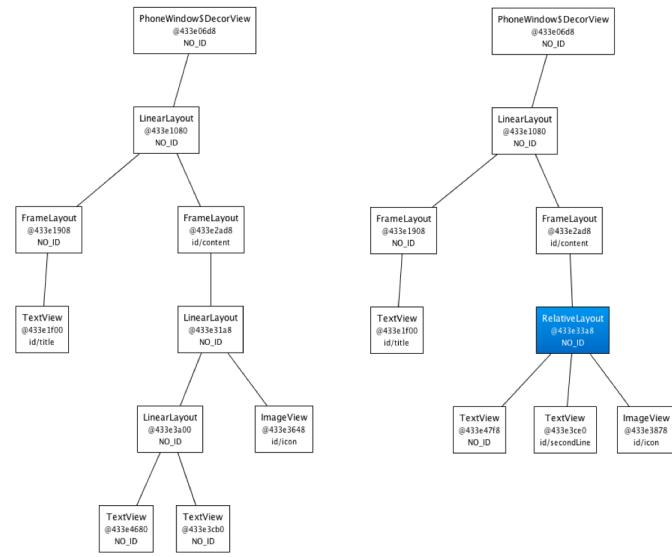




ANDROID

Layout



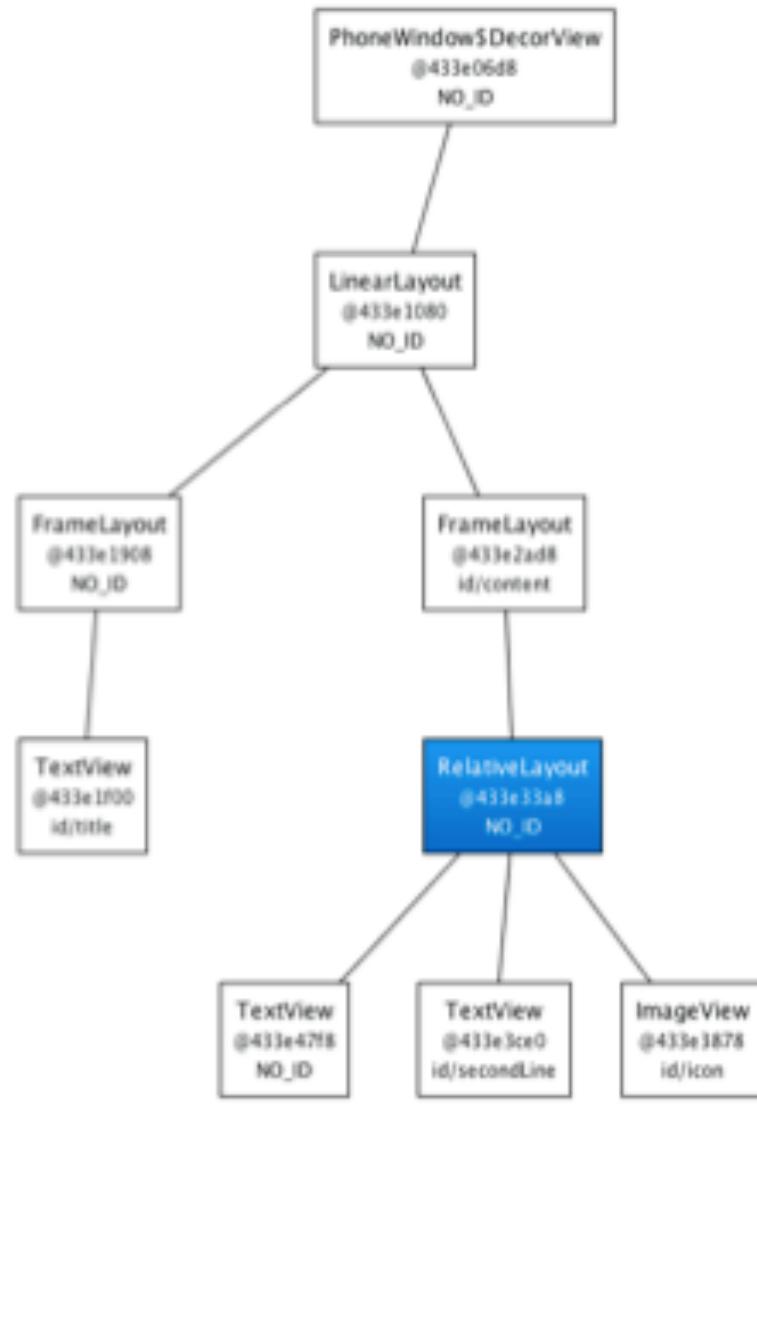
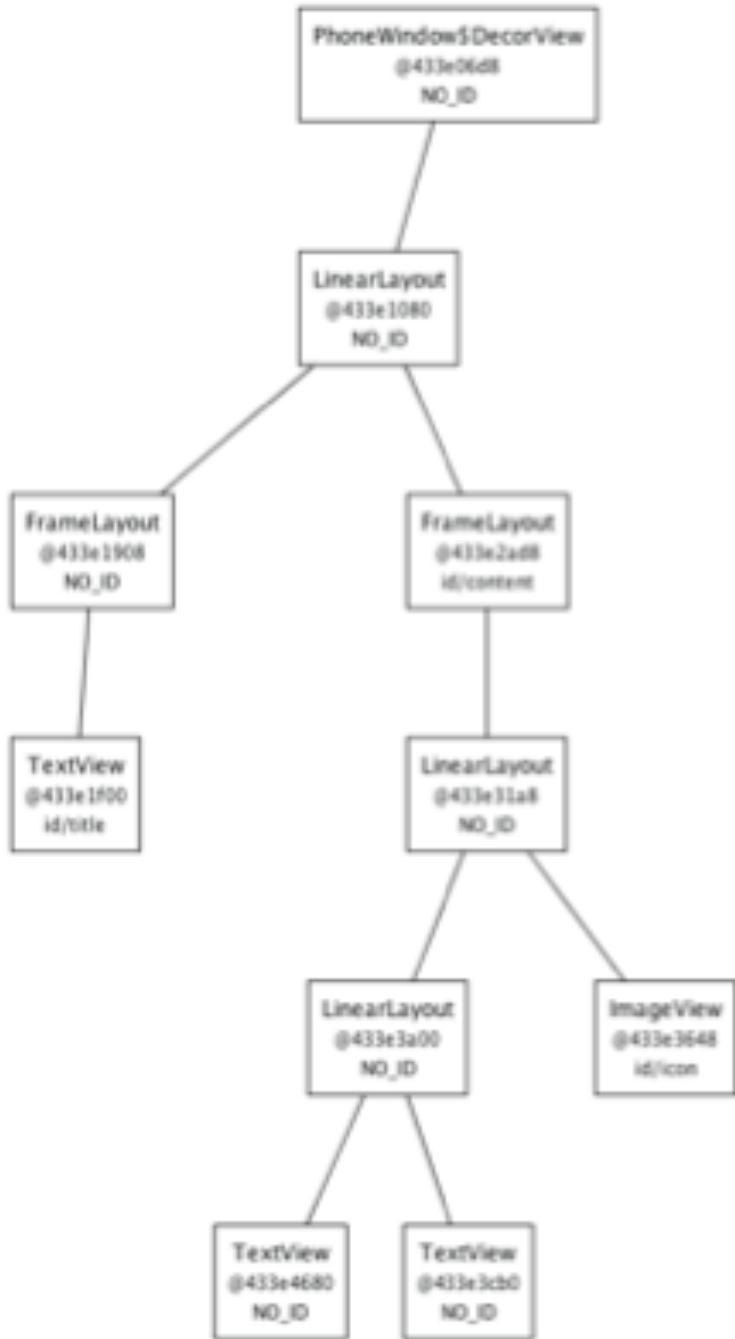
Prof. Alessandro Brawerman

LinearLayout

RelativeLayout

Introdução

- Existem dois tipos de componentes visuais, widgets e layouts.
- Widgets
 - Componente simples que herda diretamente da classe View.
Button, ImageView, ...
- Gerenciadores de Layout
 - Utilizado para organizar a disposição dos componentes na tela automaticamente.
 - Subclasses de android.view.ViewGroup.



