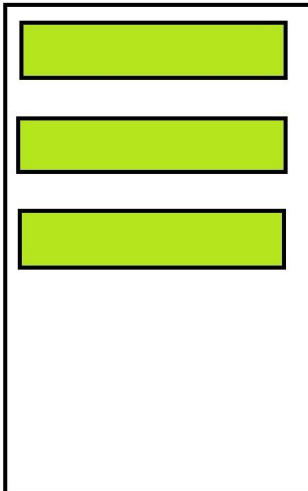
 Copiar código

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <!-- Conteúdo aqui -->
</LinearLayout>
```

Linear Layout Horizontal



Linear Layout Vertical



Tag RelativeLayout

A tag RelativeLayout permite posicionar visualizações com base em outras visualizações.

Alguns exemplos:

android:layout_alignParentTop

Se "true", faz com que a borda superior dessa visualização corresponda à borda superior do pai.

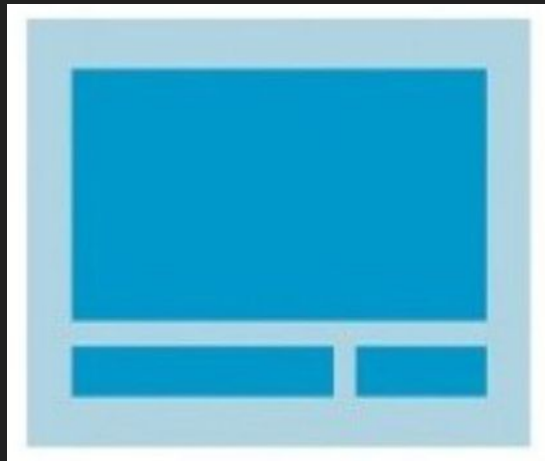
android:layout_centerVertical

Se "true", centraliza esse filho na vertical no pai.

xml

 Copiar código

```
<RelativeLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <!-- Conteúdo aqui -->  
</RelativeLayout>
```



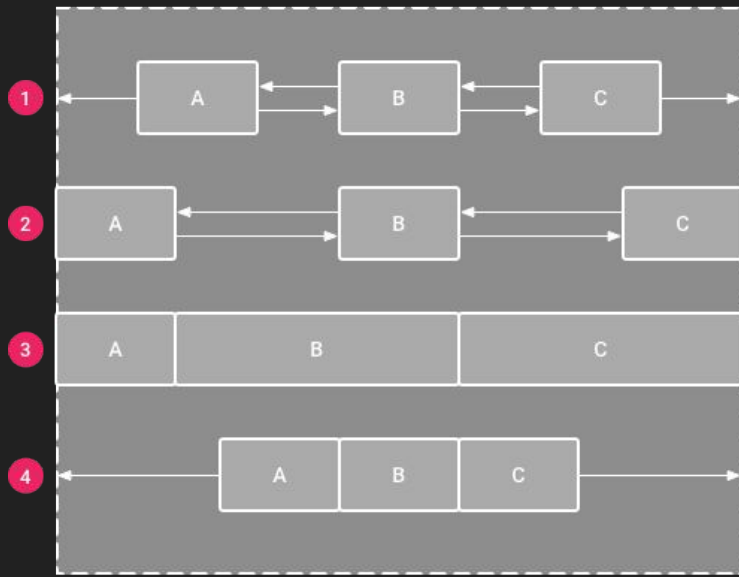
Tag ConstraintLayout

ConstraintLayout oferece flexibilidade para criar layouts complexos definindo relações de posicionamento entre os elementos e os widgets (como botões, caixas de texto etc).

xml

Copiar código


