



Programação para dispositivos móveis

Danilo de Sousa Barbosa



Curso Técnico em Informática

Educação a Distância

2020



Programação para dispositivos móveis

Danilo de Sousa Barbosa

Curso Técnico em Informática

Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa

Educação a Distância

Recife

1.ed. | Mai. 2020



Licença Pública Creative Commons
Atribuição-NãoComercial-Compartilhável 4.0 Internacional

Professor Autor

Danilo de Sousa Barbosa

Revisão

Aline Chagas Rodrigues Marques
Rebecca Cristina Linhares de Carvalho

Coordenação de Curso

José Américo Teixeira de Barros

Coordenação Design Educacional

Deisiane Gomes Bazante

Design Educacional

Ana Cristina do Amaral e Silva Jaeger
Helisangela Maria Andrade Ferreira
Izabela Pereira Cavalcanti
Jailson Miranda
Roberto de Freitas Morais Sobrinho

Descrição de imagens

Sunnye Rose Carlos Gomes

Catálogo e Normalização

Hugo Cavalcanti (Crb-4 2129)

Diagramação

Jailson Miranda

Coordenação Executiva

George Bento Catunda
Renata Marques de Otero
Manoel Vanderley dos Santos Neto

Coordenação Geral

Maria de Araújo Medeiros Souza
Maria de Lourdes Cordeiro Marques

**Secretaria Executiva de
Educação Integral e Profissional**

**Escola Técnica Estadual
Professor Antônio Carlos Gomes da Costa**

Gerência de Educação a distância

Maio, 2020

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISDB

C871s

CDU – 658.56



Sumário

Introdução	6
1.Competência 01 Conhecer o Activity em detalhes	8
1.1 Criando um novo projeto com Activity.....	8
1.2 Conceito de Activity.....	10
1.3 Ciclo de vida de uma Activity.....	12
1.4 Navegabilidade entre Activity	17
1.5 Navegabilidade Simples de uma Activity para outra.....	19
1.6 Navegação entre telas com passagem de parâmetros.....	22
2.Competência 02 Manipular Action bar e temas.....	27
2.1 Criando um novo projeto com Action Bar	27
2.2 Conceito de Action Bar	31
2.3 Conceito de temas.....	32
2.4 Customização de Action Bar com botão.....	33
2.5 Modificando Action Bar com temas	36
3.Competência 03 Conhecer os conceitos de interface gráfica - gerenciadores de layout.....	39
3.1 Criando um novo projeto com a classe ViewGroup	39
3.2 Conceito da classe View	40
3.3 Conceito da classe ViewGroup	41
3.3.1 LinearLayout	44
3.3.2 ConstraintLayout	50
3.4 Dimensões dos componentes do layout	56
4.Competência 04 Utilizar recursos avançados de gerenciadores de layout.....	58
4.1 Recursos Avançados da classe ViewGroup.....	58
4.1.1 FrameLayout.....	58
4.1.2 TableLayout	62



4.1.3	RelativeLayout	65
5.	Competência 05 Conhecer os conceitos de interface gráfica – view.....	70
5.1	Criando um novo projeto com a classe Widget.....	70
5.2	Conceito da classe Widget.....	71
5.2.1	TextView	72
5.2.2	EditText.....	75
5.2.3	Imageview.....	82
6.	Competência 06 Utilizar recursos avançados de view	86
6.1	Avanços da Widget	86
6.1.1	Button	86
6.1.2	ImageButton	89
6.1.3	CheckBox	91
6.1.4	RadioButton	94
6.1.5	Spinner	97
7.	Competência 07 Planejar e executar o compartilhamento do APK	102
7.1	Versionamento da aplicação	102
7.2	Compatibilidade para instalação	104
7.3	Assinatura da aplicação com certificado digital	106
7.4	Assinatura da aplicação para publicá-la no Google Play	109
7.5	Publicando a sua aplicação no Google Play	111
	Conclusão	114
	Referências	115
	Minicurriculo do Professor	116



Introdução

Caro estudante, gostaria de dar a você as boas-vindas à disciplina Programação para Dispositivos Móveis. Os assuntos que serão tratados nessa disciplina fazem parte de uma área da Informática chamada de Desenvolvimento para Dispositivos Móveis e envolvem a utilização do sistema operacional *Android*.

Nesta disciplina você aprenderá sobre o sistema operacional *Android*, suas principais classes, características e comportamentos. Vale ressaltar que a plataforma *Android* é utilizada neste curso de Desenvolvimento de Sistemas por ser considerada bastante difundida, madura, robusta e acessível, tanto relacionada a área de desenvolvimento de aplicativos quanto utilizada pelo usuário em diversas plataformas, como *tablets*, *Smartphone*, TV, relógios, carros, entre outros.

Para que você entenda um pouco da importância do *Android*, ele simplesmente é considerado o sistema operacional líder mundial no segmento de dispositivos móveis! Em 2017, a Google I/O anunciou que existem mais de dois bilhões de dispositivos *Android* ativados no mundo. Apesar de ser considerado uma plataforma criada pela Google, o *Android* é considerado como um sistema livre e de código aberto, ou seja, permite que você baixe o código do sistema operacional *Android* e realize a alteração da maneira que você desejar.

Mas como desenvolvedores podem publicar aplicativos construídos no *Android*? você pode desenvolver os seus aplicativos para a plataforma *Android*, depois disso pode utilizar o *Google Play*, que é considerado um site fornecido pelo Google aos desenvolvedores para disponibilizar os seus aplicativos. Caso você crie o seu próprio aplicativo, pode pagar a taxa de 25 dólares e concordar com os termos de uso. Depois disso, o aplicativo pode ser publicado e instalado pelos usuários.

E caso você pretenda trabalhar na área de desenvolvimento de aplicativos, essa disciplina de Desenvolvimento para Dispositivos Móveis é uma excelente oportunidade para você! A empregabilidade de desenvolvedores de aplicativos está entre as mais altas do mundo, como exemplo nos EUA, os salários são aproximadamente U\$100/h (cem dólares a hora) e mesmo dentro do Brasil, as empresas pagam salários de R\$ 5.000 para bons desenvolvedores de aplicativos.

Além disso, você pode criar uma *startup* e desenvolver um aplicativo inovador! Existem vários casos de sucessos de *startups* que criaram aplicativos inovadores e conseguiram milhões de usuários e estão bem-sucedidos. O criador do *Wave*, popular aplicativo de mapas e rotas possui mais de 40 milhões de usuários e foi comprada pela Google por 1,3 bilhões de dólares. Outra empresa de



sucesso é a *Evernote*, com 100 milhões de usuários que geram 150 milhões de dólares por ano. Já pensou você criando uma startup e ela ser um caso de sucesso como esses que foram citados? Excelente ideia não é!

Mas para tudo isso ocorrer, você tem um grande desafio que é aprender uma nova linguagem para Desenvolvimento para Dispositivos Móveis. Na aprendizagem da linguagem *Android*, principalmente, exige dedicação, persistência, para estudar os conteúdos disponibilizados e fazer as atividades propostas, além de curiosidade para buscar mais informações. Assim, o aprendizado do ser humano será construído por repetição, ou seja, quanto mais você praticar, mais você vai assimilar o conteúdo.

Então, vamos começar com o conteúdo do nosso e-book. Você vai aprender sobre as classes de interface gráfica mais importantes e utilizadas no desenvolvimento de aplicativos em *Android*: na competência 01 será mostrado o conceito da classe *Activity*; na competência 02 será visto como manipular *Action bar* e temas; na competência 03 será abordado os conceitos de interface gráfica dos *layouts ConstraintLayout* e *LinearLayout*; competência 04 será apresentado alguns recursos avançados de gerenciadores de *layout* dos *layouts* como *FrameLayout*, *TableLayout* e *RelativeLayout*.

Como continuidade desse e-book vamos aprofundar sobre o conceito de interface gráfica utilizando classe *view*. Na competência 05 será visto alguns conceitos da classe *view* usando os elementos *TextView*, *EditText* e *ImageView*; na competência 06 será abordado os recursos avançados de *View* usando os elementos *ImageButton*, *Button*, *Checkbox*, *RadioButton* e *Spinner* e por fim, na competência 07 será mostrado como planejar e executar o compartilhamento de aplicação do *Android* no *Google Play*.

Na disciplina Introdução para Dispositivos Móveis você realizou o processo de instalação do *Android Studio*, a qual será a plataforma utilizada para darmos continuidade ao seu processo de aprendizagem. Como já foi visto, para fazer o download do *Android Studio*, basta acessar o site <https://developer.android.com/studio/index.html?hl=pt-br> e fazer o download da versão mais recente. Caso você ainda não tenha o *SDK Android*, deve baixar o *Android Studio* com o *SDK*. Caso já tenha, basta baixar apenas o *Android Studio*.

E então caro estudante, vamos começar a nos inspirar nesse mundo?



1.Competência 01 | Conhecer o Activity em detalhes

Caro estudante, nesta competência você vai aprender com criar um novo projeto com *Activity*, a descrição do conceito da classe *Activity* e o funcionamento do ciclo de vida de uma *Activity*.

Além disso, será mostrado como realizar a navegabilidade entre *Activity*, e será visto como realizar navegabilidade simples de uma *Activity* para outra e a navegação entre telas com passagem de parâmetros.

Vamos começar!

1.1 Criando um novo projeto com Activity

Na disciplina Introdução a programação para dispositivos móveis você aprendeu a criar um novo projeto no *Android Studio*. Você lembra? Na construção desse projeto criado por você foram gerados os arquivos *AndroidManifest.xml*, *MainActivity.java* e o *activity_main.xml*. A declaração da sintaxe da nova *Activity* ocorre no arquivo *AndroidManifest.xml* com a execução da *Activity* sendo realizada no arquivo *MainActivity.java* através da seguinte sintaxe `<activity android:name=".MainActivity">`. O código do *AndroidManifest.xml* do novo projeto com *Activity* pode ser visualizado na Figura 1.1 abaixo:

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

    <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"/>
    </application>
</manifest>
```

Figura 1.1 – Arquivo *AndroidManifest.xml* do novo projeto com *Activity*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* possuindo as principais declarações da aplicação e a declaração da *activity* no novo projeto.



No arquivo *MainActivity.java* é executado o método *setContentView(view)* para especificar a *View* que contém os elementos de layout com acesso ao arquivo de gerenciamento de layout *activity_main.xml*. O código do arquivo *MainActivity.java* pode ser visualizado na Figura 1.2 abaixo:

```
package com.example.eadpernambucopdm;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Figura 1.2 – Arquivo MainActivity.java

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo o acesso ao arquivo de layout com o nome *activity_main*.

O arquivo *activity_main.xml* é responsável por configurar o gerenciador de layout, ou classe *ViewGroup*, além de configurar os elementos da classe *Widgets*. O código do *activity_main.xml* do novo projeto com *Activity* pode ser visualizado na Figura 1.3 abaixo:

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

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 1.3 – Arquivo activity_main.xml

Fonte: Android Studio 3.5.2



Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* possuindo a declaração do layout *ConstraintLayout* com o componente de texto chamado *TextView*.

Até aqui você aprendeu como criar um novo projeto com declaração da classe *Activity*. Nas seções a seguir, você vai aprender sobre os conceitos da classe *Activity*, ciclo de vida de uma *Activity* e a navegabilidade entre *Activity*.

1.2 Conceito de Activity

A classe *Activity* é responsável por controlar os eventos da tela dos aplicativos e definir qual componente visual será responsável pela interface gráfica do usuário. Mas o que é uma classe *Activity*? A classe *Activity* deve herdar da classe Java *android.app.Activity* ou alguma subclasse Java desta, essa classe *Activity* é representada por alguma tela da aplicação, assim como é responsável por tratar todos os eventos gerados nessa tela, por exemplo, os cliques de botão, seleção de *checkbox*, cliques no *menu* e mudanças de tela.

Essa classe *Activity* possui implementado o método *onCreate(bundle)*, responsável por realizar a inicialização dos elementos necessários para exibir uma tela do aplicativo. Esse método *onCreate(bundle)* é executado no método *setContentView(view)*, responsável por configurar o *layout XML (Extensible Markup Language)* que contém os elementos da tela, tais como as suas imagens e botões. Por fim, cada *Activity* criada deve ser declarada no arquivo *AndroidManifest.xml*, como será mostrado nos exemplos abaixo. O código do *AndroidManifest.xml* com a declaração *MainActivity* pode ser visualizado na Figura 1.4 abaixo:

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

    <application >
        <activity
            android:name=".MainActivity" />
    </application>

</manifest>
```

Figura 1.4 – Arquivo *AndroidManifest.xml* no Android Studio com a declaração *MainActivity*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* possuindo a declaração da *Activity* com o nome *MainActivity*.



Nessa configuração, esse “.” (ponto) antes do nome da sua *Activity* indica que o pacote da classe Java é relativo ao pacote do projeto. Caso esse pacote estivesse em um pacote diferente, o nome desse pacote deveria ser especificado, por exemplo, *br.com.cursoandroid.hello.activity*. O código do *AndroidManifest.xml* com a declaração *activity.MinhaClasseActivity* pode ser visualizado na Figura 1.5 abaixo:

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

    <application >
        <activity
            android:name=".activity.MinhaClasseActivity" />

    </application>
</manifest>
```

Figura 1.5 – Arquivo *AndroidManifest.xml* no Android Studio com a declaração *activity.MinhaClasseActivity*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* possuindo a declaração da *Activity* com o nome *MinhaClasseActivity* na pasta *Activity*.

Se achar necessário, pode definir o nome completo da classe Java, pois não é considerado uma declaração errada. O código do *AndroidManifest.xml* com a declaração da classe Java completa *br.com.cursoandroid.hello.activity.MinhaClasseActivity* pode ser visualizado na Figura 1.6 abaixo:

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

    <application>

        <activity android:name="br.com.cursoandroid.hello.activity.MinhaClasseActivity" />

    </application>

</manifest>
```

Figura 1.6 – Arquivo *AndroidManifest.xml* no Android Studio com a declaração *br.com.cursoandroid.hello.activity.MinhaClasseActivity*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* possuindo a declaração da *Activity* com o nome da classe *br.com.cursoandroid.hello.activity.MinhaClasseActivity*.

Na prática podemos relacionar a palavra “*Activity*” à palavra “tela”. Diante desse raciocínio, caso seja necessário criar uma nova tela do seu projeto, você deverá criar uma nova *Activity*, e essa classe deverá ser declarada no arquivo *AndroidManifest.xml*. Esse arquivo é



responsável por controlar os eventos e elementos da tela, que é especificada no método `setContentView(view)`, que já vimos anteriormente.

Por fim, lembre-se de configurar corretamente o nome da classe *Activity*, pois o compilador é *case sensitive*, ou seja, ele diferencia as letras maiúsculas das minúsculas, dessa forma para ele “MinhaActivity” é diferente de “minhaActivity”. Para ter certeza de que o nome e o pacote foram especificados corretamente, segure a tecla *ctrl* (control) do seu teclado e passe o mouse em cima do nome da sua *Activity*, se o texto ficar selecionado, você pode clicar para abrir o código-fonte da classe. Se o editor abrir, essa execução está correta.

1.3 Ciclo de vida de uma Activity

Na execução do ciclo de vida de uma *Activity* é importante entender em qual estado ele se encontra e como você pode controlar o comportamento da sua aplicação. Tenha em mente o seguinte conceito: quem cuida da execução do fluxo do ciclo de vida é o sistema operacional *Android*, mas você pode customizar esses estados, por exemplo, criando outros estados como: executando, temporariamente interrompida em segundo plano e completamente destruída.

Imagine que você tenha desenvolvido um jogo espetacular para *Android* e enquanto o usuário está jogando ele receba uma ligação, então o que vai acontecer com o jogo? Bom, o sistema operacional vai interromper o jogo temporariamente e colocá-lo em segundo plano para que o usuário possa atender a ligação. Depois que ligação for encerrada a grande pergunta é: o que deve acontecer com o jogo? as informações do jogo serão salvas? Bom, isso vai depender das especificações realizada por você. O *Android* fornece toda a estrutura necessária para isso, basta você entender o ciclo de vida da *Activity*.

Mas o que acontecerá quando for executado diversos aplicativos no *Android*, como o *browser*, a tela inicial (*Home*) e a tela para discar um número? Bem, cada execução possui uma *Activity* inicial sendo inserido no topo de uma pilha, chamada de pilha de atividades. Assim, nessa pilha de atividade, sempre que uma nova *Activity* é inserida no topo da pilha, a *Activity* anterior que estava em execução fica logo abaixo da nova, ou seja, a *Activity* que está no topo da pilha é a *Activity* em execução no momento, as demais podem estar executando no segundo plano, estar no estado pausado ou totalmente paradas.



Agora você vai aprender como utilizar na prática o ciclo de vida de uma *Activity*. Vamos começa? Alguns métodos da classe *Activity* pode ser utilizado para controlar o estado da tela. Dentre eles, existem o método *onCreate()*, que é responsável por inicializar a *Activity* e iniciar o seu ciclo de vida. Além desse método, os outros métodos importantes do ciclo de vida são *onStart()*, *onRestart()*, *onPause()*, *onStop()* e *onDestroy()*. A Figura 1.7, abaixo, mostra o fluxo do ciclo de vida de uma *Activity* e seus possíveis estados.

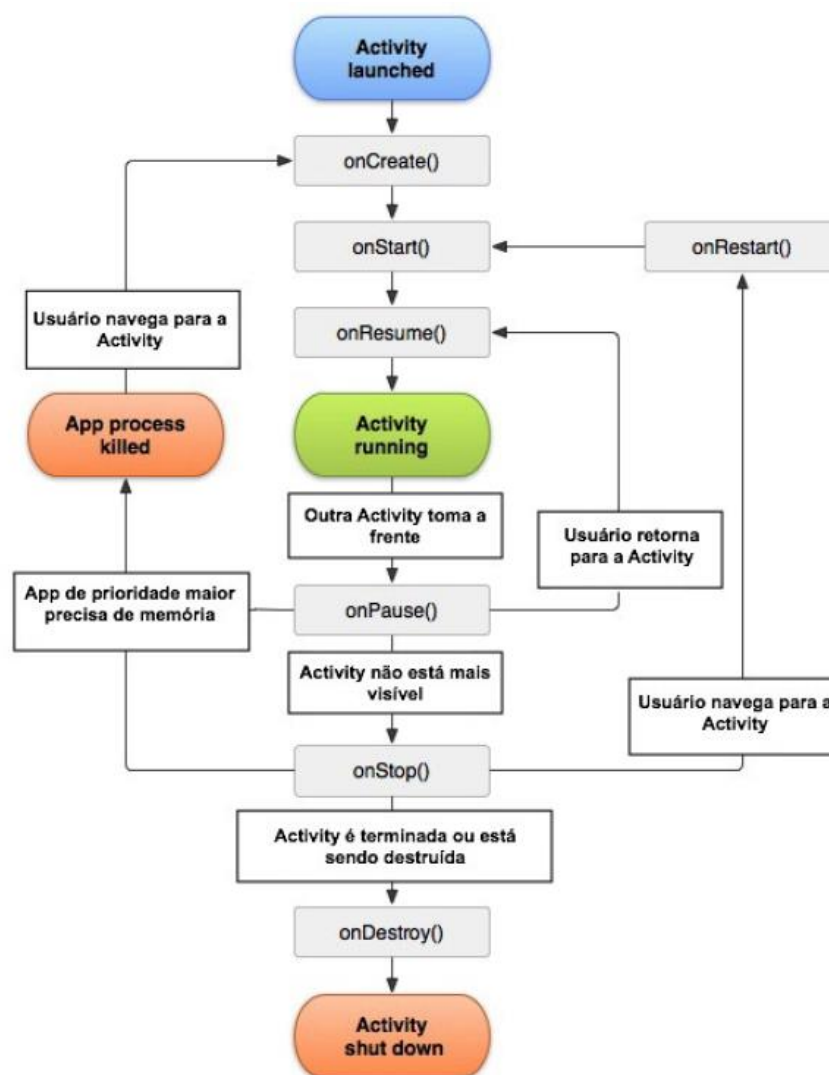


Figura 1.7 – Ciclo de vida de uma activity

Fonte: o autor

Descrição: a imagem apresenta todas as fases do ciclo de vida de uma aplicação *Android*. Quando uma *activity* é iniciada o seu ciclo de vida tem início e o método *onCreate* é executado, em seguida é executado o *onStart* e o *onResume*. Caso outra *activity* seja iniciada, a atual é pausada, então, é executado o método *onPause*. Se a *activity* anterior voltar a executar, ela chamará novamente o *onResume*, caso contrário, é chamado o método *onStop* e em seguida pode ser chamado o *onDestroy*, que encerra a *activity* ou pode ser chamado o *onRestart*, se a *Activity* voltar a executar ou caso ela tenha ido para segundo plano no *onStop* e encerrada pelo SO. No caso dela voltar a ser exibida, o método *onCreate* é chamado novamente.



Preste atenção nesse momento, pois agora você vai aprender sobre cada um dos métodos que se encontram no ciclo de vida de uma *Activity* na Tabela 1.1, abaixo. É importante que você leia a descrição do método e acompanhe o fluxograma da Figura 1.7, acima.

Método	Descrição
<i>onCreate(bundle)</i>	Esse método é considerado obrigatório no ciclo de vida da <i>Activity</i> , por isso deve ser iniciado apenas uma vez. O objetivo desse método é realizar a inicialização necessária para executar um aplicativo. Neste método, deve-se criar uma <i>view</i> e chamar o método <i>setContentView(view)</i> para configurar os elementos que serão exibidos no layout da tela dos usuários. Após a execução do <i>onCreate(bundle)</i> é chamado o método <i>onStart</i> .
<i>onStart()</i>	Esse método é chamado quando uma <i>Activity</i> está criada, assim como a sua <i>view</i> , ou seja, após o método <i>onCreate(bundle)</i> ocorre uma execução quando a <i>activity</i> está visível ao usuário. Como mostrado no fluxo acima, esse método também é chamado depois dos métodos <i>onRestart</i> ou <i>onCreate</i> , dependendo assim do estado atual o aplicativo.
<i>onRestart()</i>	Esse método é chamado quando uma <i>Activity</i> é parada temporariamente e esteja sendo iniciada novamente, então uma aplicação em Android chama o método <i>onRestart</i> que logo após executa o método <i>onStart</i> .
<i>onResume()</i>	Esse método é chamado quando a <i>Activity</i> principal já está executando e está interagindo com o usuário. Além disso, quando esse método está sendo executado, o usuário está vendo a tela. Por isso, esse método é frequentemente utilizado para consultar o banco de dados e atualizar as informações da tela do aplicativo.
<i>onPause()</i>	Esse método é chamado quando a tela da <i>Activity</i> fecha. Isso pode acontecer quando o usuário pressiona o botão voltar para sair do aplicativo. Outra maneira ocorre quando você recebe uma ligação telefônica. Nesse momento, o método é chamado para salvar o estado do aplicativo, para posteriormente, quando essa <i>activity</i> voltar a executar, tudo possa ser recuperado no método <i>onResume</i> .
<i>onStop()</i>	Esse método é chamado quando a <i>Activity</i> está sendo encerrada e não está mais visível para o usuário. Depois de parada, a <i>Activity</i> pode ser reiniciada. Caso isso ocorra, o método <i>onRestart</i> realiza a reinicialização da <i>activity</i> . Caso essa <i>Activity</i> fique muito tempo parada em segundo plano e o sistema operacional Android precise liberar a memória, o método <i>onDestroy</i> pode ser automaticamente chamado para remover completamente <i>Activity</i> .



<i>onDestroy()</i>	Esse método encerra a execução de uma <i>Activity</i> . Com isso, esse método pode ser chamado automaticamente pelo sistema operacional para liberar recursos alocados pela <i>Activity</i> como valores de atributos e demais recursos. Após o método <i>onDestroy</i> ser executado, a <i>Activity</i> será removida completamente e o seu processo no sistema operacional será completamente encerrado.
--------------------	--

Tabela 1.1 – Ciclo de vida de uma aplicação Android

Fonte: o autor

Descrição da imagem: a descrição dos métodos do ciclo de vida de uma *Activity* e seus respectivos conceitos.

Depois de mostrar o ciclo de vida de uma *Activity* e descrever o funcionamento dos seus métodos. Você deve ter ficado curioso para entender a execução desse ciclo de vida em código! Bem, a seguir será mostrado um exemplo do código de uma *activity* que implementa os métodos: *onCreate(bundle)*, *onStart()*, *onRestart()*, *onResume()*, *onPause()*, *onStop()* e *onDestroy()* do ciclo de vida que mostramos anteriormente.

Recomendo seguir esse exemplo e implementar todos os métodos. Dessa maneira, você pode fazer diversos testes. Como exemplo, realizar ligação para o seu celular, encerrar uma aplicação e acompanhar esses métodos chamados, através das mensagens de log que serão exibidas na janela do *LogCat*. Para testar isso, você pode criar um projeto Android e executá-lo usando o emulador do Android ou diretamente no seu próprio celular. O código do *MainActivity* com a descrição do Logcat pode ser visualizado na Figura 1.8 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Log.i(getLocalClassName(), "onCreate()");

        TextView olaMundo = new TextView(this);
        olaMundo.setText("Olá mundo! :)");
        setContentView(olaMundo);
        setContentView(R.layout.activity_main);
    }
    @Override
    protected void onStart() {
        super.onStart();
    }
}
```



```
Log.i(getLocalClassName(), "onStart()");
}
@Override
protected void onRestart() {
    super.onRestart();
    Log.i(getLocalClassName(), "onRestart()");
}
@Override
protected void onResume() {
    super.onResume();
    Log.i(getLocalClassName(), "onResume()");
}
@Override
protected void onPause() {
    super.onPause();
    Log.i(getLocalClassName(), "onPause()");
}
@Override
protected void onStop() {
    super.onStop();
    Log.i(getLocalClassName(), "onStop()");
}
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.i(getLocalClassName(), "onDestroy()");
}
}
```

Figura 1.8 – estrutura do logcat no arquivo MainActivity.java do Android Studio.

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a declaração dos métodos *onCreate*, *onStart*, *onRestart*, *onResume*, *onPause*, *onStop* e *onDestroy*.

Na execução do código mostrada na Figura 1.8, acima, o *Android Studio* mostra no *LogCat* a descrição dos métodos *onCreate()*, *onStart()* e *onResume()*. A execução do *LogCat* do *Android Studio* pode ser visualizada na Figura 1.9 abaixo:

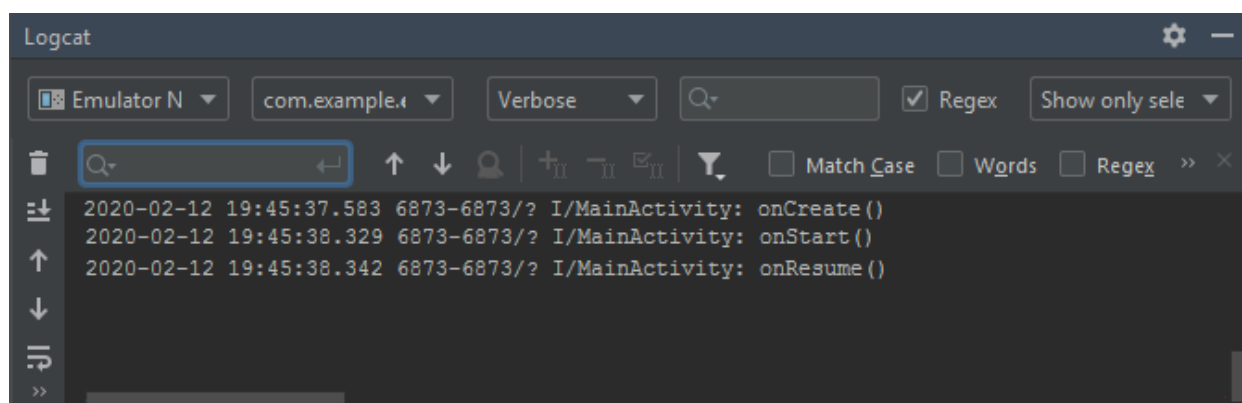


Figura 1.9 – descrição de logs no LogCat do Android Studio.



Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com a tela de *logCat* com logs mostrando a descrição do *Logcat* com os métodos *onCreate()*, *onStart()* e *onResume()* que foram chamados a partir da exibição de uma *Activity*

Entender o ciclo de vida de uma *Activity* é fundamental para você dominar o desenvolvimento de aplicações *Android*. O conhecimento desses métodos é considerado importante principalmente quando você está jogando no seu celular. A seguir será mostrado dois exemplos do ciclo de vida de uma *Activity* no momento da execução do jogo.

Imagine que você está jogando e ocorre uma interrupção por receber uma ligação, ou a bateria do seu celular acaba quando você estiver jogando. Nessas situações, o ideal seria que o jogo fosse salvo ao ser interrompido pela falta de bateria ou pausado no momento da ligação, para assim o usuário não ser prejudicado com interrupções externas.

1.4 Navegabilidade entre Activity

Até esse momento abordamos conceitos do que significa uma *Activity*, para que serve uma *Activity* e o funcionamento do ciclo de vida da *Activity*. Agora chegou o momento de aprender como criar uma *Activity*, assim como a navegabilidade de uma para outra. Essas *activities* são representadas por telas. Diante disso, os aplicativos utilizam um conjunto de telas diferentes, então para cada tela desse aplicativo será criado uma nova *Activity*.

A navegabilidade entre *activities* pode ocorrer utilizando os métodos *startActivity(intent)* ou *startActivityForResult(intent, código)*. Ambos os métodos utilizam uma instância da classe *Intent*. Essa classe *Intent* significa intenção, nesse caso a intenção é mudar de *Activity*, ou seja, a navegabilidade da transição da tela de origem para uma tela de destino com envio de informação entre elas.

Mas como ocorre a execução desses dois métodos? Bem, o método *startActivity* é responsável pela mudança de uma tela de origem para uma tela de destino, porém a navegabilidade entre *activities* não vincula relação com a *Activity* anterior. A Figura 1.10, abaixo, mostra a navegabilidade do método *startActivity(intent)*. Com isso, a informação da *Activity* de origem é enviada para *Activity* de destino sem retorno de informação.

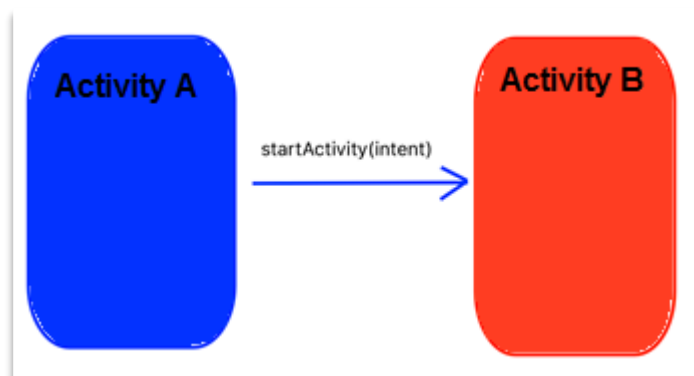


Figura 1.10 – navegabilidade do método `startActivity(intent)`.

Fonte: o autor

Descrição da imagem: A imagem apresenta uma caixa azul representando a tela azul com a *Activity A* e uma caixa vermelha representando a tela vermelha com a *Activity B*. Entre as telas azul e vermelha possui uma seta azul com a navegabilidade do método `startActivity(intent)` que realiza a transição entre a tela azul (*Activity A*) e a tela vermelha (*Activity B*).

Já o método `startActivityForResult` é responsável pela mudança de uma tela de origem para uma tela de destino, porém a navegabilidade entre *activities* retorna informações da tela de destino para a tela de origem. A Figura 1.11, abaixo, mostra a navegabilidade do método `startActivityForResult`. Com isso, a informação da *Activity* de origem (*Activity A*) é enviada para *Activity* de destino (*Activity B*). Em seguida, a *Activity* de destino (*Activity B*) faz o tratamento do parâmetro recebido e retorna à informação para *Activity* de origem.

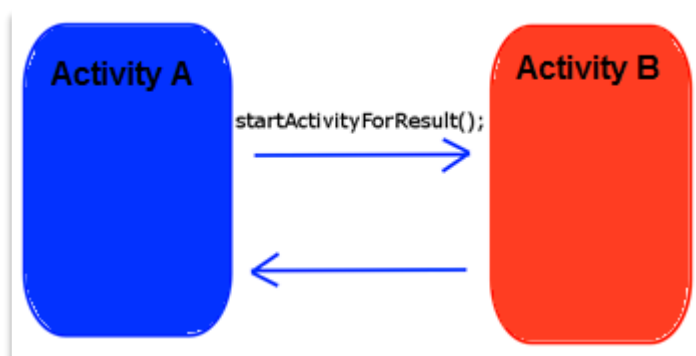


Figura 1.11 – navegabilidade do método `startActivityForResult()`.

Fonte: o autor

Descrição da imagem: A imagem apresenta uma caixa azul representando a tela azul com a *Activity A* e uma caixa vermelha representando a tela vermelha com a *Activity B*. Entre as telas azul e vermelha possui uma seta com a navegabilidade do método `startActivityForResult()` que realiza a transição entre a tela azul e a tela vermelha e outra seta azul com o retorno da tela vermelha para azul.



1.5 Navegabilidade Simples de uma Activity para outra

A navegação simples de uma *Activity* para outra necessita de duas telas. Mas como isso ocorre a nível de código? O código da Figura 1.12, abaixo, mostra como a partir de um clique é possível abrir uma nova tela. Para isso ocorrer é necessário criar uma segunda *Activity* como mostrado na Figura 1.13, abaixo. Nesse código será necessário criar duas *activities* para navegar de uma tela para outra. A primeira *Activity* será chamada de *MainActivity* e a segunda *Activity* de *SegundaTela*.

```
package com.example.eadpernambucopdm;

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

public class MainActivity extends AppCompatActivity implements
View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Button olaMundo = new Button(this);
        olaMundo.setText("EAD Pernambuco! :)");
        olaMundo.setOnClickListener(this);
        setContentView(olaMundo);

    }

    public void onClick(View v) {
        Intent tela2 = new Intent(this, SegundaTela.class);
        startActivity(tela2); }

}
```

Figura 1.12 – Arquivo MainActivity.java com navegabilidade simples de uma Activity

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* com navegabilidade simples de uma *Activity* para outra, por meio da classe *MainActivity.java* que realiza a navegabilidade entre a tela de origem representada por *MainActivity* para a tela de destino representada por *SegundaTela* usando a classe *intent*.



```
package com.example.eadpernambucopdm;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class SegundaTela extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tela2 = new TextView(this);
        tela2.setText("Bem vindo a SegundaTela.");
        setContentView(tela2); }
}
```

Figura 1.13 – Arquivo SegundaTela.java com navegabilidade simples de uma Activity

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *SegundaTela.java* com navegabilidade simples de uma *Activity* para outra, por meio da classe *SegundaTela* que mostra o resultado da tela de destino representada por *SegundaTela* e mostrada na mensagem ("Bem vindo a SegundaTela").

Depois de criar as *activities* dessas classes é necessário criar a declaração dessas *activities* no *AndroidManifest.xml*. Mas ocorre algum erro caso não declare essas *activities* no *AndroidManifest.xml*? Sim, a criação da nova *Activity(tela)* deve ser declarada no *AndroidManifest.xml* Caso você não realize essa declaração no *AndroidManifest*, a seguinte mensagem de erro será exibida no seu *LogCat*. A descrição do erro no Logcat pode ser visualizada na Figura 1.14 abaixo:

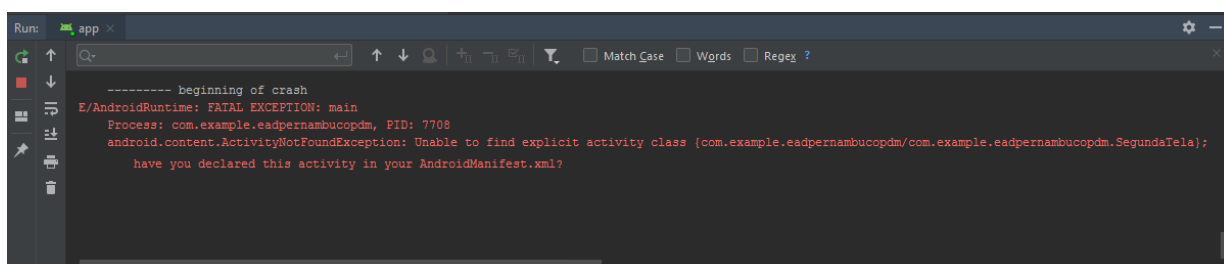


Figura 1.14 – Descrição do erro no Logcat

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com a tela de *logCat* com log de erro mostrando a seguinte mensagem “você declarou essa *activity* no seu *AndroidManifest.xml*?”.

Perceba nessa mensagem de exceção fatal a explicação do erro em inglês, traduzindo: “você declarou essa *Activity* no seu *AndroidManifest.xml*?”. Para resolver esse problema adicione as seguintes linhas no seu *AndroidManifest.xml* `<activity android:name=".SegundaTela"/>` e



`<activity android:name=".MainActivity">`. O código do *MainActivity* com a declaração da *Activity* pode ser visualizado na Figura 1.15 abaixo:

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

    <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">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Figura 1.15 - Arquivo *AndroidManifest.xml* com a declaração da *Activity*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* com a declaração da *Activity* com o nome *MainActivity*.

Depois da atualização dessa duas *activities* no *AndroidManifest.xml* o erro anterior foi resolvido e ocorreu a navegabilidade da primeira tela (Figura 1.16) para a segunda tela (Figura 1.17).

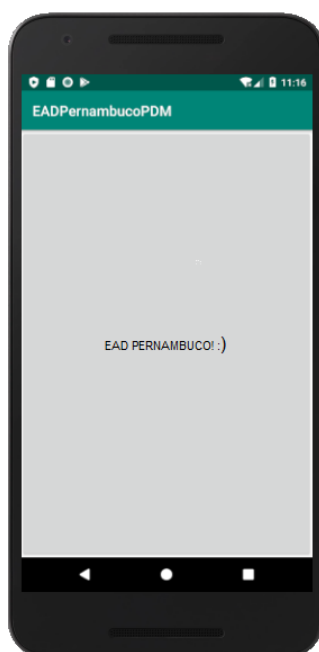




Figura 1.16 – tela da classe MainActivity.java no Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta uma tela do celular da classe MainActivity.java mostrado na parte superior da tela a mensagem EADPernambucoPDM, na barra inferior da tela três botões do celular com símbolos da seta, bola e quadrado e na parte do meio da tela a mensagem "EAD Pernambuco! :)".

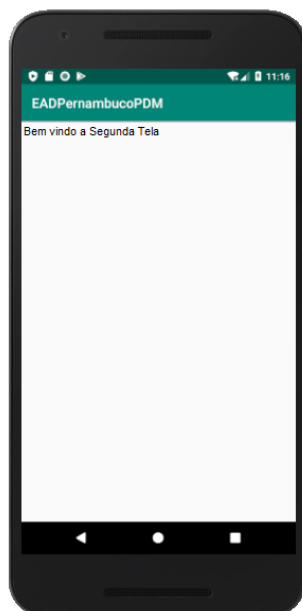


Figura 1.17 - tela da classe SegundaTela.java no Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta uma tela do celular da classe SegundaTela.java mostrado na parte superior da tela a mensagem EADPernambucoPDM, na barra inferior da tela três botões do celular com símbolos da seta, bola e quadrado e na parte do meio da tela a mensagem ("Bem vindo a SegundaTela.").

1.6 Navegação entre telas com passagem de parâmetros

Agora você vai aprender sobre como funciona a navegação entre telas com passagem de parâmetros. Mas como ocorre o envio de uma ou mais informações da primeira tela para a segunda tela? Nessa navegação é utilizada a classe *Bundle* que é semelhante a uma *HashTable* (estrutura de dados do tipo chave com valor). Essa classe utiliza o método *putString(chave, valor)* para configurar as informações com formato de texto para a segunda tela. O código do *MainActivity* com a navegabilidade entre telas com passagem de parâmetro pode ser visualizado na Figura 1.18 abaixo:

```
package com.example.eadpernambucopdm;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
```



```
import android.view.View;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener {

    public void onClick(View v) {
        Intent tela2 = new Intent(this, SegundaTela.class);
        Bundle informacoes = new Bundle();

        informacoes.putString("disciplina", "Programação de desenvolvimento de
sistemas");
        informacoes.putString("professor", "Danilo ");
        tela2.putExtras(informacoes);
        startActivity(tela2);
    }
}
```

Figura 1.18 - Arquivo MainActivity.java com navegabilidade entre telas com passagem de parâmetros

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* com navegabilidade entre telas com passagem de parâmetros, por meio do método *putString* que configurar o formato de texto para a segunda tela.

Nesse exemplo anterior utilizamos o método *putString*, porém a classe *Bundle* possui outras opções de tipo primitivo como *putInt*, *putChar*, *putBoolean*, entre outros. Até esse momento você aprendeu como enviar as informações entre as telas, em seguida você vai aprender como conseguir informação da segunda tela. O código da SegundaTela com a navegabilidade entre telas com passagem de parâmetro pode ser visualizado na Figura 1.19 abaixo:

```
package com.example.eadpernambucopdm;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class SegundaTela extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tela2 = new TextView(this);
        Intent it = getIntent(); String disciplina = "", professor = "";
        if(it != null){
            Bundle informacoes = it.getExtras();
            if(informacoes != null){
                disciplina = informacoes.getString("disciplina");
                professor = informacoes.getString("professor");
            }
        }
    }
}
```




```
tela2.setText("Bem vindo " + disciplina + " " + professor);  
setContentView(tela2); }  
}
```

Figura 1.19 – Arquivo SegundaTela.java com navegabilidade entre telas com passagem de parâmetros

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo SegundaTela.java com navegabilidade entre telas com passagem de parâmetros, por meio do parâmetro do *startActivity* e do método *onCreate* da segunda tela retornamos essa mesma intent da tela2 e extraímos as informações relacionadas a “disciplina” e “professor”.

No método *onClick* da primeira tela (*MainActivity*) executamos a *Intent* (chamado tela2) com as informações que desejamos transferir para segunda tela. Como parâmetro do *startActivity* e no método *onCreate* da segunda tela (*SegundaTela*) retornamos essa mesma *Intent*(tela2) e extraímos as informações relacionadas as chaves “disciplina” e “professor”.

Em linguagem de programação existem diversas maneiras de realizar a mesma funcionalidade, isso depende do seu conhecimento e da sua flexibilidade sobre a lógica utilizada. Diante disso, você pode realizar a passagem de informação entre as telas da seguinte maneira. O código com a navegabilidade entre telas com passagem de parâmetro utilizando *putExtra* pode ser visualizado na Figura 1.20 abaixo:

```
package com.example.eadpernambucopdm;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
  
import android.view.View;  
  
public class MainActivity extends AppCompatActivity implements  
View.OnClickListener {  
  
    public void onClick(View v) {  
        Intent tela2 = new Intent(this, SegundaTela.class);  
        tela2.putExtra("disciplina", "Programação de desenvolvimento de  
sistemas");  
        tela2.putExtra("professor", "Danilo ");  
        startActivity(tela2);  
    }  
}
```

Figura 1.20 - Arquivo MainActivity.java com navegabilidade entre telas com passagem de parâmetro utilizando *putExtra*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* com navegabilidade entre telas com passagem de parâmetros, por meio do parâmetro *putExtra* que extrair informações relacionadas a “disciplina” e “professor”.



No exemplo da Figura 1.20, acima, não utilizamos o *Bundle* e colocamos os parâmetros diretamente no *Intent*. Depois da modificação dessas classes de *activities* ocorreu a navegabilidade da primeira tela para a segunda tela. A tela com a execução da segunda tela pode ser visualizada na Figura 1.21 abaixo:

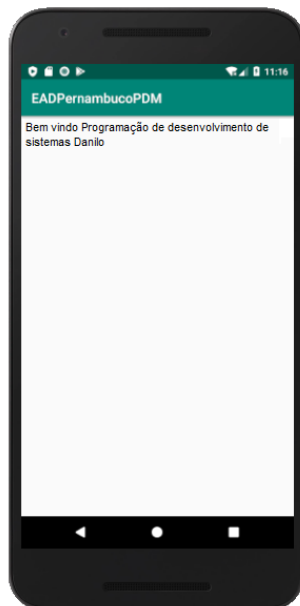


Figura 1.21 - tela da classe SegundaTela.java no Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta uma tela do celular da classe SegundaTela.java mostrado na parte superior da tela a mensagem EADPernambucoPDM, na barra inferior da tela três botões do celular com símbolos da seta, bola e quadrado e na parte do meio da tela a mensagem “Bem vindo Programação de desenvolvimento de sistemas Danilo”.

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- Criar um novo projeto com *Activity*
- Conceito de *Activity*
- Ciclo de vida de uma *Activity*
- Navegabilidade entre *Activity*
- Navegabilidade Simples de uma *Activity* para outra
- Navegação entre telas com passagem de parâmetros



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 1?
Acesse o AVA e assista a videoaula da competência 1.



Agora, acesse o AVA e responda as questões da competência 1



Ficou com alguma dúvida na competência 1? Acesse o **Fórum - “Competência 1”** para saná-las com os professores e discutir com seus colegas sobre os assuntos estudados.

Nosso próximo passo é seguir para competência 2. Na competência 02 você vai aprender sobre *Action Bar*, a barra de ação dos aplicativos e sobre o tipo de estilo temas. Vamos juntos?



2.Competência 02 | Manipular Action bar e temas

Caro estudante, nesta competência você vai aprender com criar um novo projeto com *Action Bar* e os conceitos de *Action Bar* e temas. Além disso, será mostrado sobre a customização de *Action Bar*. Por fim, será visto a Modificando Action Bar com temas.

Vamos continuar!

2.1 Criando um novo projeto com Action Bar

Na disciplina Introdução a programação para dispositivos móveis você aprendeu a criar um novo projeto no *Android Studio*. Você lembra? No início da construção desse novo projeto você selecionou a opção *Basic Activity* com a funcionalidade do *Action Bar*. Diante disso, esse novo projeto incluir de maneira automática uma biblioteca de suporte para utilizar o *Action Bar*. A criação do novo projeto com a opção do *Action Bar* pode ser visualizado na Figura 2.1 abaixo:

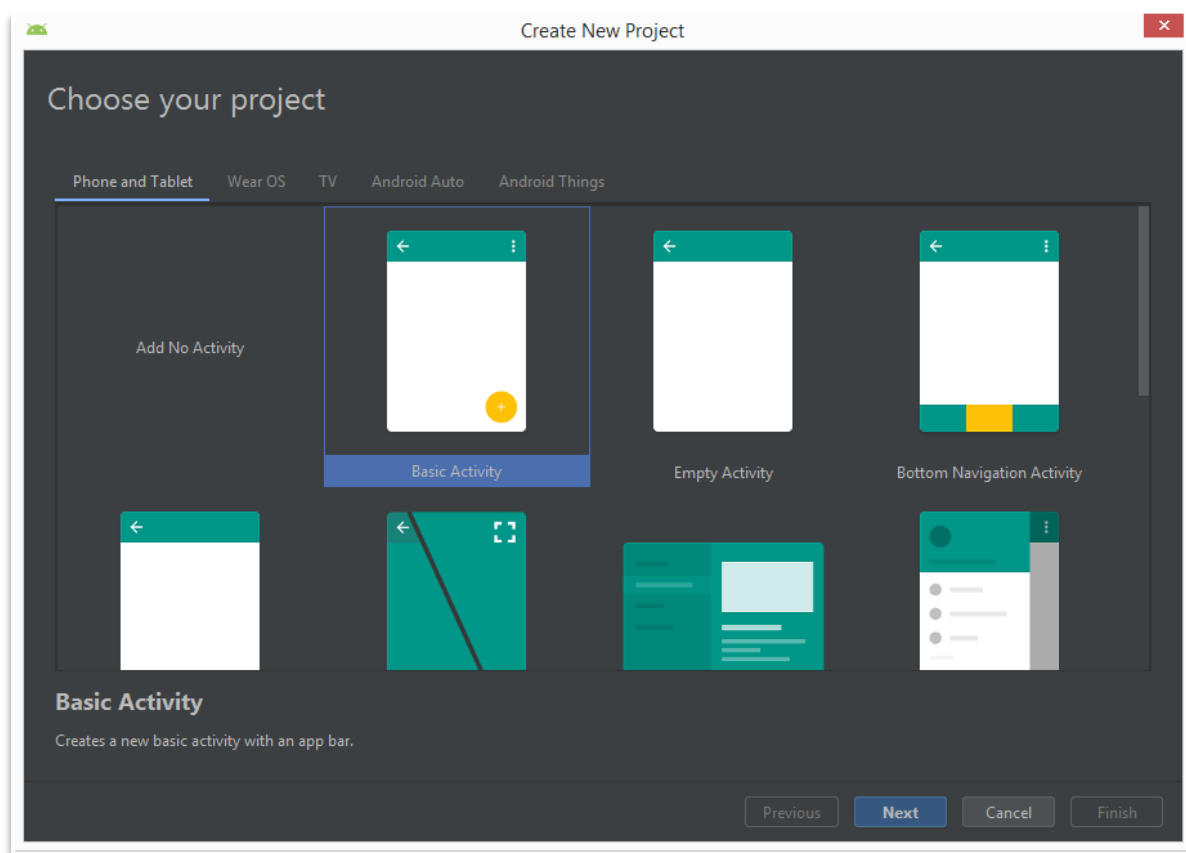




Figura 2.1 - Criando uma nova Activity com Action Bar

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta a tela do Android para criar um novo projeto com *Action Bar* possuindo a seleção de uma tela com *Activity* e a opção *next* para prosseguir na criação da nova *Activity*.

Para prosseguir a criação do projeto clique em *Next*. Em seguida, na próxima tela será necessário inserir a nova configuração do projeto. As opções modificadas nesse novo projeto são o nome do projeto, a linguagem Java e a API com a versão 26 e o *Android Oreo*. A configuração do novo projeto com *Action Bar* pode ser visualizado na Figura 2.2 abaixo:

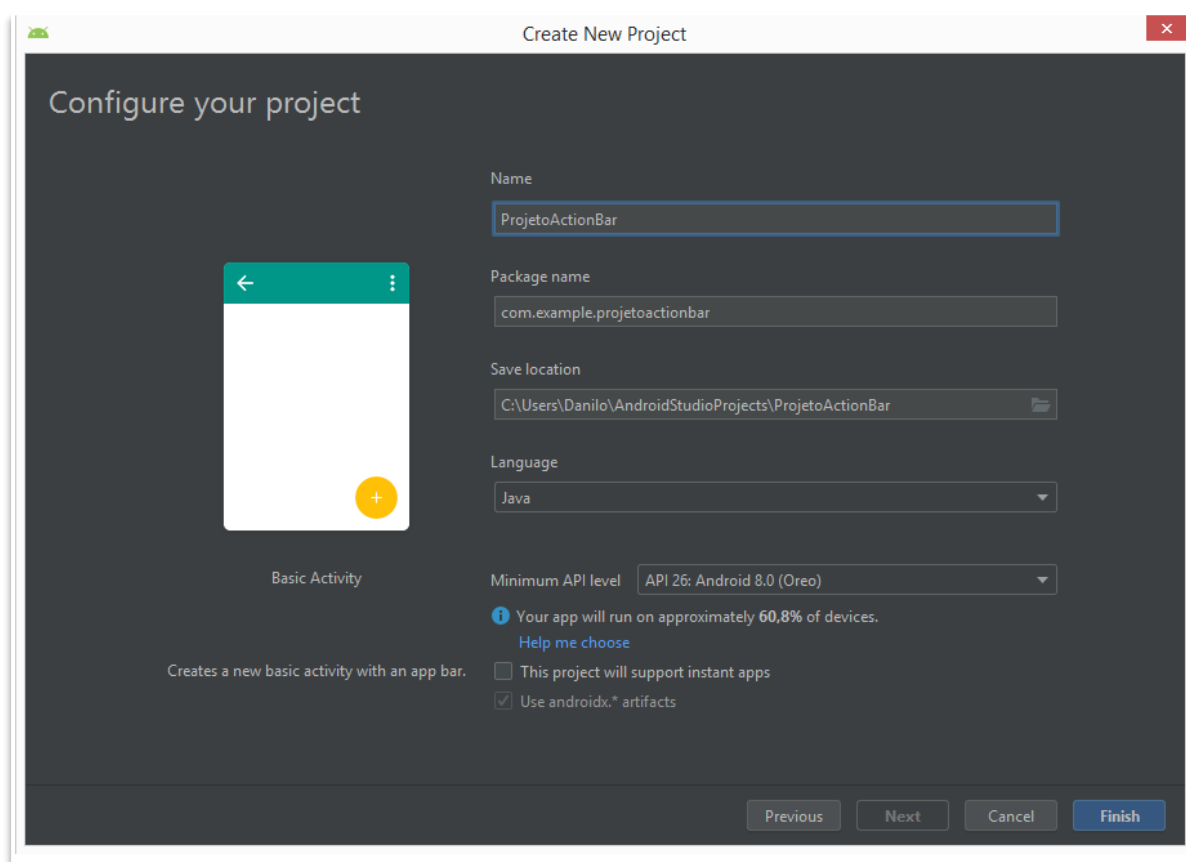


Figura 2.2 – Configurar um novo projeto com Action Bar

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta a tela do *Android* para configurar um novo projeto com *ActionBar* possuindo o preenchimento do *name* com o nome do projeto, *Package name* com a descrição do pacote do projeto, *save location* com a localização da pasta do projeto, *language* com a descrição da linguagem e a *Minimum API Level* com a descrição da mínima versão utilizada nesse projeto. Com o preenchimento desses campos deve acessar o botão *finish* para finalizar o projeto.

Em seguida, o projeto no *Android Studio* inicia a sua execução. Nesse projeto, as classes utilizadas para executar o *Action Bar* são *AndroidManifest.xml*, *styles.xml* e *colors.xml*. Agora você vai



aprender sobre esses arquivos gerados pelo *Android Studio* para realizar a configuração inicial de um *Action Bar*. Vamos iniciar explicando o arquivo *AndroidManifest.xml* com a definição do tema do *Action Bar* definido no atributo *android:theme* da tag *application*. O código do arquivo *AndroidManifest.xml* com a declaração do tema pode ser visualizado na Figura 2.3 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.eadpernambucopdm">
    <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">
        <!--Código comentado em XML: outros temas padrões
        android:theme="@android:style/Theme.Black" Padrão cor preta
        android:theme="@style/Theme.AppCompat.Light" padrão cor branca
        -->
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Figura 2.3 – Arquivo *AndroidManifest.xml* com *Action Bar*

Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a declaração da *Activity* com a tag *name* com o nome da *activity*, *label* com rótulo com a *string* da aplicação e o *theme* com o tema da aplicação.

