Aula 6 – Anatomia do Android Studio

Programação Mobile

Professor Bruno Maion

A plataforma

- **Conhecer bem ajuda a codar!**
- **Tópicos desta aula:**
  - Activities;
  - Classe R;
  - Views;

ACTIVITIES

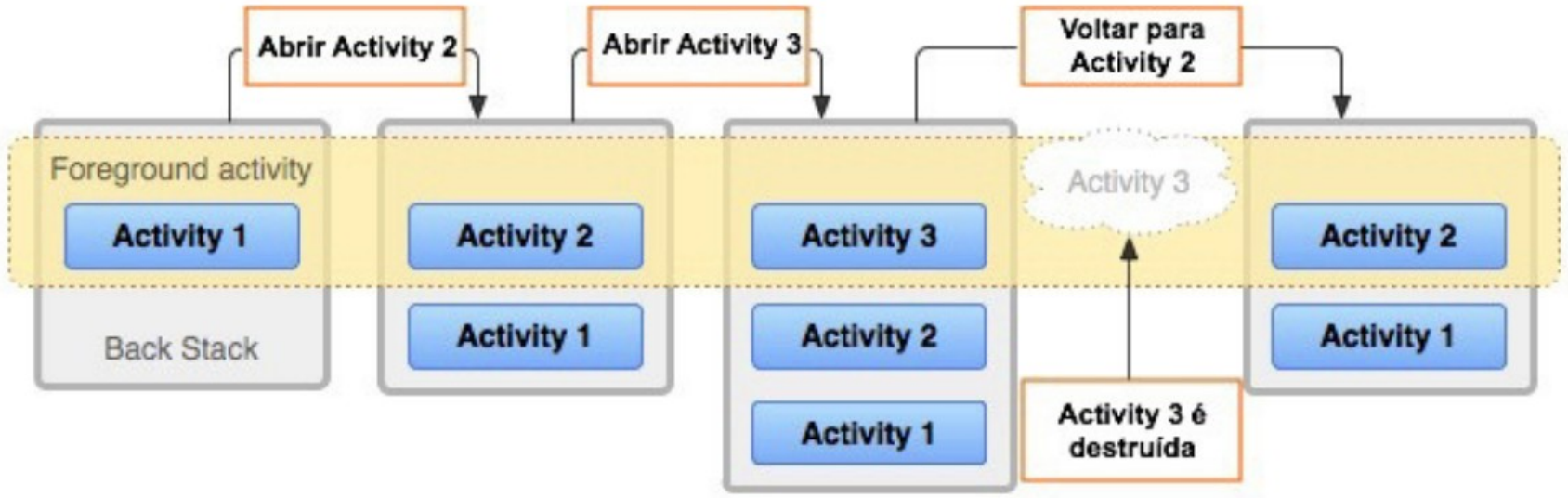
- Activity (Atividade) – É o principal componente;
- Podemos entender como uma tela de nosso aplicativo;
  - A principal é a *MainActivity.xml*.
- Colocamos nossos componentes (*Widgets*) nesta tela;

ACTIVITIES

- Cada Activity pode iniciar outro Activity;
- Exemplo: Aplicativo de mensagens:
  - A Activity **principal** seja uma lista dos contatos que você tem na agenda.
  - Ao clicar em cima de algum contato, essa Activity principal deve iniciar uma outra **Activity**, a da **conversa**.
  - Quando isso acontece, a **Activity** que foi iniciada é **empilhada** sobre a principal (Pilha/*Stack*)

ACTIVITIES

- **Pilha de navegação (*Stack*)**



ACTIVITIES

- **Erro comum:**
  - Quando precisa encerrar uma Activtity secundária o programa “chama” a Activity Principal, ao invés de apenas desempilhar;
  - O efeito seria o mesmo, porém há um consumo maior de RAM, visto que a pilha de navegação irá se enchendo.