```
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <!-- Conteúdo aqui -->  
</androidx.constraintlayout.widget.ConstraintLayout>
```



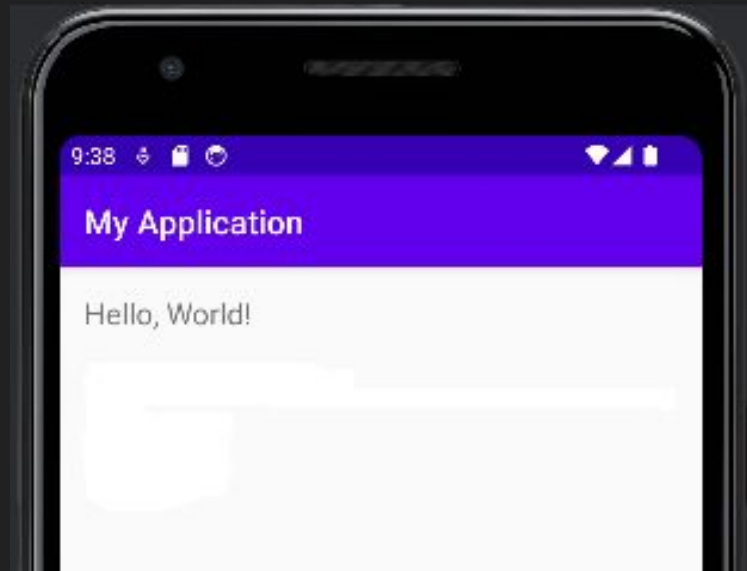
Tag TextView

A TextView permite exibir texto na tela.

xml

 Copiar código

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello, World!" />
```



Tag EditText

Permite ao usuário inserir texto.

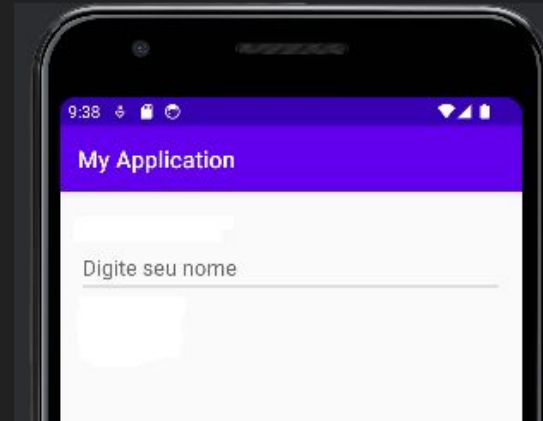
É útil para campos de entrada como formulários e pesquisas.

Podemos definir o tipo de entrada com **android:inputType** e o tamanho mínimo da área de toque com **android:minHeight**.

Com **android:hint** podemos deixar uma mensagem informando o que se espera de input.

xml

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:minHeight="48dp"  
    android:hint="Digite seu nome"  
    android:inputType="text" />
```



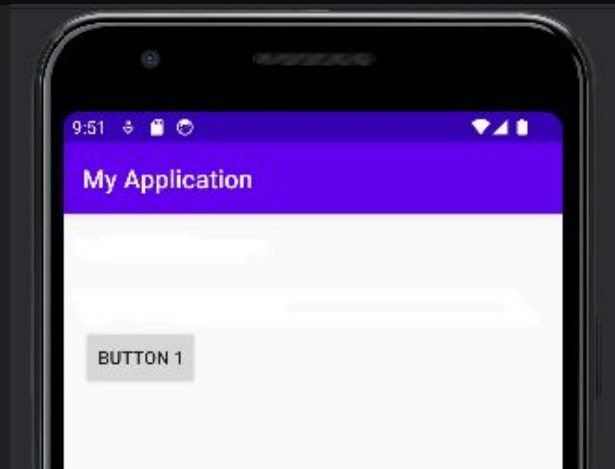
Tag Button

Essa tag representa um botão clicável.

Com **android:text** podemos definir o texto interno do botão.

Podemos referenciar botões usando IDs com **android:id="@+id/<id do botão>"** e **findViewById(R.id.<id do botão>)** no código Java.

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Button 1" />
```



Essas são apenas algumas das tags XML comuns usadas no desenvolvimento Android. Existem muitas outras, como `ImageView`, `RecyclerView`, `ListView`, entre outras, que são usadas para criar interfaces de usuário ricas e interativas.

Verifique a documentação oficial para maiores detalhes.