Classes de Layout

- **AbsoluteLayout**
 - Permite posicionar os componentes fornecendo as coordenadas x e y.
- **FrameLayout**
 - Utilizado por um componente que precisa preencher a tela inteira.
- **LinearLayout**
 - Utilizado para organizar os componentes na vertical ou horizontal, como uma pilha.
- **TableLayout**
 - É filho do LinearLayout.
 - Utilizado para organizar os componentes em uma tabela com linhas e colunas.

Classes de Layout

- **RelativeLayout**
 - Permite posicionar um componente relativo a outro, por exemplo, abaixo, acima ou ao lado de um outro componente já existente.
- Ao definir-se um layout, deve-se também informar o quanto da tela o mesmo irá ocupar.
- **android:layout_height**
 - Especifica a altura de um componente/view.
- **android:layout_width**
 - Especifica a largura de um componente/view.

Classes de Layout

- Estes dois parâmetros podem receber os valores:
 - Número: número inteiro especificando o tamanho do layout.
 - Fill_parent: layout precisa ocupar todo o tamanho definido por seu pai. Utilizar quando é necessário ocupar a tela inteira.
 - Wrap_content: ocupar apenas o tamanho necessário na tela.

FrameLayout

- É o layout mais simples.
- Um componente preenche a tela inteira.
- android.widget.FrameLayout
- O componente inserido sempre será posicionado no canto esquerdo superior e ocupará o espaço necessário.
- É possível inserir mais de um componente neste layout, mas sempre o último ficará visível, como uma pilha.
- Uma imagem pode ser inserida para background e componentes menores sobre ela.

Exemplo 1 - FrameLayout

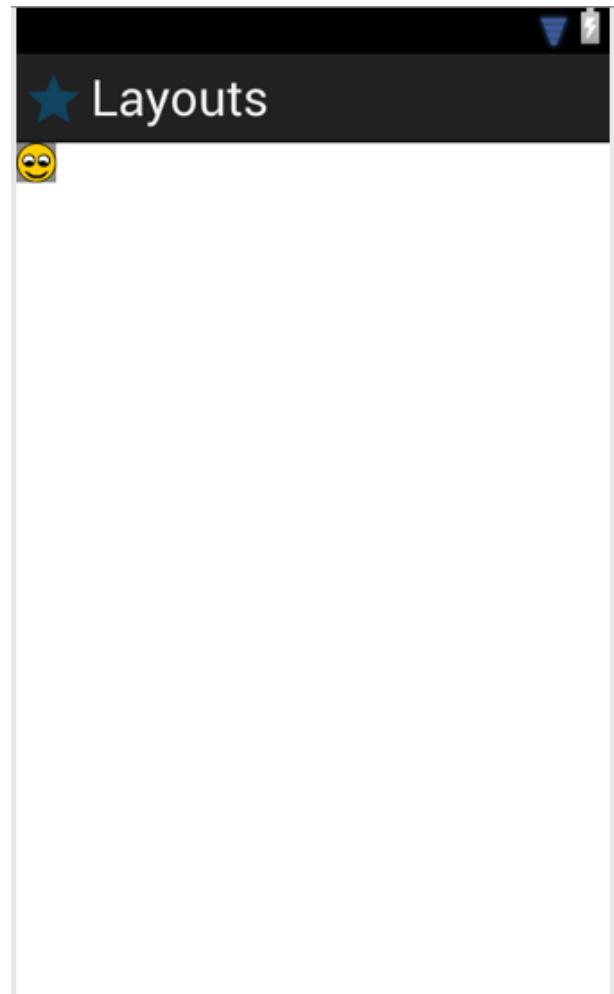
- Crie um novo projeto Layouts.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="#8B8B83" >  
  
    <ImageView  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:src="@drawable/smile1"  
    />  
  