O código do *AndroidManifest.xml* da Figura 2.3 mostra o atributo *android:theme* com a declaração no arquivo *@style/AppTheme*. No arquivo do *AndroidManifest.xml*, o arquivo *@style/AppTheme* é criado em *Theme.AppCompat* do arquivo *styles.xml*. No arquivo *styles.xml*, o *AppTheme* é considerado o nome do estilo e o tema *Theme.AppCompat.Light.DarkActionBar* é representado pela cor branca do layout. O Android Studio possui outros temas padrões como o *android:theme="@android:style/Theme.Black"* representado pela cor preta do layout e o



`android:theme="@style/Theme.AppCompat.Light"` representando pela cor branca do layout. Por fim, O `AppTheme.NoActionBar` é considerado o nome do estilo do *Action Bar*. O código do arquivo `styles.xml` com a configuração do estilo do *Action Bar* pode ser visualizado na Figura 2.4 abaixo:

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>
  <style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
  </style>
</resources>
```

Figura 2.4 - Arquivo `styles.xml` com a declaração do *Action Bar*
Fonte: Android Studio 3.5.2

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo `styles.xml` possuindo a declaração do estilo com a cor branca, com a *tag style* com nome possuindo o nome do estilo `AppTheme` e o *parent* possuindo o acesso ao arquivo do tema `Theme.AppCompat.Light`, em seguida a declaração do estilo do *Action Bar* com o nome do estilo `AppTheme.NoActionBar` relacionado ao *Action Bar*.

Outra maneira de utilizar o arquivo `styles.xml` é declarando a personalização da cor com formato de número hexadecimal. Essa declaração possui o nome o tema `AppTheme` com fundo branco e os itens `colorPrimary` com descrição hexadecimal na cor verde clara, `colorPrimaryDark` com descrição hexadecimal na cor verde escura e o `colorAccent` com descrição hexadecimal na cor rosa. O código do arquivo `styles.xml` com o tema personalizado por cor pode ser visualizado na Figura 2.5 abaixo:

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">#008577</item>
    <item name="colorPrimaryDark">#00575B</item>
    <item name="colorAccent">#D81B60</item>
  </style>
</resources>
```

Figura 2.5 - Arquivo `styles.xml` com o tema personalizado por cor
Fonte: Android Studio 3.5.2



Descrição da imagem: a imagem apresenta um fundo branco com o código do arquivo *styles.xml* possuindo a descrição do tema personalizado por cor possuindo a *tag resources* e as três *tags color* com a descrição da cor primária, cor primária escura e cor acentuada, assim como, a descrição hexadecimal da cor.

Até aqui você aprendeu como criar um novo projeto com *Action Bar*. Nas seções a seguir, você vai aprender sobre os conceitos de *Action Bar* e temas, a customização de *Action Bar* com botão e a modificando do *Action Bar* com temas.

2.2 Conceito de Action Bar

Action Bar, ou simplesmente barra de ação, é considerado um elemento de interface gráfica utilizado em aplicativos. Mas qual a função do *Action Bar*? A sua função é identificar em qual região do aplicativo você está, além de fornecer ações contextualizadas executada por você, assim como, o processo de navegação para outras telas utilizando *Activity*.

Além disso, a *Action Bar* é capaz de realizar o ajuste adequado de qualquer configuração de tela no arquivo *AndroidManifest.xml* do *Android Studio*. Os principais elementos de um *Action Bar* pode ser visualizado na Figura 2.6 abaixo:



Figura 2.6 – Elementos do Action Bar

Fonte: o autor

Descrição da imagem: a imagem apresenta uma barra de navegação com os quatro elementos do *Action bar*. A barra de navegação possui no número um o ícone do *Android* (No inglês *App icon*), no número dois possui a descrição em texto do *Action bar* que realiza a navegação com controle de telas (No inglês *View control*), no número três possui os ícones de ações (No inglês *Action buttons*) com uma lupa de pesquisa em seguida uma câmera de vídeo e no número quatro possui um submenu com as ações (No inglês *Action overflow*) possuindo três pontos com os ícones que não couberam na barra de ações.

Nesse momento você vai aprender sobre esses quatro elementos do *Action Bar* na Tabela 2.1, abaixo. Essa tabela está relacionada aos quatro elementos com números mostrado na Figura 2.6, acima.



Elemento	Descrição
(1) Ícone (App Icon)	O ícone é considerado o elemento de identificação do seu aplicativo. Além disso, esse elemento com o ícone do Android realiza o processo de navegação, por meio da seta para esquerda que realiza a navegação para cima no processamento entre telas.
(2) Navegação (View control)	A navegação é considerada o elemento de navegação entre outras telas (views) do aplicativo, permitindo assim uma rápida navegação entre telas.
(3) Botões de ações (Action buttons)	Os botões de ações são considerados os elementos com opções mais importantes e disponíveis para o usuário. Alguns exemplos de ações são acesso a lupa do celular, acesso a fotos e vídeos.
(4) Submenu de acesso (Action overflow)	O submenu de acesso é considerado um dos elementos com ações menos usadas ou ações que não são visualizadas nessa barra de ações.

Tabela 2.1 – Os elementos do Action Bar

Fonte: o autor

Descrição da imagem: a descrição dos quatro elementos do *Action Bar* ícone, navegação, botões de ações e submenu de acesso, assim como as respectivas descrições desses elementos.

2.3 Conceito de temas

O tema é considerado um tipo de estilo que pode ser configurado em toda aplicação, atividade ou hierarquia de exibição do *Android*. Mas o que é estilo? É considerado uma coleção de atributos que especificam a aparência de um único elemento visual do *Android*.

Com o estilo você pode especificar atributos como cor e tamanho da fonte e cor do plano de fundo. Quando você aplica o seu estilo com tema, todas as visualizações no aplicativo ou atividade aplicam cada atributo de estilo configurado nesse tema. Os temas também podem aplicar estilos que não são visualizados, como por exemplo, as barras de status e o plano de fundo da janela.



O tema é declarado no arquivo *styles.xml* da pasta *values* do *Android Studio*. Esse tema deve possuir a *tag* `<style>` responsável por identificar o tipo de estilo e a *tag* `<item>` responsável pela configuração de cada atributo do estilo definido.

Na *tag* `<item>` possui o item *name* responsável por especifica um atributo XML em seu *layout*. O código do *styles.xml* com exemplo de estilo com tema pode ser visualizado na Figura 2.7 abaixo:

```
<resources>

  <!-- Base application theme. -->
  <style name=" " parent=" ">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>

  </style>
</resources>
```

Figura 2.7 – Arquivo *styles.xml* do *Android Studio*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *styles.xml* possuindo o item da cor com o nome da cor primária e o acesso dessa cor representado por `@color/colorPrimary`.

2.4 Customização de Action Bar com botão

A customização do *Action Bar* com botão de ação (*Action Buttons*) necessita de configuração em sua visualização. Você pode definir se o botão sempre ficará visível, ou se ficará disponível no *menu* de *submenu* com ações (*Action overflow*). Para isso, podemos utilizar o atributo *showAsAction* responsável por definir o tipo de item que será exibido no *Action Bar*. Os itens do atributo *showAsAction* pode ser:

- ***always*** – Indica que o botão sempre será exibido no botão de ação (*Action Buttons*). Sendo considerado o item recomendado para definir as ações mais comuns do aplicativo.
- ***ifRoom*** – Exibe o item no botão de ação (*Action Buttons*) somente se houver espaço disponível. Sendo considerado o item de configuração mais adequado para manter a compatibilidade em diversos tipos de dispositivos.
- ***withText*** – Mostra o título do item ao lado do ícone caso haja espaço disponível no botão de ação (*Action Buttons*).



- **never** – Nunca exibe o item, nesse caso o item é exibido no *submenu* de ações (*Action Overflow*).

As configurações dos itens do atributo *showAsAction* pode ser definida no arquivo *activity_main.xml*. Esses itens são declarados na *tag* *menu*, com a declaração do item com a função *showAsAction* podendo ser *Always*, *ifRoom*, *withText* ou *never*. O código do arquivo *activity_main.xml* com os itens personalizados do *Action Bar* customizado pode ser visualizado na Figura 2.8 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android" >
  <!-- Search / will display always -->
  <item android:id="@+id/action_search"
        android:icon="@mipmap/ic_launcher_round"
        android:title="Android"
        app:showAsAction="always"/>

  <item android:id="@+id/camera"
        android:icon="@android:drawable/ic_menu_camera"
        android:title="camera"
        app:showAsAction="ifRoom" />

  <item android:id="@+id/help"
        android:icon="@android:drawable/ic_menu_help"
        android:title="help"
        app:showAsAction="never" />

  <item android:id="@+id/Agenda"
        android:icon="@android:drawable/ic_menu_agenda"
        android:title="Agenda"
        app:showAsAction="never" />

  <item android:id="@+id/menuJanela3"
        android:icon="@android:drawable/ic_menu_call"
        android:title="call"
        app:showAsAction="withText" />
</menu>
```

Figura 2.8 – Código *activity_main.xml* com customização do *Action Bar*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a customização do *Action Bar* possuindo a descrição dos cinco itens com nome *Android*, câmera, help, agenda e *call*.

A customização do *Action Bar* no arquivo *MainActivity.java* possui o método *getMenuInflater* que permite configurar as ações definidas no arquivo *activity_main.xml* e adicioná-las ao botão de ações do *Action Bar*. Por fim, o método *onOptionsItemSelected* é responsável por iniciar uma *Activity* e executar os botões de ações no arquivo *activity_main.xml*. O código do arquivo



MainActivity.java do *Android Studio* representando o *Action Bar* pode ser visualizado na Figura 2.9 abaixo:

```
package com.example.eadpernambucopdm;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MenuInflater;
import android.view.Menu;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_main, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

Figura 2.9– Arquivo da classe *MainActivity.java* com a customização do *Action Bar*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* com a customização do *Action Bar* possuindo a execução da tela do *Action Bar* no arquivo *activity_main.xml*

A tela do *Android Studio* representando a customização do *Action Bar* pode ser visualizado na Figura 2.10 abaixo:

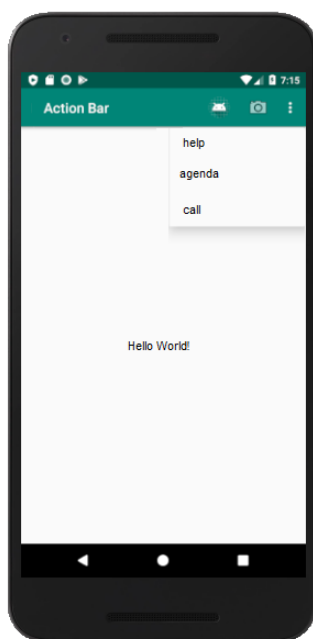


Figura 2.10 – Tela com customização do *Action Bar*



Fonte: o autor

Descrição da imagem: a imagem apresenta uma tela do celular com customização do Action bar mostrando na parte superior da tela a mensagem Action Bar e os dois botões com o símbolo do Android e da câmera e os três menus com os nomes help, agenda e call e na barra inferior da tela três botões do celular com símbolos da seta, bola e quadrado e na parte do meio da tela a mensagem "hello world".

2.5 Modificando Action Bar com temas

A aplicação de temas no *Android Studio* ocorre no arquivo *AndroidManifest.xml*. Nesse arquivo, o tema da aplicação é definido no atributo *android:theme* na *tag application*. A alteração do tema pode ser executada acessando o arquivo *themes.xml*. O código do arquivo *AndroidManifest.xml* com a modificação do tema pode ser visualizado na Figura 2.11 abaixo:

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

    <application

        android:theme="@android:style/Theme.Black">

    </application>

</manifest>
```

Figura 2.11 – Arquivo *AndroidManifest.xml* com modificação do tema

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* possuindo a *tag application* com a aplicação do tema e a descrição do *theme* com acesso específico.

O código anterior acessa o arquivo *themes.xml* e realiza a configuração do *layout* da aplicação para cor preta. O código do arquivo *themes.xml* com a configuração das cores pode ser visualizado na Figura 2.12 abaixo:

```
<resources>

    <style name="Theme.Black">
        <item name="windowBackground">@color/black</item>
        <item name="colorBackground">@color/black</item>
    </style>

</resources>
```

Figura 2.12– Arquivo *themes.xml* com a configuração das cores.

Fonte: o autor



Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *themes.xml* possuindo a *tag style* com o tema preto e duas *tags item* com os *name windowBackground* representando a cor da janela de fundo e o *colorBackground* representando a cor de fundo.

A execução desse código gera a tela do *layout* para a cor preta. A tela com a execução do *layout* para a cor preta pode ser visualizada na Figura 2.13 abaixo:

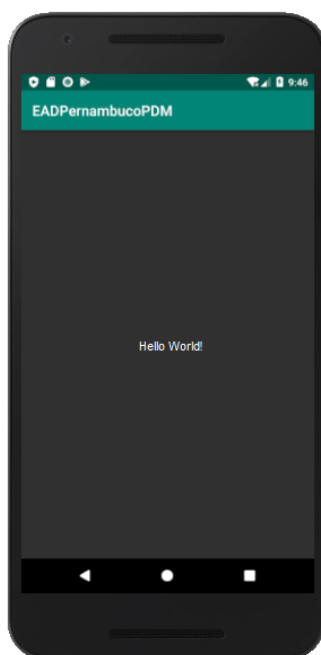


Figura 2.13 – tela com modificação de temas

Fonte: o autor

Descrição da imagem: a imagem apresenta uma tela do celular com modificação de temas mostrando na parte superior da tela a mensagem EADPernambucoPDM e na barra inferior da tela três botões do celular com símbolos da seta, bola e quadrado e na parte do meio com *layout* com fundo preto a mensagem “hello world” na cor branca.

Outra maneira de alterar a cor dos temas é acessando o arquivo *styles.xml*. Nesse arquivo, deve-se alterar a descrição do tema para *Theme.AppCompat*, dessa forma modifica a cor do layout para preto. O código do arquivo *styles.xml* com a configuração do tema com a cor preta pode ser visualizado na Figura 2.14 abaixo:

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```



Figura 2.14 – Arquivo `styles.xml` com o tema na cor preta.

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo `styles.xml` possuindo o tema com a cor preta representada pela `tag style` com o nome do estilo `AppTheme` e os três itens representando a cor primária, cor primária sombria e cor acentuada.

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- Criando um novo projeto com *Action Bar*
- Conceito de *Action Bar*
- Conceito de temas
- Modificando *Action Bar* com temas



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 2?
Acesse o AVA e assista a videoaula da competência 2.



Agora, acesse o AVA e responda as questões da competência 2.



Ficou com alguma dúvida na competência 2? **Acesse o Fórum - “Competência 2”** para saná-las com os professores e discutir com seus colegas sobre os assuntos estudados.

Nosso próximo passo é seguir para competência 3. Na competência 3 você vai aprender sobre conceitos de gerenciadores de layout ou *ViewGroup* e sobre os layouts *ConstraintLayout* e *LinearLayout*. Vamos juntos?



3.Competência 03 | Conhecer os conceitos de interface gráfica - gerenciadores de layout

Caro estudante, nesta competência você vai aprender sobre os conceitos de interface gráfica utilizando a classe *View* com os seus componentes da classe *ViewGroup* ou gerenciadores de *layout*.

Em seguida, será visto os *layouts ConstraintLayout* e *LinearLayout*. Por fim, será mostrado os componentes de dimensões do *layout*.

Vamos continuar!

3.1 Criando um novo projeto com a classe ViewGroup

Na disciplina Introdução a programação para dispositivos móveis você aprendeu a criar um novo projeto no *Android Studio*. Você lembra? Na criação desse novo projeto é gerado o arquivo *activity_main.xml* responsável por modificar a classe *ViewGroup*. A declaração da classe *ViewGroup* utiliza alguns componentes da paleta de *layouts*. Os principais *layouts* mostrados nessa paleta são *ConstraintLayout*, *LinearLayout*, *FrameLayout*, *RelativeLayout* e *TableLayout*. A declaração do identificador desses layouts e a modificação do nome do atributo *id* (identificador) ocorre na paleta atributos. Por exemplo: o identificador do *textView* declarado na Figura 3.1, abaixo, possui o *id* representado pelo nome “campo” e a sua declaração no arquivo XML é representado por *android:id="@+id/campo"*. A tela com a execução do arquivo *activity_main.xml* da classe *ViewGroup* pode ser visualizada na Figura 3.1 abaixo:

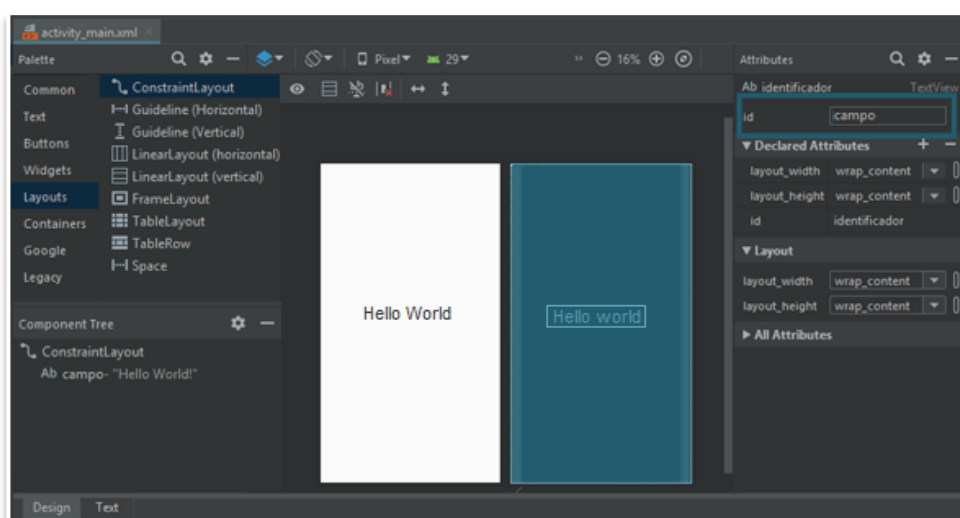




Figura 3.1 - Arquivo `activity_main.xml` gerado pelo novo projeto da classe `ViewGroup`

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo `activity_main.xml` possuindo do lado esquerdo a opção clicada na paleta de `layout` do tipo `ConstraintLayout` e o componente campo com o nome `"Hello World"`, e no lado direito duas telas da aplicação nas cores branco e verde com o nome centralizado `"Hello World"`.

Até aqui você aprendeu como criar um novo projeto com a classe `ViewGroup`. Nas seções a seguir, você vai aprender sobre o conceito da classe `View` e sobre o seu componente `ViewGroup`.

3.2 Conceito da classe `View`

A classe `View` é considerada a classe principal dos componentes visuais do *Android*, e suas subclasses são utilizadas na criação das interfaces gráficas dos aplicativos. Além disso, é uma classe base de qualquer outro componente utilizada com a respectiva classe `Activity`. Cada subclasse dessa classe `View` representada no *Android* por `android.view.View` deve realizar a implementação do método `onDraw(Canvas)`, essa classe `Canvas` utilizada no método `onDraw` define os métodos para desenhar texto, linhas, bitmaps e outras primitivas gráficas das telas dos aplicativos.

No *Android* podemos classificar a classe `View` em dois tipos de componentes: os *widgets* e os gerenciadores de layout ou `ViewGroup`. O *widget* é considerado um componente que herda diretamente da classe `View`, como exemplos existe o elemento `TextView` representada pela classe `android.widget.TextView`, além de outros elementos como `Button`, `ImageButton`, `ImageView` e o gerenciador de layout ou `ViewGroup` representado pela classe `android.view.ViewGroup`, é considerado um componente mais popular sendo representado pelos layouts dos aplicativos. A imagem representando as classes da `View` pode ser visualizada na Figura 3.2 abaixo:

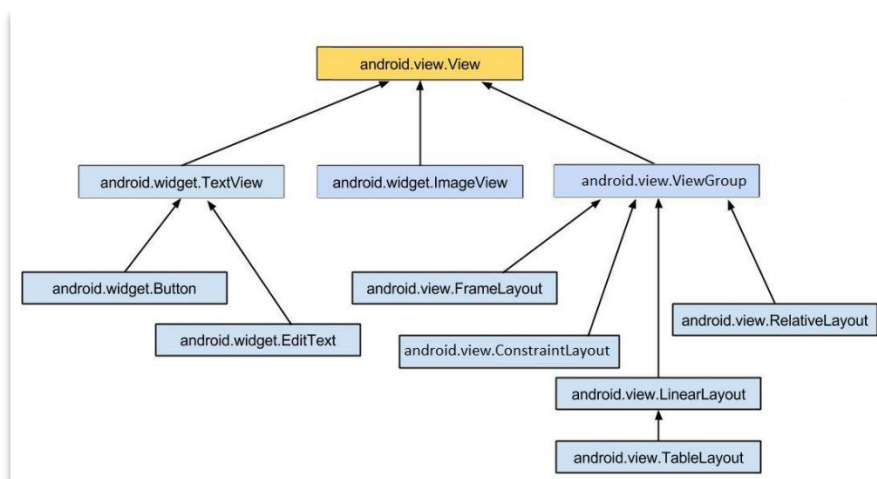




Figura 3.2 – Representação da classe View

Fonte: o autor

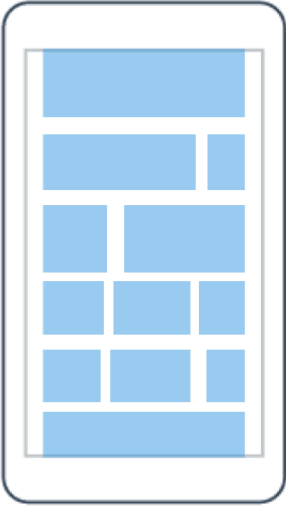
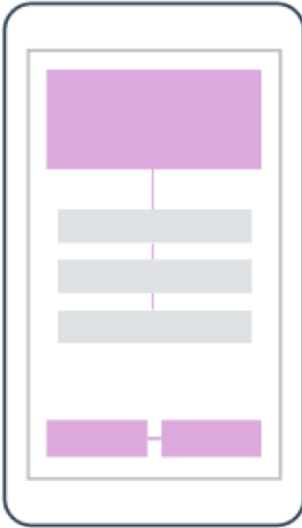
Descrição da imagem: a imagem apresenta o fluxograma de execução da classe *View*. O retângulo amarelo com a classe *android.view.View* herda do retângulo azul o componente *ViewGroup* ou gerenciador de layouts representado pela classe *android.view.ViewGroup* que herda cinco retângulo azuis com as classes *android.view.FrameLayout*, *android.view.ConstraintLayout*, *android.view.RelativeLayout* e a classe *android.view.TableLayout* herdada por *android.view.LinearLayout*. O retângulo amarelo com a classe *android.view.View* também herda os componentes da *Widget* representada pelo retângulo azul da classe *android.widget.TextView* que herda dois retângulos representados por *android.view.Button* e *android.widget.EditText*, além de herda o quadrado azul representado pela classe *android.widget.ImageView*.

3.3 Conceito da classe ViewGroup

O gerenciador de *layout*, ou classe *ViewGroup*, é considerado um componente visual responsável por organizar os componentes da tela. Essa classe é representada no Android por *android.view.ViewGroup*, sendo assim considerada a classe principal de todos os gerenciadores de *layout*. Os principais componentes da *ViewGroup* pode ser visualizado na Tabela 3.1 abaixo.

Componente	Descrição	Figura do Layout
<i>LinearLayout</i>	É um componente utilizado para organizar as telas na forma linear no sentido horizontal ou vertical. Por exemplo, o elemento vai ter que ser posicionar na telade maneira horizontalmente ou verticalmente.	



<i>TableLayout</i>	<p>É um componente semelhante ao <i>LinearLayout</i>, sendo utilizado para organizar os componentes numa tabela com linhas e colunas. Por exemplo, o layout possui forma de tabela, sendo cada componente representado por uma coluna, e pode ocupar duas ou mais colunas.</p>	
<i>RelativeLayout</i>	<p>É um componente utilizado para posicionar um componente de forma relativa a outro. Por exemplo, o elemento acima, abaixo de outro componente já existente ou ao lado de um componente já existente.</p>	




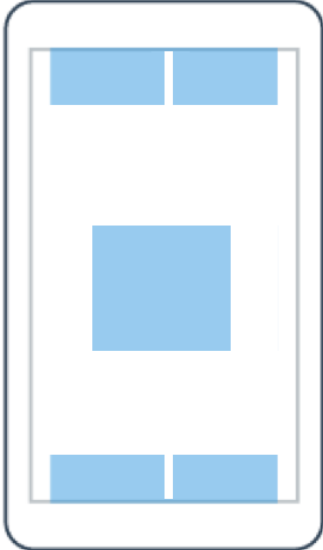
<p><i>ConstraintLayout</i></p>	<p>É similar ao componente da <i>RelativeLayout</i>, ou seja, o seu alinhamento é feito de relações entre as <i>Views</i> por meio dos seus eixos. Por exemplo: o componente de texto centralizado e os seus eixos definidos como X sendo eixo de início (esquerdo) e fim (direito) e o Y o eixo do topo (cima) e inferior (baixo).</p>	
<p><i>FrameLayout</i></p>	<p>É um componente responsável por adicionar componentes um sobre o outro e o alinhamento só pode ser feito pelas bordas do layout, podendo ser representada no topo, inferior, superior, direita ou centralizado. Por exemplo: os componentes do topo esquerdo, topo direito, inferior direito, inferior esquerdo e centralizado.</p>	

Tabela 3.1 – Os principais componentes do ViewGroup ou gerenciadores de layout

Fonte: o autor

Descrição da imagem: a descrição de cinco componentes da classe *ViewGroup* representadas pelos *layouts* *ConstraintLayout*, *LinearLayout*, *FrameLayout*, *RelativeLayout* e *TableLayout*.



Existem diversas classes que herdam de *ViewGroup*, como por exemplo *GridLayout*, *GridView* e *TableRow*. Portanto, nesse e-book de programação para dispositivos móveis você vai aprender sobre os cinco principais tipos de *layouts* mostrado na Tabela 3.1, acima. Esses componentes são considerados os mais utilizados como componente visual em telas de aplicativos.

Nas seções a seguir, você vai aprender sobre os componentes de *layout* *LinearLayout* e *ConstraintLayout*. Os componentes *FrameLayout*, *RelativeLayout* e *TableLayout* serão mostrados na próxima competência.

3.3.1 LinearLayout

O *LinearLayout* é considerado o gerenciador de *layout* mais utilizado, pois possui a possibilidade de organizar os seus componentes internos de maneira linear, posicionando objetos um abaixo do outro, quando configurado na orientação vertical ou um ao lado do outro, quando configurado na orientação horizontal. Na tela da aplicação, a orientação vertical representa o *layout* de cima para baixo e na orientação horizontal representa o *layout* da esquerda para a direita.

Toda classe do *LinearLayout* representada no *Android* por *android.widget.LinearLayout* possui os atributos *layout_width* (*android:layout_width*) e o *layout_height* (*android:layout_height*), para especificar, respectivamente, a largura e altura do *layout*. Esses atributos possuem dois valores importantes, o *match_parent* e o *wrap_content*. O valor do *match_parent* informa que o componente deve ocupar o tamanho definido pelo seu *layout*, ou seja, o *layout* deve ocupar toda a tela da aplicação e o *wrap_content* informa que o componente deve ocupar apenas o seu tamanho real, ou seja, o *layout* não possui nenhum tipo de alteração.

Além desses atributos, o *LinearLayout* possui o atributo *orientation* (*android:orientation*), com a função de especificar a orientação dos seus respectivos componentes. Esse atributo *orientation* pode ser representado por valores na horizontal (*android:orientation="horizontal"*) ou vertical (*android:orientation="vertical"*), outra maneira é o valor *default* (*android:orientation=""*), o qual é utilizado quando não é especificado nenhum valor, assim, os elementos serão exibidos de maneira horizontal. Outro atributo é o *android:gravity*, com a função de localizar a gravidade do *layout* na tela. Esse atributo *gravity* pode ser representado pelo *layout* centralizado verticalmente e horizontalmente na tela. A imagem representando os componentes *LinearLayout* pode ser visualizada na Figura 3.3 abaixo:

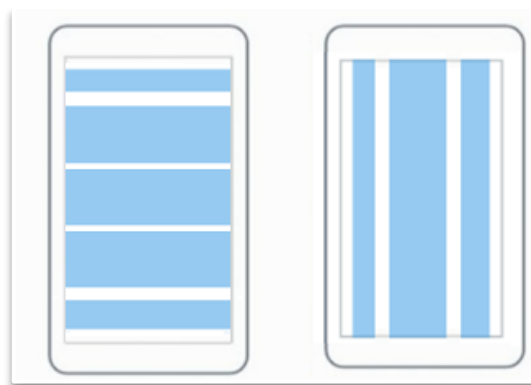


Figura 3.3 – Telas representando os componentes do LinearLayout

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas de celular na cor fundo branco e com linhas azuis representando o layout do *LinearLayout*. A primeira tela possui uma tela (lado esquerdo) com fundo branco e cinco linhas na horizontal representando o *LinearLayout* horizontal e a segunda tela (lado direito) com fundo branco e três linhas na vertical representando o *LinearLayout* vertical.

Por exemplo, imagine que existe um aplicativo de jogo com o *layout LinearLayout* com a visualização horizontal dos jogos Mario, Lol e *Barkbridge*. Diante desse exemplo, você vai aprender como utilizar a classe *LinearLayout* na visualização de imagens dos jogos na horizontal. O arquivo *activity_main.xml* mostrado na Figura 3.4, abaixo, possui o *layout LinearLayout* com orientação horizontal e três classes *ImageView* com as imagens dos jogos Mario, Lol e *Barkbridge*. Esse código do *activity_main.xml* representando o *LinearLayout* horizontal pode ser visualizado na Figura 3.4 abaixo:

```
<?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"
    android:orientation="horizontal"
    tools:context=".MainActivity">
    <ImageView
        android:id="@+id/lol"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_mario"
    />
    <ImageView
        android:id="@+id/mario"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_barkbridge"
    />
    <ImageView
```



```
android:id="@+id/barkbringe "  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:src="@mipmap/ic_launcher_lol"  
/>  
</LinearLayout>
```

Figura 3.4 - Arquivo activity_main.xml com LinearLayout horizontal

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo activity_main.xml com a representação do *LinearLayout* horizontal possuindo a representação da *tag LinearLayout* com os componentes altura, largura, orientação horizontal e o contexto do layout, e as três *tags* de imagens representadas pelas imagens do mario, barkbridge e lol.

Depois da execução do arquivo *activity_main.xml* com *LinearLayout* horizontal da Figura 3.4, acima, e gerado o *designer* da tela no *Android Studio* visualizado na Figura 3.5 abaixo:

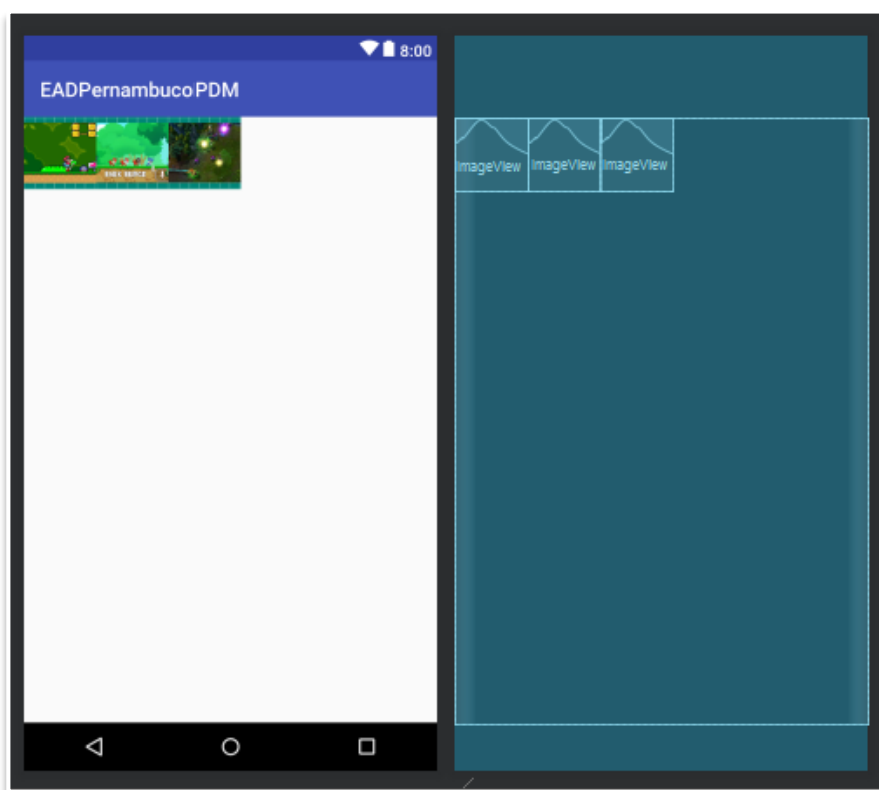


Figura 3.5 - Tela do LinearLayout horizontal

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *LinearLayout* horizontal. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio da tela três imagens dos jogos mario, barkbridge e lol organizado na sequência de maneira horizontal, a tela do lado direito possui um quadrado com fundo verde representado a tela e três quadrados pequenos representados as três imagens organizada de maneira horizontal.

O *LinearLayout* horizontal configura no arquivo *MainActivity.java* da Figura 3.6, abaixo, esse *LinearLayout* horizontal possui a declaração do método *setOrientation()* com orientação



horizontal e na imageView é utilizado o método *addView()* para visualizar uma respectiva imagem. Por fim é executado o método *setContextview()* responsável por configurar o *layout* XML com o *LinearLayout*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *LinearLayout* horizontal pode ser visualizado na Figura 3.6 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.HORIZONTAL);
        LinearLayout.LayoutParams param = new
        LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.MATCH_PARENT);
        linearLayout.setLayoutParams(param);

        ImageView imageView = new ImageView(this);
        imageView.setImageResource(R.mipmap.lol);
        linearLayout.addView(imageView);

        ImageView imageView2 = new ImageView(this);
        imageView2.setImageResource(R.mipmap.mario);
        linearLayout.addView(imageView2);

        ImageView imageView3 = new ImageView(this);
        imageView3.setImageResource(R.mipmap.barkbringe);
        linearLayout.addView(imageView3);

        setContentView(linearLayout);
    }
}
```

Figura 3.6 – Arquivo da classe *MainActivity.java* do *LinearLayout* horizontal

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *LinearLayout* horizontal no arquivo *activity_main.xml*.



Por exemplo, imagine que existe um aplicativo de jogo com o *layout* *LinearLayout* com a visualização vertical dos jogos Mario, Lol e *barkbridge*. Diante desse exemplo, você vai aprender como utilizar a classe *LinearLayout* na visualização de imagens dos jogos na vertical. O arquivo *activity_main.xml* mostrado na Figura 3.7, abaixo, possui o *layout* *LinearLayout* com orientação vertical e três classes *ImageView* com as imagens dos jogos Mario, Lol e Barkbridge. Esse código do *activity_main.xml* representando o *LinearLayout* vertical pode ser visualizado na Figura 3.7 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_mario"
    />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_barkbridge"
    />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_lol"
    />
</LinearLayout>
```

Figura 3.7 – Arquivo *activity_main.xml* com *LinearLayout* vertical

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *LinearLayout* vertical possuindo a representação da tag *LinearLayout* com os componentes altura, largura, orientação vertical e o contexto do layout, e as três *tags* de imagens representadas pelas imagens do mario, barkbridge e lol.

Depois da execução do arquivo *activity_main.xml* com *LinearLayout* vertical da Figura 3.7, acima, e gerado o *designer* da tela no *Android Studio* visualizado na Figura 3.8 abaixo:

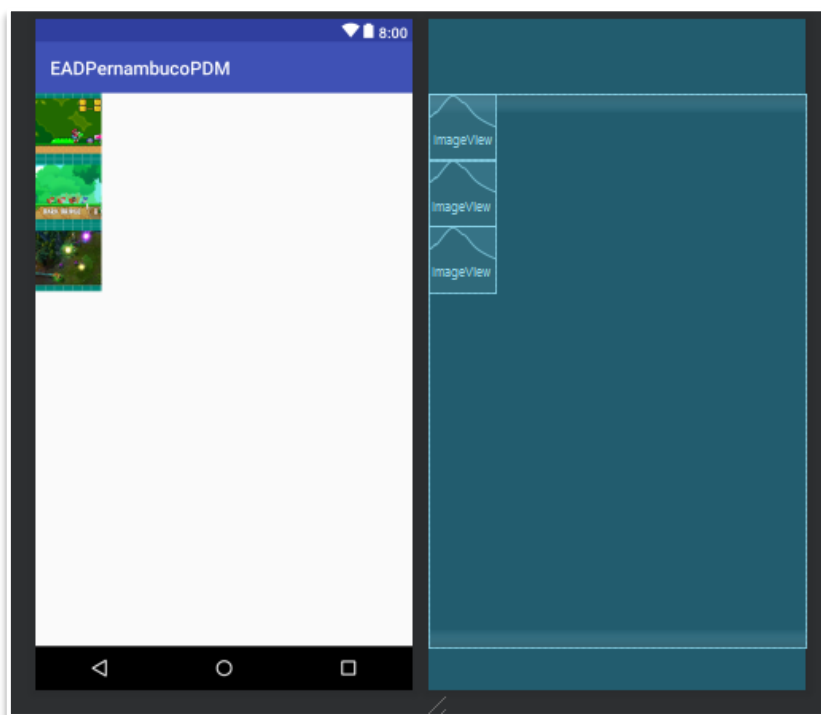


Figura 3.8 – Tela do LinearLayout vertical

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *LinearLayout* vertical. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio da tela três imagens dos jogos mario, barkbridge e lol organizado na sequência de maneira vertical, a tela do lado direito possui um quadrado com fundo verde representado a tela e três quadrados pequenos representados as três imagens organizada de maneira vertical.

O *LinearLayout* vertical configura no arquivo *MainActivity.java* da Figura 3.9, abaixo, esse *LinearLayout* horizontal possui a declaração do método *setOrientation()* com orientação vertical e na *imageView* é utilizado o método *addView()* para visualizar uma respectiva imagem. Por fim é executado o método *setContextview()* responsável por configurar o *layout* XML com o *LinearLayout*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *LinearLayout* vertical pode ser visualizado na Figura 3.9 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
```



```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    LinearLayout linearLayout = new LinearLayout(this);  
    linearLayout.setOrientation(LinearLayout.VERTICAL);  
    LinearLayout.LayoutParams param = new  
    LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,  
    LinearLayout.LayoutParams.MATCH_PARENT);  
    linearLayout.setLayoutParams(param);  
  
    ImageView imageView = new ImageView(this);  
    imageView.setImageResource(R.mipmap.lol);  
    linearLayout.addView(imageView);  
  
    ImageView imageView2 = new ImageView(this);  
    imageView2.setImageResource(R.mipmap.mario);  
    linearLayout.addView(imageView2);  
  
    ImageView imageView3 = new ImageView(this);  
    imageView3.setImageResource(R.mipmap.barkbringe);  
    linearLayout.addView(imageView3);  
    setContentView(linearLayout);  
}
```

Figura 3.9 – Arquivo da classe MainActivity.java do LinearLayout vertical

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo MainActivity.java possuindo a execução da tela do LinearLayout vertical no arquivo activity_main.xml.

3.3.2 ConstraintLayout

O *ConstraintLayout* é considerado o gerenciador de *layout* padrão do *Android*, pois possui recursos de outros gerenciadores de *layout*, assim como, uma série de outras funcionalidades desses layouts para posicionar e redimensionar os seus componentes. Nesse *layout* é possível criar *layout* planos, ou seja, sem as características de encadeamento de diversos *layouts*, como exemplo o *LinearLayout* dentro de *FrameLayout* e dentro do *RelativeLayout*. Dessa maneira, os *layouts* do *ConstraintLayout* possuem a tendência de renderizar de forma mais rápida e eficiente.

O *ConstraintLayout* representada pela classe *android.widget.ConstraintLayout* possui os atributos *layout_width* (*android:layout_width*) e o *layout_height* (*android:layout_height*), para definir, consequentemente, a sua largura e a altura. Nesses atributos é utilizado o valor *match_parent* para informar que o componente deve ocupar um tamanho definido pelo seu respectivo *layout*, ou seja, esse *layout* deve ocupar a tela inteira da aplicação. Além desses atributos, o *ConstraintLayout*



possui os atributos de *Bias* representando a inclinação do *Layout* e os atributos *Editor* representando o posicionamento do layout como visualizado na Tabela 3.2 abaixo:

Atributo	Descrição
<code>app:layout_constraintHorizontal_bias</code>	define a inclinação horizontal do <i>layout</i> da tela
<code>app:layout_constraintVertical_bias</code>	define a inclinação vertical do <i>layout</i> da tela
<code>tools:layout_editor_absoluteX</code>	define o posicionamento do eixo X na tela do <i>layout</i>
<code>tools:layout_editor_absoluteY</code>	define o posicionamento do eixo y na tela do <i>layout</i>

Tabela 3.2 – Atributos de posicionamento do *ConstraintLayout*

Fonte: o autor

Descrição da imagem: a descrição dos quatro atributos de posicionamento do *ConstraintLayout*.

Por exemplo, imagine que existe um aplicativo de jogo com o *layout ConstraintLayout* com a descrição textual do jogo no formato de texto. Diante desse exemplo, você vai aprender como utilizar a classe *ConstraintLayout* na visualização da descrição do texto. O arquivo *activity_main.xml* mostrado na Figura 3.10, abaixo, possui o *layout ConstraintLayout* e a classe *TextView* com a descrição "Descrição do jogo!". Esse código do *activity_main.xml* representando o *layout ConstraintLayout* com a classe *TextView* pode ser visualizado na Figura 3.10 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Descrição do jogo!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



Figura 3.10 - Arquivo activity_main.xml do ConstraintLayout com TextView

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *ConstraintLayout* com *TextView* possuindo a representação da *tag ConstraintLayout* com os componentes altura, largura e o contexto do layout, e a *tag* do *TextView* com os componentes altura, largura e texto com a mensagem “Descrição do jogo!”.

Depois da execução do arquivo *activity_main.xml* do *ConstraintLayout* com *TextView* da Figura 3.10, acima, e gerado o designer da tela no Android Studio visualizado na Figura 3.111 abaixo:

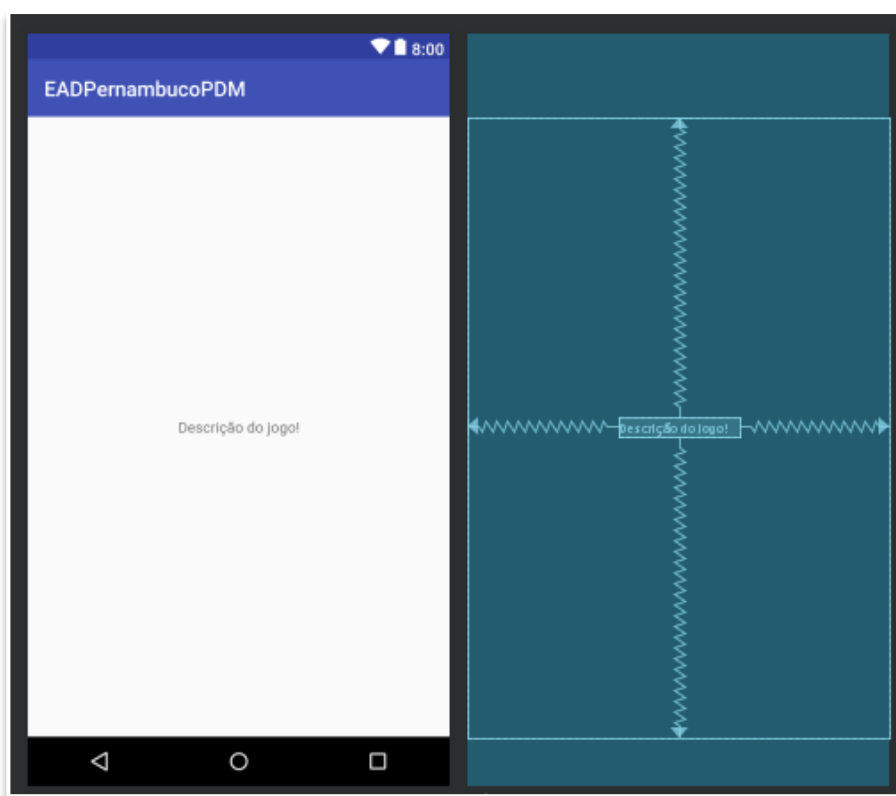


Figura 3.11 – Tela do ConstraintLayout com TextView

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do Android Studio representando o *ConstraintLayout* com *TextView*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio o texto descrição do jogo na cor cinza, a tela do lado direito possui um quadrado com fundo verde representado a tela e quadrado centralizado na tela com o nome “Descrição do jogo”.

O *ConstraintLayout* com texto configura no arquivo *MainActivity.java* da Figura 3.12, abaixo, esse *ConstraintLayout* possui a declaração do método *setId()* com o identificador do *ConstraintLayout* e no *TextView* é utilizado o método *addView()* para visualizar a mensagem do texto “descrição do jogo”. Por fim é executado o método *setContextview()* responsável por configurar o *layout* XML com o *ConstraintLayout*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *ConstraintLayout* com *TextView* pode ser visualizado na Figura 3.12 abaixo:



```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        constraintLayout.setId(R.id.constraintLayout);
        ConstraintLayout.LayoutParams param = new
        ConstraintLayout.LayoutParams(ConstraintLayout.LayoutParams.MATCH_PARENT,
        ConstraintLayout.LayoutParams.MATCH_PARENT);
        constraintLayout.setLayoutParams(param);

        TextView textView = new TextView(this);
        textView.setText("descrição do jogo");
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```

Figura 3.12 – Arquivo da classe MainActivity.java do ConstraintLayout com TextView

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo MainActivity.java possuindo a execução da tela do ConstraintLayout com TextView no arquivo activity_main.xml.

Por exemplo, imagine que existe um aplicativo de jogo com o layout *ConstraintLayout* com o botão de acesso ao jogo. Diante desse exemplo, você vai aprender como utilizar a classe *ConstraintLayout* utilizando um botão do jogo. O arquivo *activity_main.xml* mostrado na Figura 3.13, abaixo, possui o layout *ConstraintLayout* e a classe *Button* com a descrição="Acesso ao jogo". Esse código do *activity_main.xml* representando o layout *ConstraintLayout* com a classe *Button* pode ser visualizado na Figura 3.13 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match parent"
```




```
android:layout_height="match_parent"
tools:context=".MainActivity">

<Button
    android:id="@+id/constraintLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="152dp"
    android:layout_marginTop="348dp"
    android:text="@+string/Acesso ao jogo"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 3.13 – Arquivo activity_main.xml do ConstraintLayout com Button

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *ConstraintLayout* com Button possuindo a representação da *tag ConstraintLayout* com os componentes altura, largura e o contexto do layout, e a *tag* do Button com os componentes id do botão, altura, largura e texto com a mensagem “Acesso ao jogo”.

Depois da execução do arquivo activity_main.xml do *ConstraintLayout* com *Button* da Figura 3.13, acima, e gerado o designer da tela no Android Studio visualizado na Figura 3.14 abaixo:

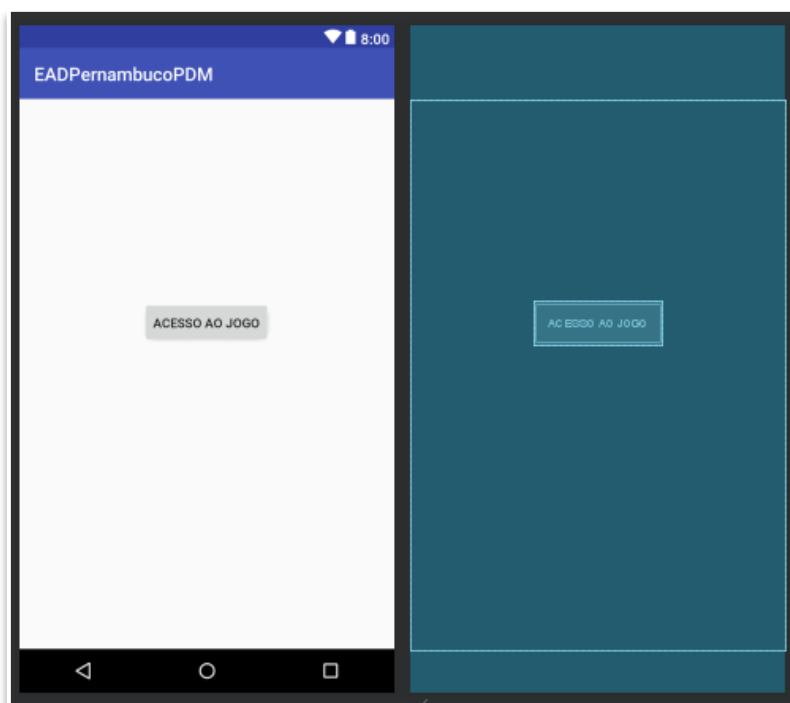


Figura 3.14- Tela do ConstraintLayout com Button

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *ConstraintLayout* com botão. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio o botão na cor azul com o texto acesso ao jogo, a tela do lado direito



possui um quadrado com fundo verde representado a tela e um quadrado verde claro centralizado na tela representando o botão com o nome acesso ao jogo na cor branca.

O *ConstraintLayout* com botão configura no arquivo *MainActivity.java* da Figura 3.15, abaixo, esse *ConstraintLayout* possui a declaração do método *setId()* com o identificador do *ConstraintLayout* e no *Button* é utilizado o método *addView()* para visualizar a mensagem do botão “Acesso ao jogo”. Por fim é executado o método *setContentView()* responsável por configurar o *layout* XML com o *ConstraintLayout*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *ConstraintLayout* com *Button* pode ser visualizado na Figura 3.15 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        constraintLayout.setId(R.id.constraintLayout);
        ConstraintLayout.LayoutParams param = new
        ConstraintLayout.LayoutParams(ConstraintLayout.LayoutParams.MATCH_PARENT,
        ConstraintLayout.LayoutParams.MATCH_PARENT);
        constraintLayout.setLayoutParams(param);

        Button button = new Button(this)
        button.setText("Acesso ao jogo");
        constraintLayout.addView(button);

        setContentView(constraintLayout);
    }
}
```

Figura 3.15 – Arquivo da classe *MainActivity.java* do *ConstraintLayout* com *Button*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *ConstraintLayout* com *Button* no arquivo *activity_main.xml*.



3.4 Dimensões dos componentes do layout

As dimensões de altura e largura geralmente utilizam valores como *match_parent* e *wrap_content*. Entretanto, outra possibilidade possível é definir os valores específicos, ou seja, valores numéricos com notações indicando a unidade que está sendo utilizado. Por exemplo: 15px (*pixels*) ou 16mm (milímetros). Dessa forma é possível modificar a dimensão do *layout* utilizando o tamanho do componente. As dimensões no *Android* e as suas respectivas notações pode ser visualizado na Tabela 3.3.

Notação	Descrição
<i>px (pixels)</i>	É uma medida que corresponde ao número de <i>pixels</i> que será ocupado na tela.
<i>in (inches)</i>	<i>Inches</i> significa polegadas, é uma medida baseada no tamanho físico em polegadas da tela.
<i>mm (milímetros)</i>	É uma medida baseada no tamanho em milímetros da tela.
<i>pt (pontos)</i>	É uma medida baseado no tamanho físico de pontos da tela. A medida pt corresponde a 1/72 de uma polegada.
<i>dp (Density-independent Pixels)</i>	É a unidade relacionada à resolução da tela. Por exemplo, caso a resolução da tela é de 160 dpi, significa que para cada dp representa 1 pixel em um total de 160. A notação dp é a mais utilizada.
<i>Sp (Scale-independent Pixels)</i>	É a unidade mais recomendada quando pretende especificar o tamanho de uma fonte, para que esta seja automaticamente ajustada conforme a resolução da tela do usuário. A notação sp é semelhante a notação dp.

Tabela 3.3 – Notações com dimensões dos componentes de layout

Fonte: o autor

Descrição da imagem: a descrição das seis notações com dimensões dos componentes de layout representadas por *px (pixels)*, *in (inches)*, *mm (milímetros)*, *pt (pontos)*, *dp (Density-independent Pixels)* e *Sp (Scale-independent Pixels)*.

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- A classe *View*
- A classe *ViewGroup*
- O layout *LinearLayout*
- O layout *ConstraintLayout*
- As dimensões dos componentes do layout



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 3?
Acesse o AVA e assista a videoaula da competência 3.



Agora, acesse o AVA e responda as questões da competência 3.



Ficou com alguma dúvida na competência 3? Acesse o **Fórum - “Competência 3”** para saná-las com os professores e discutir com seus colegas sobre os assuntos estudados.

Nosso próximo passo é seguir para competência 4. Na competência 4 você vai aprender sobre conceitos avançados de gerenciadores de layout ou *ViewGroup* e sobre os layouts *FrameLayout*, *TableLayout* e *RelativeLayout*. Vamos juntos?



4. Competência 04 | Utilizar recursos avançados de gerenciadores de layout

Caro estudante, nesta competência você vai aprender os conceitos avançados de interface gráfica utilizando a classe *View* com os componentes da classe *ViewGroup* ou gerenciadores de *layout*.

Além disso, será mostrado os layouts avançados *FrameLayout*, *TableLayout* e *RelativeLayout*.

Vamos aprofundar sobre esses layouts!

4.1 Recursos Avançados da classe *ViewGroup*

Os recursos avançados da classe *ViewGroup* utilizam alguns *layouts* com componente visual responsável por organizar a disposição dos componentes na tela. Esses *layouts* são representados pela classe *android.view.ViewGroup*, com os *layouts* *FrameLayout*, *TableLayout* e *RelativeLayout*. Nas seções a seguir você vai aprender sobre os componentes dos *layouts* *FrameLayout*, *RelativeLayout* e *TableLayout*.

4.1.1 *FrameLayout*

O *FrameLayout* é considerado um *layout* mais simples, sendo normalmente utilizado para colocar um único elemento dentro desse layout, porém, quando pretende adicionar mais componentes nesse *layout*, esses ficam sobrepostos, ou seja, o primeiro elemento ficará mais atrás no *layout* e os outros ficarão mais à frente do elemento anterior, criando assim, uma sequência de camada encadeada.

Assim, cada elemento adicionado no *FrameLayout* será posicionado na posição superior esquerdo da tela e, conseqüentemente, dependendo do tamanho definido no seu atributo, pode preencher a tela inteira da aplicação. Assim, a possibilidade de inserir outros elementos na tela, de maneira que esses elementos inseridos posteriormente fiquem por cima dos elementos inseridos anteriores, seguindo assim o conceito de pilha, no qual o último elemento fica por cima dos elementos inserido primeiro.