CRIANDO ACTIVITIES

- **Basta criar uma classe e herdar da classe Activity do Android.**
  - Conceito de herança - Programação Orientada a Objetos;
  - Todas características de uma Activity;

```
import android.app.Activity

class MainActivity : Activity(){

}
```

CRIANDO ACTIVITIES

- Para Activity funcionar é necessário o método de callback (chamada de retorno);
- Para iniciar a tela, o sistema chama o método **onCreate** (em criação);

CRIANDO ACTIVITIES

- O método **onCreate** é o único obrigatório, porque, sem ele, a tela não se constrói;

```
class MainActivity : Activity(){

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

    }

}
```

CRIANDO ACTIVITIES

- **Override –** Sobrescreve o método;
- Por parâmetro **savedInstanceState: Bundle?**
  - Quem aciona este método é o próprio sistema operacional
- A variável **savedInstanceState** guarda informações da Activity;
  - Guarda quais componentes estão na tela e informações que o usuário tenha preenchido

CRIANDO ACTIVITIES – Outros métodos

1) onCreate()

- Chamado quando a Activity é criada. Para iniciar variáveis, layout e listeners.

2) onStart()

- Chamado quando a Activity fica visível para o usuário. Ainda não está em primeiro plano.

3) onResume()

- Activity fica em primeiro plano (interação com usuário). Ideal para iniciar animações, sensores, câmera etc.

CRIANDO ACTIVITIES – Outros métodos

4) onPause()

- Chamado quando a Activity não está sendo usada, mas ainda visível. Usado para pausar animações e salvar dados temporários.

5) onStop()

- Activity não está mais visível. Liberar recursos pesados (GPS, sensores, etc).

6) onDestroy()

- Chamado antes da Activity ser finalizada. Usado para limpar recursos.

ACTIVITIES – Definindo conteúdo

- Definir o conteúdo de uma Activity significa definir o que aparecerá na tela;
- Para isso o Android possui o método **setContentView()**

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
//        enableEdgeToEdge()

        val texto = TextView( context = this)
        texto.text = "Hello Kotlin"
        setContentView(texto)
    }
}
```

ACTIVITIES – Definindo conteúdo

- Há 2 modos para definir o que será apresentado na tela;
  - Via código: exemplo anterior – Visualização somente após o código ser testado.
  - Via arquivo ***xml –*** Permite a pré-visualização dos componentes;

