

Programação para Dispositivos Móveis

Conteúdos

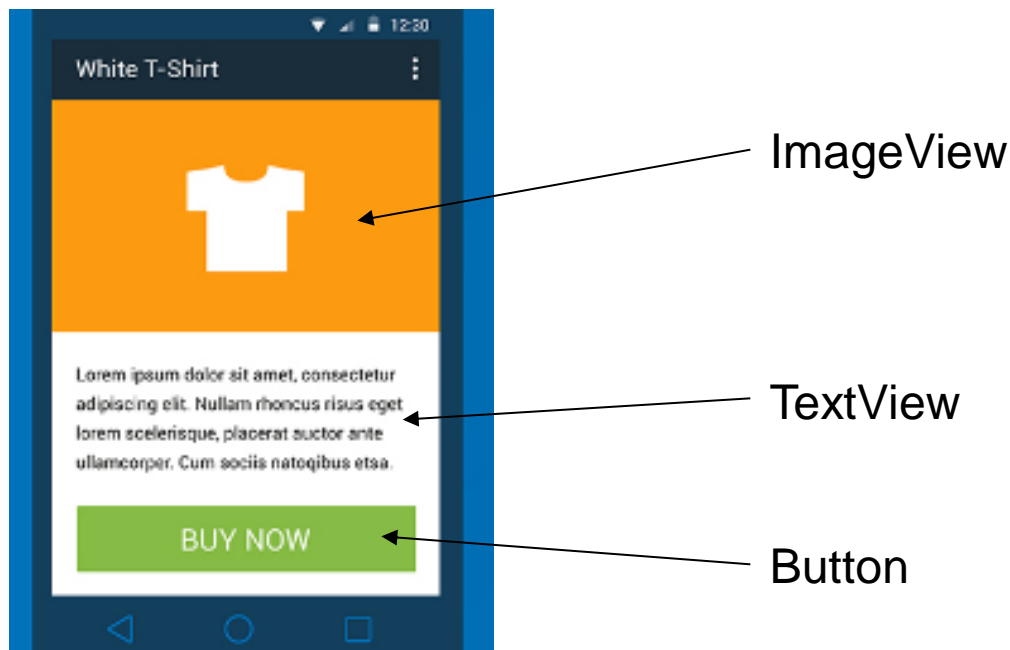
- Layouts (linear, relativo, de grade, de tabela, frame, de constraints, scrollview).
- Unidades para tamanhos (pixels, dp, sp etc.).
- Atributos: layout width, layout height, layout weight, padding e margin.
- Elementos da UI: as classes TextView, EditText, ImageView, RadioButton, RadioGroup, Checkbox, Button.
- Exemplos e exercícios.

Layouts

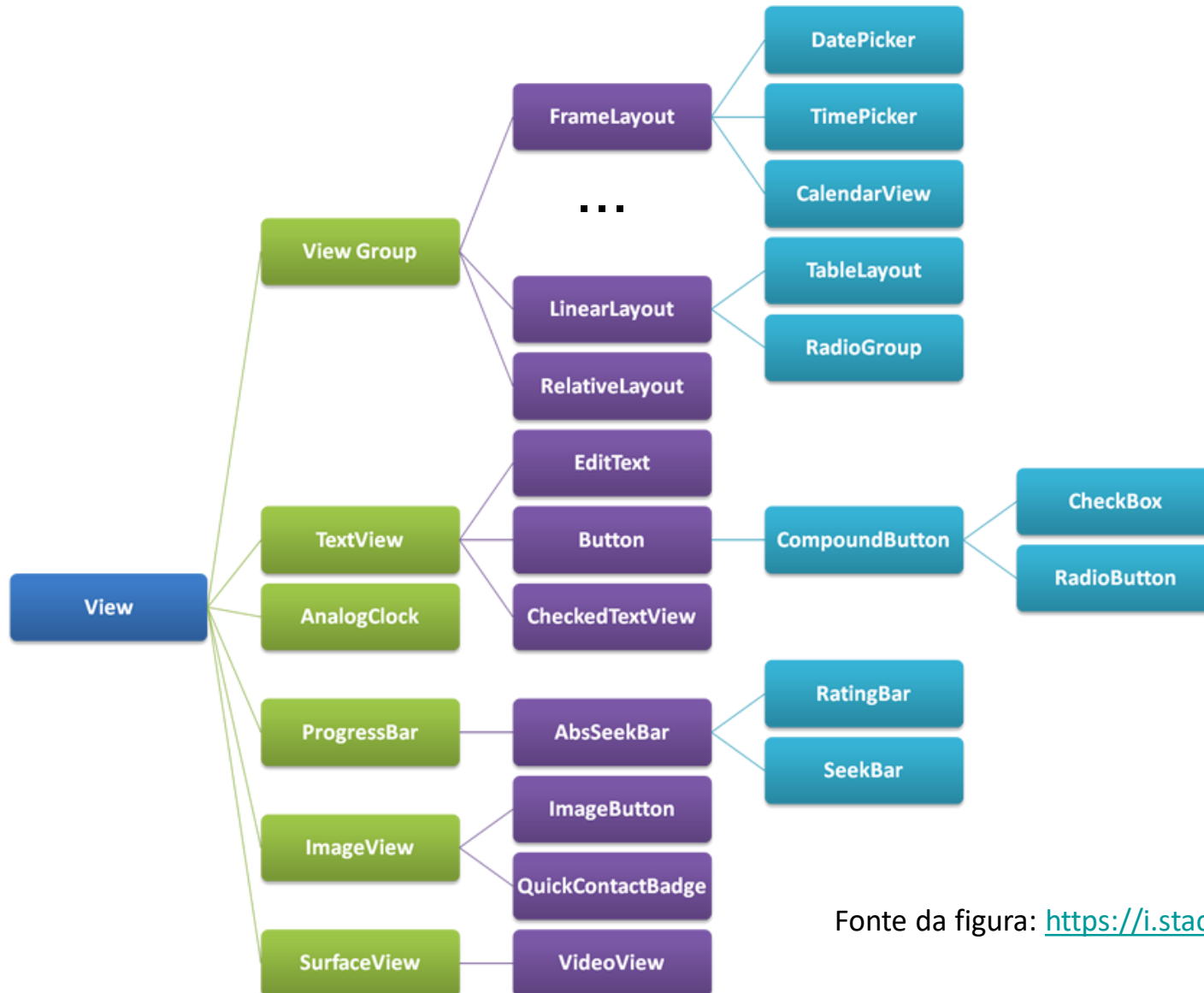
- Podemos criar os layouts de duas formas em nossas aplicações:
 1. Declarando os elementos de UI em um arquivo XML.
 2. Instanciar os elementos do layout em tempo de execução, direto no código.
- Iremos utilizar a primeira opção, pois, quando criamos nossa interface em **XML, separamos a apresentação (UI – User Interface) da aplicação (código que controla o aplicativo)**. Com isso podemos modificar nossos layouts sem ter que alterar nossos códigos, como por exemplo, podemos fazer diferentes layouts para diferentes tipos de telas, orientação, para vários idiomas etc. Também, desta forma aproveitaremos as vantagens de efetuar um projeto visual com os recursos avançados do Android Studio.

View

- Uma View é um bloco visível na tela, possui largura e altura como atributos básicos.
- Representa um componente básico da interface do usuário (botão, caixa de texto, etc).
- Muitas classes derivam de View. Veja no próximo slide.



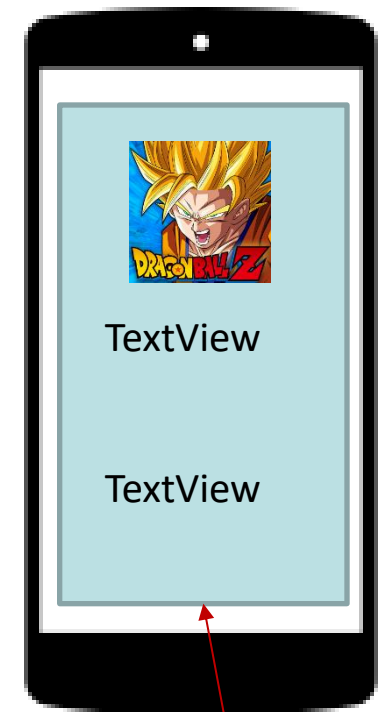
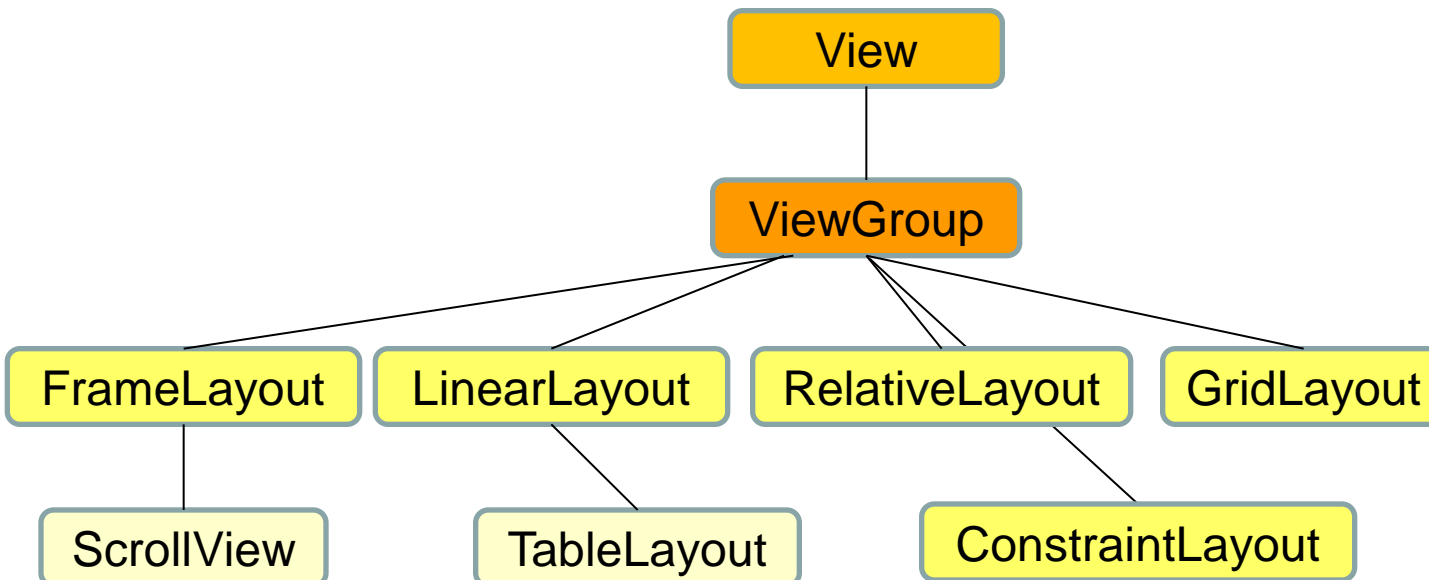
Hierarquia de classes a partir de View



Fonte da figura: <https://i.stack.imgur.com/1Km66.png>

Hierarquia de classes (layouts)

- Um layout (interface) é normalmente um objeto de uma classe derivada das classes View e ViewGroup.
- Alguns dos layouts que iremos utilizar são classes derivadas da classe ViewGroup. Os elementos (TextView, Button etc.) são classes derivadas direta ou indiretamente da classe View e serão utilizados como componentes de algum ViewGroup.



Classe derivada de
ViewGroup

Telas e resoluções

- No surgimento do Android, as telas dos dispositivos possuíam resoluções padrões. Com isso era fácil criar as UI. Basicamente se usava o layout `AbsoluteLayout`.
- Com a evolução dos aparelhos, as telas ficaram mais avançadas o que trouxe a complexidade de criar layouts que se adaptassem aos diferentes tamanhos e resoluções.
- Assim como em aplicações Web, também devemos pensar em nossos layouts para Android como responsivos.

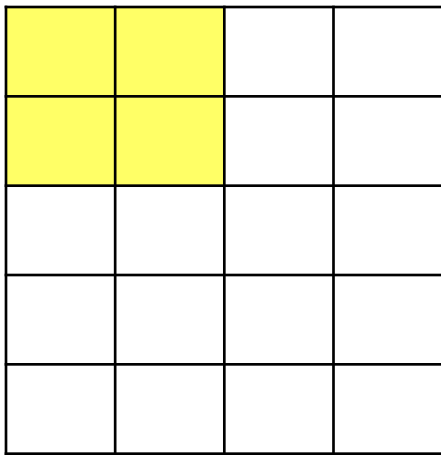
Unidades (usadas para tamanhos, dimensões)

- **px** (pixels)
corresponde ao número de pixels da tela, **evite seu uso**
- **in** (polegadas) e mm (milímetros)
baseadas no tamanho físico da tela
- **pt** (pontos)
1/72 de uma polegada, também baseado no tamanho físico da tela
- **dp (density-independent pixels, dp ou dip)**
Se baseia na densidade física da tela. Utilizar para garantir portabilidade. É uma medida baseada numa tela de 160 dpi (pontos por polegada): o sistema manipula valores em dp calculando os px equivalentes para um dispositivo específico com determinada densidade dpi com a fórmula:
$$px = dp * (dpi / 160)$$
- **sp (scale-independent pixels)**
Esta é como a unidade de dp, mas também é modificado pelo tamanho da fonte de preferência do usuário. É recomendável utilizar esta unidade ao especificar tamanhos de fontes de letras, de modo que será ajustado tanto para a densidade da tela quanto para preferência do usuário.

