# soluções mobile

*prof. Thyerri Mezzari*

# Android: Telas, Layouts, Componentes Visuais, Activity e Intents

## Parte I

# Importante

Vamos iniciar nossos slides com um "breve" *resumão* de conceitos:

**Formato XML**: Dialeto/formato padrão para composições de dicionários e componentes visuais na plataforma Android Nativa.

**Activity**: Camada lógica responsável por "montar" e controlar layouts e seus subcomponentes.

**Fragment**: Camada lógica em formato reutilizável não independente que serve para controlar trechos de layouts e seus subcomponentes.

# Importante

**Layout**: Estrutura de elementos em formato XML que compõe telas e ou trechos de telas (fragments).

**Componentes Visuais**: Elementos padronizados (ou não) de componentes comuns a serem usados em um aplicativo. *Ex: Botões, Imagens, Campos de Texto e etc.*

**Intent**: "Script" lógico que indica ao aplicativo uma "intenção" de ação. *Ex: trocar uma tela.*

# Formato XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout>
  <TextView text="Hello, I am a TextView" />
  <Button text="Hello, I am a Button" />
</LinearLayout>
```

# Formato XML

O formato utilizado pela plataforma Android Nativa é o **XML**. É um formato declarativo, de marcação, que lembra muito o HTML.

Usando o **XML** do Android, é possível projetar rapidamente layouts de interfaces e os elementos de tela do mesmo modo que se cria páginas Web — com uma série de elementos aninhados.

Cada arquivo de layout deve conter exatamente um elemento raiz, que deve ser um objeto ***View*** ou ***ViewGroup***. Com o elemento raiz definido, é possível adicionar objetos ou ***widgets*** de layout extras como elementos filho para construir gradualmente uma hierarquia de View que define o layout.

# Formato XML

Na teoria seria possível criarmos componentes e telas 100% usando código lógico, mas dada as ferramentas disponíveis no ***Android Studio***, isso seria passar trabalho de mais sem necessidade.

Sendo assim, *fica a dica* de se acostumar com o formato XML quando se trata de **Android Nativo**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout>
  <TextView text="Hello, I am a TextView" />
  <Button text="Hello, I am a Button" />
</LinearLayout>
```

# Componentes Visuais

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

# Componentes Visuais

Definiremos aqui e nas próximas aulas o conjunto de componentes interativos dispostos visualmente em uma tela de aplicativo como "**Componentes Visuais**".

Ou seja, tudo que o usuário pode enxergar ou interagir através de um toque em um app.

A ideia aqui não é passar todos os componentes existentes na biblioteca Android *(seria necessário uns 14 semestres)* mas sim dar uma "pincelada" nos principais a serem usados nos cenários mais comuns.

# Componentes Visuais

Todos os componentes visuais do Android herdam sua base da classe genérica ***View***.

Ou seja, a classe **android.view.View** e suas subclasses se encarregam de desenhar os componentes visuais.

A seguir, iremos conhecer alguns componentes e nos mais fáceis de implementar também um pouco de código.

# Button e ImageButton

Um dos componentes mais clássicos existente desde a primeira versão do SO Android, o **Button** *(e seu co-irmão **ImageButton**)* criam um bloco visual que indica ao usuário a possibilidade de clique/toque.

# Button e ImageButton

O código de um **botão simples**:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

O atributo ***android:text*** contém o conteúdo/label do botão a ser exibido em tela.

# Button e ImageButton

O código de um **botão com ícone na esquerda**:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

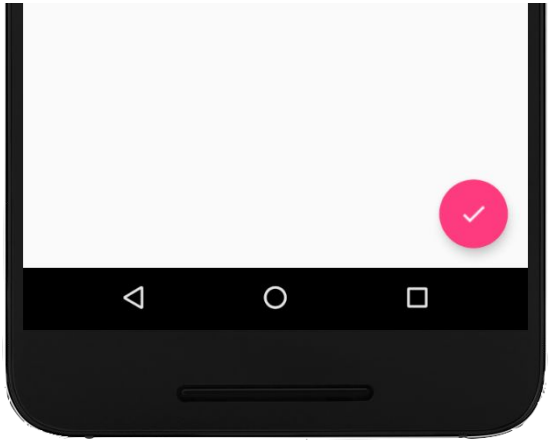
# Button e ImageButton

O código de um **botão do tipo imagem, sem texto, apenas uma imagem ou ícone**:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    android:contentDescription="@string/button_icon_desc"
    ... />
```

# FAB - Floating Action Button

Um botão de ação flutuante **(FAB, na sigla em inglês)** é um botão circular que aciona a ação principal na IU do seu app.



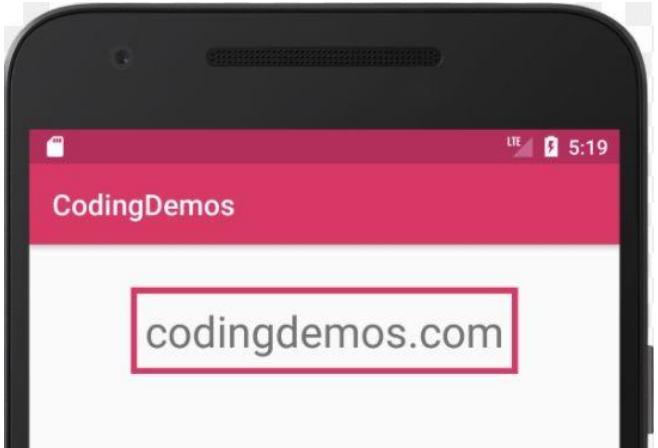
# FAB - Floating Action Button

O código a seguir mostra como o **FloatingActionButton** precisa aparecer no seu arquivo de layout:

```
<android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:src="@drawable/ic_my_icon"
        android:contentDescription="@string/submit"
        android:layout_margin="16dp" />
```

# TextView

O componente **TextView** é o mais próximo que temos de um "parágrafo" ou "label" em um comparativo direto com HTML. Ele serve para mostrar textos estáticos ao longo das telas de seu app. O uso dele é o mais variável, praticamente qualquer texto não-editável pelo usuário pode ser um **TextView**.



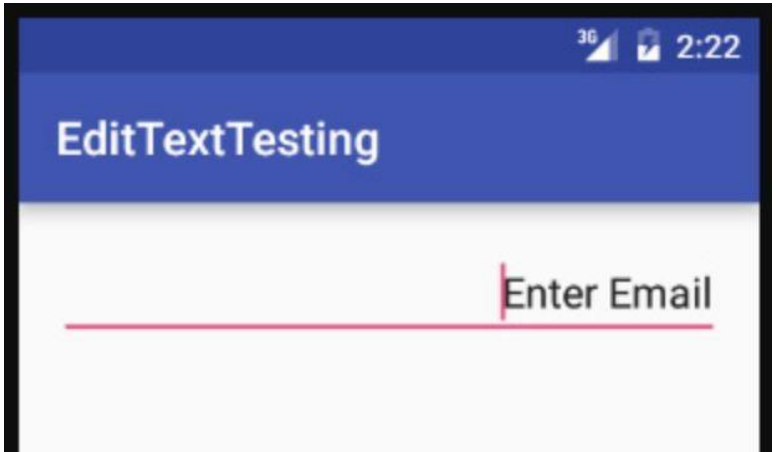
# TextView

O código básico de um **TextView**:

```
<TextView
    android:id="@+id/text_view_id"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/hello" />
```

# EditText

Sempre que precisarmos de um input de texto na tela para que o usuário insira / digite ou altere alguma informação textual o componente indicado para uso é o **EditText**. Em uma analogia direta com o HTML o **EditText** poderia ser comparado ao mesmo tempo com as tags *input* e *textarea*.



# EditText

O código básico de um **EditText**:

```
<EditText
    android:id="@+id/email"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Preencha este campo com seu e-mail"
    android:inputType="textEmailAddress" />
```

# AutoCompleteTextView

O **AutoCompleteTextView** é um *EditText* com sugestões de input/edição para o usuário.

# AutoCompleteTextView

O código básico de um **AutoCompleteTextView**:

```
<AutoCompleteTextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

As sugestões que este campo irá apresentar ao usuário devem ser definidas de forma programática.

# ImageView

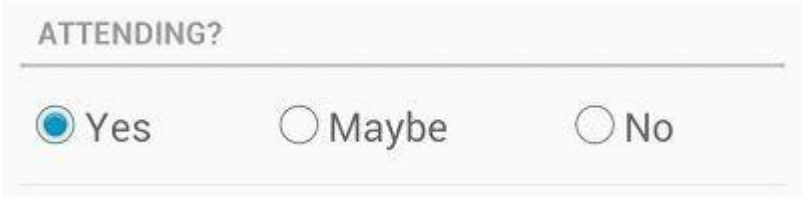
Como o nome já diz, o **ImageView** é projetado especificamente para exibir imagens na tela. Isso pode ser usado para a exibição de recursos armazenados no aplicativo ou para a exibição de imagens que são baixadas da internet.

```
<ImageView
      android:src="@drawable/imagem"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:scaleType="center"/>
```

O atributo **android:src** é o que indica dentro da pasta resources qual o arquivo de imagem deve ser exibido em tela.

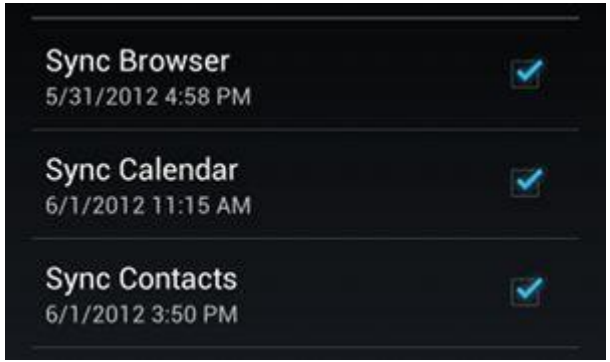
# Radio Buttons

O uso de **RadioButton** é indicado quando o usuário pode selecionar apenas uma opção, dentre poucas opções, no máximo 3 opções que caibam lado a lado em tela. Se o conjunto de opções for muito grande *(ex.: seletor de estados do Brasil)* é indicado o uso do componente ***Spinner***.



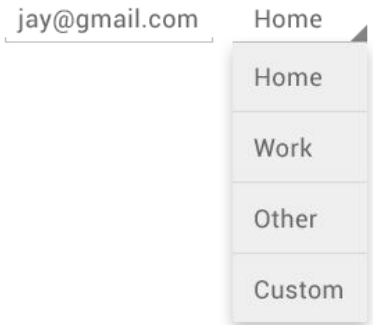
# Checkboxes

O componente **CheckBox** permite ao usuário selecionar um ou mais opções de um conjunto de opções disponíveis.



# Spinner

O componente **Spinner** ou caixa de seleção suspensa, oferece uma forma rápida de selecionar um valor de um conjunto. No estado padrão, mostra o valor selecionado naquele momento. Um toque no componente mostra um menu suspenso com todos os outros valores disponíveis. Neles, o usuário pode selecionar um novo valor.



Engenharia da Computação

# Layouts Básicos

<LinearLayout />, <RelativeLayout />, <GridLayout />,
<FrameLayout /> e <WebView />

*Fonte:* https://www.androidpro.com.br/blog/desenvolvimento-android/android-layouts-viewgroups-intro

# Layout?!

A base de todas as telas que envolvem um aplicativo Android são os **Layouts**. Em uma analogia direta com o mundo da web, os layouts são os containers mais genéricos possíveis, seria o equivalente a *DIVs* do HTML.