```
    //setContentView(texto)
    setContentView(R.layout.activity_main)
}
```

ACTIVITIES – Definindo conteúdo

- Separar o layout da tela em um arquivo ***xml*** possui diversas vantagens:
  - Responsabilidades ficam separadas - mais fácil para o programador encontrar um problema tanto no código quanto no layout
  - O arquivo pode ser reutilizado em alguma outra Activity que utilize o mesmo layout.

A CLASSE R

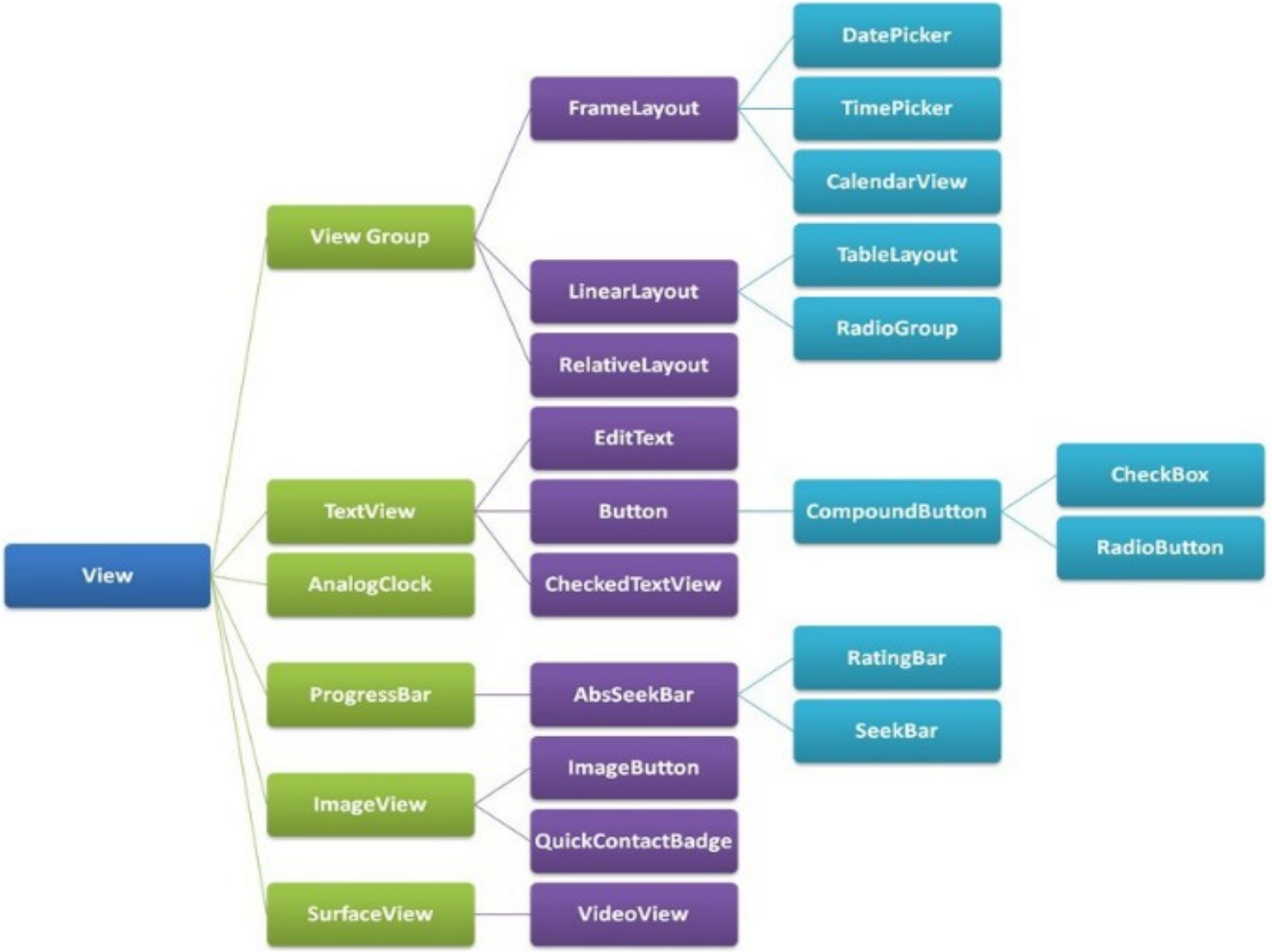
- O nome **R** é uma abreviação de "Resources" (Recursos);
- Esta classe é responsável por alterar alguns recursos em tempo real;
  - Imagens, textos, um arquivo xml;
- No exemplo anterior, utilizamos a classe para acessar o arquivo que contém o layout;
- Não é necessário colocar a extensão (**.xml**).

A CLASSE R

- Acessar uma imagem chamada ***background.png*** dentro da pasta drawable:
  - **R.drawable.background**
- Uma cor chamada azul definida no arquivo colors, podemos acessá-la com:
  - **R.color.azul**
- Essa lógica segue para qualquer recurso dentro da pasta.

VIEWS – COMPONENTES VISUAIS