Você pode entender um pouco sobre a execução do `FrameLayout` consultando este link: <https://www.youtube.com/watch?v=kH972agmmrc>

Por exemplo, imagine que existe um aplicativo de jogo com o layout `FrameLayout` com a variações dos atributos `layout_width` e `layout_height`, que representam respectivamente a largura e a altura do jogo do Mario em sete posições. Diante desse exemplo, você vai aprender como utilizar a classe `FrameLayout` na visualização de imagens do jogo do Mario em sete posições. O arquivo `activity_main.xml` mostrado na Figura 4.1, abaixo, possui o layout `FrameLayout` e sete classes `ImageView` com a imagem do jogo do Mario. Esse código do `activity_main.xml` representando o layout `FrameLayout` com as classes `ImageView` pode ser visualizado na Figura 4.1 abaixo:

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

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:src="@mipmap/ic_launcher_round"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|center"
        android:src="@mipmap/ic_launcher_round"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|left"
        android:src="@mipmap/ic_launcher_round"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|right"
        android:src="@mipmap/ic_launcher"/>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|left"
        android:src="@mipmap/ic_launcher"/>

    <ImageView
        android:layout_width="wrap_content"
```



```
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center"
        android:src="@mipmap/ic_launcher"/>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:src="@mipmap/ic_launcher"/>

</FrameLayout>
```

Figura 4.1 - Arquivo *activity_main.xml* com *FrameLayout*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *FrameLayout* possuindo a representação da *tag FrameLayout* com os componentes altura, largura e o contexto do layout, e as sete tags *de* imagens representadas pelo jogo do Mario.

Depois da execução do arquivo *activity_main.xml* com *FrameLayout* da Figura 4.1, acima, e gerado o *designer* da tela no *Android Studio* visualizado na Figura 4.2 abaixo:

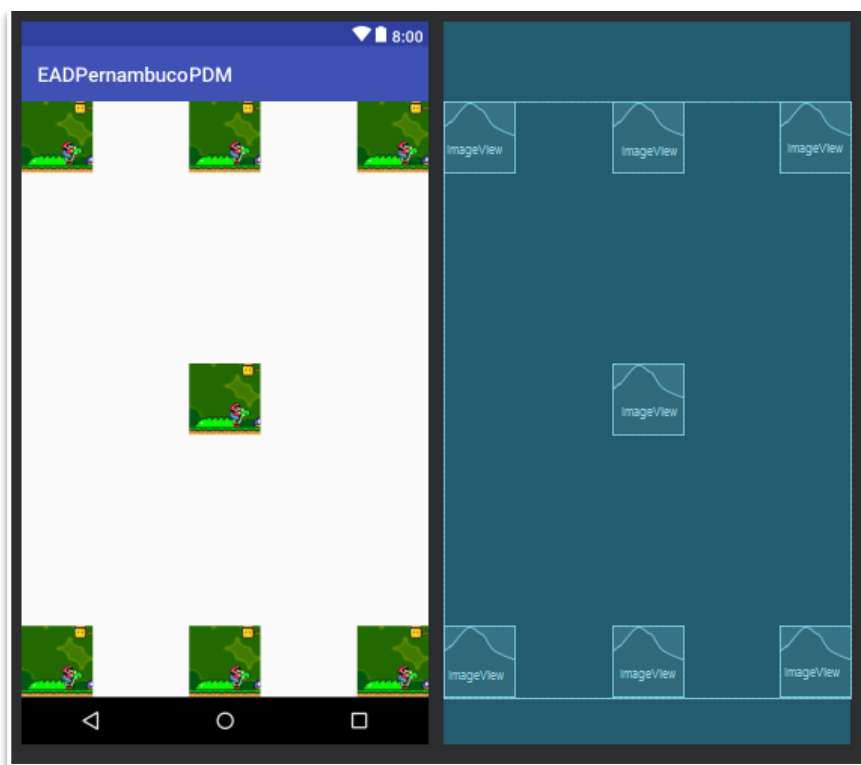


Figura 4.2 – Tela do *FrameLayout*

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *FrameLayout*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com sete imagens do jogo do Mario, sendo três na parte superior, uma no meio e três na parte inferior, a tela do lado direito possui um quadrado com fundo verde representado a tela e sete quadrado representando as imagens.



No exemplo anterior, o elemento *FrameLayout* assim como o *ImageView* possuem propriedades *layout_width* e *layout_height* identificadas como *wrap_content*. Mas o que isso significa no *FrameLayout*? Nesse layout a largura e altura representam apenas o tamanho necessário para mostrar o conteúdo inteiro, assim como a *imageView* possui apenas o tamanho para exibir a imagem do *Android_mini* e o *textView* possui apenas o tamanho do seu respectivo texto.

Por fim, a maneira para configurar o *FrameLayout* na tela inteira da aplicação. Primeiro, deve mudar as propriedades de largura e altura de *wrap_content* para *match_parent*, dessa maneira, o componente será ocupado por todo o espaço designado ao pai (parente), ocupando assim, a tela inteira e não apenas o espaço ocupado pelos seus elementos internos. O arquivo *activity_main.xml* mostrado na Figura 4.3, abaixo, possui o layout *FrameLayout* com o atributo *background* e a classe *ImageView*. Esse código XML do *Android Studio* representando o layout *FrameLayout* com *background* pode ser visualizado na Figura 4.3 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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"
    android:background="#000000"
    tools:context=".MainActivity">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_round"/>
</FrameLayout>
```

Figura 4.3 - Arquivo *activity_main.xml* com *FrameLayout* e o atributo *background*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *frameLayout* possuindo a representação da *tag frameLayout* com os componentes altura, largura, *background* representando a cor preta e o contexto do layout, e a *tag* de imagem representando o jogo do Mario.

Depois da execução do arquivo *activity_main.xml* com *FrameLayout* e o atributo *background* da Figura 4.3, acima, e gerado o designer da tela no *Android Studio* visualizado na Figura 4.4 abaixo:

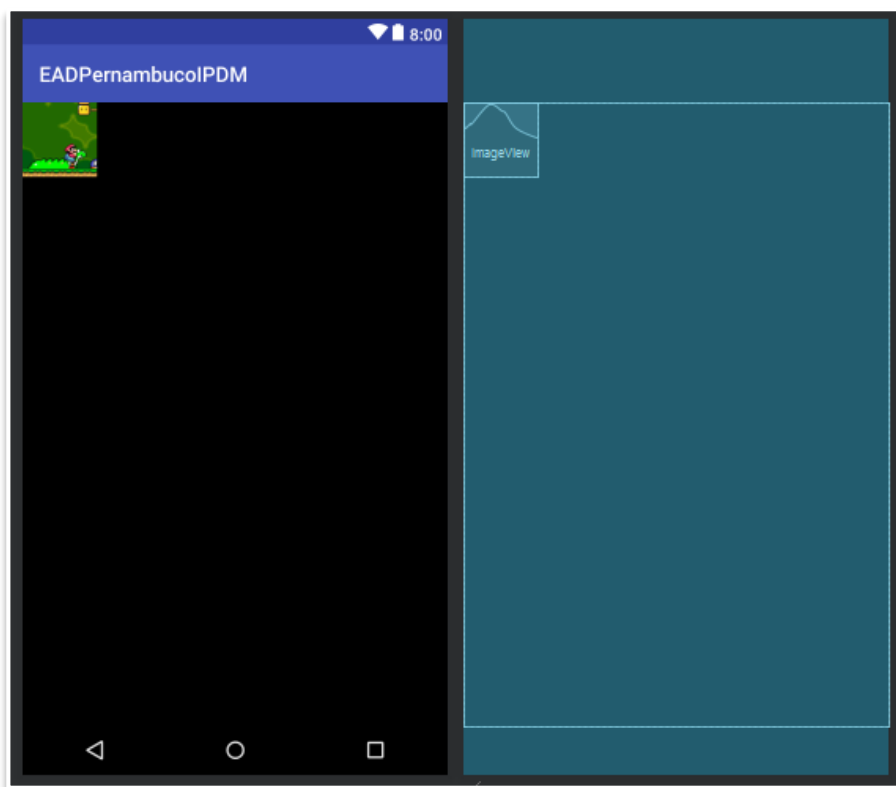


Figura 4.4- Tela *FrameLayout* com o atributo de background

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *FrameLayout*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela preta com uma imagem do jogo do Mario na parte superior, a tela do lado direito possui um quadrado com fundo verde representado a tela e um quadrado representando uma imagem.

4.1.2 *TableLayout*

O componente *TableLayout* é o *layout* utilizado para organizar os componentes numa tabela com linhas e colunas. Por exemplo, o *layout* possui uma tabela com formulário, sendo cada linha representado pela subclasse do componente *LinearLayout* representado por *android.widget.Tablerow*, e assim pode também representar outros componentes com a coluna da tabela. Esse componente também possui os atributos *android:stretchColumns* e o *android:shrinkcolumns*.

O atributo *stretchColumns* realiza nas colunas específicas um espaço disponível na tela do *layout*, sendo utilizado quando é necessário que uma determinada coluna ocupe a linha inteira. Já o atributo *shrinkcolumns* realiza nas colunas específicas uma exibição na tela do *layout*. Caso esse valor de texto seja considerado muito grande e não fique na tela, a respectiva linha e quebrada é exibida



em várias linhas na mesma coluna. Por exemplo, quando os atributos *android:shrinkColumns="1,2"* e *android:stretchColumns="0"* são definidos, o atributo *shrinkColumns* possui colunas 1 e 2 reduzidas e o *stretchColumns* possui coluna 0 esticada.



Você pode entender um pouco sobre a execução do *TableLayout* consultando este link: <https://www.youtube.com/watch?v=uraTGUn-Tzk>

Por exemplo, imagine que existe um aplicativo de jogo com o layout *TableLayout* com as variações dos atributos *shrinkColumns* e *stretchColumns*, que representam, respectivamente, a redução e extensão da coluna. Diante desse exemplo, você vai aprender como utilizar o *TableLayout* na visualização de imagens do jogo do Mario. O arquivo *activity_main.xml* mostrado na Figura 4.5, abaixo, possui o layout *TableLayout* e três classes *ImageView* com as imagens dos jogos Mario, Lol e Barkbridge. Esse código do *activity_main.xml* representando o layout *TableLayout* com as classes *ImageView* pode ser visualizado na Figura 4.5 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    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"
    android:shrinkColumns="1,2"
    android:stretchColumns="0"
    tools:context=".MainActivity">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/barkbringe" />

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/mario" />

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/lol" />

    </TableRow>
</TableLayout>
```



```
</TableRow>

</TableLayout>
```

Figura 4.5 - Arquivo activity_main.xml com TableLayout

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo activity_main.xml com a representação do *TableLayout* possuindo a representação da *tag TableLayout* com os componentes altura, largura, *shrinkColumns* com colunas 1 e 2 reduzidas, *stretchColumns* com coluna 0 esticada e o contexto do layout, e as três *tags* de imagens representadas pelos jogos do Mario, Lol e Barkbridge.

Depois da execução do arquivo activity_main.xml com *TableLayout* da Figura 4.5, acima, e gerado o designer da tela no Android Studio visualizado na Figura 4.6 abaixo:

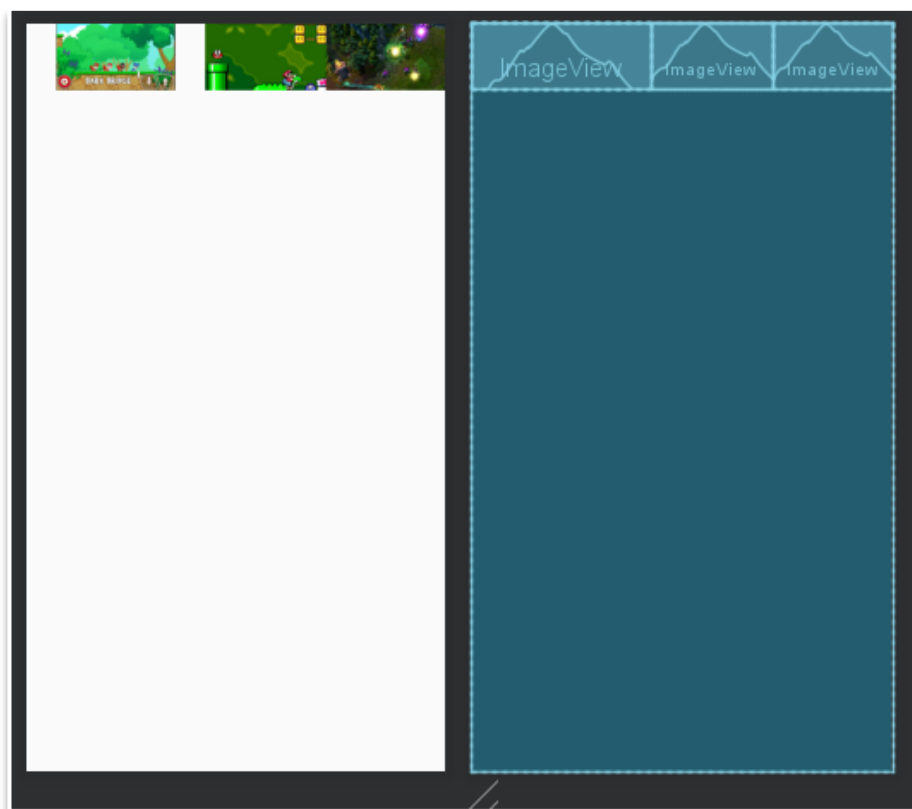


Figura 4.6 – Tela do TableLayout

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *TableLayout*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com duas linhas na parte superior representando duas imagens do jogo do Mario, a tela do lado direito possui um quadrado com fundo verde representado a tela e dois retângulos representando duas imagens.



4.1.3 RelativeLayout

O componente *RelativeLayout* também é bastante utilizada, através desse *layout* é possível posicionar um componente em relação ao outro. Nesse *layout* também é possível especificar um determinado elemento para aparecer ao lado direito ou lado esquerdo de uma determinada imagem, assim como pode ser posicionado acima ou abaixo dessa imagem.

Mas como esse componente deve aparecer ao lado, abaixo ou acima de outro componente já existente? Para realizar isso você deve definir um *ID* (identificador) para cada componente da tela, pois o posicionamento de um conjunto depende do outro. Com isso, é possível com o *RelativeLayout* dizer para um componente “A” fique abaixo do componente “B”. Os atributos importantes do *RelativeLayout* podem ser visualizados na Tabela 4.1, abaixo.

Atributo	Descrição
<i>android:layout_below</i>	Posiciona o elemento abaixo de outro.
<i>android:layout_above</i>	Posiciona o elemento acima de outro.
<i>android:layout_toLeftOf</i>	Posiciona o elemento à esquerda de outro.
<i>android:layout_toRightOf</i>	Posiciona o elemento à direita de outro
<i>android:layout_marginTop</i>	Utilizado para definir um espaço (margem) na parte superior (<i>Top</i>) do componente
<i>android:layout_marginBottom</i>	Utilizado para definir um espaço (margem) na parte inferior (<i>Bottom</i>) do componente.
<i>android:layout_marginRight</i>	Utilizado para definir um espaço (margem) na parte direita (<i>Right</i>) do componente.
<i>android:layout_marginLeft</i>	Utilizado para definir um espaço (margem) na parte esquerda (<i>Left</i>) do componente

Tabela 4.1 – Atributos de posicionamento do RelativeLayout

Fonte: o autor

Descrição da imagem: a descrição dos oito atributos de posicionamento do *RelativeLayout*.



Você pode entender um pouco sobre a execução do `RelativeLayout` consultando este link: <https://www.youtube.com/watch?v=rNyrFNxVwZs>

Por exemplo, imagine que existe um aplicativo de jogo com o layout *RelativeLayout* com um formulário de cadastro do jogo. Diante desse exemplo, você vai aprender como utilizar a classe *RelativeLayout* no formulário de cadastro o jogo com os atributos de posicionamento *layout_marginBottom* e *layout_marginLeft*. O arquivo *activity_main.xml* mostrado na Figura 4.7, abaixo, possui o layout *RelativeLayout*, a classe *TextView* possuindo a descrição "Formulário de cadastro de jogo", as três classes do *TextView* possuindo os nomes Login, senha e confirmação da senha e as três classes do *EditText* como preenchimento desse formulário. Esse código do *activity_main.xml* representando o layout *RelativeLayout* com as classes *ImageView* e *EditText* pode ser visualizado na Figura 4.7 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtViewDescricao"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="25dp"
        android:gravity="center"
        android:textSize="18dp"
        android:text="Formulário de cadastro de jogo" />

    <TextView
        android:id="@+id/txtViewLogin"
        android:layout_width="155dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtViewDescricao"
        android:layout_marginBottom="20dp"
        android:layout_marginLeft="15dp"
        android:text="Login" />

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



```
        android:layout_alignTop="@+id/txtViewLogin"
        android:layout_toRightOf="@+id/txtViewLogin"
        android:background="@android:drawable/editbox_background"/>

<TextView
    android:id="@+id/txtViewSenha"
    android:layout_width="155dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/txtViewLogin"
    android:layout_marginBottom="20dp"
    android:layout_marginLeft="15dp"
    android:text="Senha"/>

<EditText
    android:id="@+id/editTxtSenha"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/txtViewSenha"
    android:layout_toRightOf="@+id/txtViewSenha"
    android:background="@android:drawable/editbox_background"/>

<TextView
    android:id="@+id/txtViewConfirmacaoDeSenha"
    android:layout_width="155dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editTxtSenha"
    android:layout_marginBottom="20dp"
    android:layout_marginLeft="15dp"
    android:text="Confirmação de senha"/>

<EditText
    android:id="@+id/editTxtConfirmacaoDeSenha"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@android:drawable/editbox_background" />

</RelativeLayout>
```

Figura 4.7 - Arquivo *activity_main.xml* com *RelativeLayout*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *RelativeLayout* possuindo a representação da tag *RelativeLayout* com os componentes altura, largura, e as quatro tags de *TextView* com a descrição do texto do formulário e as três tags do *EditText* com campos para preencher o login, senha e a confirmação da senha do formulário.

Depois da execução do arquivo *activity_main.xml* com *RelativeLayout* da Figura 4.7, acima, e gerado o designer da tela no *Android Studio* visualizado na Figura 4.8 abaixo:

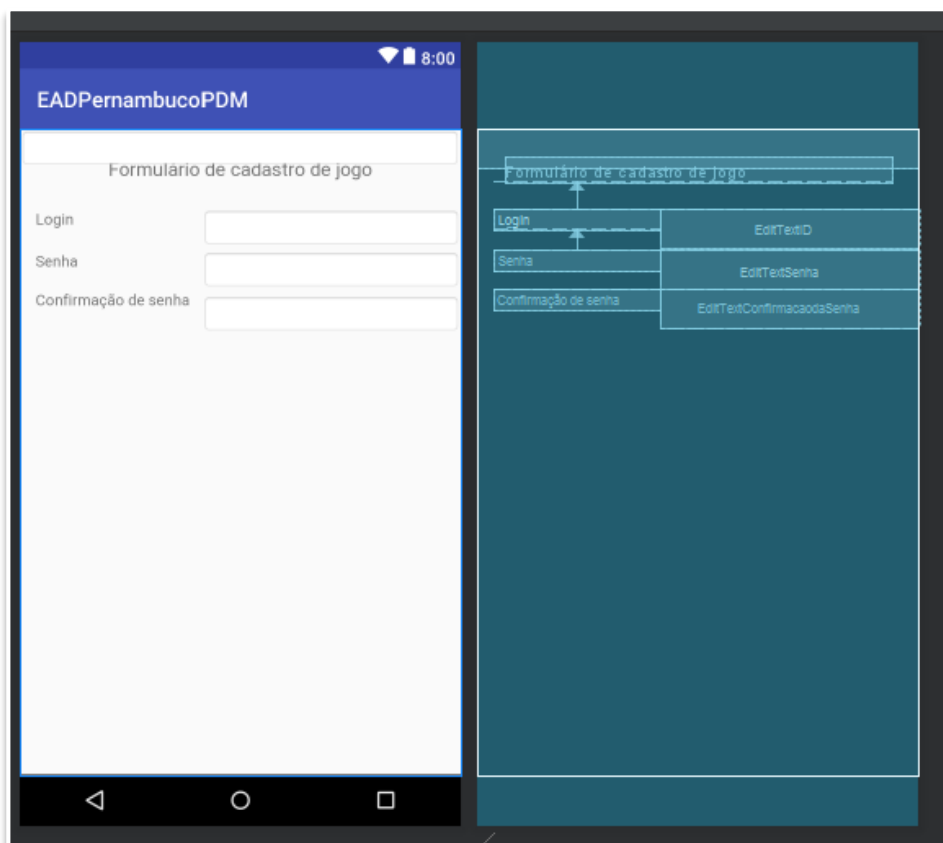


Figura 4.8 – Tela do RelativeLayout

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o *RelativeLayout*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com um campo de texto na parte superior com a descrição “formulário de cadastro de jogo”, em baixo um campo de texto login e um campo para preencher o login, em seguida o campo de texto senha e um campo para preencher a senha, por fim o campo com o texto confirmação da senha, a tela do lado direito possui um quadrado na parte superior com o nome “formulário de cadastro de jogo”, em baixo dois quadrados com login e o campo para editar o login, em seguida dois quadrados com o texto senha e o campo para editar a senha, por fim um quadrado com o texto de confirmação da senha

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- Recursos Avançados da classe *ViewGroup*
- O layout *FrameLayout*
- O layout *TableLayout*
- O layout *RelativeLayout*



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 4?
Acesse o AVA e assista a videoaula da competência 4.



Agora, acesse o AVA e responda as questões da competência 4.



Ficou com alguma dúvida na competência 4? Acesse o **Fórum - “Competência4”**
para
saná-las com os professores e discutir com seus colegas sobre os assuntos
estudados.

Nosso próximo passo é seguir para competência 5. Na competência 5 você vai aprender sobre conceitos da classe *Widget* gerada pela classe *View* e sobre os componentes *TextView*, *EditText* e *ImageView*. Vamos juntos?



5. Competência 05 | Conhecer os conceitos de interface gráfica – view

Caro estudante, nesta competência você vai aprender os conceitos de interface gráfica utilizando a classe *widget* que herda diretamente da classe *view*. A classe *widget* possui os elementos *TextView* e *EditText* relacionados para texto. Além disso, será mostrado o componente *ImageView* relacionado a imagem.

Vamos entender sobre *view*!

5.1 Criando um novo projeto com a classe Widget

Na disciplina Introdução a programação para dispositivos móveis você aprendeu a criar um novo projeto no *Android Studio*. Você lembra? Na criação desse novo projeto é gerado o arquivo *activity_main.xml* responsável por modificar a classe *Widget*. A declaração da classe *Widget* utiliza alguns componentes nas paletas *Common*, *Text* e *Buttons*. Os principais componentes mostrados nessa paleta são *textView*, *editText*, *imageView*, *button*, *imageButton*, *checkBox* e *radioButton*. A declaração do identificador desses componentes ocorre na paleta atributos com a modificação do nome do atributo *id* (identificador). Por exemplo: o identificador do *textView* declarado na Figura 5.1, abaixo, possui o *id* representado pelo nome “identificador” e a sua declaração no arquivo XML é representado por `android:id="@+id/identificador"`. A tela com a execução do arquivo *activity_main.xml* da classe *Widgets* pode ser visualizada na Figura 5.1 abaixo:

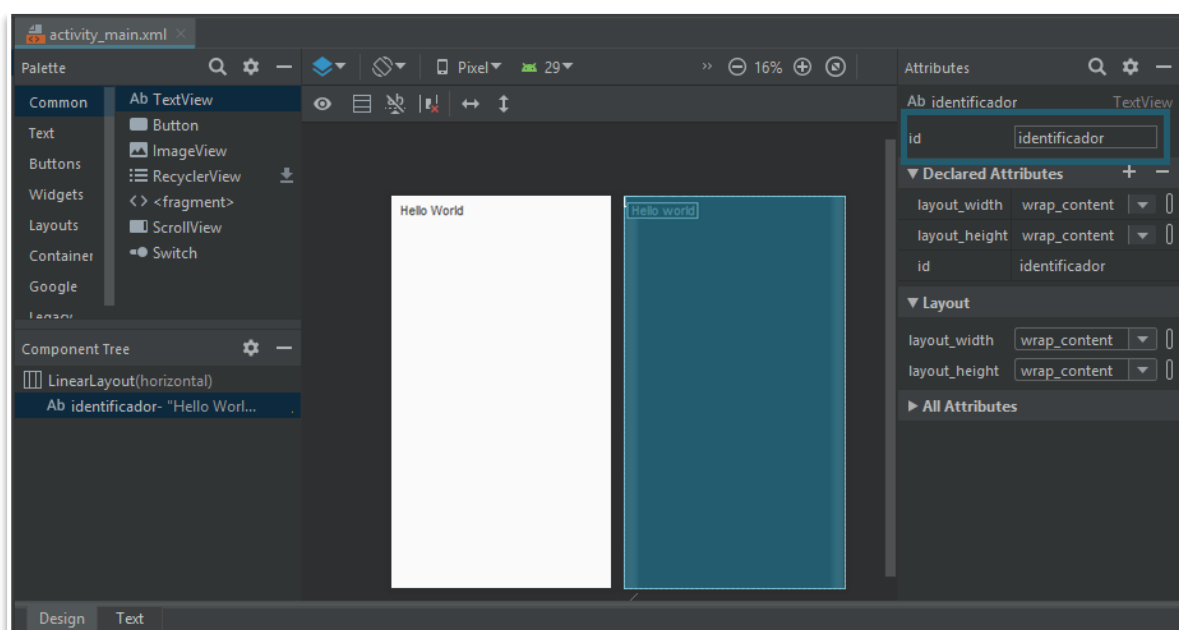




Figura 5.1 - Arquivo `activity_main.xml` gerado pelo novo projeto da classe `Widgets`

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo `activity_main.xml` possuindo do lado esquerdo a opção clicada na paleta *Common* do tipo *TextView* e o componente campo com o nome *Hello World*, e no lado direito duas telas da aplicação nas cores branco e verde com o nome centralizado *Hello World*.

Até aqui você aprendeu como criar um novo projeto com a classe *Widgets*. Na seção a seguir, você vai aprender sobre os conceitos da classe *Widgets* com a descrição dos seus principais métodos.

5.2 Conceito da classe *Widget*

A classe *widgets* é responsável por herdar da classe *View* e são utilizados para estruturar o *layout* das telas de uma aplicação *Android*. Essa subclasse de *View* implementa o método *onDraw* (*Canvas*) que é responsável por desenhar o componente na tela. Essa classe *Canvas* utilizada no *onDraw* define os métodos para desenhar texto, linhas, *bitmaps* e outras primitivas gráficas. Geralmente utilizamos alguns componentes prontos do próprio *Android*, como *TextViews*, *EditText* e *Buttons*, mas você pode criar o seu próprio componente, por meio da sua própria *View*, com a classe que herde de *View* e implementar o método *onDraw* (*Canvas*). Os métodos mais importantes da classe *View* pode ser visualizado na Tabela 5.1 abaixo:

Método	Descrição
<i>onDraw(Canvas)</i>	O método <i>onDraw</i> é responsável por desenhar algum componente de <i>layout</i> na tela do <i>Android</i> .
<i>setPadding(direita, abaixo, esquerda, acima)</i>	O método <i>setPadding</i> é responsável por informar o espaçamento à direita, abaixo, à esquerda e acima em <i>pixels</i> que devem ser colocados na tela do <i>Android</i> antes de inserir algum componente. No <i>XML</i> esse método equivale a função <i>android:padding</i> que define um único valor e esse mesmo valor ser aplicado a todos os parâmetros. Caso necessite informar um valor diferente para cada espaçamento, deve utilizar os seguintes atributos <i>android:paddingLeft</i> , <i>android:paddingTop</i> , <i>android:paddingRight</i> e <i>android:paddingBottom</i> , respectivamente.



<i>setVisibility(visibility)</i>	O método <i>setVisibility</i> pode receber três valores definidos pelas constantes <i>View.VISIBLE</i> , <i>View.INVISIBLE</i> e <i>View.GONE</i> . A constante <i>View.VISIBLE</i> é responsável pela exibição do componente na tela, já as constantes <i>View.INVISIBLE</i> e <i>View.GONE</i> são responsáveis pela não exibição do componente na tela. A diferença entre essas duas constantes é que a constante <i>View.INVISIBLE</i> não exibe o elemento, porém deixa o espaço que ele usaria reservado na tela (em branco), enquanto a constante <i>View.GONE</i> faz com que se remova tanto o elemento quanto o espaço utilizado na tela. No <i>XML</i> esse método é equivalente ao <i>android:visibility</i> que recebe os seguintes valores: <i>visible</i> , <i>invisible</i> e <i>gone</i> .
----------------------------------	---

Tabela 5.1 – Métodos da classe *View*

Fonte: o autor

Descrição da imagem: a descrição dos três métodos da classe *View*.

Nas seções a seguir, você vai aprender sobre os componentes *TextView*, *EditText* e *ImageView*. Além disso, os componentes *button*, *ImageButton*, *CheckBox* e *RadioButton* serão mostrados na próxima competência.

5.2.1 *TextView*

A classe *TextView* é considerada uma subclasse da classe *View* responsável por desenhar um determinado texto na tela das aplicações. A classe *TextView* no arquivo *XML* é representado pela seguinte sintaxe **<TextView/>**. Para escrever um texto normal na tela é utilizada a seguinte declaração `android:text="Hello World"`. Assim como a declaração do endereço de site é utilizada a seguinte declaração `android:text="https://ead.educacao.pe.gov.br/"`. A seguir será mostrado alguns exemplos para utilizar a classe *TextView* com descrição de nome, descrição de site e utilizando nomes com diversas cores.

Por exemplo, imagine que existe um determinado *layout* de aplicativo possuindo textos com a descrição de nome, descrição de site e os nomes com diversas cores. Diante desse exemplo, você vai aprender como utilizar a classe *TextView* para editar formato de texto. O arquivo



activity_main.xml mostrado na Figura 5.2, abaixo, possui o layout *LinearLayout* e classe *TextView*. Esse código do *activity_main.xml* representando o layout *LinearLayout* e a classe *TextView* com descrição de nome, descrição de site e utilizando nomes com diversas cores pode ser visualizado na Figura 5.2 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/text01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
    />

    <TextView
        android:id="@+id/text02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:autoLink="web"
        android:text="https://ead.educacao.pe.gov.br/"
    />

    <TextView
        android:id="@+id/text03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#0000ff"
        android:textColor="#ffffff"
        android:text="Texto branco (text_color) e fundo azul (background)"
    />
</LinearLayout>
```

Figura 5.2 – Arquivo *activity_main.xml* do *TextView*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *TextView* possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e as três tags de *TextView* representadas pelas propriedades de texto normal com o nome *Hello World!*, texto com o site <https://ead.educacao.pe.gov.br/> e com o texto personalizado por cores com o nome Texto branco (*text_color*) e fundo azul (*background*).