A ideia é que cada tela, fragmento e componente tenha pelo menos um (ou mais) layouts aninhados contendo diversos componentes dentro dele (botões, imagens, campos de texto e etc).

Porém diferente do HTML ao invés de usar um DIV genérica e posicionar os elementos usando CSS, a *Android SDK* possui um layout específico para a maioria das necessidades a cerca de distribuição e posicionamento de elementos.

# LinearLayout

O **LinearLayout** é o tipo de layout mais simples e direto (e também um dos mais usados) para Android. Basicamente ele serve para agrupar e dispor componentes de forma ***horizontal*** ou ***vertical***.

Para ajustar esse posicionamento utilizamos o atributo ***android:orientation*** passando o valor ***vertical*** ou ***horizontal*** para posicionar as Views uma embaixo da outra ou lado-a-lado.

# LinearLayout

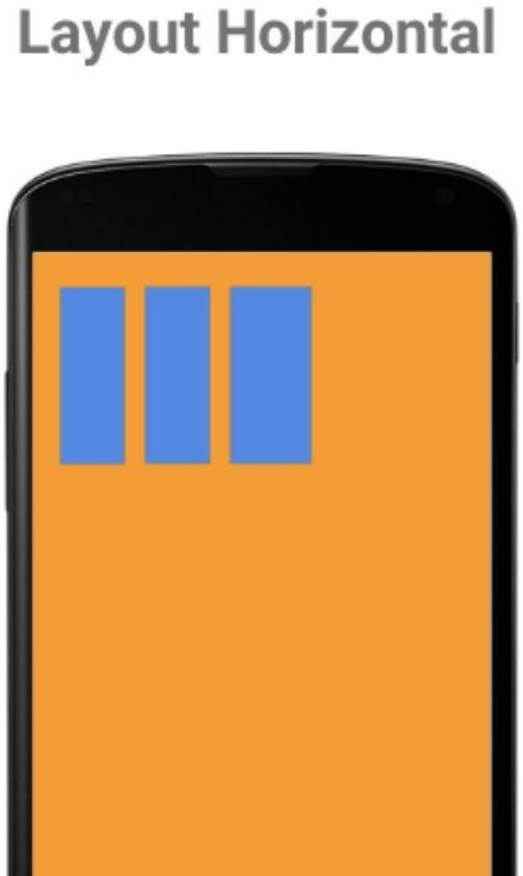
Layout Vertical:

```
<LinearLayout
    xmlns:android="..."
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TextView
        android:text="Ola  Androideiro!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp"    />
</LinearLayout>
```

# LinearLayout

Layout Horizontal:

```
<LinearLayout
    xmlns:android="..."
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TextView
        android:text="Ola  Androideiro!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp"    />
</LinearLayout>
```



# LinearLayout

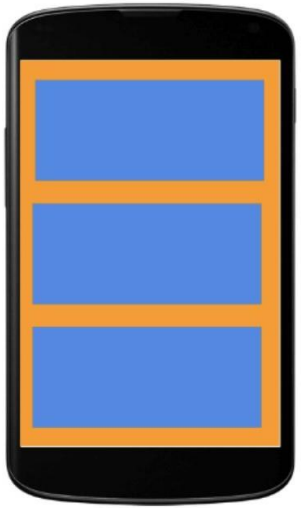
Além da orientação, outro atributo importante do *LinearLayout* é o ***android:layout_weight***, ele é responsável por definir o peso *(importância visual)* que cada View tem referente a distribuição dentro do *LinearLayout*.

A utilização desse atributo depende de como você está utilizando o LinearLayout, para um layout vertical, precisamos definir a **altura igual a zero e um peso para cada uma das Views**.

Caso esteja usando um layout horizontal, precisamos definir a **largura igual a zero e um peso para cada uma das Views**.

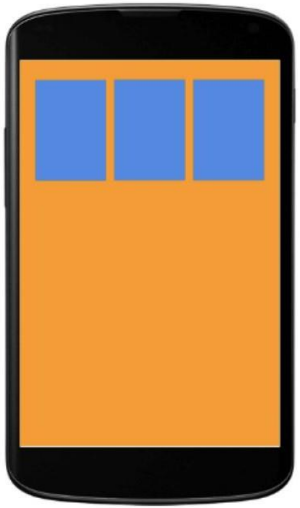
# LinearLayout

Layout Vertical



03 Views
- height: 0 dp
- weight: 1

Layout Horizontal



03 Views
- Width: 0 dp
- weight: 1

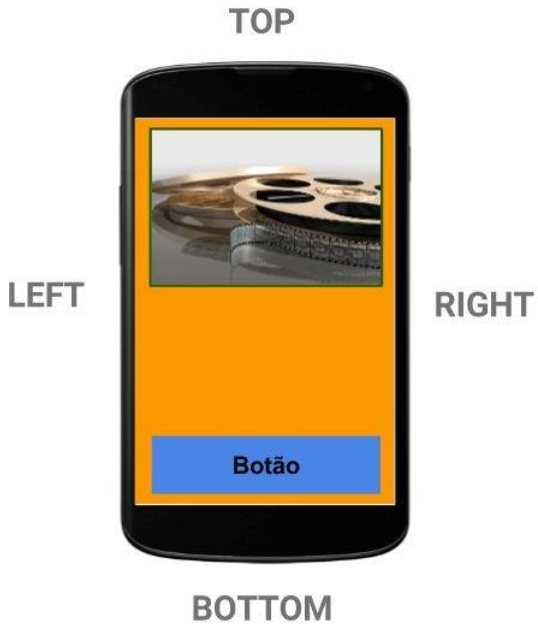
# RelativeLayout

Outro tipo de layout de uso bem comum é o **RelativeLayout**. A ideia deste layout é disponibilizar um posicionamento de elementos um pouco mais "flexível" não só apenas ou  "tudo na vertical" ou "tudo na horizontal".