- As **Views** (visualizações) são todos os **componentes visuais** que podem ser usados na criação de um aplicativo.
- A classe View é a classe **base** de qualquer outro componente que podemos usar em uma **Activity**.
  - Botões, caixas de texto, caixas de seleção, objetos de imagens etc.,
  - Todos são derivados da classe View .

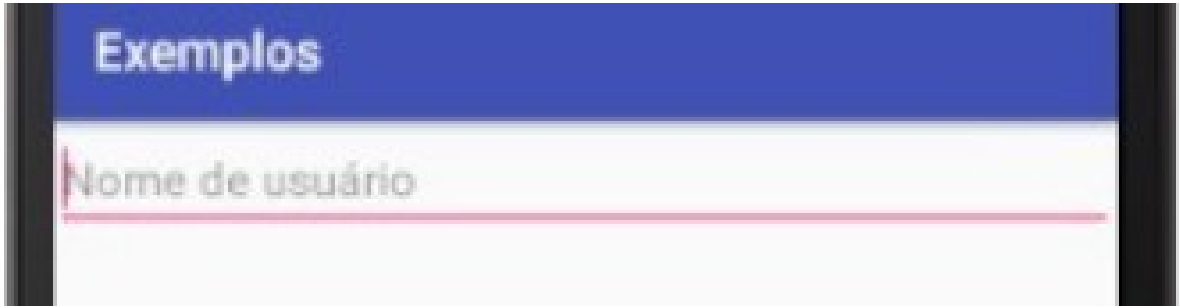

View
View Group
FrameLayout
DatePicker
TimePicker
CalendarView
LinearLayout
TableLayout
RadioGroup
RelativeLayout
TextView
EditText
Button
CompoundButton
CheckBox
RadioButton
CheckedTextView
AnalogClock
ProgressBar
AbsSeekBar
RatingBar
SeekBar
ImageView
ImageButton
QuickContactBadge
SurfaceView
VideoView

VIEWS – TextView

- O **TextView** é um componente de visualização de texto usado quando queremos mostrar alguma informação escrita para o usuário.
- Ele pode ser definido pela tag **<TextView />**
- Podemos usar sua propriedade **text** para definir o texto exibido na tela.

VIEWS – EditText

- O **EditText** é o componente que usamos quando queremos que o usuário informe algo ao aplicativo;
- Conhecido também como **caixa de texto.**
- O EditText pode ser definido pela tag **<EditText/>**