9. Ciclo de Vida da Atividade e Fragmento:

O ciclo de vida da atividade e do fragmento é um aspecto fundamental do desenvolvimento Android. As atividades representam as diferentes telas de um aplicativo e os fragmentos são componentes modulares que podem ser reutilizados em várias atividades.

O ciclo de vida de uma atividade ou fragmento inclui uma série de estados, como **onCreate()**, **onStart()**, **onResume()**, **onPause()**, **onStop()** e **onDestroy()**. Compreender o ciclo de vida é crucial para gerenciar corretamente os recursos, lidar com eventos de sistema e manter o estado da aplicação de forma consistente.

Os estados do ciclo de vida de uma atividade ou fragmento não ficam no arquivo XML. Eles são métodos que fazem parte da classe Java que representa a atividade ou o fragmento.

Esses métodos são parte integrante da API do Android e são chamados automaticamente pelo sistema operacional Android em resposta a eventos específicos que ocorrem durante a vida útil da atividade ou do fragmento.

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main); // Referencia o layout XML  
    }  
}
```

onCreate é chamado quando a atividade é criada pela primeira vez. Este método é onde você deve fazer toda a configuração inicial necessária para sua atividade, como definir o layout da interface do usuário, inicializar componentes, configurar listeners, e restaurar o estado anterior da atividade, se houver.

10. Manipulação de Eventos:

A manipulação de eventos é essencial para criar aplicativos interativos e responsivos. No desenvolvimento Android, eventos podem ser gerados por toques na tela, cliques em botões, movimentos de dedos, entre outros. Você pode definir ouvintes de eventos para responder a esses eventos e executar ações específicas, como atualizar a interface do usuário, navegar para outra tela ou processar dados de entrada do usuário.