Com o **RelativeLayout**, você pode posicionar os ***componentes*** em relação ao próprio layout, como por exemplo posicionar a ***View (componente)*** no topo ou no fim do layout. A outra opção, é posicionar as ***Views (componentes)*** em relação a outras Views dentro do mesmo RelativeLayout.

# **RelativeLayout:** Em relação ao Layout Pai

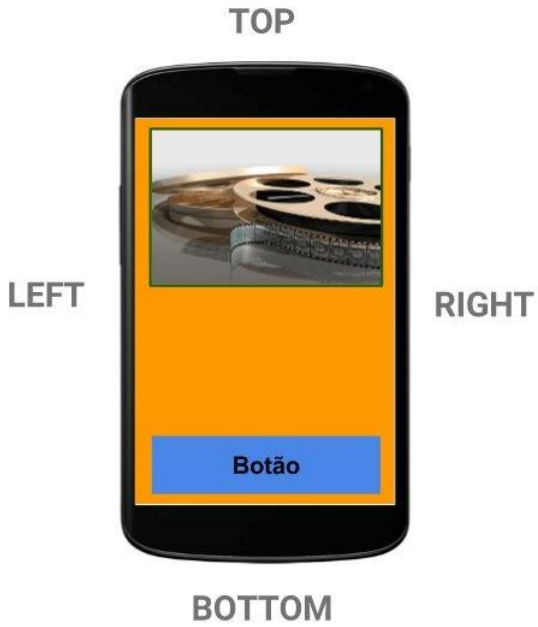
As ***Views (Componentes)*** dentro do **RelativeLayout** principal podem ser posicionadas as bordas **esquerda** do Pai, **superior**, **direita** ou **inferior**.



# **RelativeLayout:** Em relação ao Layout Pai

Os atributos utilizados para alinhamento nas bordas são:

- android:**layout_alignParentTop**="true/false"
- android:**layout_alignParentBottom**="true/false"
- android:**layout_alignParentLeft**="true/false"
- android:**layout_alignParentRight**="true/false"
- android:**layout_centerHorizontal**="true/false"
- android:**layout_centerVertical**="true/false"



# **RelativeLayout:** Em relação a outros componentes

As ***Views (Componentes)*** podem ser posicionadas em relação a outras Views, é possível adicionar restrições a sua posição. Por exemplo, um **TextView** deve estar acima de outra **TextView**, ou um **ImageView** deve ser a a esquerda de uma outra **TextView**.

# RelativeLayout

```
<RelativeLayout
    xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/froyo_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:textSize="24sp"
        android:text="Froyo" />
</RelativeLayout>
```

# GridLayout

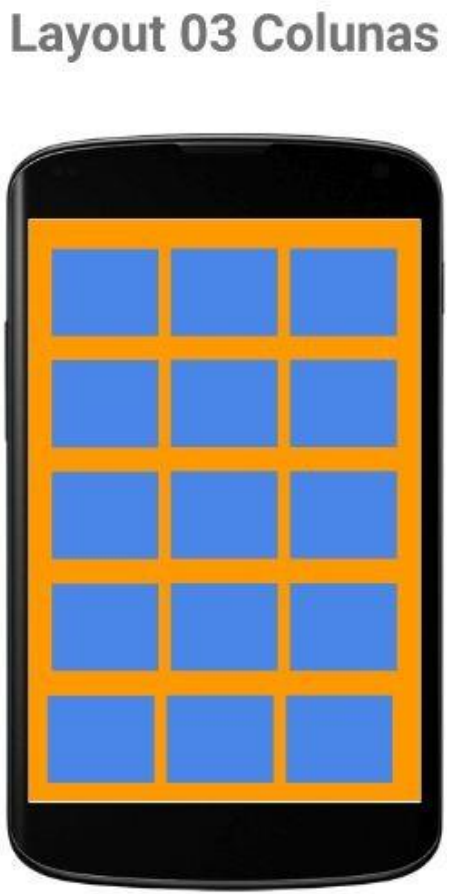
O objetivo do **GridLayout** é permitir posicionar as ***Views (Componentes)*** em uma disposição de grade. Basicamente consiste em um número de linhas de horizontais e verticais que servem para dividir a visualização do layout em forma de “grade”, com cada linha e coluna formando uma célula que pode, por sua vez, conter uma ou mais Views.

As linhas e colunas são definidas utilizando os atributos ***android:columnCount*** e ***android:rowCount***.

# GridLayout

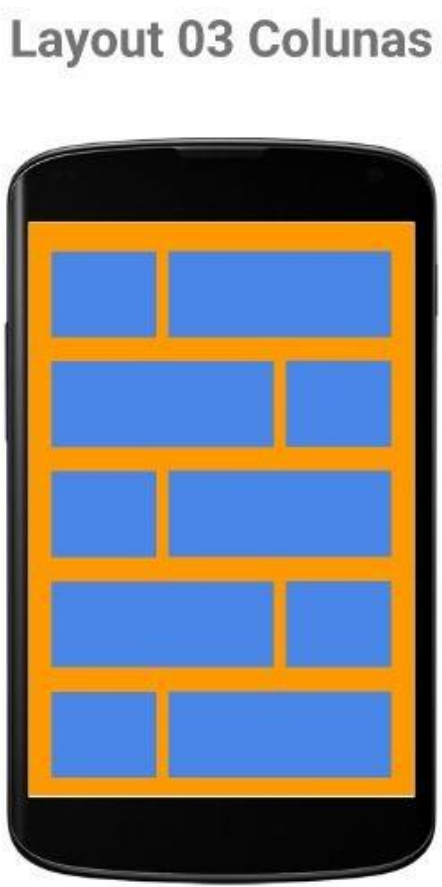
Na imagem ao lado podemos ver como fica a configuração de ***android:columnCount="3"*** e ***android:rowCount="5"***.