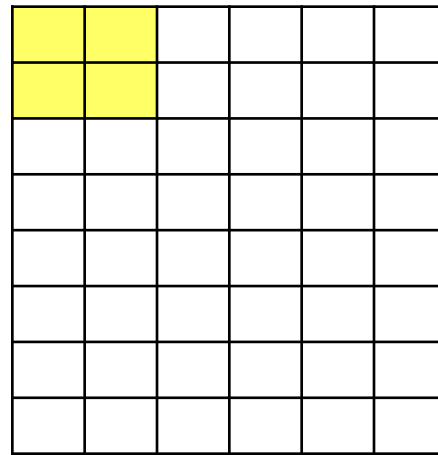
Unidades - **pixels** (evite utilizar)

os itens ficam diferentes dependendo da resolução do dispositivo

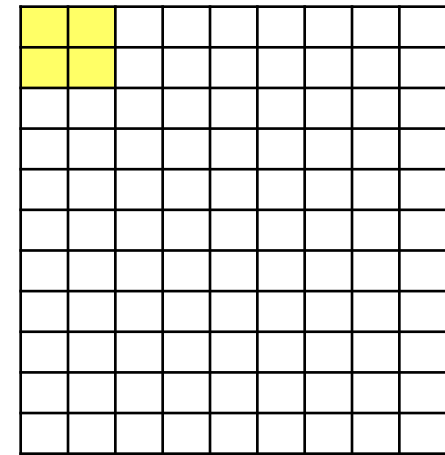
por exemplo, uma figura de 2 x 2 pixels ficaria assim em diferentes aparelhos:



média



alta



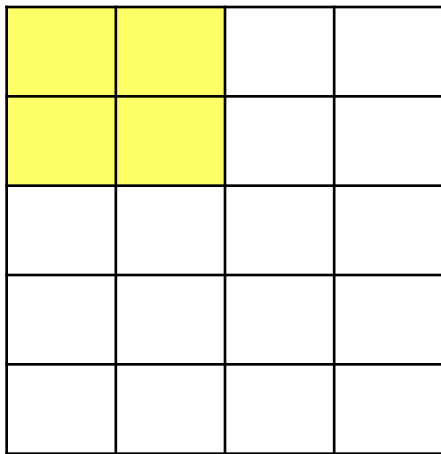
extra-alta

Resolução do dispositivo

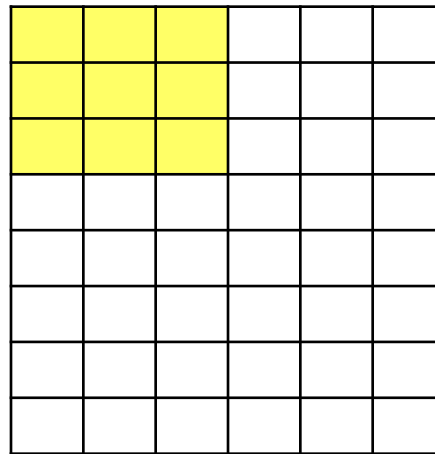
Unidades - **dp** e **sp** (preferidas)

os itens ficam semelhantes, independente da resolução do dispositivo

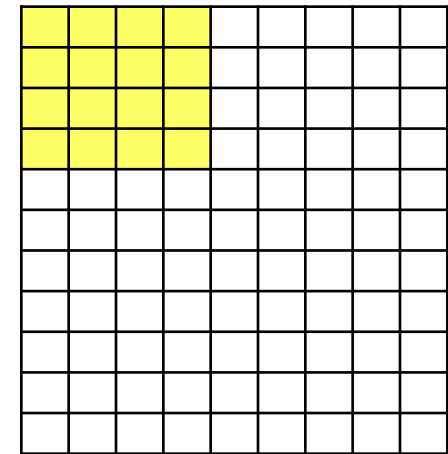
por exemplo, uma figura de 2 x 2 pixels ficaria assim em diferentes aparelhos



média



alta



extra-alta

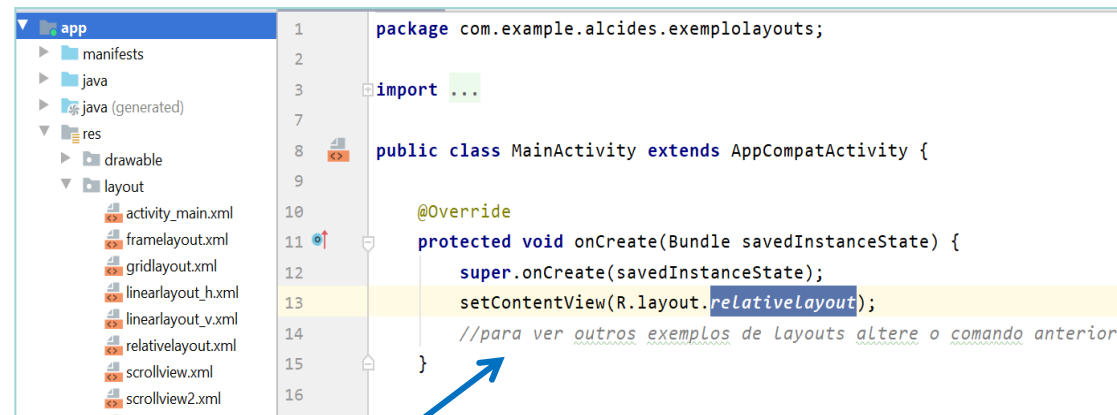
Resolução do dispositivo

$px = dp * (dpi / 160)$. Por exemplo, em uma tela de 240 dpi, 1 dp equivale a 1,5 pixels físicos. (https://developer.android.com/guide/practices/screens_support.html?hl=pt-br)

Alguns dos tipos de layouts

- Alguns layouts comuns são:

- LinearLayout
- RelativeLayout
- FrameLayout
- GridLayout, TableLayout
- ConstraintLayout

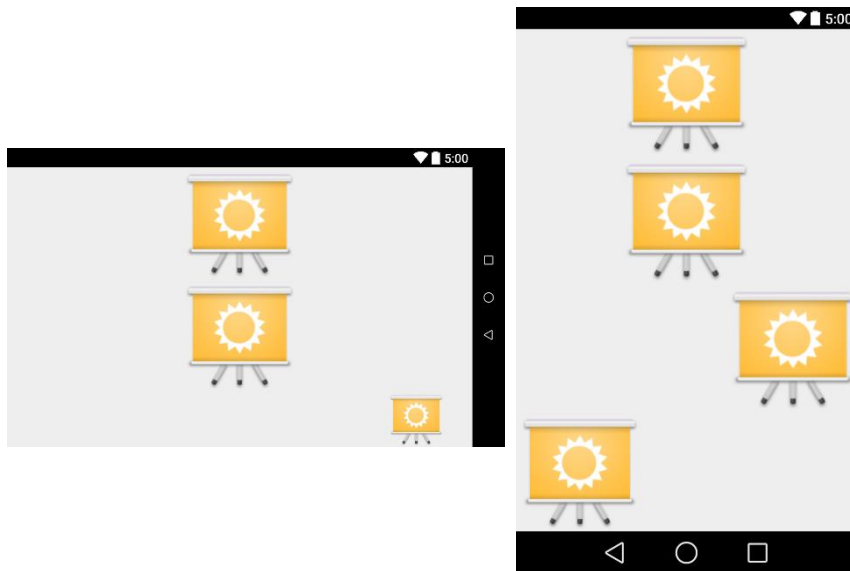


- É possível trabalhar com esquemas de layouts dentro de outros, por exemplo, um LinearLayout dentro de um RelativeLayout.

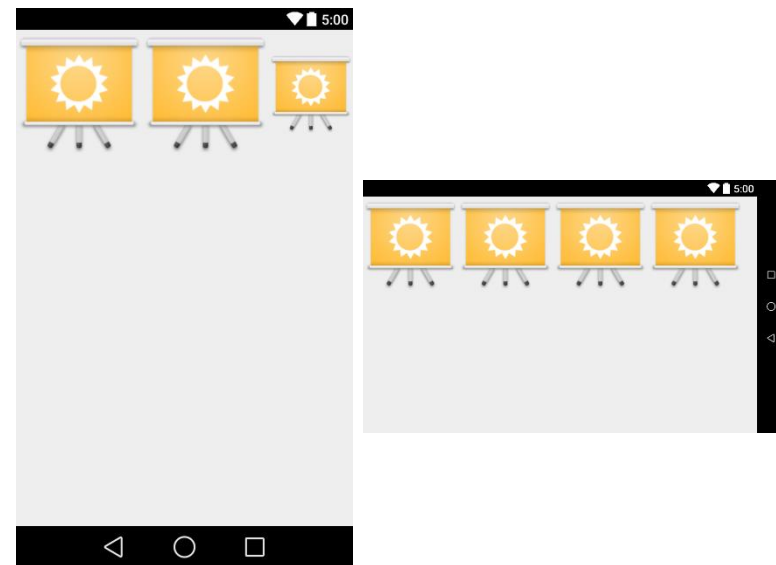
Veja um exemplo que demonstra diferentes tipos de layouts no projeto **ExemploLayouts**. Para selecionar outro layout altere o comando `setContentView(R.layout.activity_main);` no arquivo `MainActivity.java`.

LinearLayout (API 1)

- LinearLayout organiza os elementos de forma linear, podendo ser de maneira horizontal ou vertical.
- É possível configurar o layout para dispor os elementos proporcionalmente.
- Os elementos que ultrapassam o tamanho da tela ficam ocultos.



vertical



horizontal

LinearLayout (API 1)

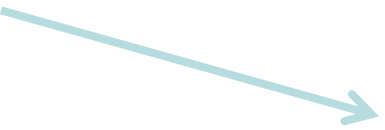
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
```

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/tempo" />
```

```
<ImageView
    android:id="@+id/imageView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/tempo" />
```

...

```
</LinearLayout>
```



layout linear com
orientação horizontal

Veja o exemplo no
projeto ExemploLayouts
(use *linearlayout_h.xml*)

LinearLayout (API 1)

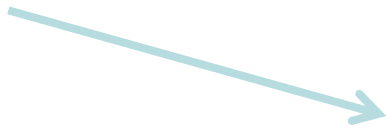
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:src="@drawable/tempo" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:src="@drawable/tempo" />

    ...