No desenvolvimento Android, existem vários ouvintes de eventos comuns que são usados para responder a diferentes tipos de interações do usuário. Alguns dos ouvintes de eventos mais comuns incluem:

- **OnClickListener:** Usado para detectar cliques em elementos de interface do usuário, como botões, imagens, etc. Este ouvinte é usado para executar ações quando um elemento é clicado.
- **OnLongClickListener:** Semelhante ao OnClickListener, mas usado para detectar cliques longos (pressionando e segurando) em elementos de interface do usuário.
- **TouchListener:** Usado para detectar eventos de toque, como pressionar, deslizar e soltar, em elementos de interface do usuário. Este ouvinte é usado para implementar interações personalizadas com base nos eventos de toque.

- **OnCheckedChangeListener:** Usado em elementos de interface do usuário que têm estados alternáveis, como caixas de seleção (CheckBox) e botões de alternância (Switch). Este ouvinte é usado para detectar mudanças no estado desses elementos.
- **TextWatcher:** Usado para detectar alterações no texto de elementos de interface do usuário, como campos de texto (EditText). Este ouvinte é usado para realizar ações em tempo real à medida que o texto é digitado ou alterado.
- **SeekBar.OnSeekBarChangeListener:** Usado para detectar mudanças no valor de uma barra de progresso (SeekBar). Este ouvinte é usado para responder a alterações no valor da barra de progresso.

Esses são apenas alguns exemplos dos ouvintes de eventos mais comuns no desenvolvimento Android.

Cada um desses ouvintes é usado para lidar com tipos específicos de interações do usuário e permite que você crie aplicativos interativos e responsivos. Dependendo das necessidades do seu aplicativo, você pode implementar ou combinar esses ouvintes para criar a experiência do usuário desejada.



11. Rodando a aplicação

O Android Studio utiliza um emulador android para testar as aplicações.

Podemos conectar um aparelho via wifi ou usb ou utilizarmos um sistema virtualizado.