Depois da execução do arquivo *activity_main.xml* com *TextView* com descrição de nome, descrição de site da Figura 5.2, acima, e gerado o designer da tela no *Android Studio* visualizado na Figura 5.3 abaixo:

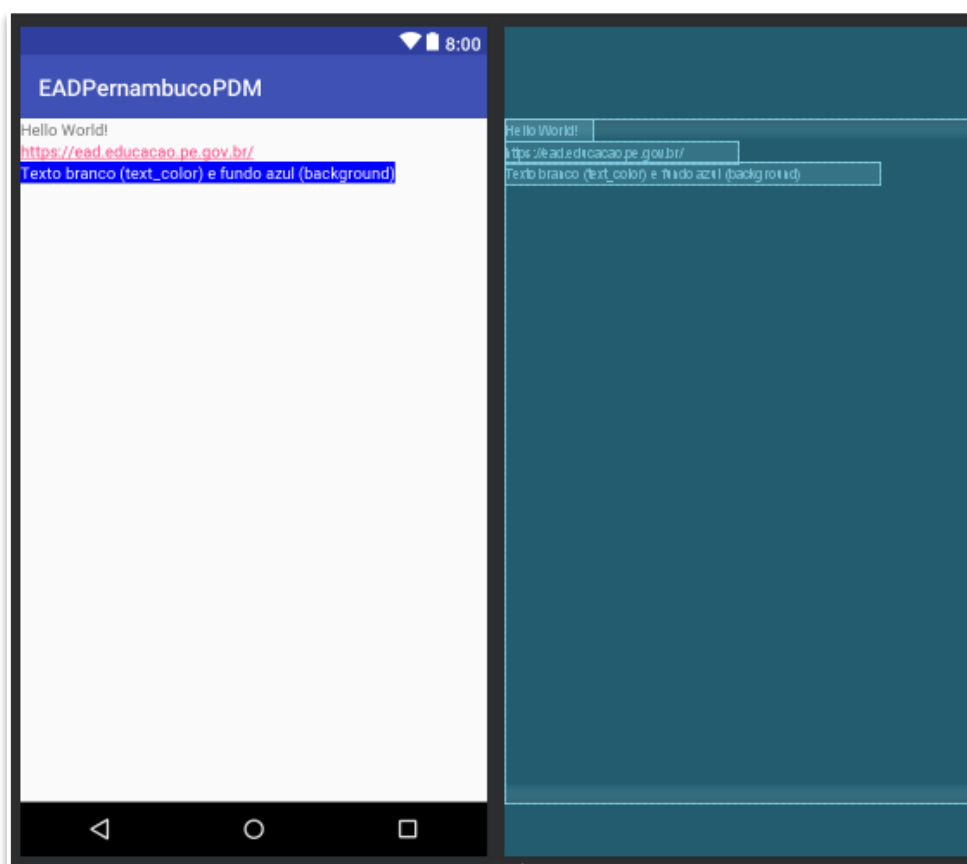


Figura 5.3 – Tela do TextView

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do Android Studio representando o *TextView*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com a descrição do nome *hello world* com de texto preta, a descrição do site <https://ead.educacao.pe.gov.br/> com cor vermelha e o texto *Texto branco (text_color)* e a descrição “*Texto branco (text_color)* e fundo azul (*background*)” com cor de texto branca e negrito azul, a tela do lado direito possui um quadrado com fundo verde representado a tela e três retângulos representando as três descrições.

O *TextView* configura no arquivo *MainActivity.java* da Figura 5.4, abaixo, esse *TextView* possui um ID (identificador) definido pelo arquivo *XML* para recuperar um determinado componente no código Java, dessa maneira é possível executar o identificador *R.id.text01* utilizando o método *findViewById()*, em seguida é utilizado o método *setText()* para mostrar o texto Hello World!. O código do arquivo *MainActivity.java* do *Android Studio* representando o *TextView* pode ser visualizado na Figura 5.4 abaixo:

```
package com.example.eadpernambucopdm;  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;
```




```
import android.widget.TextView

public class MainActivity extends AppCompatActivity {

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

        TextView textView = (TextView)findViewById(R.id.text01);
        textView.setText("Hello World!");
    }
}
```

Figura 5.4 – Arquivo da classe MainActivity.java do TextView

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *TextView* no arquivo *activity_main.xml*.

5.2.2 EditText

A classe *EditText* é utilizada quando se pretende criar um campo, onde um determinado usuário deve digitar uma informação do texto. A classe *EditText* no arquivo *XML* é representada pela seguinte sintaxe **<EditText/>**. A propriedade *android:inputType* é responsável por indicar o tipo de entrada mais adequado para a digitação dos campos. Para escrever um texto normal é utilizada a seguinte declaração *android:inputType = "text"*. Os principais valores utilizando o *EditText* pode ser visualizado na Tabela 5.2 abaixo:

Propriedade	Descrição
<i>text</i>	Responsável pela descrição de texto normal
<i>textEmailAddress</i>	Responsável pela descrição do endereço de e-mail
<i>textUri</i>	Responsável pela descrição de endereços com recursos com <i>URL</i> .
<i>textPassword</i>	Responsável pela descrição de campo com senha.
<i>number</i>	Responsável pela descrição de apenas números sem sinal.
<i>NumberSigned</i>	Responsável pela descrição de apenas números com sinal.
<i>phone</i>	Responsável pela descrição de número de telefone.



Tabela 5.2 – Representação das propriedades do EditText

Fonte: o autor

Descrição da imagem: a descrição das seis propriedades do *EditText*.

Essa classe *EditText* é utilizada também no processo de criação de formulários. A seguir será mostrado alguns exemplos para utilizar a classe *EditText* com os componentes *text*, *textEmailAddress*, *textUri*, *textPassword*, *number*, *NumberSigned* e *phone*.

Por exemplo, imagine que existe um determinado *layout* com campos de descrição do jogo, e-mail do jogo, site do jogo, senha do jogo, número de acesso do jogo e telefone de contato do jogo. Diante desse exemplo, você vai aprender como utilizar a classe *EditText* na visualização de campos de edição utilizado nesse jogo. O arquivo *activity_main.xml* mostrado na Figura 5.5, abaixo, possui o *layout LinearLayout* e as classes *TextView* e *EditText*. Esse código do *activity_main.xml* representando o *layout LinearLayout* e as classes *TextView* e *EditText* pode ser visualizado na Figura 5.5 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/text01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="Descrição do jogo:"
    />

    <EditText
        android:id="@+id/edit01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:inputType="text"
    />
</LinearLayout>
```

Figura 5.5 – Arquivo *activity_main.xml* do *EditText*

Fonte: O autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *EditText* possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e a *tag TextView* representando por texto a mensagem “descrição do jogo” e a *tag EditText* representando como tipo de entrada um texto.



Depois da execução do arquivo *activity_main.xml* com *EditText* da Figura 5.5, acima, e gerado o designer da tela no *Android Studio* visualizado na Figura 5.6 abaixo:

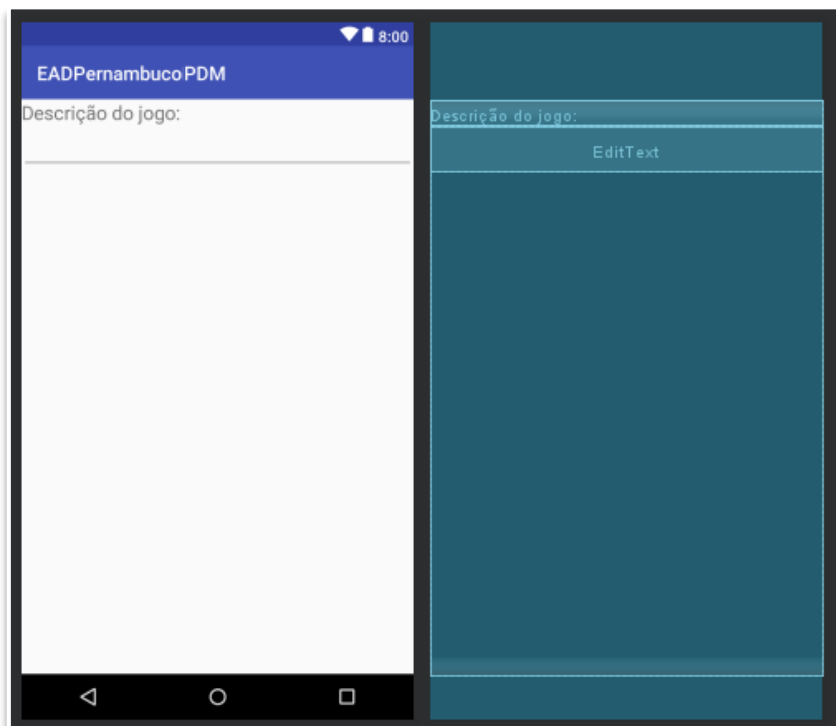


Figura 5.6 – Tela do EditText

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do Android Studio representando o *EditText*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com a descrição do texto com a mensagem “descrição do jogo” e uma linha representando o tipo de entrada com texto, a tela do lado direito possui um quadrado com fundo verde representado a tela e dois retângulos representando a descrição do texto e o campo de entrada de texto representada por *EditText*.

O *EditText* configura no arquivo *MainActivity.java* da Figura 5.7, abaixo, esse *EditText* possui um ID (identificador) definido pelo arquivo XML para recuperar um determinado componente no código Java, dessa maneira é possível executar o identificador *R.id.edit01* utilizando o método *findViewById()*, em seguida é utilizado o método *setText()* para mostrar o texto o jogo possui três telas com acesso ao jogo do mario, Lol e barkbringe!". O código do arquivo *MainActivity.java* do *Android Studio* representando o *EditText* pode ser visualizado na Figura 5.7 abaixo:

```
package com.example.eadpernambucopdm;  
import androidx.appcompat.app.AppCompatActivity;
```



```
import android.os.Bundle;
import android.widget.EditText

public class MainActivity extends AppCompatActivity {

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

        EditText editText = (EditText) findViewById((R.id.edit01)) ;
        editText.setText("O jogo possui três telas com acesso ao jogo do
mario, Lol e barkbringe!");

    }
}
```

Figura 5.7 – Arquivo da classe MainActivity.java do EditText

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *EditText* no arquivo *activity_main.xml*.

O código XML do *Android Studio* representando o layout *LinearLayout* e a classe *TextView* com *inputType textEmailAddress* pode ser visualizado na Figura 5.8 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="Email do jogo:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:inputType="textEmailAddress"
    />

</LinearLayout>
```

Figura 5.8 - Arquivo *activity_main.xml* do EditText com *textEmailAddress*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a



representação do *EditText* com endereço de e-mail possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e a *tag TextView* representando por texto a mensagem “Email do jogo” e a *tag EditText* representando como tipo de entrada uma endereço de e-mail.

O código XML do *Android Studio* representando o *layout LinearLayout* e a classe *TextView* com *inputType textUri* pode ser visualizado na Figura 5.9 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="site do jogo:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:inputType="textUri"
    />

</LinearLayout>
```

Figura 5.9 - Arquivo *activity_main.xml* do *EditText* com *textUri*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *EditText* com endereço de site possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e a *tag TextView* representando por texto a mensagem “site do jogo” e a *tag EditText* representando como tipo de entrada uma endereço de site.

O código XML do *Android Studio* representando o *layout LinearLayout* e a classe *TextView* com *inputType textPassword* pode ser visualizado na Figura 5.10 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="senha do jogo:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:inputType="textPassword"
    />

</LinearLayout>
```



```
        android:textSize="20dp" android:text="Senha do jogo:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
    />

</LinearLayout>
```

Figura 5.10 – Arquivo activity_main.xml do EditText com textPassword

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação da senha de acesso possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e a tag *TextView* representando por texto a mensagem “senha do jogo” e a tag *EditText* representando como tipo de entrada uma senha de acesso.

O código XML do *Android Studio* representando o layout *LinearLayout* e a classe *TextView* com *inputType number* pode ser visualizado na Figura 5.11 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:text="Número de acesso do jogo:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
    />

</LinearLayout>
```

Figura 5.11 - Arquivo activity_main.xml do EditText com number

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do número possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e a tag *TextView* representando por texto a mensagem “Número de acesso do jogo” e a tag *EditText* representando como tipo de entrada uma senha de acesso.



O código XML do *Android Studio* representando o *layout LinearLayout* e a classe *TextView* com *inputType NumberSigned* pode ser visualizado na Figura 5.12 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="Pontuação do jogo:"
    />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberSigned"
    />
</LinearLayout>
```

Figura 5.12- Arquivo activity main.xml do EditText com NumberSigned

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do número com ponto possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e a *tag TextView* representando por texto a mensagem “Pontuação do jogo” e a *tag EditText* representando como tipo de entrada uma pontuação do jogo.

O código XML do *Android Studio* representando o *layout LinearLayout* e a classe *TextView* com *inputType phone* pode ser visualizado na Figura 5.13 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp" android:text="telefone de contato do jogo:"
    />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="phone"
    />
</LinearLayout>
```

Figura 5.13 - Arquivo activity main.xml do EditText com phone



Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do telefone possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e a tag *TextView* representando por texto a mensagem “Telefone de contato do jogo” e a tag *EditText* representando como tipo de entrada um numeração com telefone.

5.2.3 Imageview

A classe *Imageview* é utilizada quando se pretende visualizar uma determinada imagem. A classe *Imageview* é representada pela seguinte sintaxe **<ImageView/>**. Para declarar um identificador com o nome da imagem é utilizada a seguinte declaração `android:id="@+id/imageView"`; para a declaração da altura da imagem é utilizada a seguinte declaração `android:layout_width="match_parent"` e da largura da imagem a seguinte declaração `android:layout_height="wrap_content"`. Cada imagem possui uma respectiva pasta com o caminho da imagem representada pela seguinte declaração `android:src="@drawable/ic_launcher"`. Essa imagem pode ser representada por imagem normal (com apenas a figura), imagem da internet através de URL e imagem por representação de catálogo.

Por exemplo, imagine que existe um determinado *layout* com a visualização das imagens dos jogos Mario e Lol. Diante desse exemplo, você vai aprender como utilizar a classe *Imageview* na visualização de imagens de jogos. O arquivo *activity_main.xml* mostrado na Figura 5.14, abaixo, possui o layout *LinearLayout* e a classe *ImageView*. Esse código do *activity_main.xml* representando o layout *LinearLayout* e a classe *Imageview* pode ser visualizado na Figura 5.14 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imgBtn01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_mario" />
    <ImageView
        android:id="@+id/imgBtn02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher_lol" />
</LinearLayout>
```




Figura 5.14 – Arquivo *activity_main.xml* do *ImageView*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação da *ImageView* possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e as duas *tags ImageView* representada pelas imagens do jogo do Mario e o jogo do Lol.

Depois da execução do arquivo *activity_main.xml* com *ImageView* da Figura 5.14, acima, e gerado o designer da tela no *Android Studio* visualizado na Figura 5.15 abaixo:

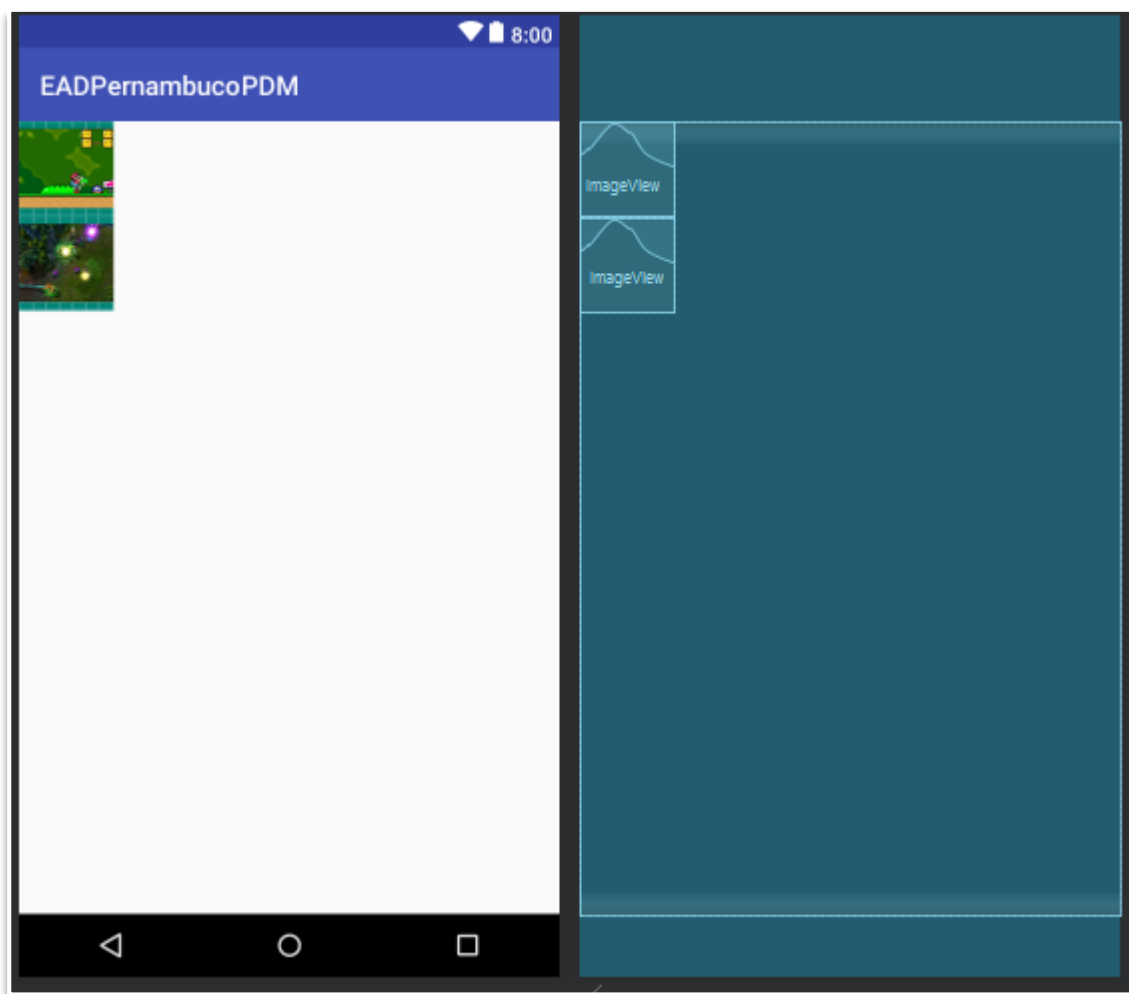


Figura 5.15 – Tela da *Imageview*

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando a *Imageview*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome *EADPernambucoPDM*, na barra inferior três botões do celular e na parte do meio possui uma tela branca com as imagens dos jogos Mario e Lol, a tela do lado direito possui um quadrado com fundo verde representado a tela e dois retângulos com o nome *ImageView* representando as duas imagens.

A *ImageView* configura no arquivo *MainActivity.java* da Figura 5.16, abaixo, essa *ImageView* possui um ID (identificador) definido pelo arquivo *XML* para recuperar um determinado



componente no código Java, dessa maneira é possível executar os identificadores *R.id.imgBtn01* e *R.id.imgBtn02* utilizando o método *findViewById()*, em seguida é utilizado o método *setImageResource()* para acessar as imagens do mario e lol na pasta do Android Studio. O código do arquivo *MainActivity.java* do *Android Studio* representando a *ImageView* pode ser visualizado na Figura 5.16 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

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

        ImageView imageView1 = (ImageView) findViewById(R.id.imgBtn01);
        imageView1.setImageResource(R.mipmap.mario);
        ImageView imageView2 = (ImageView) findViewById(R.id.imgBtn02);
        imageView2.setImageResource(R.mipmap.lol);

    }
}
```

Figura 5.16 – Arquivo da classe MainActivity.java do ImageView

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *ImageView* no arquivo *activity_main.xml*.

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- A classe *Widgets*
- O componente *TextView*
- O componente *EditText*
- O componente *ImageView*



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 5?
Acesse o AVA e assista a videoaula da competência 5.



Agora, acesse o AVA e responda as questões da competência 5;



Ficou com alguma dúvida na competência 5? Acesse o **Fórum - "Competência5"**
para
saná-las com os professores e discutir com seus colegas sobre os assuntos
estudados.

Nosso próximo passo é seguir para competência 6. Na competência 6 você vai aprender sobre conceitos avançados da classe *Widget* gerada pela classe *View* e sobre os componentes *Button*, *ImageButton*, *CheckBox*, *RadioButton* e *Spinner*. Vamos juntos?



6.Competência 06 | Utilizar recursos avançados de view

Caro estudante, nesta competência você vai aprender os conceitos avançados da classe *widget* que herda diretamente da classe *view*. A classe *widget* possui outros elementos avançados utilizando botão, como *ImageButto* e o *Button*, além dos componentes *Checkbox*, *RadioButton* e *Spinner*.

Vamos aprofundar mais sobre os avanços da classe *widget*!

6.1 Avanços da Widget

A classe *widget* possui diversos outros componentes que herdam diretamente da classe *view*. Os componentes que utilizam botões como o *Button* e o *ImageButton*, além dos componentes com *CheckBox*, *RadioButtone* e *Spinner*. O *CheckBox* é responsável por desenhar uma caixa de seleção ou caixa de verificação utilizando elemento da interface gráfica, *RadioButton* responsável por selecionar uma única opção de uma determinada lista e o *Spinner* é responsável por desenhar na tela um *combobox*, ou seja, um conjunto de opções. Nas seções a seguir você vai aprender sobre os componentes *Button*, *ImageButton*, *CheckBox*, *RadioButton* e *Spinner*.

6.1.1 Button

A classe *Button* é responsável pela criação de novos botões na tela das aplicações. A classe *Button* no arquivo *XML* é representado pela seguinte sintaxe **<Button />**. Para declarar um botão é necessário a identificação do botão com a seguinte sintaxe `android:id="@+id/bt01"`. A declaração de altura e largura são representadas pelas seguintes sintaxes `android:layout_width="match_parent"` e `android:layout_height="wrap_content"` e para declarar o nome do texto do botão é utilizada a seguinte sintaxe `android:text="Botão 01"`.

Por exemplo, imagine que existe um determinado *layout* de aplicativo possuindo botão de acesso dos jogos de Mario e Lol. Diante desse exemplo, você vai aprender como utilizar a classe *Button* para criar um novo botão. O arquivo *activity_main.xml* mostrado na Figura 6.1, abaixo, possui o layout *LinearLayout* e duas classes *Button* com os botões “botão de acesso do Mario” e “botão de acesso do Lol”. Esse código do *activity_main.xml* representando a classe *Button* pode ser visualizado na Figura 6.1 abaixo:



```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btn01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botão de acesso do Mario"/>

    <Button
        android:id="@+id/btn02"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botão de acesso do Lol"/>

</LinearLayout>
```

Figura 6.1 – Arquivo activity main.xml do Button

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *Button* possuindo a representação da *tag LinearLayout* com os componentes altura, largura e orientação vertical, e as duas tags *Button* representada pelos botões com o nome “Botão de acesso do Mario” e “Botão de acesso do Lol”.

Depois da execução do arquivo *activity_main.xml* com *Button* da Figura 6.1, acima, e gerado o *designer* da tela no *Android Studio* visualizado na Figura 6.2 abaixo:

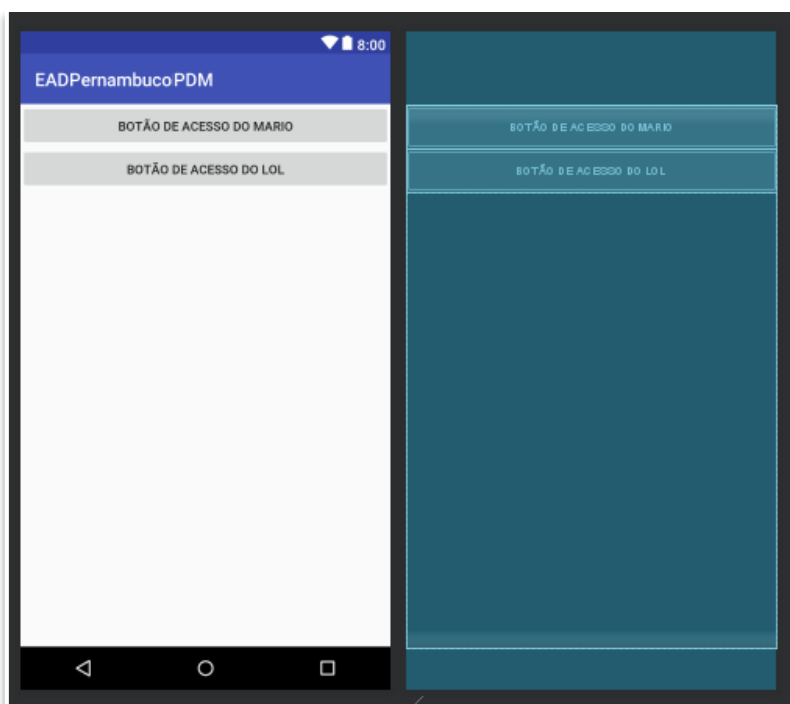




Figura 6.2 – Tela do Button

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o componente *button*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com dois botões na cor cinza com os nomes “BOTÃO DE ACESSO DO MARIO” e “BOTÃO DE ACESSO DE LOL”, a tela do lado direito possui um quadrado com fundo verde representado a tela e dois retângulos representando os botões com os nomes “BOTÃO DE ACESSO DO MARIO” e “BOTÃO DE ACESSO DE LOL”.

O *Button* é executado no arquivo *MainActivity.java* da Figura 6.3, abaixo, possuindo um ID (identificador) definido pelo arquivo *XML* para recuperar um determinado componente no código Java, dessa maneira é possível executar os identificadores *R.id.btn01* e *R.id.btn02* utilizando o método *findViewById()*, em seguida é utilizado o método *setText()* para mostrar o texto do botão 01 “Botão de acesso do Mario e do botão 02 “Botão de acesso do Lol”. O código do arquivo *MainActivity.java* do *Android Studio* representando o *Button* pode ser visualizado na Figura 6.3 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

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

        Button botao1 = (Button) findViewById(R.id.btn01);
        botao1.setText("Botão de acesso do Mario");
        Button botao2 = (Button) findViewById(R.id.btn02);
        botao2.setText("Botão de acesso do Lol");

    }
}
```

Figura 6.3 – Arquivo da classe MainActivity.java do Button

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *Button* no arquivo *activity_main.xml*.