Cada linha da grade é referenciada por índices, que são numeradas a partir de 0 contando de baixo para cima. As células (linha/coluna) também tem numeração e começam em 0 a partir do da célula no canto superior esquerdo da grade.



# **GridLayout:** RowSpan e ColumnSpan

Uma ***View Filha (Componente dentro de Layout)*** também pode ser configurada para ocupar várias linhas e colunas do ***GridLayout Pai*** através do uso dos atributos ***android:layout_rowSpan*** e ***android:layout_columnSpan***.



# GridLayout

```
<GridLayout xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="4"
    android:rowCount="2"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_columnSpan="2"
        android:text="Botão 01" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botão 02" />
</GridLayout>
```

# FrameLayout

O **FrameLayout** têm um objetivo muito simples, "*empilhar*" ***Views (Componentes)*** um em cima do outro, como camadas sobrepostas.

É possível posicionar os elementos filhos do **FrameLayout** por toda a extensão dele, mas por padrão os elementos são empilhados *(quando levando em consideração o eixo Z)* independente do posicionamento nos eixos X e Y.

No seu uso padrão o último elemento adicionado dentro do layout é o que ficará por cima de todos os outros.

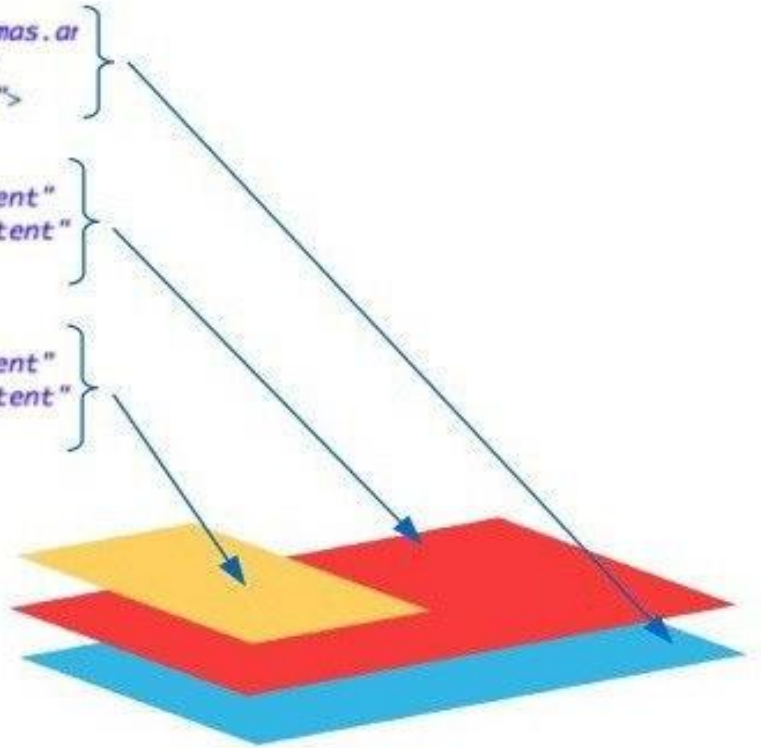
# FrameLayout

```
<FrameLayout xmlns:android="http://schemas.a
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

</FrameLayout>
```



# WebView

A **WebView** não é exatamente um layout, pode ser considerada mais um componente. Porém diversos aplicativos utilizam ela de forma que ocupe a tela toda, logo semelhante a um layout.

A ideia da **WebView** é criar uma janela web, uma espécie de navegador "embutido" dentro de sua aplicação Android, apenas mostrando o conteúdo de um endereço na web, sem barras de ferramenta, endereço ou botões de voltar e avançar.

Dependendo do tipo de conteúdo carregado em uma webview dificilmente o usuário notará que é uma página da web ao invés de um conteúdo nativo do aplicativo.

# WebView

A classe **WebView** é uma extensão da classe ***View*** do Android, que permite exibir páginas da Web como parte do layout de atividades.

Usar **WebView** pode ser útil quando você quer fornecer informações que talvez precisem ser atualizadas, por exemplo, como um contrato de usuário final ou um guia do usuário. No app para Android, você pode criar uma *Activity* que contenha uma **WebView** e usá-la para exibir o documento hospedado on-line.

# WebView

```
<WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />
```

O jeito mais comum de determinar qual URL uma webview irá abrir é acessando a mesma programaticamente e utilizando o método **loadUrl**.

# WebView

O jeito mais comum de determinar qual URL uma webview irá abrir é acessando a mesma programaticamente e utilizando o método **loadUrl**.

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

# Outros Layouts

# ScrollView e HorizontalScrollView

Sempre que você precisa dispor em um app conteúdo que é maior que a tela do seu usuário, será necessário criar uma "rolagem". Para este tipo de interação existe o layout/componente **ScrollView**.

O componente ***ScrollView*** aceita apenas uma View filha direta, logo recomenda se que você adicione um **LinearLayout** ou **RelativeLayout** como filho da ScrollView e administre o conteúdo da tela dentro destes últimos.

Caso você precise de uma rolagem horizontal o componente indicado é o **HorizontalScrollView**.

# ListView e RecyclerView

Você precisa dispor em tela uma lista enorme de produtos? Uma lista de contatos? Fazer um carrinho de compras que pode ter 1 ou até 100 itens? Normalmente sempre que temos um componente em comum que se repete por várias vezes (um abaixo do outro normalmente) o componente indicado é a **ListView**.