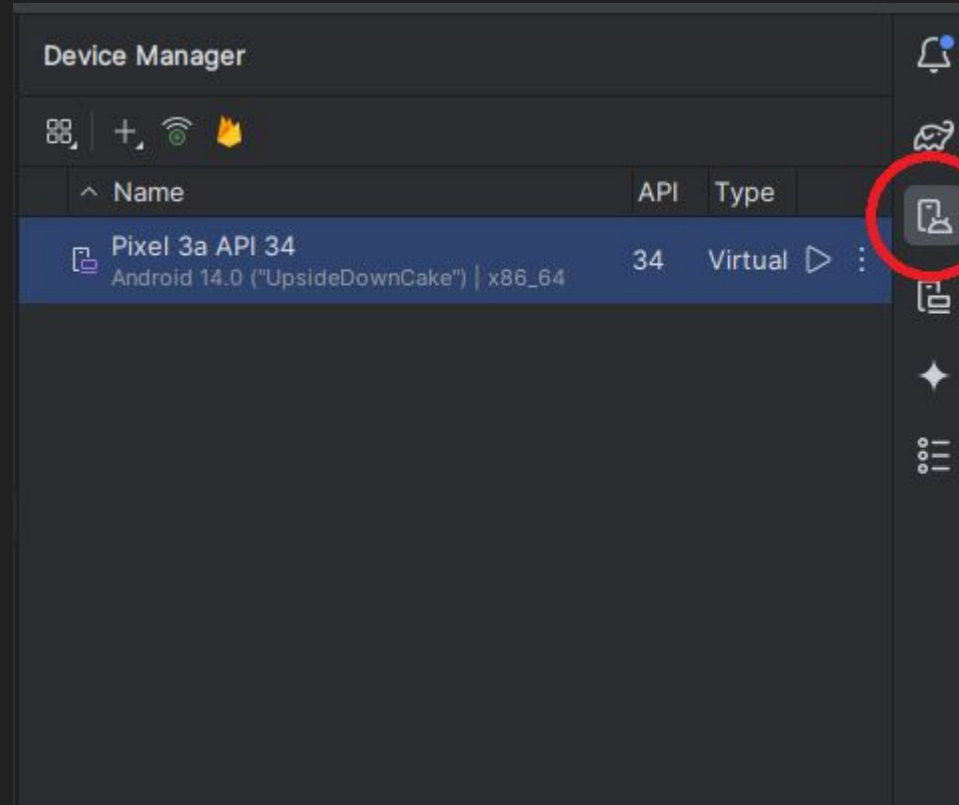
To mirror a physical device, connect it via USB cable or over WiFi, click **+** and select the device from the list. You may also select the **Activate mirroring when a new physical device is connected** option in the [Device Mirroring settings](#).

To launch a virtual device, click **+** and select the device from the list, or use the [Device Manager](#).

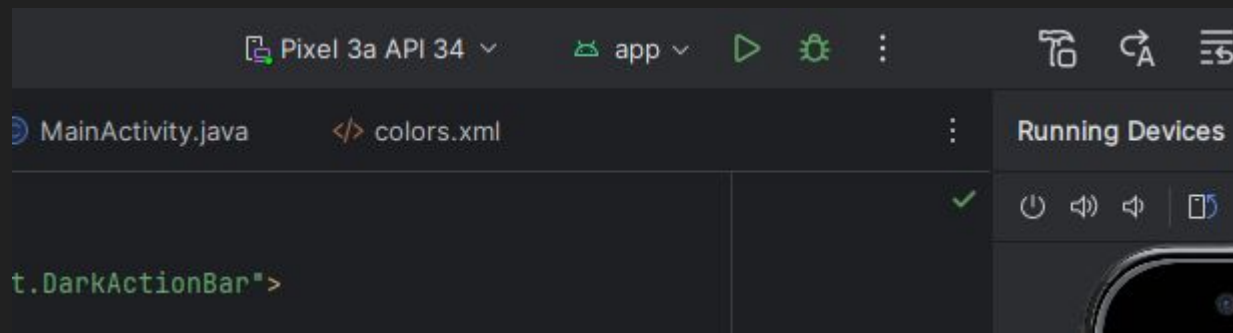
Sistema virtualizado

Em **device manager** podemos selecionar um sistema virtualizado para testar a aplicação.

O Android Studio vai inicializar o sistema e rodar a aplicação automaticamente.

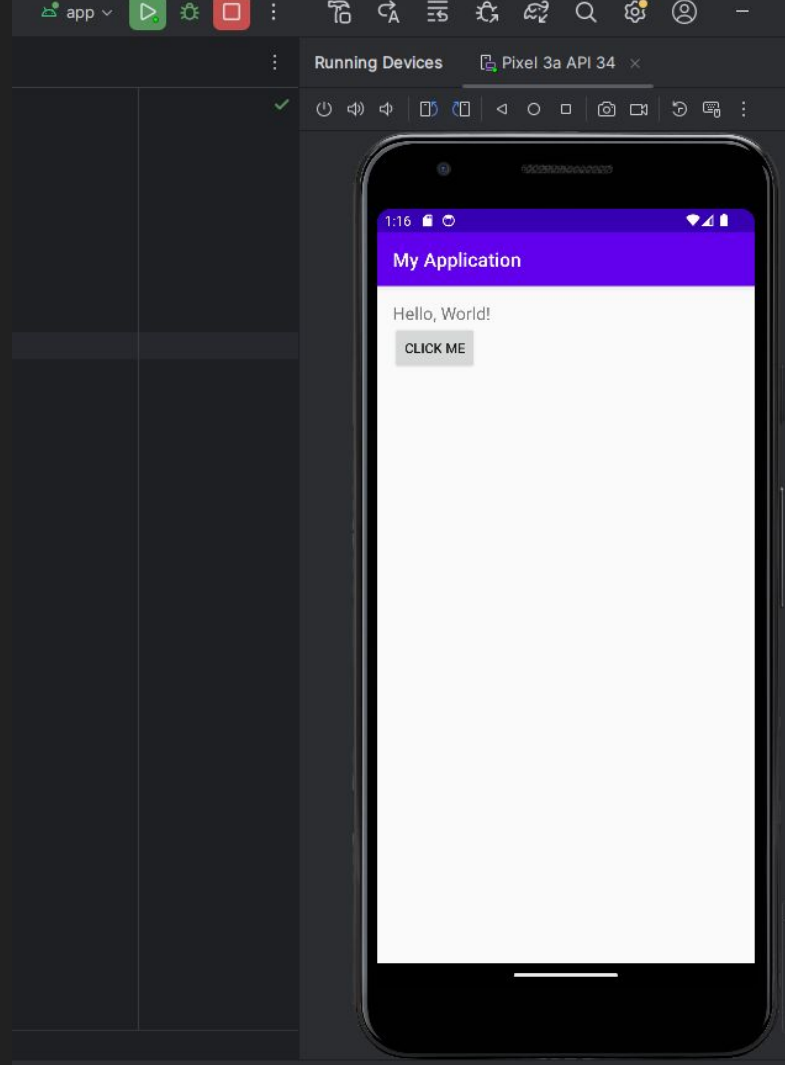


Com a máquina virtual inicializada, podemos rodar a aplicação e ver o efeito na tela simulada.



Nossa máquina virtual funciona como um smartphone, aceitando cliques do mouse.

Se clicarmos no botão, teremos a mensagem **Button Clicked!** no local do **Hello World!**



Esperamos que este ebook tenha sido útil e inspirador, capacitando você a criar aplicativos incríveis e desbloquear todo o potencial da plataforma Android.

Agora, é hora de seguir em frente e continuar explorando as possibilidades emocionantes que o desenvolvimento de aplicativos móveis oferece.

Boa sorte em suas futuras aventuras de desenvolvimento!

That's all folks!



Tecnologias usadas para criação desse ebook

A criação desse ebook foi incentivada no desafio do bootcamp **Bootcamp Nexa - Fundamentos de IA Generativa e Claude 3** da DIO, Natural ou Fake Natty? Como Vencer na Era das IAs Generativas, que tinha a proposta de usar IA para criar algum material.

Foram utilizadas as seguintes AI:

Chat GPT e Bing: criação dos textos contidos no ebook

Leonardo: criação das imagens

Outras ferramentas:

Photopea: edição de imagens

Referências:

<https://javatutorial.net/>

<https://developer.android.com/get-started/overview?hl=pt-br>