VIEWS – Button

- Cria um **botão** na tela e pode disparar alguma ação.
- Imagine uma tela de cadastro, em que o usuário preenche todas as informações e, ao final, é comum haver um botão cadastrar , que dispara uma ação no código, que faz o cadastro do usuário.
- Um botão pode ser definido pela tag **<Button />**

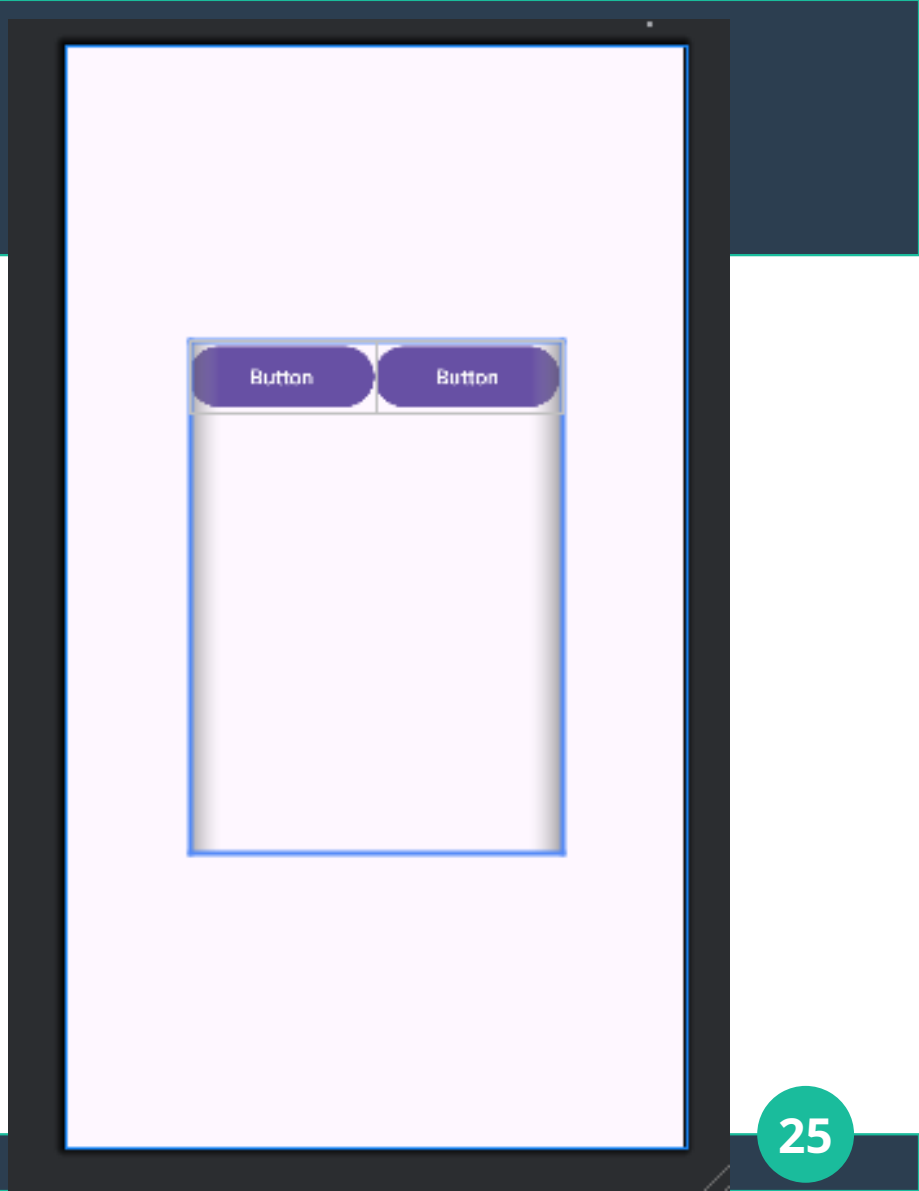
VIEWS – LinearLayout

- O LinearLayout é um pouco diferente das outras Views porque ele faz parte da família do **ViewGroup** (grupo de visualização).
- Sua função é **agrupar** outros componentes dentro dele.
- Podemos definir um layout linear através da tag **<LinearLayout /> .**

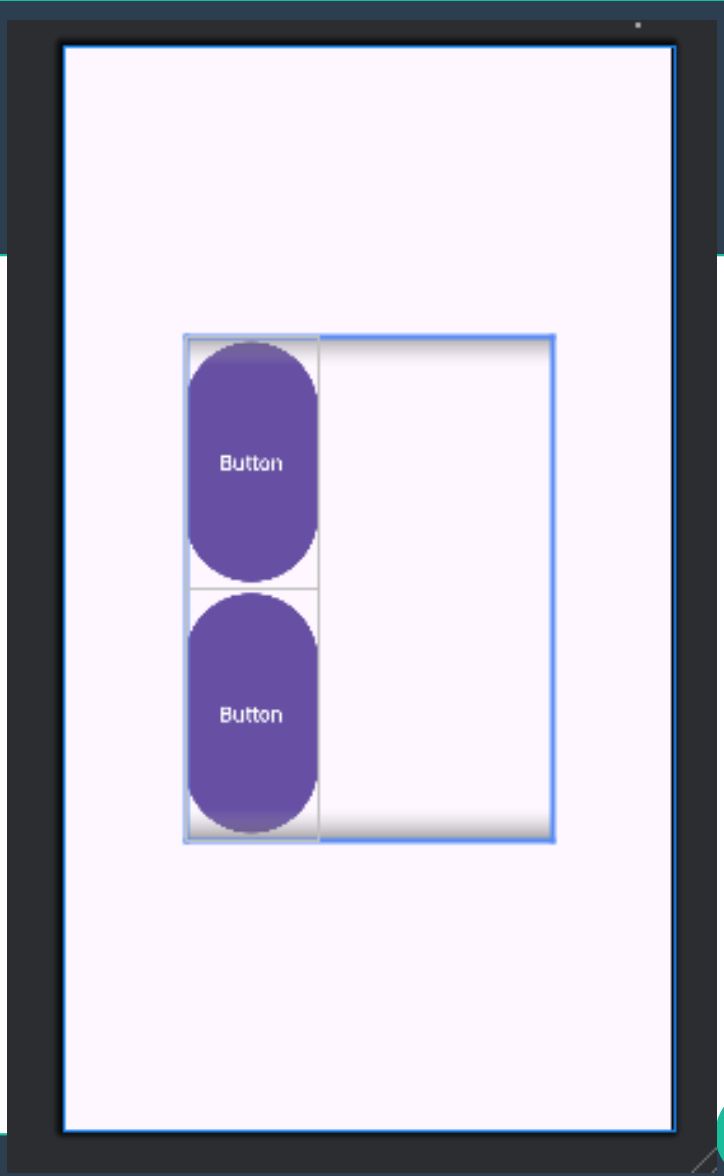
VIEWS – LinearLayout

- Possui uma propriedade chamada orientation (orientação) que define como os componentes serão exibidos dentro do layout.
- Essa definição pode acontecer de duas formas: com orientação **horizontal** ou **vertical**.
  - Layout vertical, todos os componentes dentro dele ficarão um **embaixo** do outro;
  - Layout horizontal, todos os componentes ficarão um **ao lado** do outro.