O layout **ListView** serve para listar de maneira repetitiva o mesmo componente, e assim como o componente ScrollView, também criará uma barra de rolagem caso necessário.

Ainda na mesma necessidade, nos últimas versões do Android surgiu o componente RecyclerView um pouco mais complexo de ser implementado, porém com os mesmos objetivos da ListView e segundo o Google muito mais performance.

# TabLayout e ViewPager

Se você deseja criar uma clássica navegação por abas, você deverá combinar o componente **TabLayout** e o layout **ViewPager**.

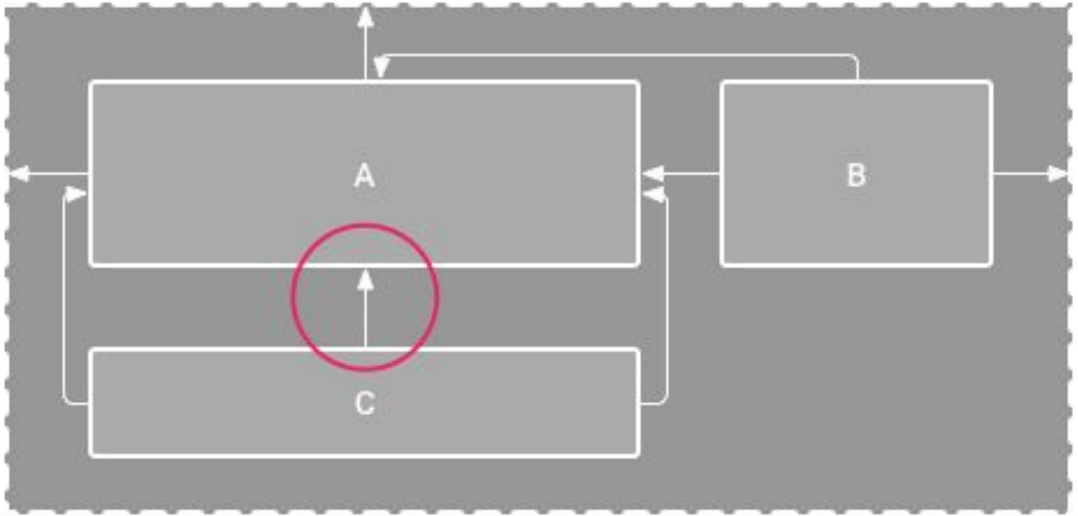
Nesta combinação o **TabLayout** fica com parte dos "botões de aba" e o **ViewPager** cuida da transição de conteúdo entre as telas de cada aba.

# ConstraintLayout

O **ConstraintLayout** permite que você crie layouts grandes e complexos com uma hierarquia de visualização plana (sem grupos de visualização aninhados).

Ele é semelhante a ***RelativeLayout***: todas as visualizações são dispostas de acordo com as relações entre visualizações irmãs e layout pai, mas são mais flexíveis que RelativeLayout e mais fáceis de usar com o **Layout Editor do Android Studio**.

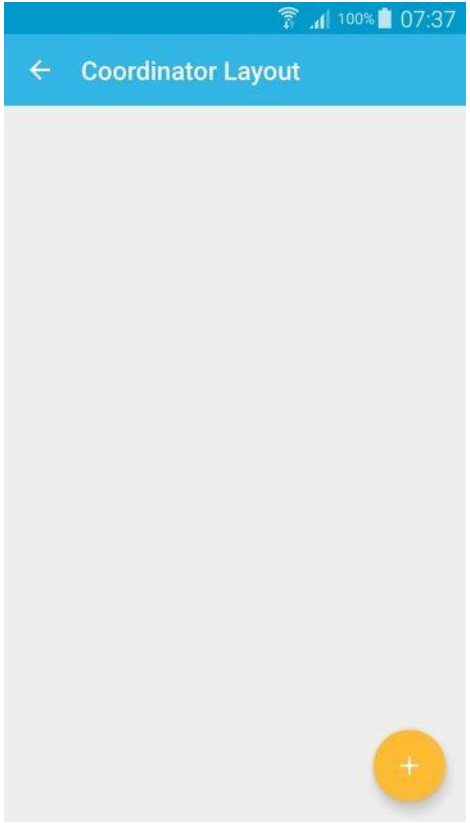
# ConstraintLayout



# CoordinatorLayout

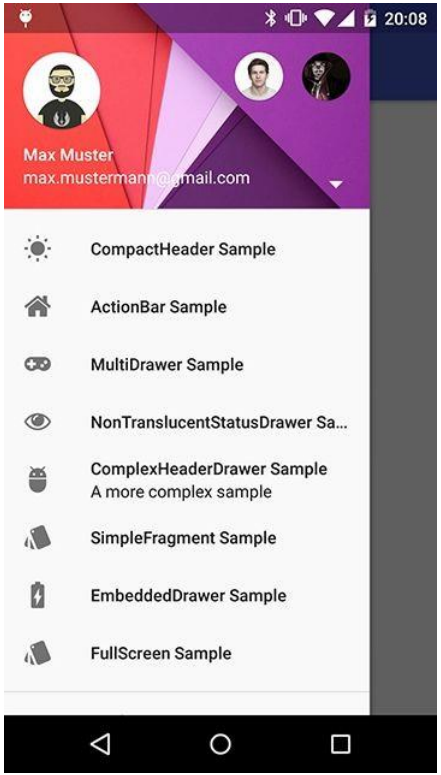
Um **CoordinatorLayout** têm uma função bem específica como layout. De acordo com a documentação do google ele é um  FrameLayout com superpoderes.

A ideia principal do **CoordinatorLayout** é agregar um layout base, normalmente com barra de topo e algum elemento flutuante como um botão de ação.



# DrawerLayout

O **DrawerLayout** em combinação com a **NavigationView** servem basicamente para criar uma navegação clássica com menu do tipo Drawer. Aquela caixa suspensa que é "sacada" da lateral e passa por cima de todo o conteúdo.