</LinearLayout>
```

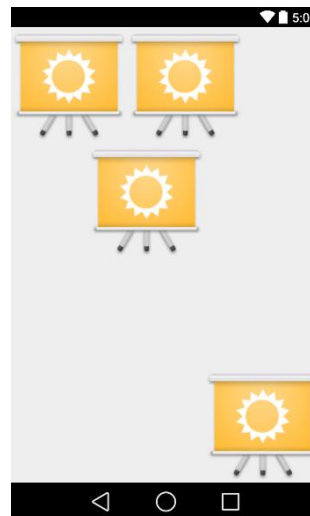


layout linear com
orientação vertical

Veja o exemplo no
projeto ExemploLayouts
(use *linearlayout_v.xml*)

RelativeLayout (API 1)

- RelativeLayout (layout relativo), poderoso porém complexo. No RelativeLayout posicionamos os elementos por referência (com relação) a outros elementos. Por exemplo, dizemos se o botão estará abaixo de um campo de texto, do lado direito ou até mesmo em cima dele.
- Neste tipo de layout colocamos os itens em posições relativas a outros itens. As tags `android:layout_toRightOf`, `android:layout_toLeftOf`, `android:layout_below`, `android:layout_above` etc. permitem especificar a colocação relativa dos componentes.



RelativeLayout (API 1) - um exemplo

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

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/tempo" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/tempo"
        android:layout_below="@+id/imageView1"
        android:layout_toStartOf="@+id/imageView3" />

    <ImageView
        android:id="@+id/imageView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/tempo"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true" />

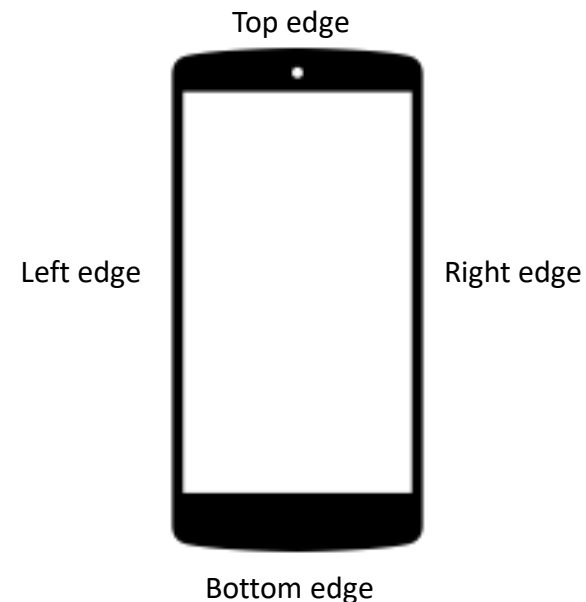
    <ImageView
        android:id="@+id/imageView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/tempo"
        android:layout_toRightOf="@+id/imageView1" />

</RelativeLayout>
```

Veja o exemplo no
projeto ExemploLayouts
(use relativelayout.xml)

RelativeLayout (API 1) - atributos de posicionamento

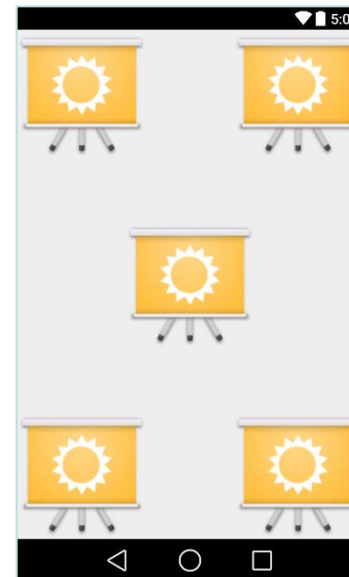
- Atributos com relação ao elemento pai (parent)
 - android:layout_alignParentBottom
 - android:layout_alignParentLeft
 - android:layout_alignParentRight
 - android:layout_alignParentTop
 - android:layout_centerHorizontal
 - android:layout_centerInParent
 - android:layout_centerVertical
- Atributos com relação a outro elemento
 - android:layout_above
 - android:layout_below
 - android:layout_toLeftOf
 - android:layout_toRightOf
- Atributos de alinhamento com base em outro elemento:
 - android:layout_alignBaseline
 - android:layout_alignBottom
 - android:layout_alignLeft
 - android:layout_alignRight
 - android:layout_alignTop



FrameLayout (API 1)

- Layout para visualização simples, utilizado quando queremos exibir somente um elemento (ListView por exemplo), entretanto podemos inserir nove (9) elementos e controlar sua posição com o atributo gravity, para especificar as posições disponíveis, mas tenha cuidado para não provocar sobreposição. As posições podem ser especificadas com as constantes: left, center_horizontal, right; top, center_vertical, bottom. Ver outras constantes (center etc.) em:

<https://developer.android.com/reference/android/widget/FrameLayout>).



FrameLayout (API 1) - um exemplo

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

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageView5"
        android:layout_gravity="center"
        android:src="@drawable/tempo" />

</FrameLayout>
```

Veja o exemplo no
projeto ExemploLayouts
(use framelayout.xml)

FrameLayout (API 1) - um exemplo

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

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

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView5"  
    android:layout_gravity="center"  
    android:src="@drawable/tempo" />
```

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView1"  
    android:layout_gravity="left|top"  
    android:src="@drawable/tempo" />
```

Veja o exemplo no projeto ExemploLayouts
(use *framelayout.xml*, tire os comentários
no xml)

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView7"  
    android:layout_gravity="left|bottom"  
    android:src="@drawable/tempo" />
```

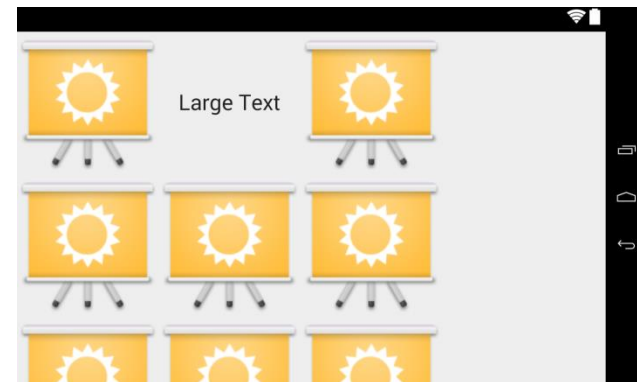
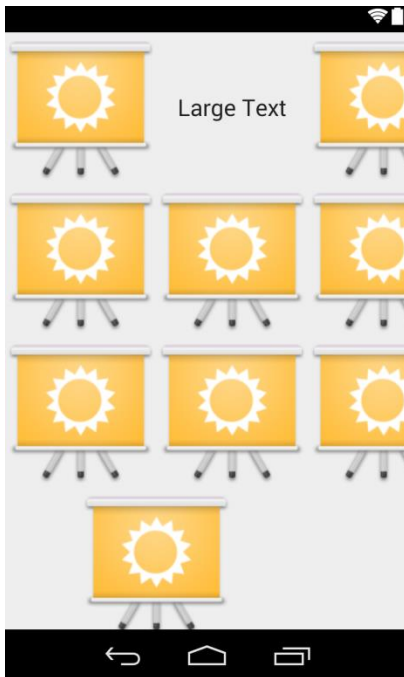
```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView9"  
    android:layout_gravity="right|bottom"  
    android:src="@drawable/tempo" />
```

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView3"  
    android:layout_gravity="right|top"  
    android:src="@drawable/tempo" />
```

```
</FrameLayout>
```

GridLayout (API 14)

- Dispõe os elementos em uma grade, formada logicamente por linhas e colunas.



GridLayout (API 14) - um exemplo

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="3"
    android:rowCount="4"
    android:orientation="horizontal"
    ><!-- ou vertical-->

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView0"
        android:src="@drawable/tempo"
        android:layout_row="0"
        android:layout_column="0"
    />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/textView"
        android:layout_row="0"
        android:layout_column="1"
        android:layout_gravity="center_vertical|center_horizontal"/>

    ...

</GridLayout>
```

Veja este exemplo no
projeto ExemploLayouts
(use gridlayout.xml)

TableLayout (API 1)

É semelhante ao GridLayout, mas a ordem dos elementos na 'tabela' estará dada pela ordem em que especificamos os componentes.



TableLayout (API 1) - um exemplo

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

    <TableRow>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imageView0"
            android:src="@drawable/tempo"
        />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imageView1"
            android:src="@drawable/tempo"
        />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imageView2"
            android:src="@drawable/tempo"
        />
    </TableRow>
```

Veja este exemplo no projeto ExemploLayouts (use tablelayout.xml)