```
<LinearLayout
    android:layout_width="247dp"
    android:layout_height="339dp"
    android:orientation="horizontal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

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

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button" />
</LinearLayout>
```



```xml
<LinearLayout
    android:layout_width="247dp"
    android:layout_height="339dp"
    android:orientation="vertical"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

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

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button" />
</LinearLayout>
```



VIEWS – Tamanho de uma view

- Em todas as Views, é **obrigatório** que se definam duas propriedades,
  - **layout_width** (largura) e **layout_height** (altura).
  - Caso você não as defina, o código não vaicompilar.
- Não há um valor padrão para elas, ou seja, você sempre terá que as definir.

VIEWS – Tamanho de uma view

- Há 3 maneiras:

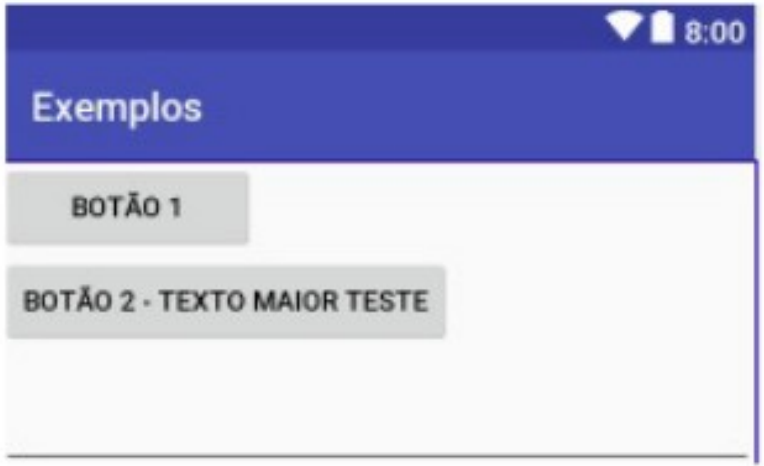
  1. Definindo uma largura ou altura fixa em **dp** (pixel independente de densidade);

     Garante que os componentes sempre terão o mesmo tamanho

VIEWS – Tamanho de uma view

2. Usar a constante **wrap_content** .
   - Será **variável** de acordo com o conteúdo dela.

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botão 2 - Texto Maior Teste"
        />
```

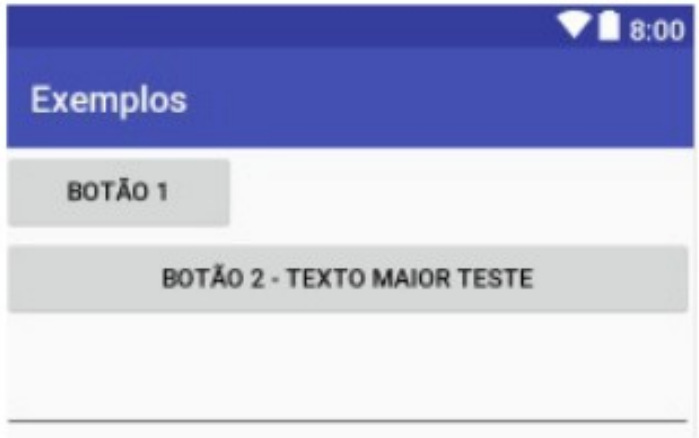


VIEWS – Tamanho de uma view

3. Usar a constante **match_parent** .

O componente tomará todo o espaço disponível em relação ao layout em que ele está inserido.

```
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Botão 2 - Texto Maior Teste"
        />
```



VIEWS – findViewById

- Para acessar algum elemento que foi definido no arquivo de **layout** pelo código em Kotlin, precisamos localizar aquele elemento pelo seu **id**.
- O método **findViewById** localiza no layout uma View a partir de um id passado.

```
findViewById<Button>(R.id.btn_login)
```

OU

```
val btn_login = findViewById<Button>(R.id.btn_login)
```

Referências

- RESENDE, K. Kotlin com Android. [s.l.] Editora Casa do Código, 2018.

Obrigado!