Engenharia da Computação

# Unidades de medida, Tamanhos e posicionamento

# Unidades de medida

Nos poucos exemplos que foram demonstrados nesta aula foi possível notar que em alguns casos ao invés de usar pixels ou centímetros uma nova unidade foi definida para setar largura, altura, padding e margin. Essa unidade é o **DP**.

Breves definições:

- **px** é pixel ~ mesmo usado na WEB.
- **sp** é scale-independent pixels (pixels independente de escala).
- **dip** ou **dp** é density-independent pixels (pixels independente de densidade).

# Unidades de medida - dp

**dp**: É uma unidade abstrata que se baseia na densidade física da tela. Estas unidades são relativas a ***160dpi*** *(pontos por polegada)* de tela, em que **1dp** é aproximadamente igual a **1px**. Quando executado em uma tela de maior densidade, o número de pixels usados para desenhar 1dp é dimensionada por um adequado fator da tela.

A proporção de dp para pixel pode mudar com a densidade da tela, mas não necessariamente em proporção direta. Usando unidades **dp** *(em vez de unidades de pixel)* é uma solução simples para a criação de views em  seu layout capazes de redimensionar adequadamente para diferentes densidades de tela.

# Unidades de medida - sp

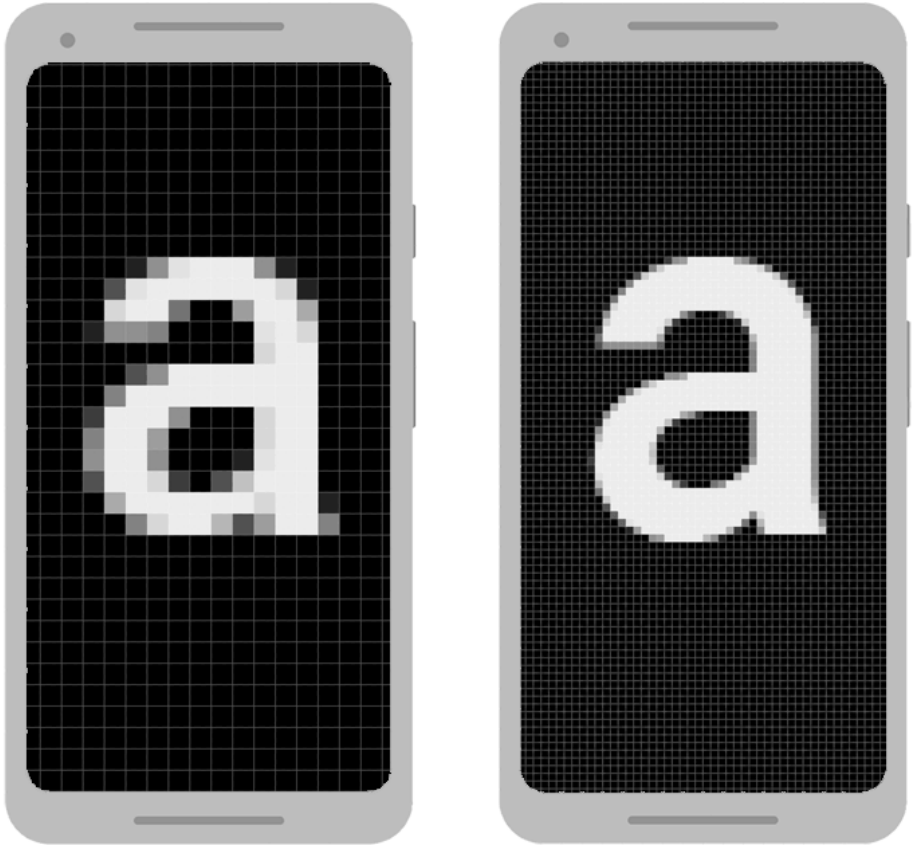
**sp**: Este é como a unidade de dp, mas também é escalado pelo tamanho da fonte a preferência do utilizador.

É recomendável utilizar esta unidade ao especificar tamanhos de fonte, de modo que será ajustado tanto para a densidade da tela quanto para preferência do usuário.

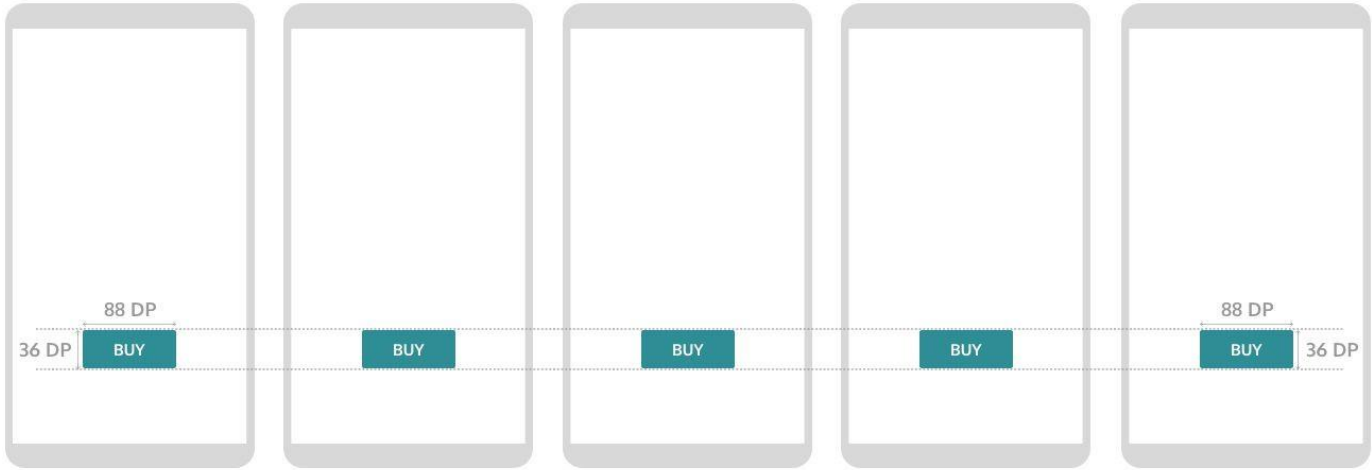
# Unidades de medida - px

**px**: Corresponde à pixels reais da tela.

Esta unidade de medida não é recomendado porque a representação real pode variar entre os dispositivos; cada um dos dispositivos podem ter um número diferente de pixels por polegada e podem ter mais ou menos total de pixels disponíveis na tela.