6.1.2 ImageButton

A classe *ImageButton* é responsável pela criação de novos botões representada por imagem na tela das aplicações. A classe *Button* no arquivo *XML* é representado pela seguinte sintaxe `<ImageButton/>`. Para declarar um botão com imagem é necessário a identificação do botão com a seguinte sintaxe `android:id="@+id/btImg01"`. A declaração de altura e largura são representadas pelas seguintes sintaxes `android:layout_width="match_parent"` e `android:layout_height="wrap_content"` e para declarar o caminho da imagem do botão é utilizado a seguinte sintaxe `android:src="@drawable/android_mini2"`.

Por exemplo, imagine que existe um determinado *layout* de aplicativo possuindo botão com imagem dos jogos de Mario e Lol. Diante desse exemplo, você vai aprender como utilizar a classe *ImageButton* para criar o botão com imagem. O arquivo *activity_main.xml* mostrado na Figura 6.4, abaixo, possui o layout *LinearLayout* e duas classes *ImageButton* com os botões com imagens dos jogos de Mario e Lol. Esse código do *activity_main.xml* representando a classe *ImageButton* pode ser visualizado na Figura 6.4 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageButton
        android:id="@+id/imgBtn01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@mipmap/mario"
        android:contentDescription="todo" />

    <ImageButton
        android:id="@+id/imgBtn02"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@mipmap/lol"
        android:contentDescription="todo" />

</LinearLayout>
```

Figura 6.4 - Arquivo *activity_main.xml* do *ImageButton*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *ImageButton* possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e as duas tags *ImageButton* representada pelos botões com imagens dos jogos Mario e Lol.



Depois da execução do arquivo *activity_main.xml* com *ImageButton* da Figura 6.4, acima, e gerado o *designer* da tela no *Android Studio* visualizado na Figura 6.5 abaixo:

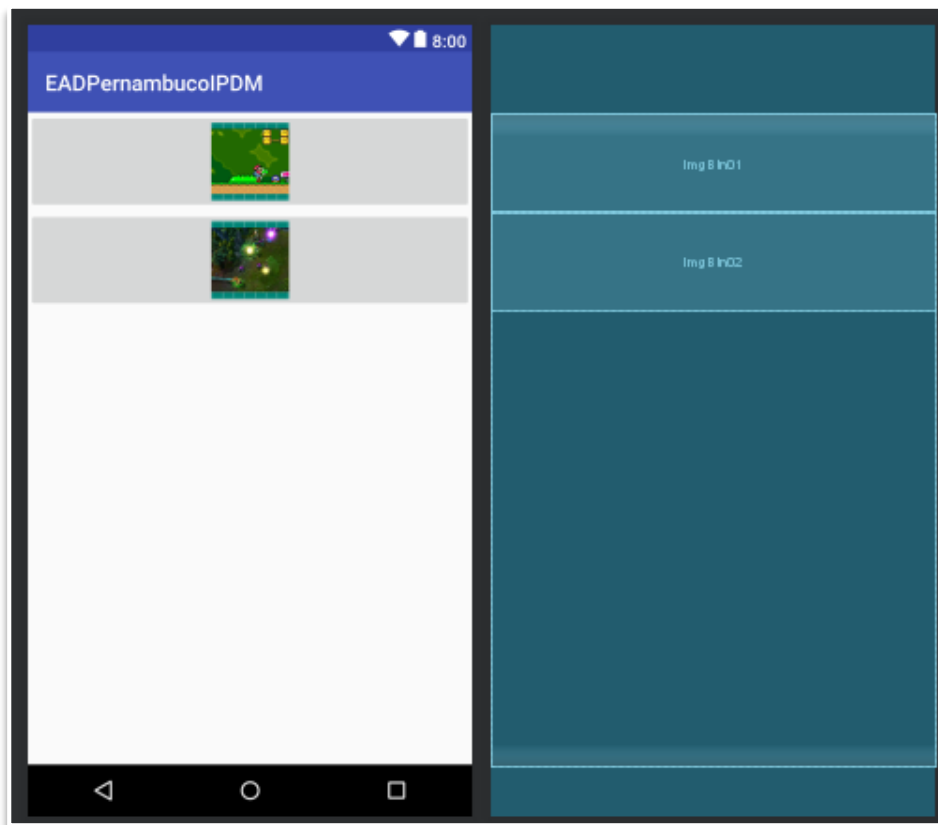


Figura 6.5– Tela do ImageButton

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o componente *ImageButton*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoIPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com dois botões de imagens dos jogos Mario e Lol”, a tela do lado direito possui um quadrado com fundo verde representando a tela e dois retângulos representando os botões de imagem com o nome das imagens dos jogos Mario e Lol.

A *ImageButton* é executado no arquivo *MainActivity.java* da Figura 6.6, abaixo, possuindo um ID (identificador) definido pelo arquivo *XML* para recuperar um determinado componente no código Java, dessa maneira é possível executar os identificadores *R.id.imgBtn01* e *R.id.imgBtn02* utilizando o método *findViewById()*, em seguida é utilizado o método *setImageResource()* para acessar as imagens do mario e lol na pasta do Android Studio. O código do arquivo *MainActivity.java* do *Android Studio* representando a *ImageButton* pode ser visualizado na Figura 6.6 abaixo:



```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ImageButton;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

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

        ImageButton imageButton1 = (ImageButton)
findViewById(R.id.imgBtn01);
        imageButton1.setImageResource(R.mipmap.mario);
        ImageView imageView1 = (ImageView) findViewById(R.id.imgBtn01);
        ImageButton imageButton2 = (ImageButton)
findViewById(R.id.imgBtn02);
        imageButton2.setImageResource(R.mipmap.lol);

    }
}
```

Figura 6.6 – Arquivo da classe MainActivity.java do ImageButton

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *ImageButton* no arquivo *activity_main.xml*.

6.1.3 CheckBox

O *Checkbox* é um componente do *Android* responsável pelo estado de marcado e desmarcado. Geralmente, esse componente é utilizado quando se pretende fazer uma escolha mútua, no qual não seja mutuamente exclusiva. O componente *Checkbox* no arquivo *XML* é representado pela seguinte sintaxe **<CheckBox />**. Para declarar um componente *Checkbox* é necessário a declaração de altura e largura representadas pelas seguintes sintaxes *android:layout_width = "match_parent"* e *android:layout_height="wrap_content"* e texto com a opção do *Checkbox* é utilizado a seguinte sintaxe *android:text="opcaocheckbox"*.

Por exemplo, imagine que existe um determinado formulário que pergunta quais tipos de jogos você domina ou conhece e existem várias opções: mortal kombat, kung fu for, MMA Fighter e Gangsters, caso você conheça mais de um jogo de luta é ideal utilizar o componente *Checkbox* para representar essas várias opções. Diante desse exemplo, você vai aprender como utilizar o componente *Checkbox* para escolher uma opção desse jogo.



O arquivo *activity_main.xml* mostrado na Figura 6.7, abaixo, possui o layout *LinearLayout*, a classe *TextView* com a descrição "Assinale os jogos que você domina ou conhece:" e quatro componentes *Checkbox* com as opções dos jogos mortal kombat, kung fu for, MMA e Gangsters. Esse código do *activity_main.xml* representando o *Checkbox* pode ser visualizado na Figura 6.7 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Assinale os jogos que você domina ou conhece:"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="mortal kombat"
            />
        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="kung fu for"
            />
        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="MMA Fighter "
            />
        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="Gangsters"
            />

    </LinearLayout>
</LinearLayout>
```

Figura 6.7 - Arquivo *activity_main.xml* do *Checkbox*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *Checkbox* possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, e a tag *TextView* com a descrição "Assinale os jogos que você domina ou conhece:" e quatro tags *Checkbox* com as opções Mortal Kombat, kung fu for, MMA Fighter e Gangsters.

O *Checkbox* é executado no arquivo *MainActivity.java* da Figura 6.8, abaixo, possuindo um ID (identificador) definido pelo arquivo *XML* para recuperar um determinado componente no



código Java, dessa maneira é possível verificar caso o *checkbox* está marcado ou não está marcado. Isso é realizado através do método *isChecked()*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *Checkbox* pode ser visualizado na Figura 6.8 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.CheckBox;

public class MainActivity extends AppCompatActivity {

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

        CheckBox chkBox = (CheckBox) findViewById(R.id.checkboxJava);
        boolean checkboxSelecionado = chkBox.isChecked();
    }
}
```

Figura 6.8 – Arquivo da classe *MainActivity.java* do *Checkbox*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* possuindo a execução da tela do *CheckBox* no arquivo *activity_main.xml*.

Caso o *Checkbox* esteja marcado com a seleção dos jogos *Mortal kombat* e *MMA* indica que os valores de *Mortal kombat* e *MMA* estão selecionados. A tela do *Android Studio* representando o componente de *Checkbox* para identificar mais de um jogo de luta pode ser visualizado na Figura 6.9 abaixo:

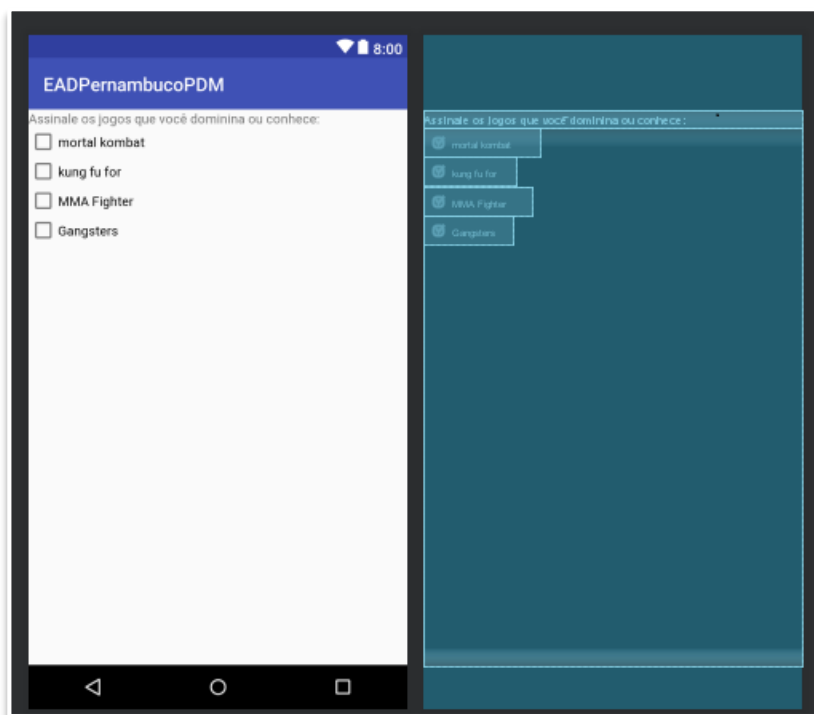


Figura 6.9 – Tela do checkbox

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o componente *ImageButton*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com a descrição do nome na cor preta "Assinale os jogos que você domina ou conhece:", e quatro *checkbox* com nos nomes Mortal Kombat, kunf fu for, MMA Fighter e Gangsters, a tela do lado direito possui um quadrado com fundo verde representado a tela e cinco quadrados retângulos representando a descrição do nome e quatro *checkbox*.

6.1.4 RadioButton

O *RadioButton* é um componente do *Android* responsável por selecionar uma única opção de uma determinada lista. Geralmente, esse componente é utilizado quando o usuário pretende fazer uma única escolha, ou seja, as opções são mutuamente exclusivas, não sendo assim possível ter mais de uma resposta para a mesma pergunta. O componente *RadioButton* no arquivo *XML* é representado pelas seguintes classes `<RadioGroup />` e `<RadioButton />`. A classe *RadioGroup* é responsável por definir um grupo que contém uma lista de opções, no qual cada opção é representada por um respectivo *RadioButton*.

Para declarar um componente *RadioGroup* é necessário realizar a declaração de altura e largura com as seguintes sintaxes `android:layout_width="match_parent"` e `android:layout_height="wrap_content"`, assim como a orientação do *layout* com a seguinte sintaxe



`android:orientation="horizontal"`. Com relação ao componente *RadioButton* é necessário realizar a declaração do ID (identificador) do componente com a seguinte sintaxe `android:id="@+id/radioM"`. Já a declaração de altura e largura devem ser realizadas com as seguintes sintaxes `android:layout_width="match_parent"` e `android:layout_height="wrap_content"` e o texto com a opção de sexo com a seguinte sintaxe `android:text="Feminino"`.

Por exemplo, imagine que existe um determinado formulário de jogo que pergunta o sexo dos seus jogadores, este pode ser masculino ou feminino, ou quando se pretende fazer uma pergunta de execução do jogo e a resposta é sim ou não. Diante desse exemplo, você vai aprender como utilizar o componente *RadioButton* para escolher o sexo dos jogadores desse jogo.

O arquivo *activity_main.xml* mostrado na Figura 6.10, abaixo, possui o layout *LinearLayout*, a classe *TextView* com a descrição "Sexo:", o componente *RadioGroup* e dois componentes *RadioButton* com as opções Masculino e Feminina. Esse código do *activity_main.xml* representando o layout *LinearLayout* e os componentes *RadioGroup* e *RadioButton* pode ser visualizado na Figura 6.10 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/lblSexo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Sexo:"
    />

    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <RadioButton
            android:id="@+id/radioM"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Masculino"/>

        <RadioButton
            android:id="@+id/radioF"
            android:layout_width="wrap_content"
```




```
        android:layout_height="wrap_content"
        android:text="Feminina"
    />
</RadioGroup>
</LinearLayout>
```

Figura 6.10 - Arquivo activity_main.xml do RadioButton

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo activity_main.xml com a representação do RadioButton possuindo a representação da tag LinearLayout com os componentes altura, largura e orientação vertical, e a tag RadioGroup com os componentes altura, largura e orientação vertical e duas tags do RadioButton representando as opções masculino e feminina.

O *RadioGroup* é executado no arquivo *MainActivity.java* da Figura 6.11, abaixo, possuindo ID (identificador) definido pelo arquivo *activity_main.xml* para identificar um componente no código Java, dessa maneira quando se escolhe um *RadioButton* o outro vai ser automaticamente desmarcado. Isso é verificado através do método do *RadioGroup* chamado *getCheckedRadioButtonId()*. O código do arquivo *MainActivity.java* do *Android Studio* representando o *RadioButton* pode ser visualizado na Figura 6.11 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.RadioGroup;

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

        RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
        boolean radioM_Checked = R.id.radioM ==
radioGroup.getCheckedRadioButtonId();
    }
}
```

Figura 6.11 - Arquivo da classe MainActivity.java do RadioButton

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* do *RadioButton* possuindo a execução da tela do *RadioButton* no arquivo *activity_main.xml*.

Caso o valor de *RadioButton* masculino seja considerado verdadeiro indica que o *RadioButton* com o valor Masculino está selecionado, da mesma maneira pode-se fazer para o

RadioButton Feminino. A tela do *Android Studio* representando o componente *RadioButton* com o sexo dos jogadores pode ser visualizado na Figura 6.12 abaixo:

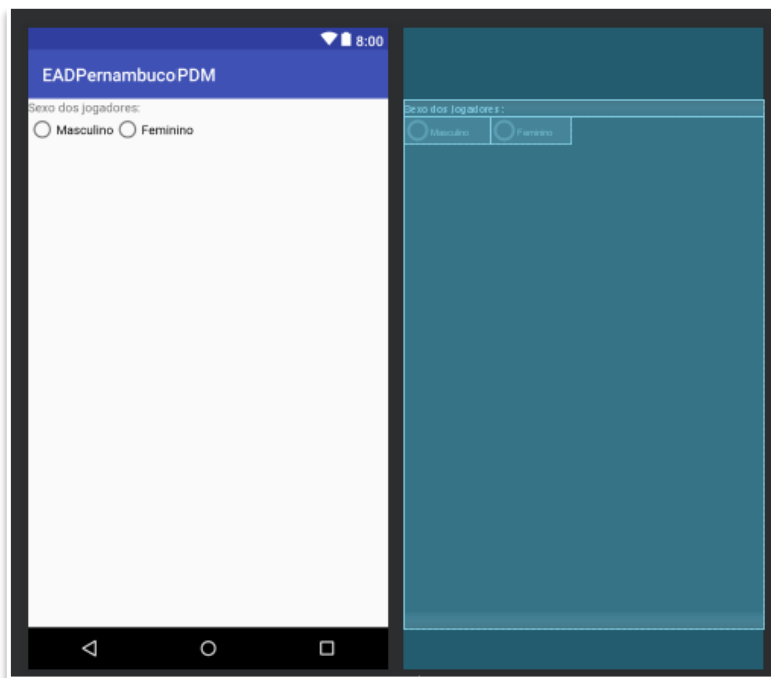


Figura 6.12 – Tela do *RadioButton*

Fonte: o autor.

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o componente *RadioButton*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três botões do celular e na parte do meio possui uma tela branca com a descrição do nome na cor preta "Sexo dos jogadores:", e dois *RadioButton* representando as opções Masculino e Feminino, a tela do lado direito possui um quadrado com fundo verde representado a tela, uma linha representando o texto "Sexo dos jogadores:" e dois quadrado representando os dois *RadioButton*.

6.1.5 Spinner

O *Spinner* é um componente do Android responsável por selecionar um valor a partir de um conjunto de dados de um *combobox*. No estado padrão, esse *Spinner* mostra o seu valor atualmente selecionado nesse *combobox*. A seleção do *Spinner* exibe um menu com todos os outros valores disponíveis, a partir do qual o usuário pode selecionar um novo.

O componente *Spinner* no arquivo XML é representado pela seguinte classe `<Spinner/>`. No *Spinner* é necessário declarar a altura e largura utilizando as seguintes sintaxes `android:layout_width="match_parent"` e `android:layout_height="wrap_content"`. Além disso, o identificador do *combobox* possui a seguinte declaração `android:id="@+id/combobox"`.



Por exemplo, imagine que existe uma determinada lista de jogos e ao selecionar o jogo a aplicação irá exibir a imagem do jogo. Diante desse exemplo, você vai aprender como utilizar o componente *Spinner* para selecionar um jogo e mostrar a imagem desse jogo.

O arquivo *activity_main.xml* mostrado na Figura 6.13, abaixo, possui o layout *LinearLayout*, a classe *TextView* com a descrição "Selecione um jogo:", o componente *Spinner* com a opção de *combobox* dos jogos Barkbridge, Lol e Mario e o componente *ImageView* com a imagem do jogo selecionado. Esse código do *activity_main.xml* representando o layout *LinearLayout* e os componentes *TextView*, *ImageView* e *Spinner* pode ser visualizado na Figura 6.13 abaixo:

```
<?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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Selecione um jogo:"
    />

    <Spinner
        android:id="@+id/combobox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />

    <ImageView
        android:id="@+id/imgJogo"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</LinearLayout>
```

Figura 6.13 - Arquivo activity_main.xml do Spinner

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *activity_main.xml* com a representação do *RadioButton* possuindo a representação da tag *LinearLayout* com os componentes altura, largura e orientação vertical, a tag *TextView* representando o texto "Selecione um jogo:", *Spinner* representando o *combobox* e a tag *ImageView* representando a imagem do Mario.

O *Spinner* é executado com imagens para preencher o seu *combobox*, para utilizar essas imagens você precisa baixá-las e colocá-las na pasta *drawable*. Sempre que você quiser exibir uma



imagem no seu aplicativo, você pode colocá-la nessa pasta e, para ter acesso a essa imagem, basta digitar `R.drawable.NOME_DA_IMAGEM` sem a extensão do arquivo (.jpg, .png, etc.). O Spinner configura no arquivo `MainActivity.java` da Figura 6.14, abaixo, as imagens dos jogos barkbridge, lol e mario na pasta do `drawable`, além disso, utiliza o método `onCreate()` para configurar os itens do `combobox` do jogo, o `onItemSelected()` para executar um item da seleção quando tiver sido selecionado e o método `onNothingSelected()` para executar um item da seleção quando não tiver sido selecionado na exibição do seu `combobox`. O código do arquivo `MainActivity.java` do *Android Studio* representando o *Spinner* pode ser visualizado na Figura 6.14 abaixo:

```
package com.example.eadpernambucopdm;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.Spinner;

public class MainActivity extends AppCompatActivity {

    private int[] imagens = {R.drawable.ic_launcher_barkbridge_background,
R.drawable.ic_launcher_lol_background,
R.drawable.ic_launcher_mario_background};
    private String[] jogos = new String[]{"barkbridge", "lol", "mario"};

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

        final ImageView imgJogo = (ImageView) findViewById(R.id.imgJogo);
        Spinner combobox = (Spinner) findViewById(R.id.combobox);
        ArrayAdapter adaptador = new ArrayAdapter<String>
            (this, android.R.layout.simple_spinner_item, jogos);

        adaptador.setDropDownViewResource(android.R.layout.simple_spinner_item);
        combobox.setAdapter(adaptador);

        combobox.setOnItemSelectedListener(new
        AdapterView.OnItemSelectedListener() {

            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int
            position, long id) {

                imgJogo.setImageResource(imagens[position]);
            }

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

```
}  
    }  
    }  
}
```

Figura 6.14 – Arquivo da classe MainActivity.java do Spinner

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *MainActivity.java* do *Spinner* possuindo a execução do *Spinner* com as três opções do jogo barkbridge", "lol", "mario".

Caso a opção selecionada no jogo seja Mario indica a visualização do arquivo do Mario executado na pasta *drawable* do *Android Studio*. A tela do *Android Studio* representando o componente de *Spinner* com as opções de jogos barkbridge, Lol e Mario pode ser visualizado na Figura 6.15 abaixo:

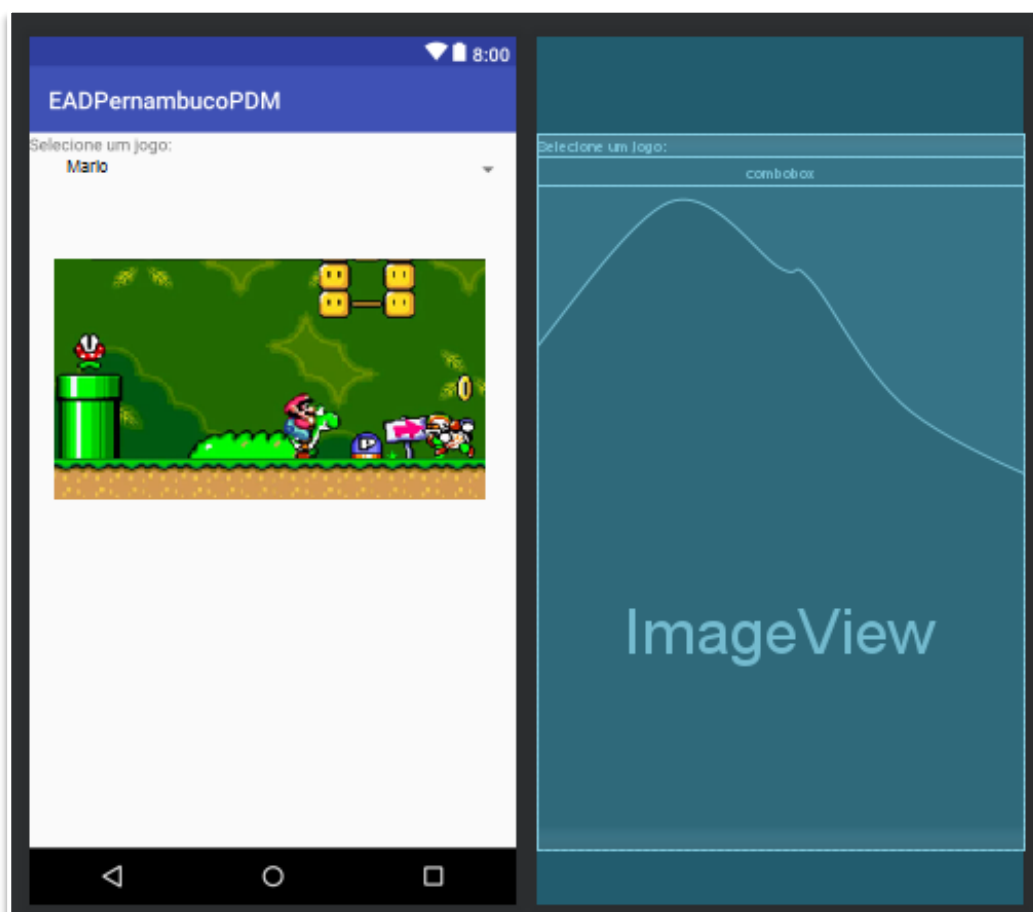


Figura 6.15 – Tela do Spinner

Fonte: o autor

Descrição da imagem: a imagem apresenta duas telas do *Android Studio* representando o componente *ImageButton*. A tela do lado esquerdo possui na parte superior uma barra azul com o nome EADPernambucoPDM, na barra inferior três



botões do celular e na parte do meio possui uma tela branca com a descrição do nome na cor preta "Selecione um jogo:", a tag *Spinner* com a opção Mario selecionada e uma imagem representando o jogo Mario, a tela do lado direito possui um quadrado com fundo verde representando a tela, duas linhas representando o texto "Selecione um jogo:", e a opção do *Spinner*, e um quadrado representando a imagem pela tag *ImageView*.

E aí, estudante, foi legal o caminho até aqui! O bom é que até aqui você aprendeu sobre:

- A classe *Widgets*
- O componente *Button*
- O componente *ImageButton*
- O componente *CheckBox*
- O componente *RadioButton*
- O componente *Spinner*



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 6?
Acesse o AVA e assista a videoaula da competência 6.



Agora, acesse o AVA e responda as questões da competência 6.



Ficou com alguma dúvida na competência 6? Acesse o **Fórum - "Competência 6"** para saná-las com os professores e discutir com seus colegas sobre os assuntos estudados.

Nosso próximo passo é seguir para competência 7. Na competência 7 você vai aprender sobre como planejar e executar o compartilhamento da sua aplicação no *Google Play*. Vamos juntos?



7.Competência 07 | Planejar e executar o compartilhamento do APK

Caro estudante, nesta competência você vai aprender sobre como planejar e executar o compartilhamento do APK (*Android Package*) que representa um pacote destinado a uma nova aplicação gerada no *Android Studio*.

O planejamento e a execução do compartilhamento do APK utilizam arquivos dos aplicativos no sistema operacional *Android*. Esse aplicativo necessita da definição do versionamento da aplicação e a assinatura do certificado digital para ser publicado no *Google Play*.

Vamos entender como realizar a publicação de APK no *Google Play*!

7.1 Versionamento da aplicação

O versionamento da aplicação em *Android* possui como extensão de arquivo a sigla *APK* (*Android Package*). *APK* é considerado um pacote destinado a uma nova aplicação gerada pelo *Android*. A primeira modificação para realizar a publicação do seu aplicativo é definir ou atualizar a sua versão no arquivo *AndroidManifest.xml*. O código do arquivo *AndroidManifest.xml* do *Android Studio* com versionamento da aplicação pode ser visualizado na Figura 7.1 abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.eadpernambucopdm"
    android:versionCode="1"
    android:versionName="0.0.1-beta">

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

Figura 7.1 - Arquivo *AndroidManifest.xml* com versionamento da aplicação

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o código do arquivo *AndroidManifest.xml* com a



declaração do versionamento da aplicação na *tag manifest* possuindo as configurações das propriedades *versionCode* e *versionName*.