```
<TableRow>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView3"
        android:src="@drawable/tempo"
    />
    <TextView
        android:id="@+id/password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Senha: "
    />
    <EditText
        android:id="@+id/txtpassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="90sp"
        android:text="Digite a senha aqui..."
    />
</TableRow>
</TableLayout>
```

ScrollView (mecanismo de rolagem)

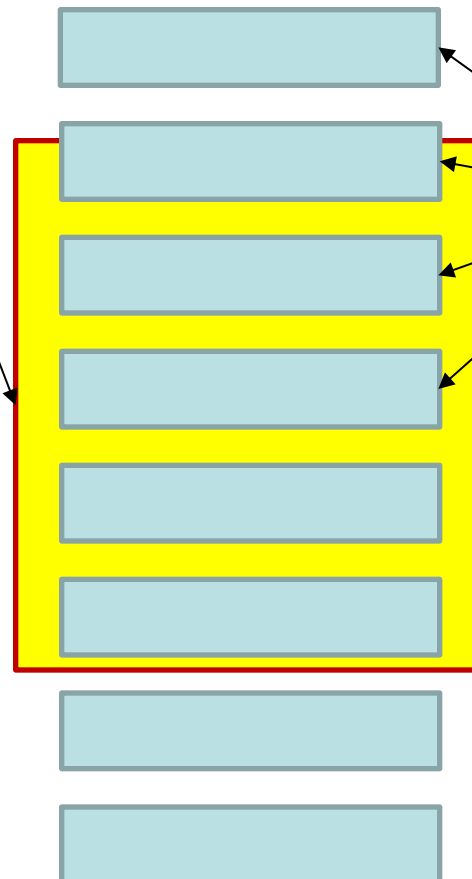
- Componente que permite rolar o conteúdo.
- Dentro dele só é permitido um filho, que poderá ser um esquema de layout como container para outros elementos.
- Existe também o HorizontalScrollView.

ScrollView

Componente que poderia dar problema, pois exige muita memória, o que tornaria a rolagem menos sutil.

Mesmo os elementos que não estão visíveis, são carregados para a memória.

Para listas longas, podemos resolver isso com AdapterView e usando ListView, GridView ou RecyclerView (veremos em uma próxima aula).



Elementos da View

ScrollView

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

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/scrollView" >

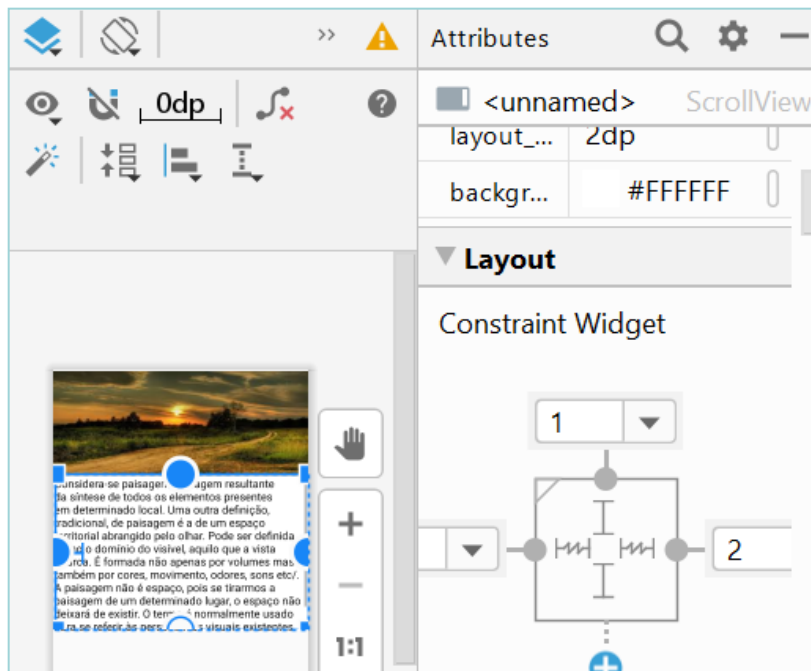
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <ImageView
                android:id="@+id/imageView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:src="@drawable/tempo" />

            ...
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

Veja este exemplo no projeto ExemploLayouts (use scrollView.xml ou scrollView2.xml)

ConstraintLayout

- Com o Android Studio 2.2 (ou superior) o Google adicionou uma nova ferramenta visual para processar o layout ConstraintLayout (classe derivada da classe ViewGroup), o que facilita a criação de uma UI responsiva. Embora seja novo, muitos dos templates dos projetos do Android Studio são baseados neste tipo de layout.

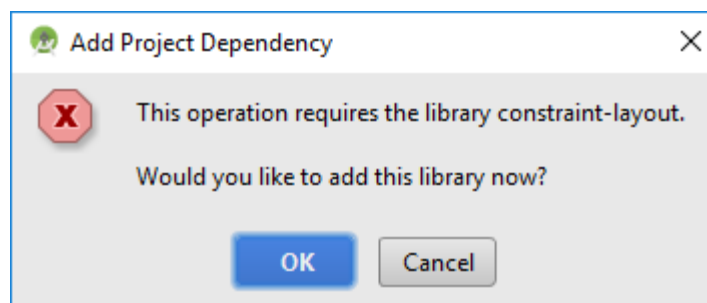
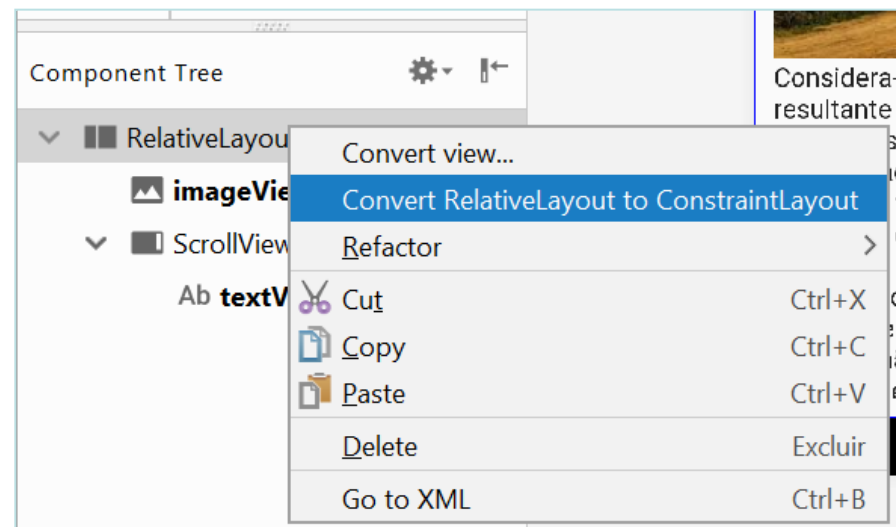
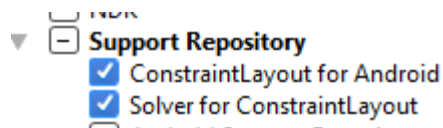


Para estabelecer as 'restrições', selecione o componente e procure a ficha **Layout** em **Attributes**. Clicando nos botões **+** pode selecionar ou não alguma restrição, e alterar os valores das margens nos quatro campos de texto.

Constraint Layout

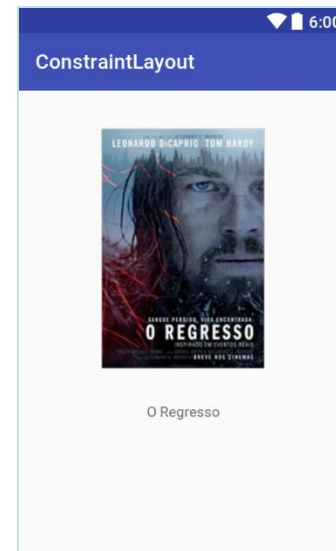
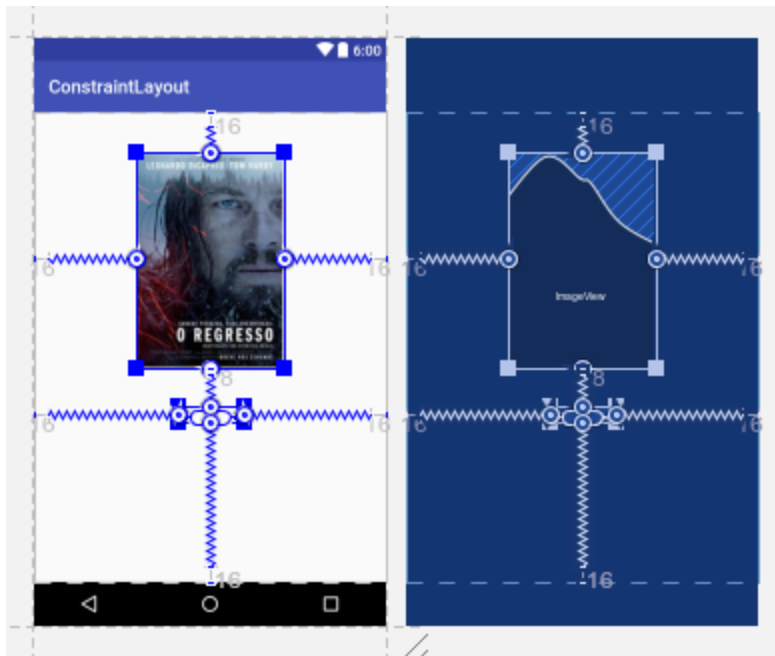
- É possível fazer a conversão do Relative Layout (ou outro layout) para o Constraint Layout.

Obs: Poderia ser necessário adicionar essa biblioteca. Se for necessário acesse o SDK Manager na aba SDK Tools e escolha Support Repository.

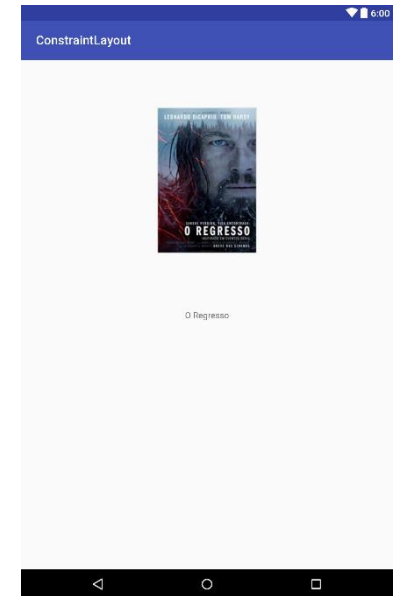


Constraint Layout

- Insira um ImageView e um TextView conforme imagem abaixo e adicione as restrições. Lembre-se que cada View deve ter pelo menos duas restrições: uma horizontal e uma vertical. Observe os pequenos círculos nos lados dos componentes, que serão utilizados para estabelecer as restrições.



Nexus S



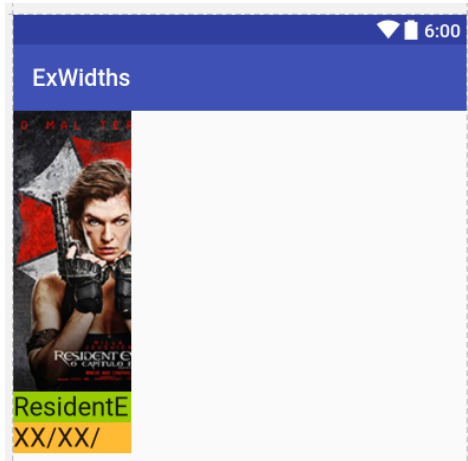
Nexus 7

Obs: Nas novas versões do ConstraintLayout é possível colocar o mesmo dentro de um ScrollView.

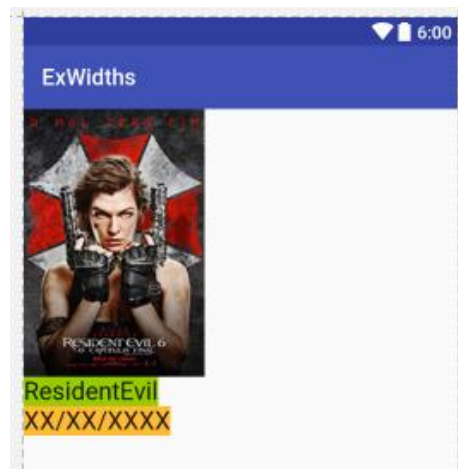
Atributos `layout_width` e `layout_height`

- Todos os elementos da UI deverão possuir os atributos: `layout_width` e `layout_height`.
- Para definir esses atributos, se utilizam os valores:
 - `wrap_content` – especifica para o elemento que sua dimensão será ajustada conforme seu conteúdo
 - `match_parent` (antigo `fill_parent`) – diz para o elemento ocupar a região máxima que seu elemento pai (contêiner) permitir
- Definir esses atributos com valores **em pixels não é aconselhável**.
- Se for colocar valores numéricos, **utilize unidades dp**, com isso você garante que sua UI será exibida corretamente em uma variedade de tamanhos de tela.

Layout width (atributo layout_width)



100dp



wrap_content



match_parent

Veja este exemplo no projeto ExWidths.

Altere o comando a seguir para testar os três casos anteriores:

```
setContentView(R.layout.activity_main);  
setContentView(R.layout.activity_main_2);  
setContentView(R.layout.activity_main_3);
```

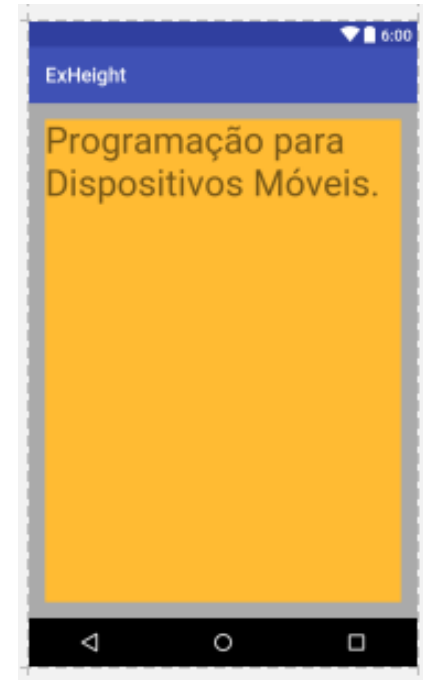
Layout height (atributo layout_height)



200dp



wrap_content



match_parent

Veja este exemplo no projeto ExHeight.

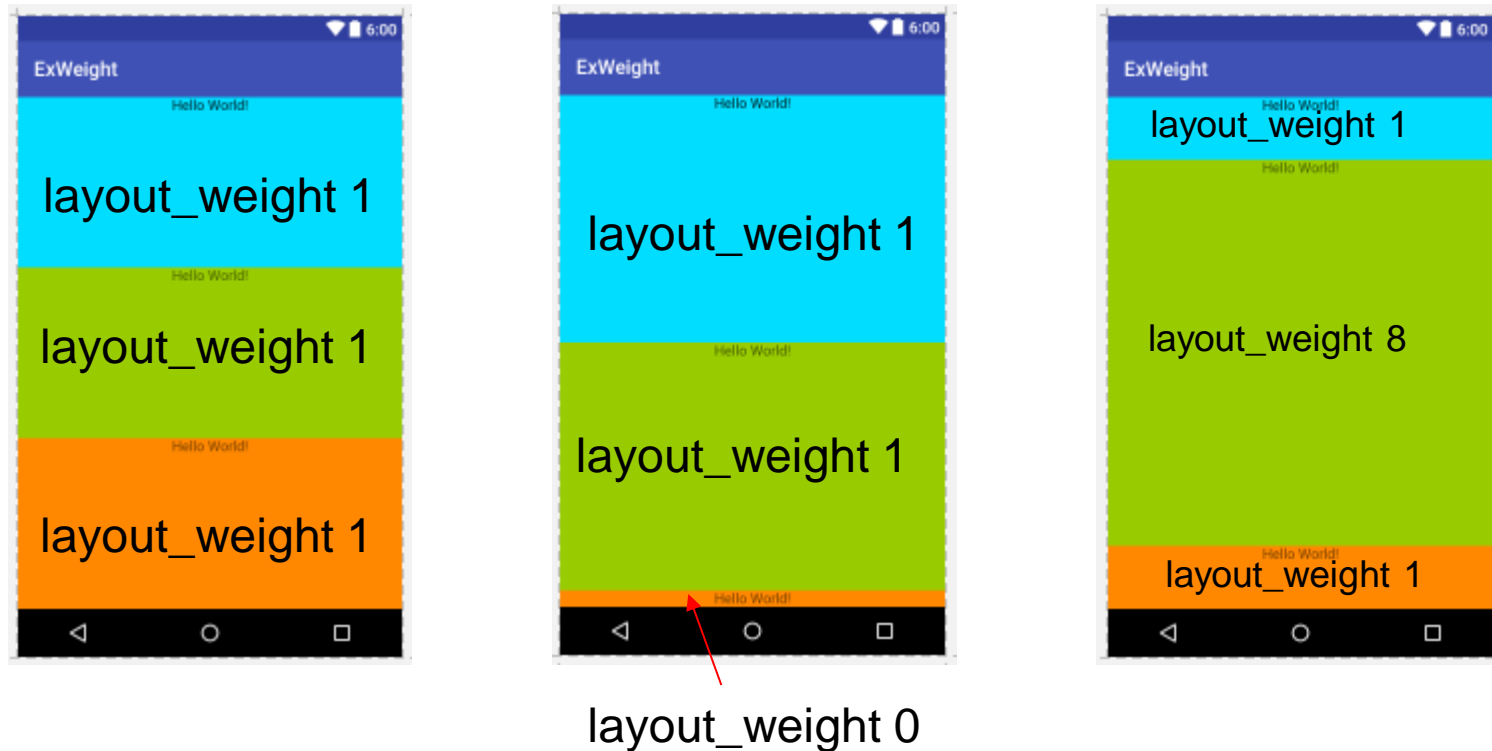
Altere o comando a seguir para testar os três casos anteriores:

```
setContentView(R.layout.activity_main);  
setContentView(R.layout.activity_main_2);  
setContentView(R.layout.activity_main_3);
```

Layout weight (atributo layout_weight)

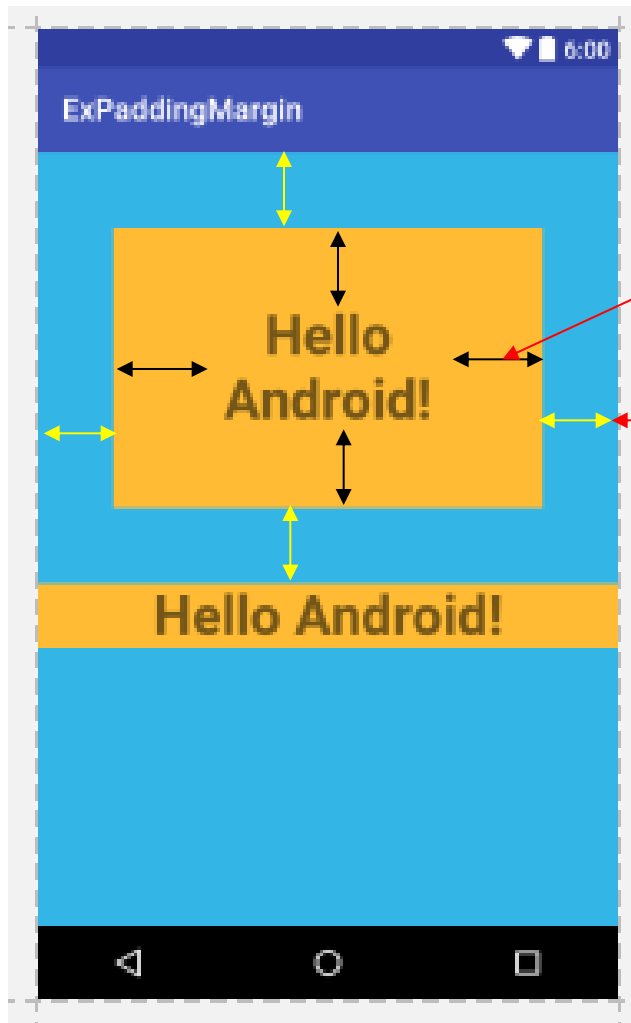
- Atributo (peso) que distribui proporcionalmente os elementos na tela. É utilizado com o `LinearLayout`.
- Como usar:
 - Se você quiser dividir o espaço igualmente entre os elementos com o mesmo peso, defina o `width` ou `height` com `0dp` para todos os elementos filhos e definir o valor `1` para o `weight`.
 - Se você quiser definir com o `wrap_content`, os tamanhos dos elementos irão depender dos pesos e do conteúdo dentro de cada elemento.
 - Para ter um controle melhor da distribuição proporcional, utilize `0dp` para `width` ou `height`.
 - Basicamente o total da altura ou largura é dividido pela quantidade de pesos e a altura ou largura de cada componente dependerá de seu peso.

Layout weight (exemplo)



Veja este exemplo no projeto ExWeight. Todos os TextView foram definidos com `android:layout_height="wrap_content"`, mas com valores de `layout_weight` diferentes, como mostrados nas figuras acima.

Atributos padding e layout_margin



padding

layout_margin

Observe que **padding** é a distância entre o texto do componente e sua borda. O atributo **layout_margin** é a distância entre o componente e outros componentes. Semelhante a CSS.

`android:layout_margin="50dp"`

`android:padding="50dp"`

Veja o exemplo no projeto ExPaddingMargin

TextView

Serve para escrevermos um texto na tela do Android, como um label, rótulo. Através do código podemos alterar seu valor em tempo de execução.

```
...  
<TextView  
    android:id="@+id/textView"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Olá Mundo!"  
    android:textSize="28sp"  
    android:textColor="#12FF88"  
    android:textStyle="italic|bold"  
>  
...
```

Atributos extras:

android:gravity, especifica como alinhar o texto no eixo x e/ou y quando o conteúdo for menor que o componente.

android:inputType, informa o tipo de entrada de dados, útil para configurar o teclado virtual, por exemplo, Number, Phone, Password etc. Muito útil para objetos da classe EditText.

EditText

É uma subclasse da TextView que permite edição, entrada de dados.

Enquanto o TextView exibe texto, como um JLabel de Java SE, quem oferece ao usuário um campo para ele digitar texto é o EditText (assim como o JTextField em Java SE) que é uma classe filha de TextView.

Este componente não declara nenhum atributo XML novo, mas usa os do TextView. A diferença é que ele permite edição.

Para tornar ele editável ou não, usamos o atributo android:editable com true ou false (true é o default).

...

```
<EditText
    android:id="@+id/editView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="..."
    android:textSize="28sp"
    android:textColor="#12FF88"
    android:textStyle="italic|bold" />
```

...

Button

É um simples botão com algo escrito assim como a classe JButton de Java SE. É através de um Button que o usuário sabe que alguma ação será realizada quando o acionar.

...

<Button

```
    android:id="@+id/button"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Clique!"  
    android:textSize="28px"  
    android:textColor="#12FF88"  
    android:textStyle="italic|bold" />
```

...

CheckBox

- É um botão que permite ao usuário alternar entre as opções marcado / desmarcado. Utilizado para selecionar uma ou várias opções.
- Esta classe também deriva, indiretamente, do TextView, e o valor atribuído em 'android:text' é o texto visível ao lado do CheckBox.

<CheckBox

```
    android:id="@+id/checkBox2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Você gosta do Android"  
    android:checked="false"
```

/>

Android CheckBox Example

- ☐ Android
- ☒ Java
- ☒ XML
- ☐ HTML

RadioButton

São botões parecidos com o CheckBox que permitem marcar o componente, mas não é possível desmarcá-lo. Para se obter o efeito de seleção única, é necessário colocar todos os radiobuttons dentro de um RadioGroup.

`<RadioGroup`

```
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/grb1">
```

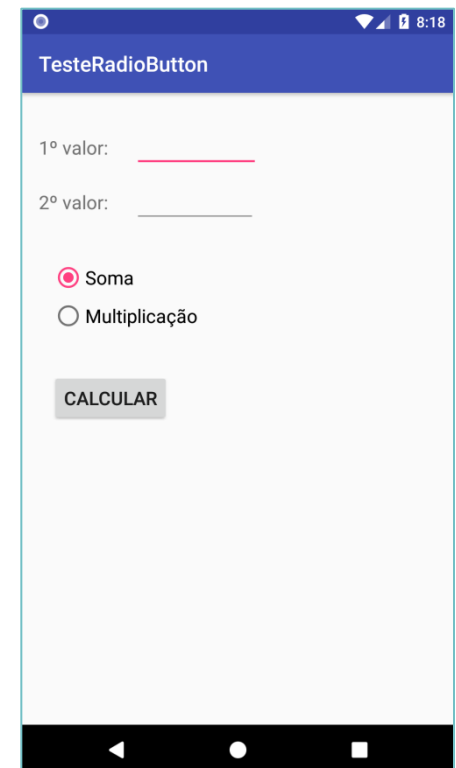
`<RadioButton`

```
    android:id="@+id/rb1"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Opção 1"  
    android:textSize="28sp"  
    android:checked="true"/>
```

`<RadioButton`

```
    android:id="@+id/rb2"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Opção 2"  
    android:textSize="28sp" />
```

`</RadioGroup>`



ImageView

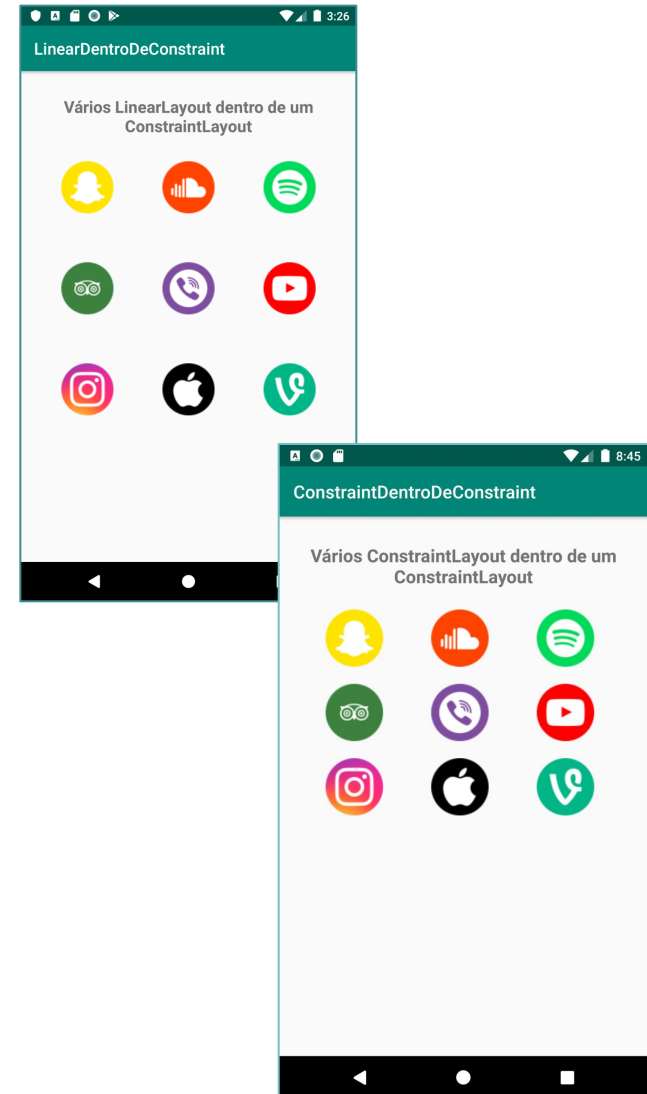
- O ImageView, como o próprio nome diz, é um componente que nos permite trabalhar com imagens.
- As imagens são colocadas na pasta drawable, posteriormente, a classe R.java mapeia esses recursos.
- Outro atributo interessante é **scaleType** (valores fitXY, center etc; por exemplo: android:scaleType="fitXY").

```
<ImageView  
    android:id="@+id/imageView1"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:src="@drawable/ic_launcher" />
```


Exemplos finais

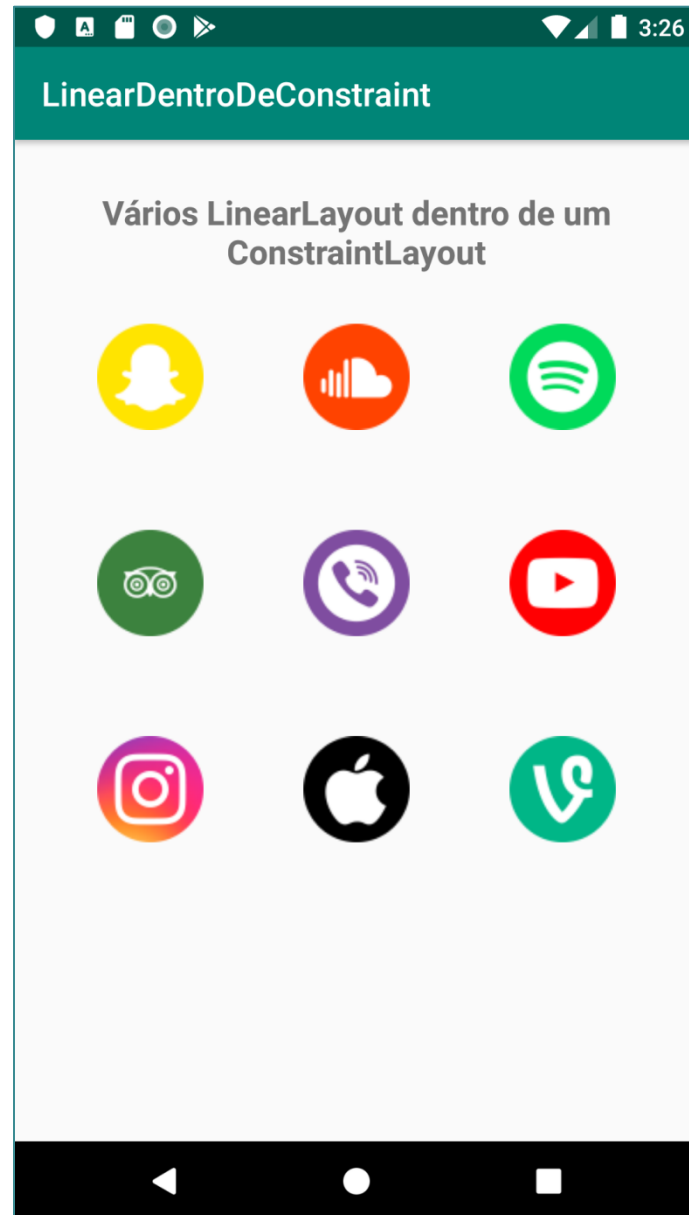
- LinearDentroDeConstraint
- ConstraintDentroDeConstraint

Estes exemplos estão completamente resolvidos nos projetos com estes mesmos nomes, mas sugerimos tentar reproduzir estas telas a partir das explicações fornecidas, criando novos projetos para praticar.



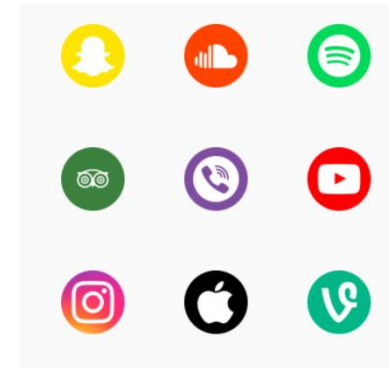
Um exemplo de
combinação de layouts:
vários LinearLayout
dentro de um
ConstraintLayout

(projeto **LinearDentroDeConstraint**)



Queremos deixar uma margem entre as figuras, tanto na horizontal como na vertical. Cada figura poderia especificar as quatro margens:

```
android:layout_marginLeft="30dp"  
android:layout_marginRight="30dp"  
android:layout_marginTop="30dp"  
android:layout_marginBottom="30dp"
```



Mas, se queremos que todas as margens sejam iguais podemos resumir com um único atributo :

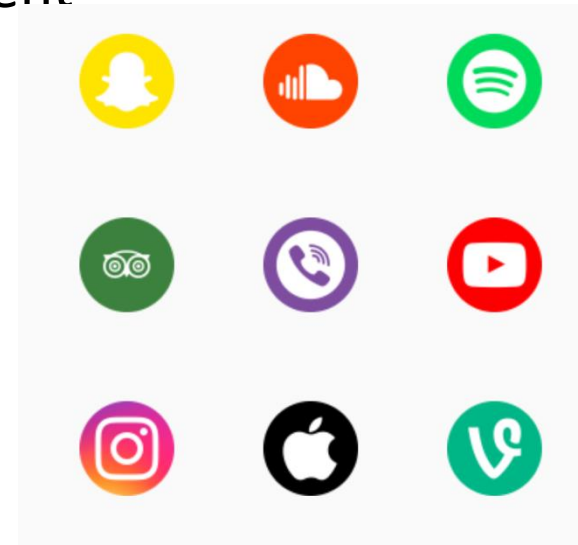
```
android:layout_margin="30dp"
```

No layout landscape (land) faremos algumas alterações nas margens, para evitar colocar um ScrollView.

A tag de cada figura (com identificadores imageView1, ... imageView9) (nos arquivos f1.png, ... f9.png) poderia ficar inicialmente assim:

<ImageView

```
android:id="@+id/imageView1"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_margin="30dp"  
android:src="@drawable/f1" />
```



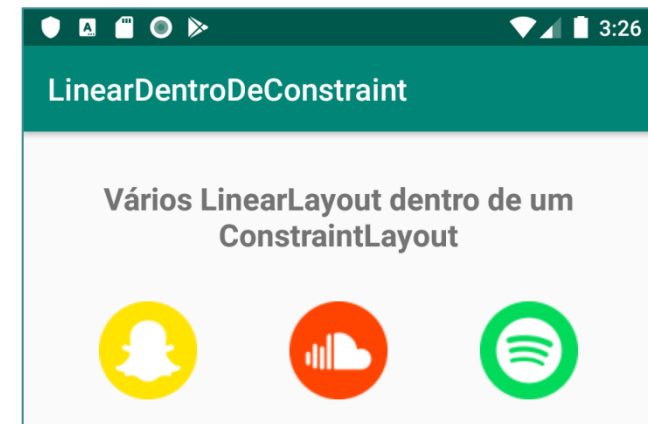
O título especificará duas restrições (constraints) com relação ao parent (o ConstraintLayout) e também especificará margens, um tamanho de letra de 20sp, negrito, centralizado e ocupará a largura do parent (match_parent):

<TextView

```

android:id="@+id/cabecalho1"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
android:text="@string/titulo_principal"
android:textSize="20sp" android:textStyle="bold"
android:layout_margin="30dp"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />

```



Cada **LinearLayout** terá três figuras dentro.

Todos os LinearLayout estarão centralizados, ocuparão a largura do parent (match_parent), e terão orientação horizontal por causa do atributo android:orientation="horizontal".

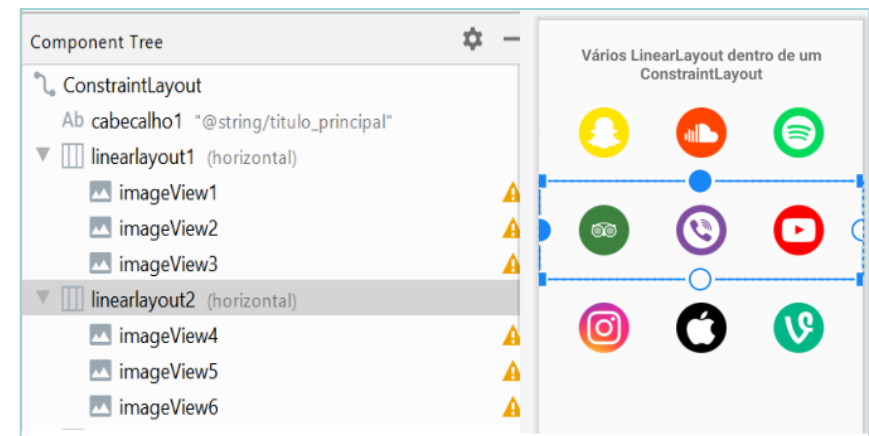
Prepararemos cada LinearLayout com duas restrições (constraints) para que fique colocado depois do LinearLayout anterior, mas observe que o primeiro LinearLayout ficará colocado depois do título inicial que tem id **cabecalho1**.

<LinearLayout

```
android:id="@+id/linearlayout1"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:gravity="center_horizontal"  
android:orientation="horizontal"  
app:layout_constraintStart_toStartOf="@id/cabecalho1"  
app:layout_constraintTop_toBottomOf="@id/cabecalho1">
```

... as figuras deste LinearLayout estarão aqui dentro...

</LinearLayout>



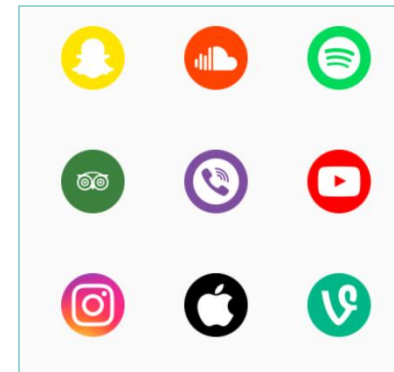
Os restantes LinearLayout também terão duas restrições, especificando sua posição vertical e horizontal com relação ao LinearLayout anterior. Por exemplo, veja o segundo LinearLayout:

<LinearLayout

```
android:id="@+id/linearlayout2"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:gravity="center_horizontal"  
android:orientation="horizontal"  
app:layout_constraintStart_toStartOf="@id/linearlayout1"  
app:layout_constraintTop_toBottomOf="@id/linearlayout1">
```

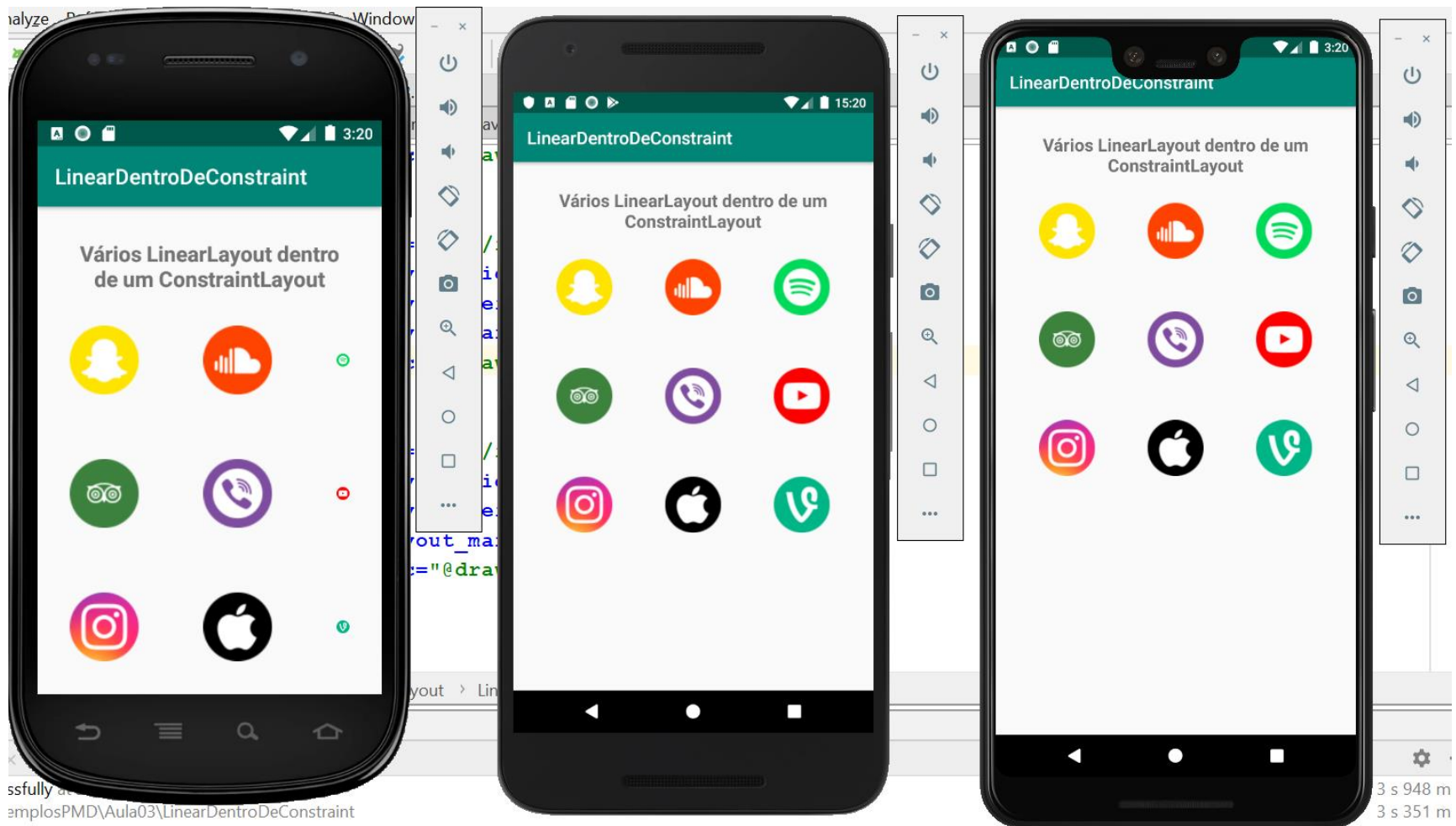
... as figuras estão aqui dentro...

</LinearLayout>



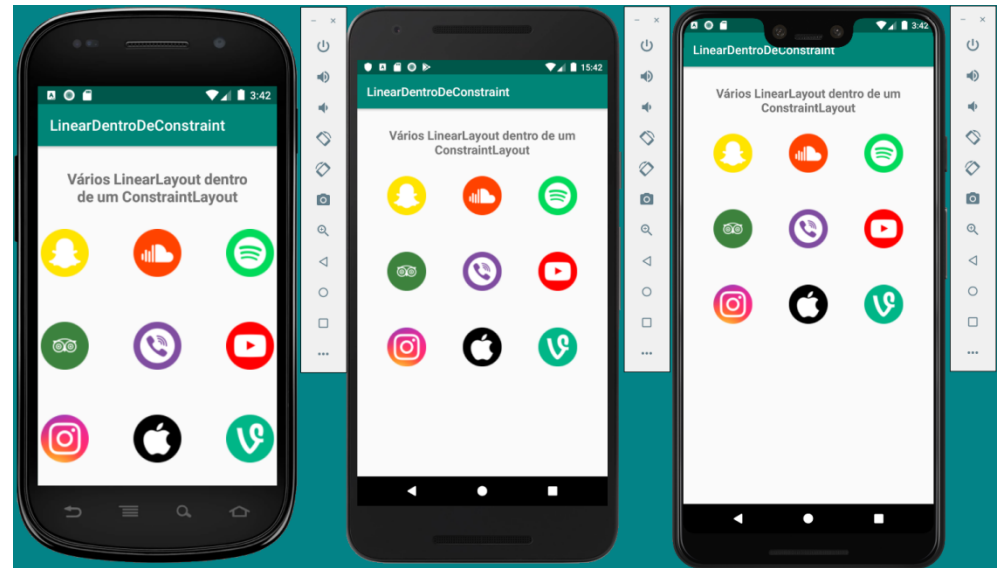
A solução completa deste exemplo se encontra no projeto **LinearDentroDeConstraint**.

Está tudo perfeito? Não! Como utilizamos as mesmas figuras para qualquer tela, os resultados poderão ser diferentes em alguns aparelhos. Veja a seguir três casos diferentes: Nexus S (480x800), Nexus 5X (1080x1920) e Pixel 3 XL (1440x2960).

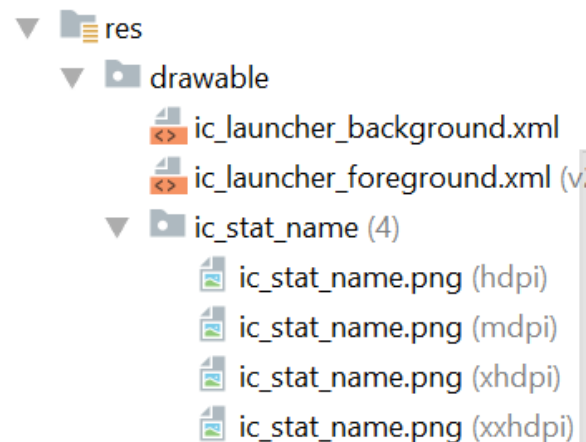


Especificando um tamanho correto para as figuras, em unidades **dp**, podemos ter melhores resultados:

```
<ImageView
    android:id="@+id/imageView2"
    android:layout_width="64dp"
    ... />
```

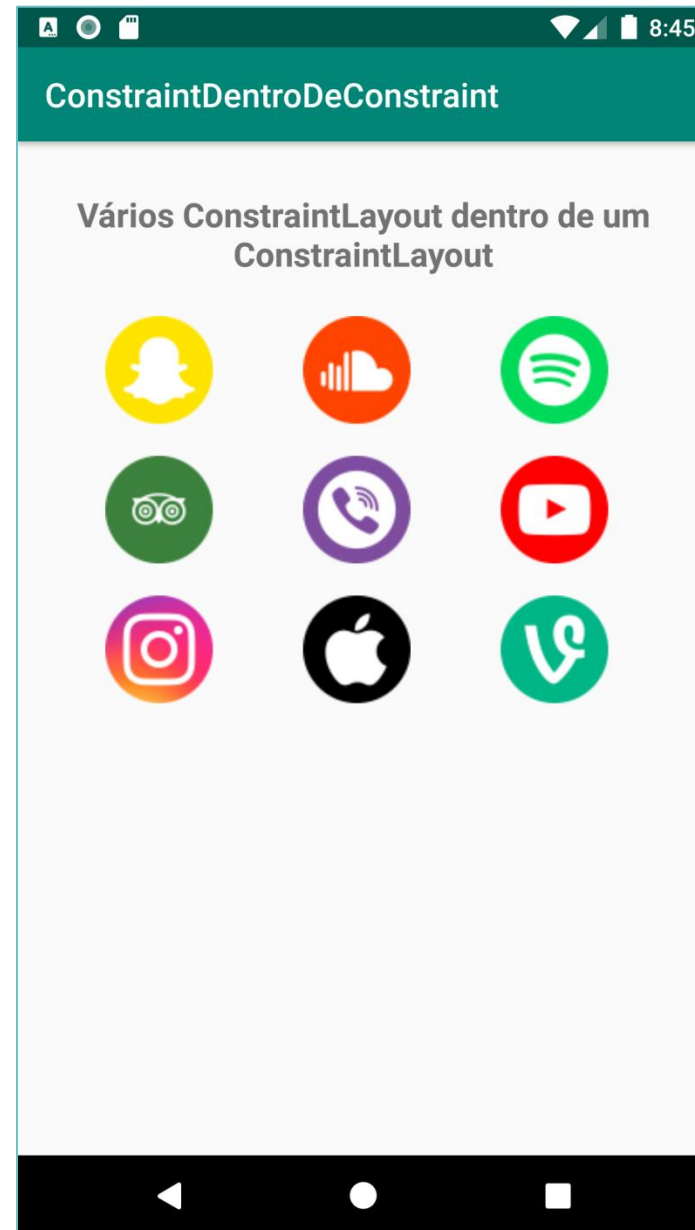


Também, poderíamos criar uma versão de cada figura para diferentes densidades de tela, com o assistente do Image Asset (aula posterior).

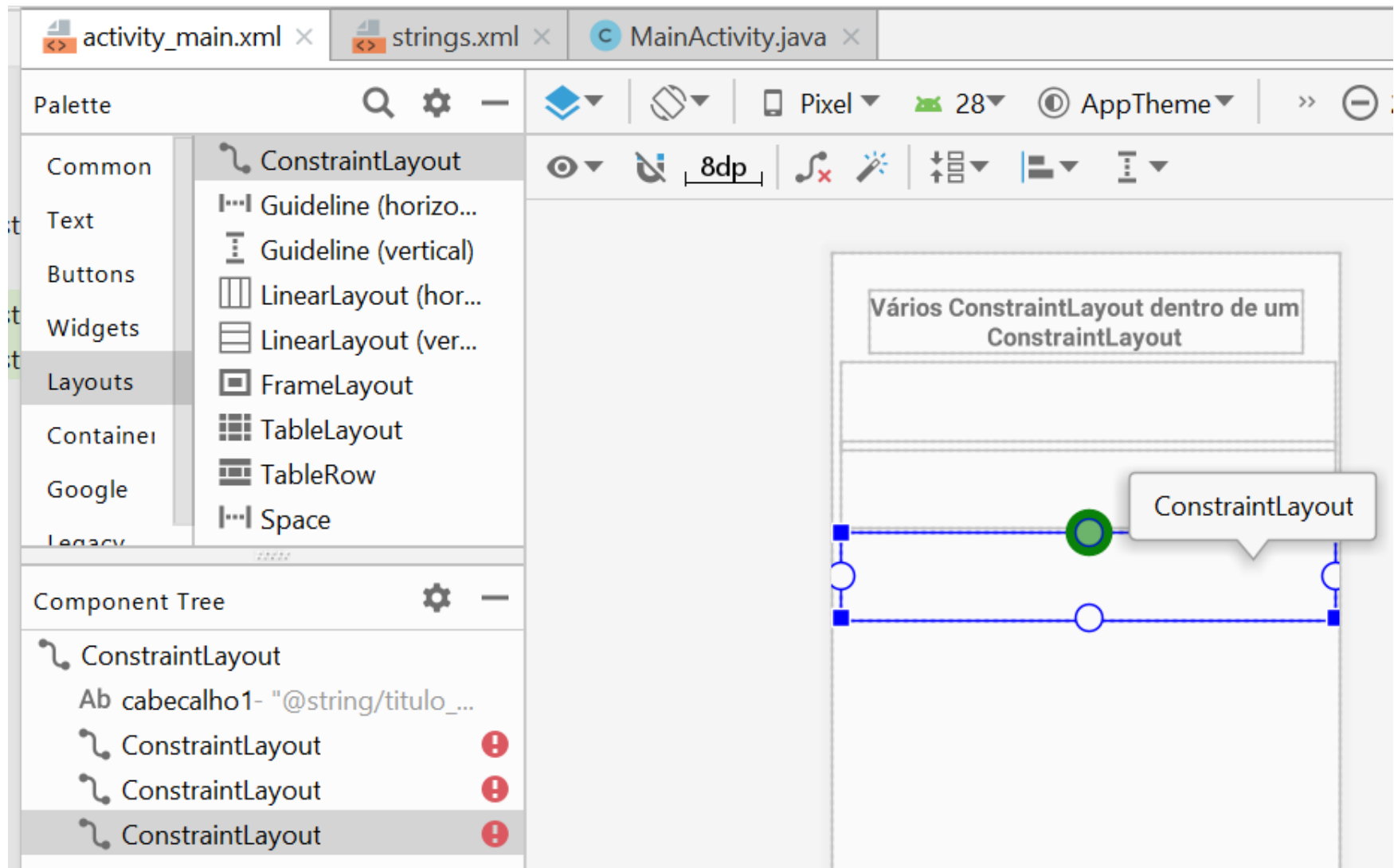


Outro exemplo de
combinação de layouts:
vários ConstraintLayout
dentro de um
ConstraintLayout

(exemplo no projeto
ConstraintDentroDeConstraint)

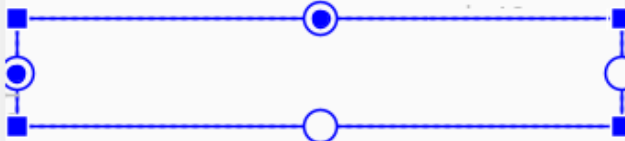


Crie um novo projeto e arraste um TextView e três ConstraintLayout

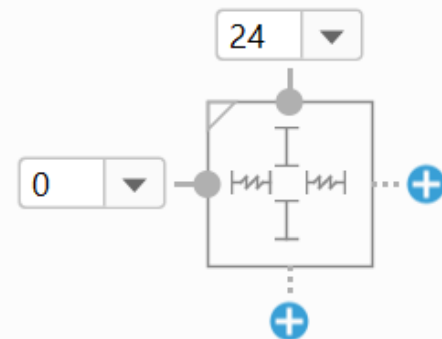


Configure duas restrições para o TextView e para cada um dos ConstraintLayout internos (será mais fácil clicar nos botões **+**). Modifique o atributo **layout_width** para **match_parent**, de forma que cada layout interno ocupe a largura total do layout externo (parent).

Vários ConstraintLayout dentro de um ConstraintLayout



▼ Layout

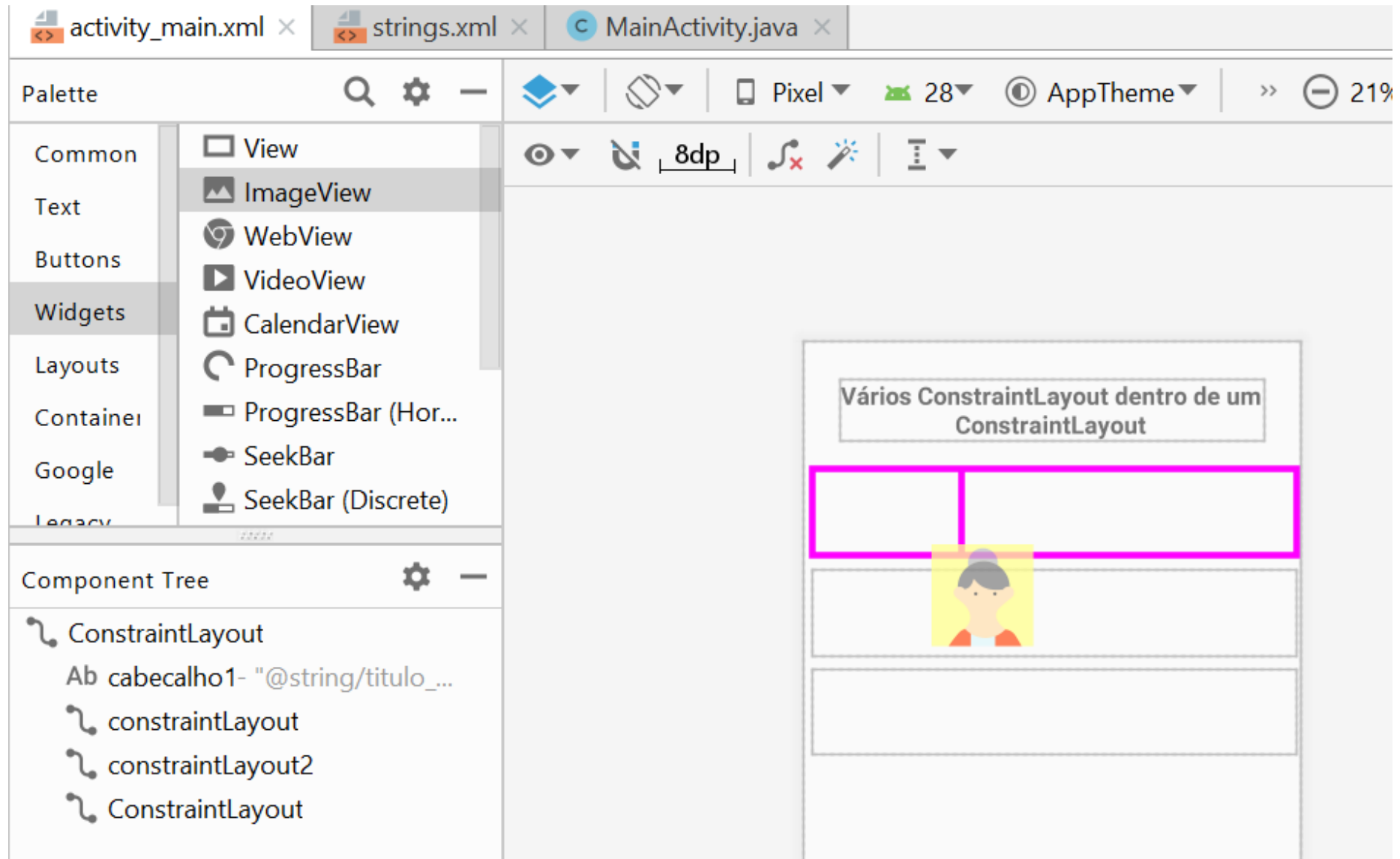


match_parent

layout_width 400dp

layout_height 70dp


Arraste três figuras para cada ConstraintLayout interno:



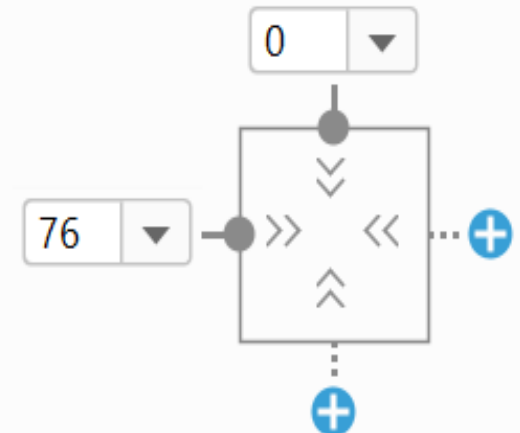
Cada vez que arraste e solte uma figura, selecione o arquivo da figura desejada (f1, f2... das figuras png já colocadas na pasta drawable do projeto) e configure pelo menos duas restrições: veja que clicando nos botões + será mais simples, depois digite o valor de margem desejado.

Vários ConstraintLayout dentro de um
ConstraintLayout

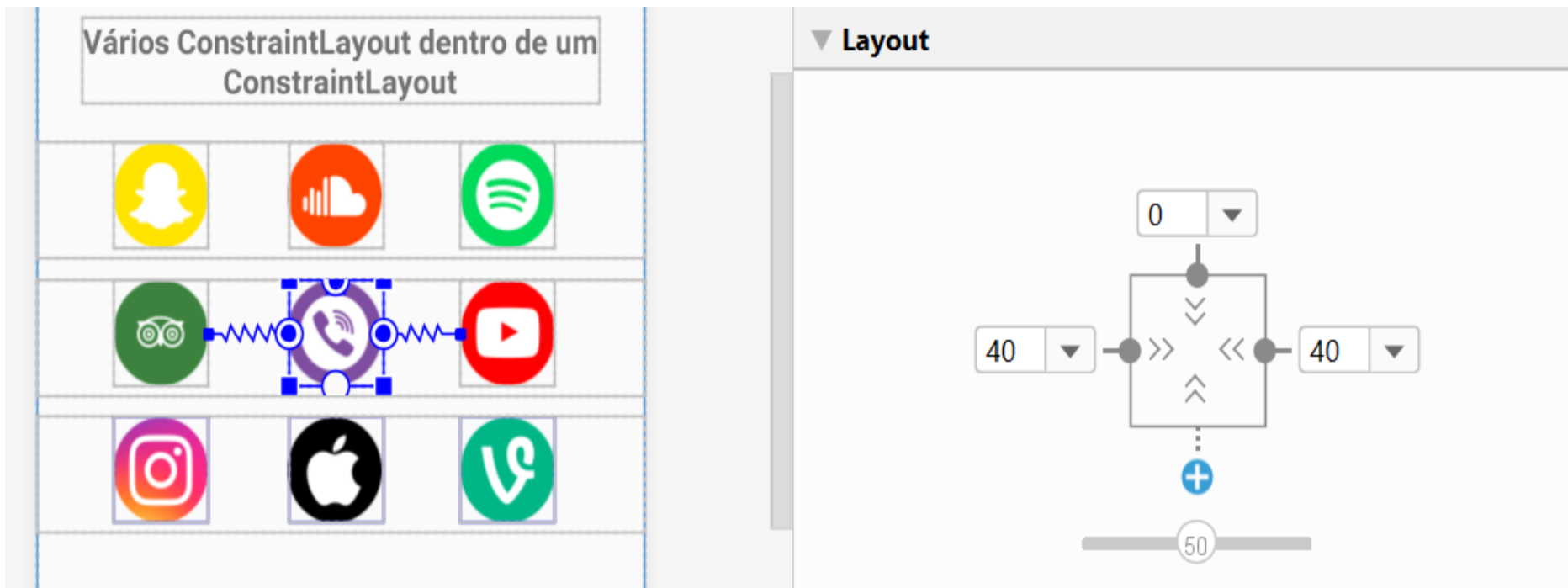


id	imageView2
srcCompat	 @drawable/f2

▼ Layout



Nas figuras centrais pode dar melhor resultado configurar a margem esquerda e direita, com relação às figuras vizinhas ou as margens com relação ao layout externo. Na figura esquerda configuremos a margem esquerda e na figura direita configuremos a margem direita.



Finalmente crie a versão landscape e faça os ajustes necessários.

Exercício 1



Crie um cartão de aniversário com a imagem fornecida, siga o modelo ao lado.

Utilize:

2 TextView

1 ImageView


Use RelativeLayout ou o ConstraintLayout padrão

A ordem influencia a visualização dos elementos.

Exercício 2 (utilize pesos)



para enviar



Crie um form conforme modelo ao lado para envio de mensagem.

Utilize:

4 EditText

Sugerido LinearLayout

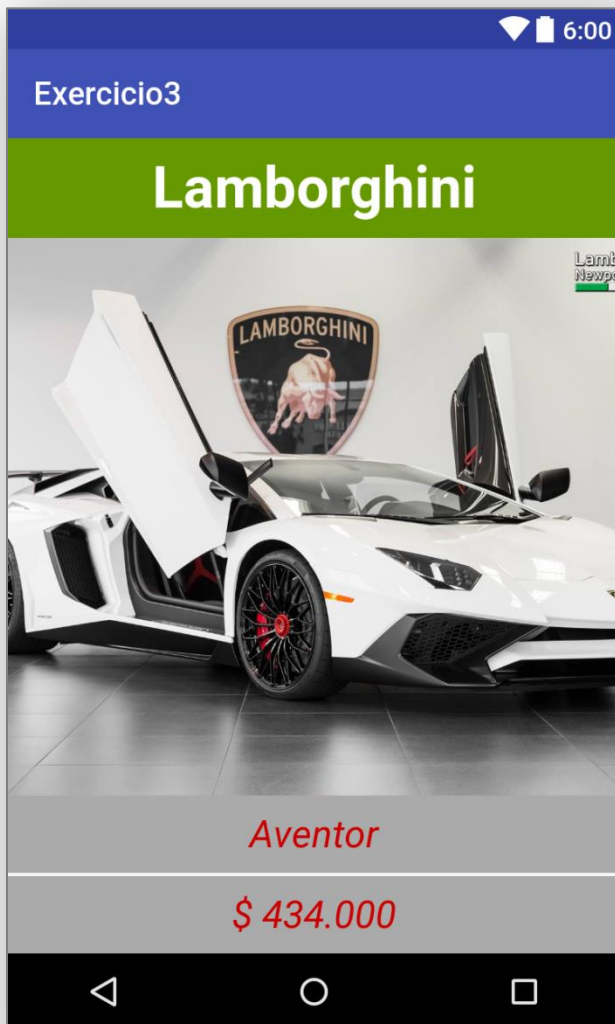