# Density-independent-pixels (DP)




| 1x | 1.5x | 2x | 3x | 4x |
|---|---|---|---|---|
| (~160 DPI) | (~240 DPI) | (~320 DPI) | (~480 DPI) | (~640 DPI) |
| 1 DP = 1 PX | 1 DP = 1.5 PX | 1 DP = 2 PX | 1 DP = 3 PX | 1 DP = 4 PX |
| 88 x 36 DP = 88 x 36 PX | 88 x 36 DP = 132 x 54 PX | 88 x 36 DP = 176 x 72 PX | 88 x 36 DP = 264 x 108 PX | 88 x 36 DP = 352 x 144 PX |

# Unidades de medida

**Utilitário:**

**https://pixplicity.com/dp-px-converter**

https://developer.android.com/training/multiscreen/screendensities?hl=pt-br

# Textos use SP

Sempre devemos utilizar a unidade de medida SP para declarar tamanhos de textos.

O usuário Android pode altar o tamanho de fonte base nas configurações do Smartphone ou ainda, algumas opções de acessibilidade podem permitir o aumento do “tamanho do texto”.

Ao utilizar a unidade sp o sistema Android irá renderizar um tamanho de fonte considerando as alterações feitas pelo usuário.

```
<TextView android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20sp" />
```

# Definindo o tamanho com Width e Height

Sempre utilizaremos os atributos ***android:layout_width*** e ***android:layout_height*** para definir largura e altura respectivamente.

Para definir a altura e largura fixa das Views (Componentes), podemos usar um valor fixo com a medida em DP (ex: 150dp).

Outra opção é fazer o tamanho da Views (Componentes) se adequar ao conteúdo utilizando ***wrap_content*** ou fazer a View expandir ao máximo ocupando todo espaço disponível no layout pai utilizando ***match_parent***.

# Definindo o tamanho com Width e Height

```
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/clickme"
    android:layout_marginTop="20dp" />
```

```
<TextView android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20sp" />
```

# Definindo o tamanho com Width e Height

150 dp


wrap_content


match_content

# Espaços com Padding e Margin

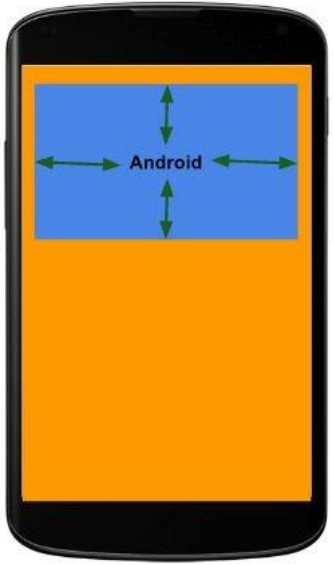
Caso você necessite de acrescentar um pouco de espaço entre as ***Views*** poderá utilizar dois atributos chamado ***android:padding*** e ***android:layout_margin***. Estes se assemelham muito ao comportamento de margin e padding do CSS para WEB e também devem receber valores inteiros com a unidade de medida DP.

O **Padding** é o espaço dentro da View, entre a borda e o conteúdo. Podemos adicionar esse espaço na parte superior, inferior, nos lados direito e esquerdo.

A **Margin** são basicamente as margens das Views, entre a parte de fora e os outros elementos próximos a View. Podemos adicionar margens na parte superior, inferior, direita e esquerda.

# Espaços com Padding e Margin

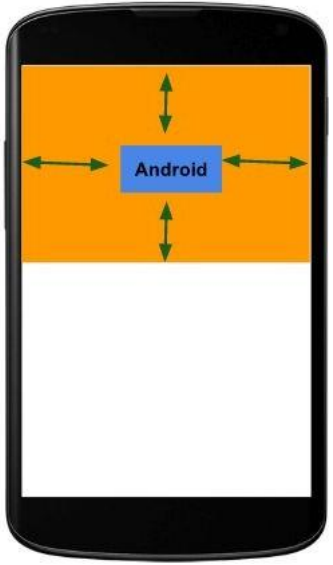
Padding



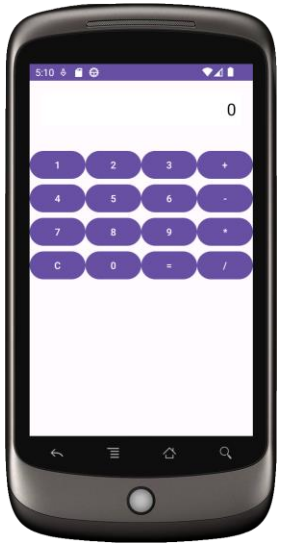

TextView

Android

Margin



# Exercício 1 (parte 1)




Desenvolva o layout de uma calculadora simples no Android Studio.

Aplique os conceitos que aprendemos:

- Linear layout, Relative layout, etc.
- Padding e Margin
- Defina tamanhos com Width e Height
- Não é necessário desenvolver a parte lógica (funcionamento da calculadora). Neste momento vamos apenas criar o layout.

# obrigado 🚀