As propriedades modificadas no arquivo *AndroidManifest.xml* são *android:versionCode* e *android:versionName*. A propriedade *versionCode* é utilizada para modificar a versão da aplicação no *Google Play*. Por exemplo, imagine que você lançou um aplicativo com versão "2.0" e uma semana depois precisou fazer uma correção de *bug* (erro no código) e lançará uma nova versão. Nesse caso, você pode nomeá-la como "2.1" e lançar a atualização. Já a propriedade *versionName* indica ao usuário o versionamento da aplicação que está instalada. Dessa maneira, o usuário pode apresentar uma versão anterior à versão da loja e ele pode identificar essa desatualização pelo nome da versão apresentado através da propriedade *versionName*.

Caso você esteja executando o Android Studio, existe um arquivo chamado *build.gradle* que fica em *Gradle Scripts* que contém as configurações de *build* da sua *APK*. Nesse arquivo, assim como no *AndroidManifest.xml* você também pode definir o *versionCode* e o *versionName* da sua aplicação. Esse arquivo *AndroidManifest.xml* possui prioridade sobre as configurações de *build.gradle* do projeto, ou seja, caso você defina o *versionName* e *versionCode* no *AndroidManifest.xml*, mas o seu projeto também possui o arquivo *build.gradle* localizado no *Gradle Scripts*, as informações que serão usadas serão as informações contidas no *build.gradle*, como visualizado na Figura 7.2 abaixo. O código do arquivo *build.gradle* do *Android Studio* com a descrição da versão pode ser visualizado na Figura 7.3 abaixo:

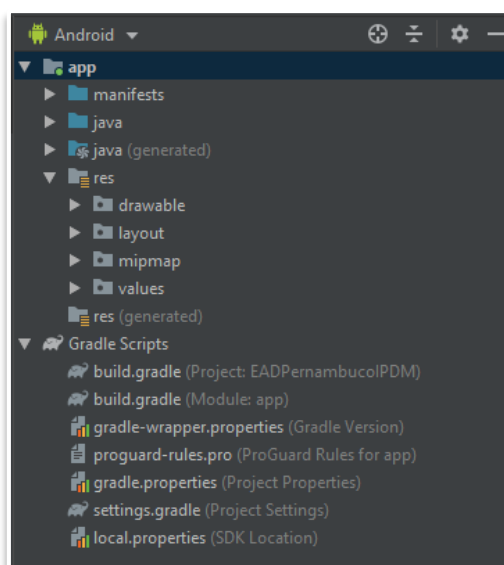


Figura 7.2 - Estrutura de um projeto do Android Studio com a configuração do arquivo *build.gradle*



Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com a estrutura de pastas de um projeto criado no *Android Studio* e mostra onde o arquivo *build.gradle* está localizado.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.eadpernambucopdm"
        minSdkVersion 26
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

Figura 7.3 - arquivo *build.gradle* com a descrição da versão

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o arquivo *build.gradle* possuindo destacando o valor dos atributos *versionCode* na versão 1 e o valor do atributo *versionName* na versão "1.0".

7.2 Compatibilidade para instalação

A compatibilidade da versão é uma configuração importante no momento da geração do *build.gradle* ou versão da sua aplicação. Dessa maneira, esse arquivo *build* possui uma propriedade chamada *minSdkVersion* que especifica a menor versão que a aplicação deve suportar, ou seja, especifica a versão pela qual sua aplicação pode ser instalada. Caso a versão do *Android* seja inferior a ela, a *Google Play* não vai permitir que o usuário instale a aplicação. Existe também a propriedade *targetSdkVersion* que especifica a última versão que a sua aplicação pode ser instalada. Os códigos a seguir mostram como especificar o *minSdkVersion* e *targetSdkVersion* no arquivo



AndroidManifest.xml visualizado na Figura 7.4, abaixo, e o arquivo *build.gradle.java* visualizado na Figura 7.5 abaixo:

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

    ...> ...
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="21" />

</manifest>
```

Figura 7.4 - arquivo *AndroidManifest.xml* com a descrição do *minSdkVersion* e *targetSdkVersion*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o arquivo *AndroidManifest.xml* possuindo a descrição do *minSdkVersion* na versão 10 e do *targetSdkVersion* na versão 21.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.eadpernambucopdm"
        minSdkVersion 26
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

Figura 7.5- arquivo *build.gradle*

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com o arquivo *build.gradle* possuindo as configurações identificador do aplicativo com o nome "com.example.eadpernambucopdm", com a descrição do *versionCode* com a versão 1, *versionName* com a versão. "1.0"



7.3 Assinatura da aplicação com certificado digital

O processo de instalação de uma aplicação em um celular ou emulador necessita da assinatura digital. Quando você compila usando o modo *Debug* esse processo é realizado de maneira automática no *Android Studio*. Contudo para realizar a assinatura de uma *APK* no *Google Play* é necessário gerar o seu próprio certificado através do *JDK (Java Development Kit)*, dessa maneira você deve possuir uma chave privada e com essa chave deve assinar *APK* durante o *build*.

No *Android Studio* você deve clicar no menu *Build* e depois em *Generate Signed APK* como visualizado na Figura 7.6 abaixo. Em seguida, a nova janela será aberta *Generate signed Bundle or APK*, como visualizado na figura 7.7 abaixo:

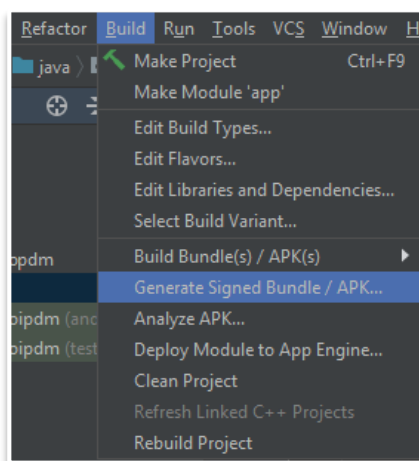


Figura 7.6 – Tela geração de certificado do Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com a tela do *Android Studio* de geração de certificado possuindo a opção *Build* na cor azul e executando a opção *Generate Signed Bundle / APK*.

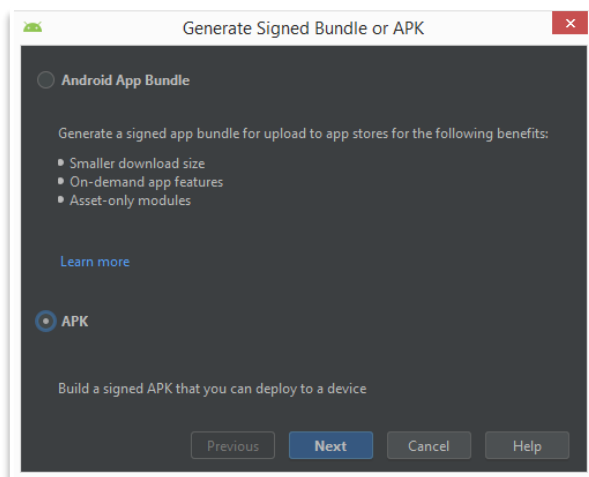




Figura 7.7 - Tela geração de certificado do Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com a tela do *Android Studio* de geração de certificado possuindo a seleção da opção *APK*, em seguida na parte de baixo da tela a opção *next* para prosseguir.

Em seguida, você deve clicar em *next* como mostrado na figura 7.7, acima, para iniciar o processo de criação da assinatura digital. A tela para gerar o certificado digital pode ser visualizada na Figura 7.8 abaixo:

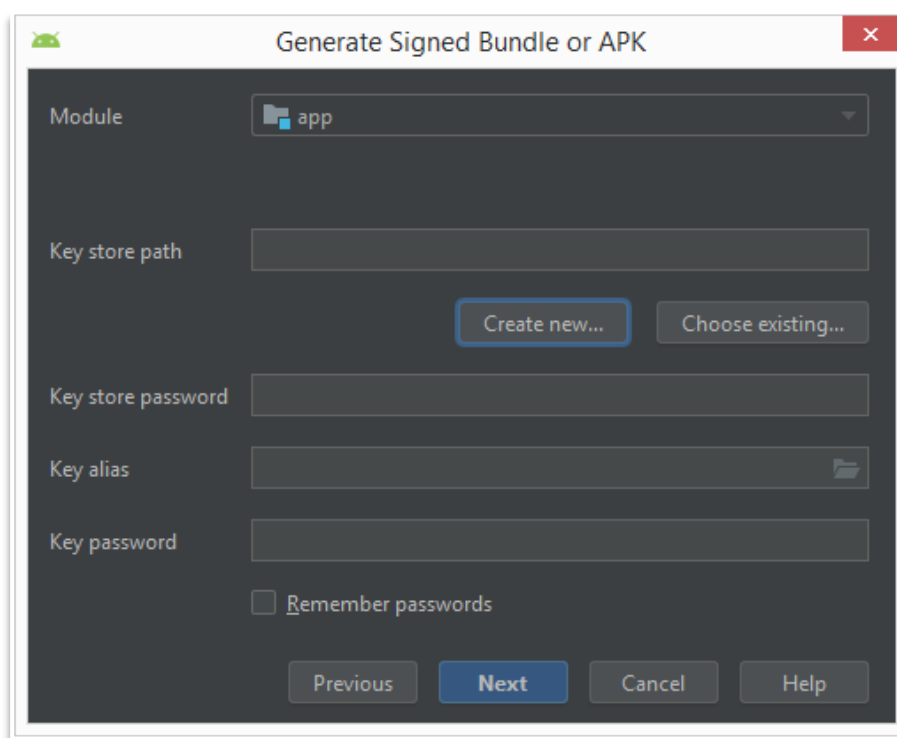


Figura 7.8 – Tela geração de certificado do Android Studio

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com a tela do *Android Studio* de geração de certificado possuindo a seleção do campo módulo com a opção *app* e a opção do botão *create new*, em seguida na parte de baixo da tela a opção *next* para prosseguir.

Na Figura 7.8, acima, você pode escolher uma assinatura existente ou criar um novo arquivo de assinatura no formato jks (Keystore Java) que serve como repositório de certificados e chaves privadas. Caso seja um aplicativo novo, crie um novo arquivo através do botão *Create new...* para assiná-lo; se for um arquivo que já foi assinado utilize o botão *Choose existing...* para selecionar o arquivo anterior. Vamos seguir os passos de criação de um novo arquivo clicando no botão *Create new...* A tela com o novo arquivo chave pode ser visualizada na Figura 7.9 abaixo:

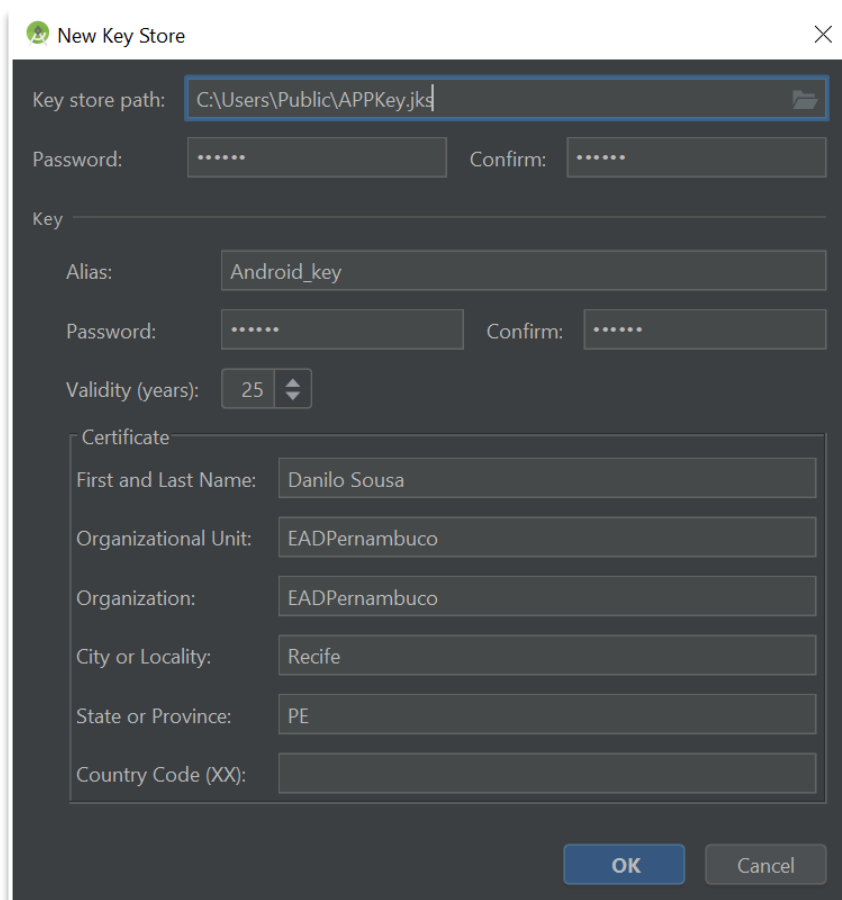


Figura 7.9 – Tela do novo armazenamento de chave preenchido

Fonte: o autor

Descrição da imagem: a imagem apresenta um fundo preto com a tela do *Android Studio* com seguintes campos chave *alias*, *password*, *validaty*, *first and last name*, *organization unit*, *organization*, *city or locality*, *state or province*, em seguida na parte debaixo da tela clica na opção de botão com a seleção OK.

Na Figura 7.9, acima, você deve preencher alguns campos e clicar no botão OK.

- *Key store path* : caminho para salvar o arquivo no formato jks.
- *Password* : senha para o arquivo.

Key (Chave):

- *Alias* : um "apelido" para essa chave; esse nome é de escolha do desenvolvedor. Eu geralmente deixo o Alias com o mesmo nome do App.
- *Password* : senha para a chave que está sendo criada. Essa senha pode ser diferente da senha anterior, do arquivo, ou você pode manter a mesma senha para não confundir, no entanto, para melhor segurança é bom ter senhas diferentes.
- *Validity* : validade desse certificado em anos.
- *First and Last Name* : primeiro e último nome, pode ser o seu nome.



- *Organizational Unit* : nome da sua unidade organizacional (campo opcional).
- *Organization* : nome da sua organização (campo opcional).
- *City or Locality* : cidade ou localização (campo opcional).
- *State or Province* : Estado ou província (campo opcional).
- *Country Code(XX)* : código do país (campo opcional).

7.4 Assinatura da aplicação para publicá-la no Google Play

Com o certificado criado existe a possibilidade de utilizar essa assinatura na sua *APK*. Mas como realizar isso? Para assinar a sua respectiva *APK* utilizando o certificado que você criou, basta importá-lo no *Android Studio* clicando no botão *Choose existing...*, digitar a sua senha e gerar a sua *APK* assinada clicando no botão *Next*. A tela utilizando o botão *Choose existing* pode ser visualizada na Figura 7.10 abaixo:

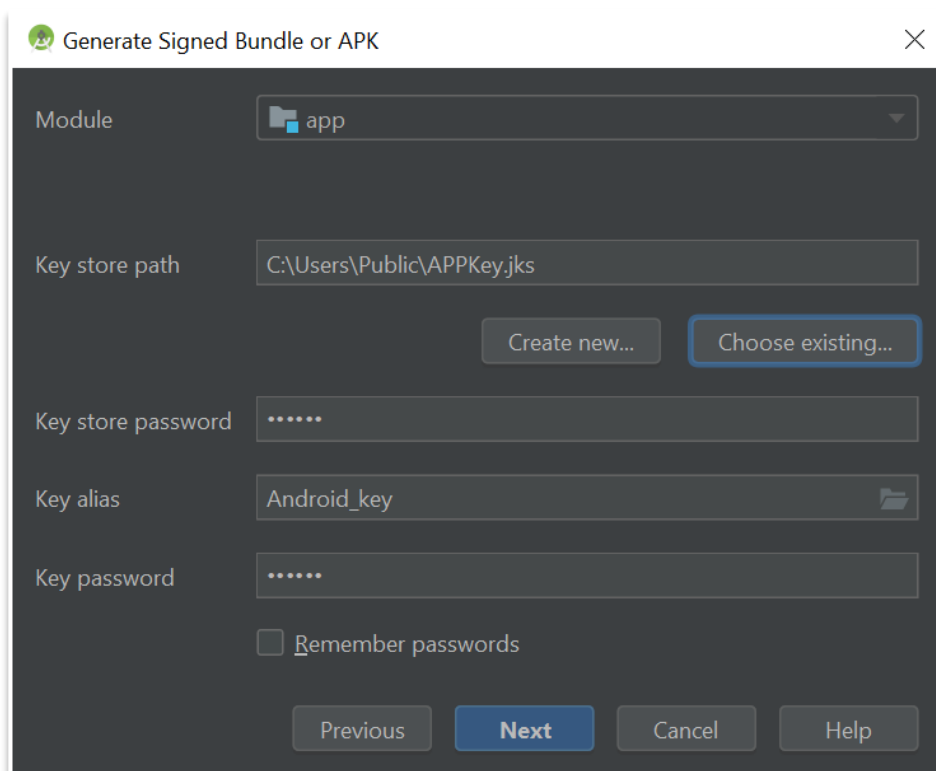


Figura 7.10 – Tela do Assinando APK com as configurações preenchidas

Fonte: o autor.

Descrição da imagem : a imagem apresenta um fundo preto com a tela assinando uma *APK* do *Android Studio* com os configurações preenchidas possuindo os campos *Module* com a opção *app*, *key store path* com a opção preenchida *AppKey.jks*, *key store password* com a senha preenchida, *key alias* com a opção *alias* e *key password* com a senha preenchida, em seguida na parte de baixo da tela a opção *next* para prosseguir.



Na figura 7.10, acima, você deve clicar no botão *next* para aparecer uma janela pedindo para informar o local onde você deseja que a sua *APK* assinada seja gerada e ao final do processo basta clicar em *Finish*. A tela com o processo final da sua *APK* assinada pode ser visualizada na Figura 7.11 abaixo.

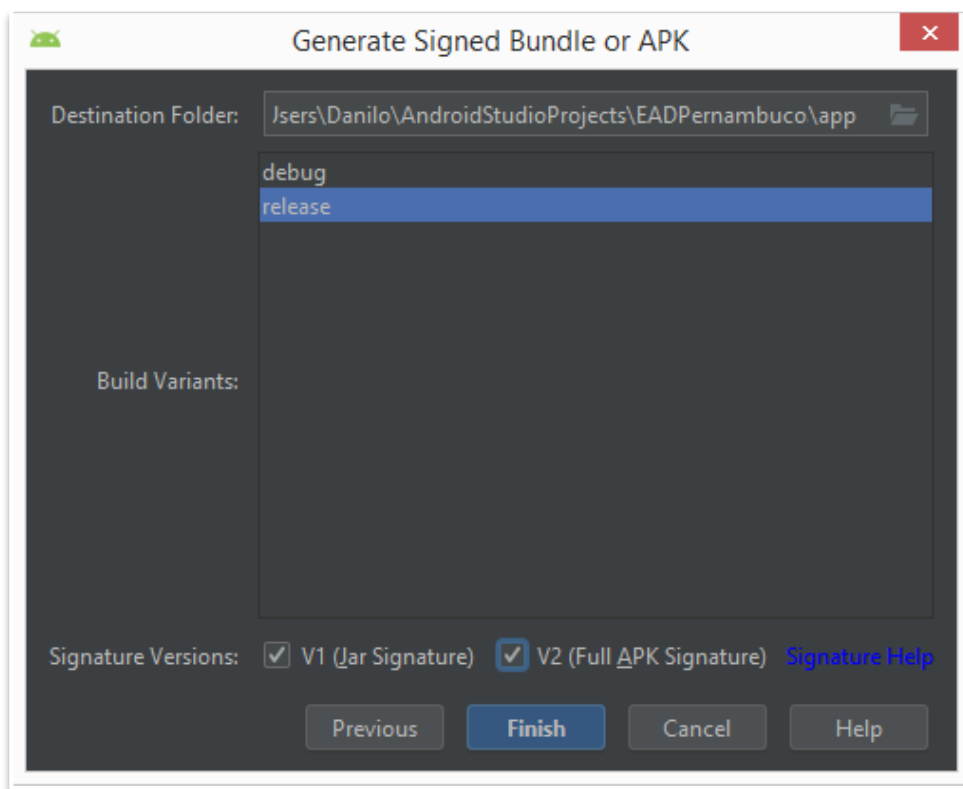


Figura 7.11 – Tela da assinando APK com a finalização da assinatura

Fonte: o autor

Descrição da imagem a imagem apresenta um fundo preto com a tela assinando uma *APK* do *Android Studio* no final da assinatura com os campos *Destination folder* com a descrição preenchida, a opção *release* selecionada com a cor azul, em seguida na parte de baixo da tela a opção *finish* para finalizar a assinatura.

Na Figura 7.11, acima, você deve escolher o *Build type* (tipo de compilação) entre *release* (lançamento) ou *debug* (desenvolvimento). Como vamos enviar à loja, este é um *build de release*. Em *Signature Versions* (versões de assinatura) selecione as duas opções V1 e V2. O tipo de assinatura V2 é relativamente novo no *Android*, antes o padrão era a V1; a V2 veio junto com a versão 7.0 do *Android*, e de acordo com a documentação do *Android* devemos escolher esses dois tipos. Por fim, clique em *Finish* (finalizar).



7.5 Publicando a sua aplicação no Google Play

O processo para publicar uma aplicação na *Google Play* é considerado um processo bem trivial, basta entrar no site do *Google Play console* <https://play.google.com/apps/publish/>. Caso você esteja logado na conta do Google já abrirá a página da Figura 7.12, abaixo, caso contrário você precisará logar a sua conta no *Google*, na qual a sua aplicação será associada na sua conta.

Figura 7.12 - Tela de aceitação do contrato como desenvolvedor da Google Play

Fonte: o autor

Descrição da imagem: a imagem apresenta a tela do site com informações da conta da *Google* do usuário que está logado, com o valor a ser cobrado pela licença de desenvolvedor, com os termos de uso e um botão para continuar o pagamento.

O processo para aceitar o contrato do desenvolvedor necessita de quatro passos. O primeiro é necessário fazer o login com uma conta Google, depois aceitar os termos de uso do contrato disponível em: <https://play.google.com/about/developer-distribution-agreement.html>, pagar a taxa de 25 dólares pelo aplicativo e, por fim, fornecer os detalhes da conta como é exibido na Figura 7.13 abaixo.



A interface de pagamento é exibida em um modal com o título "Conclua sua compra". No topo, há uma barra de status com um ícone de fechar (X) e um menu de opções (três pontos). Abaixo, o item a ser pago é "Developer Registration Fee" com o valor "US\$ 25,00".

Logo abaixo, há uma seção para "Adicionar novo cartão de crédito". O formulário contém:

- Um campo para o "Número do cartão" com uma máscara "# _____" e ícones para American Express, Discover, JCB, Mastercard e Visa.
- Campos para "MM / AA" (validade) e "CVC".
- Campos para "Nome do titular do cartão" e "endereço de faturamento".

No rodapé do modal, há um texto de aviso: "Ao continuar, você adiciona esta forma de pagamento à sua conta do Google Payments e concorda com os [Termos de Serviço do Google Payments](#) e o [Aviso de privacidade](#)." e um botão azul "COMPRAR".

Figura 7.13 - Tela de pagamento, basta inserir o cartão de crédito e pagar 25 dólares.

Fonte: o autor

Descrição da imagem: a imagem apresenta a tela do site com as informações do cartão de crédito juntamente com o valor da licença para o pagamento.

Após o pagamento do aplicativo confirmado e de realizar o cadastro das suas informações você pode realizar a publicação da sua aplicação. Em seguida, deve preencher um formulário com uma breve descrição da sua aplicação e enviar algumas telas dessa aplicação. Depois disso, deve preencher algumas informações do aplicativo e fazer o *upload* da *APK* assinada, escolher a categoria da sua aplicação e informar se essa aplicação é paga ou gratuita.

Por fim, depois de publicar essa aplicação no *Google Play*, essa será disponibilizada imediatamente para qualquer outro usuário instalar. A cada nova versão dessa aplicação que você atualizar é preciso modificar os parâmetros *versionCode* e *versionName*, como mostrado na seção versionamento de aplicação. Assim, os usuários dessas aplicações serão notificados sobre a existência de uma nova versão, e assim podem realizar a atualização quando desejarem.



Saiba mais!

Para você saber como publicar um Aplicativo Android na Play Store 2020.

Acesse

<https://www.youtube.com/watch?v=yY9hQhE18K4>

E aí, estudante, foi legal o caminho até aqui. O bom é que até aqui você aprendeu sobre:

- Versionamento da aplicação
- Compatibilidade para instalação
- Assinatura da aplicação com certificado digital
- Assinatura da aplicação para publicá-la no *Google Play*
- Publicando a sua aplicação no *Google Play*



Videoaula!

Pronto para aprender por vídeo aula sobre a competência 7?

Acesse o AVA e assista a videoaula da competência 7.



Agora, acesse o AVA e responda as questões da competência 7.



Ficou com alguma dúvida na competência 7? Acesse o **Fórum - "Competência7"** para saná-las com os professores e discutir com seus colegas sobre os assuntos estudados.

Agora, o nosso próximo passo é focar na avaliação da disciplina de PDM. Vamos lá?



Conclusão

Chegamos ao final da disciplina programação para dispositivos móveis! Espero que os conhecimentos acumulados em nossa disciplina tenham lhe ajudado a alcançar a compreensão inicial sobre gerenciamento de *layout* e outros conhecimentos essenciais no desenvolvimento de aplicativos para a plataforma *Android*.

Durante essas sete semanas de curso você compreendeu a finalidade do gerenciamento de *layout* utilizada em aplicações *Android*, com a utilização da classe *Activity* responsável pela navegabilidade entre telas, em seguida você aprendeu sobre como manipular *Action bar* e temas.

Por conseguinte, você viu alguns conceitos mais técnicos relacionados aos gerenciadores de *layout* usando classe *View* com *ViewGroup* representados pelos *layouts ConstraintLayout* e *LinearLayout*. Em seguida, os recursos avançados de gerenciadores de *layout* representada pelos *layouts FrameLayout*, *TableLayout* e *RelativeLayout*.

Em seguida, você entendeu os recursos de *view* da classe *widget* com os elementos *TextView*, *EditText* e *ImageView*. Assim como, os recursos avançados da classe *widget* com os elementos *ImageButton*, *Button*, *Checkbox*, *RadioButton* e *Spinner*. Por fim, você aprendeu como planejar e executar o compartilhamento do *APK* no *Google play*.

Portanto, espero que os conceitos visualizados nesta disciplina de programação para dispositivos móveis possam agregar em sua carreira profissional a tentar, de forma criativa, utilizar os conhecimentos para criar *layouts* de aplicações com propostas diferentes dos exemplos construídos nesta disciplina.

Bons estudos e nos veremos na próxima!



Referências

LECHETA, Ricardo. **Google Android**. Aprenda A Criar Aplicações Para Dispositivos Móveis Com O Android SDK. São Paulo: Novatec, 2010.

GLAUBER, Nelson. **Dominado o Android**. São Paulo: Novatec, 2015.

LECHETA, Ricardo. **Android Essencial com Kotlin**. São Paulo: Novatec, 2018.

GLAUBER, Nelson. **Dominado o Android com Kotlin**. São Paulo: Novatec, 2019.

QUEIRÓS, Ricardo. **Android Profissional**. Desenvolvimento Moderno de Aplicações. Lisboa: FCA, 2018.

Site oficial de desenvolvedores Android: <https://developer.android.com/?hl=pt-br>



Minicurrículo do Professor

Danilo de Sousa Barbosa

Mestre em Engenharia da Computação na universidade Federal de Pernambuco (UPE), Especialização em Gestão e Qualidade em Tecnologia da Informação e Comunicação no Instituto Federal de Pernambuco (IFPE). Possui graduação em Bacharelado em Sistemas de Informação pela Universidade de Pernambuco (UPE). Atualmente possui doutorado em andamento em Ciência da Computação no Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE) e professor de ensino à distância do curso técnico de Desenvolvimento de Sistemas da Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa (ETEPAC)