Considere o atributo `layout_weight`, sendo que o campo mensagem deve ser o maior.

Procure usar os atributos `gravity`, `hint` para o texto e no `background drawable` editbox

Exercício 3



para enviar



Crie o layout conforme modelo.

Utilize:

1 ImageView

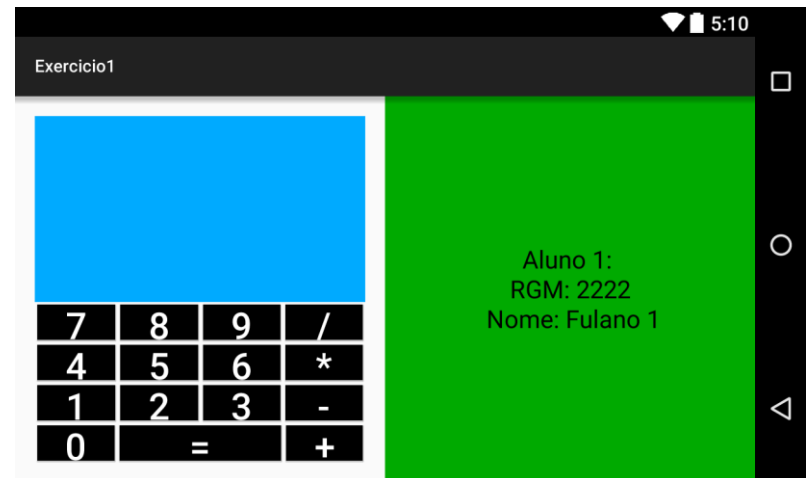
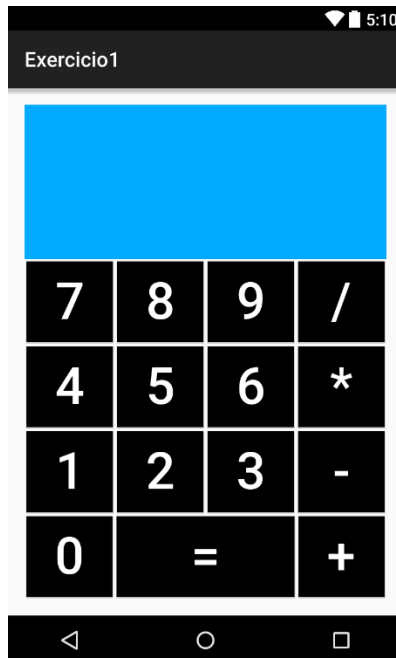
3 TextView

Utilize LinearLayout ou outros

Utilize margin, padding e outras configurações que achar necessárias.

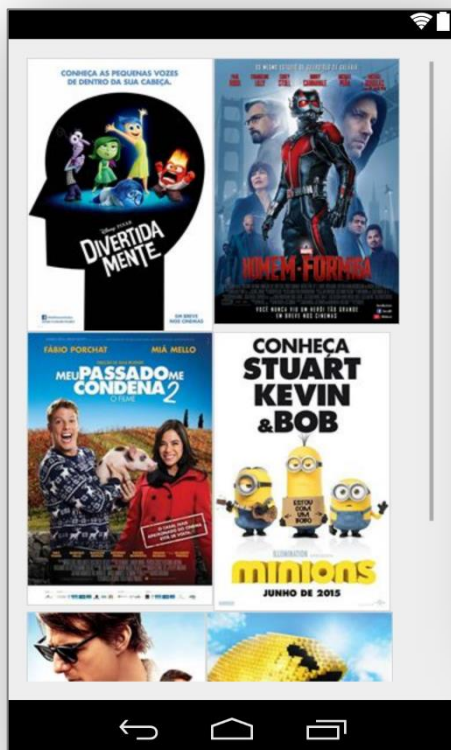
Exercício 4

- Desenvolva o visual da calculadora abaixo conforme formatos para as telas retrato e paisagem. O layout deverá se adaptar a diferentes tipos de telas.
- Para testar seu exercício, utilize um AVD.
- Para mudar a orientação do simulador, pressione Ctrl+F11.



Exercício 5

- O cinema UCI do Anália Franco quer um aplicativo para disponibilizar seus filmes em cartaz, o aplicativo deverá mostrar as imagens dos filmes no layout portrait e as imagens/sinopses no layout landscape.
- Você deverá trabalhar com barra de rolagem em ambas as orientações.



A disposição Imagem/Sinopse deve ser intercalada, segue exemplo:

Imagem	Sinopse
Sinopse	Imagem
Imagem	Sinopse
Sinopse	Imagem

Bibliografia sugerida sobre Android

- ANDROID. Android Developers. Disponível em <http://developer.android.com>.
- LECHETA, RICARDO R. Google Android, Aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec, 2010.
- MEDNIEKS, Z. et. al. Desenvolvimento de Aplicações Android. São Paulo: Novatec, 2009.
- LEE, Wei-Meng. Introdução ao Desenvolvimento de Aplicativos para o Android. Rio de Janeiro: Editora Ciência Moderna, 2011

https://developers.google.com/android/for-all/vocab-words/?utm_source=udacity&utm_medium=course&utm_campaign=android_basics

Vocabulary Glossary

This glossary of Android and Java vocab words supplements the [Udacity Android for Beginners](#) course. This course is targeted at those who are new to programming but want to start building Android apps.

Superclass or Base Class

System Log

Text Localization

TextView

Theme

User Interface

Variable

Variable Declaration

Variable Name

Variable Scope

...

<https://material.io/guidelines/style/typography.html#>

<https://material.io/guidelines/style/color.html#color-color-palette>

<https://material.io/guidelines/layout/metrics-keylines.html#metrics-keylines-keylines-spacing>