</FrameLayout>
```

- Wrap_content – tamanho do layout é somente o necessário para desenhar todos os seus componentes.

Exemplo 1 - FrameLayout

- O fundo cinza do layout, atrás da imagem, mostra o tamanho ocupado pelo layout principal.
- Mesmo tamanho da imagem, neste caso.
- Use o atributo `background` quando quiser saber o tamanho que o layout está ocupando na tela.



Exemplo 2 - FrameLayout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:background="#8B8B83" >  
  
    <ImageView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:src="@drawable/smile1"  
    />  
  
</FrameLayout>
```

Exemplo 2 - FrameLayout



Exemplo 3 - FrameLayout

- Configure a largura e altura da imagem para ser o mesmo tipo da layout em que está inserida.
- É possível combinar os valores `fill_parent` e `wrap_content` para expandir os componentes na vertical ou horizontal.



LinearLayout

- Um dos gerenciadores mais utilizados.
- android.widget.LinearLayout.
- É possível organizar uma sequência de componentes na horizontal (padrão) ou vertical.
- Além dos parâmetros de altura e largura, que todos os layouts têm, o LinearLayout possui também o android:orientation, para saber se os componentes serão exibidos na horizontal ou vertical.

Exemplo 1 - LinearLayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#8B8B83" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/smile1"
    />

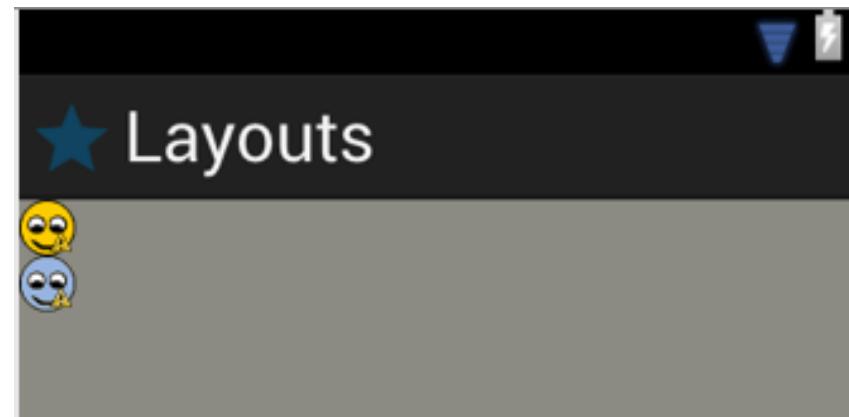
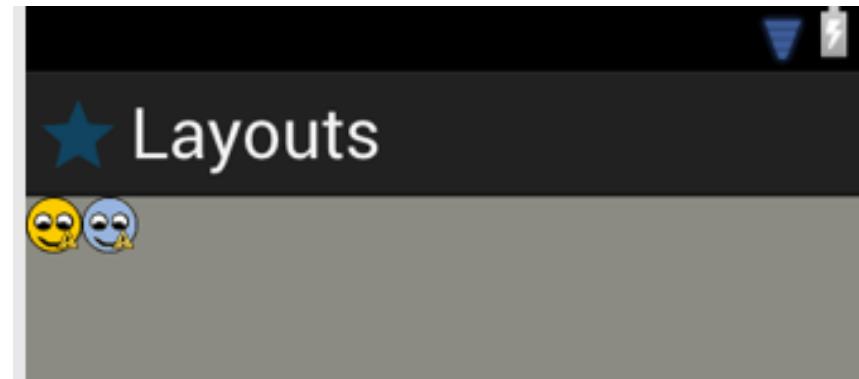
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/smile2"
    />

</LinearLayout>
```

Exemplo 1 - LinearLayout

- Por padrão a orientação é horizontal.
- Cuidado caso haja muitos componentes, eles poderão ir para fora da tela.
- Para modificar o sentido para vertical basta incluir a linha de comando abaixo:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="#8B8BB3" >
```



Linear Layout – Controle de Alinhamento

- Como alinhar os componentes de forma centralizada, a esquerda ou a direita?
- Deve-se usar o atributo android:layout_gravity.
- Valores válidos:
 - Top, bottom, center_vertical, fill_vertical
 - Left, right, center_horizontal, fill_horizontal
 - Center e fill

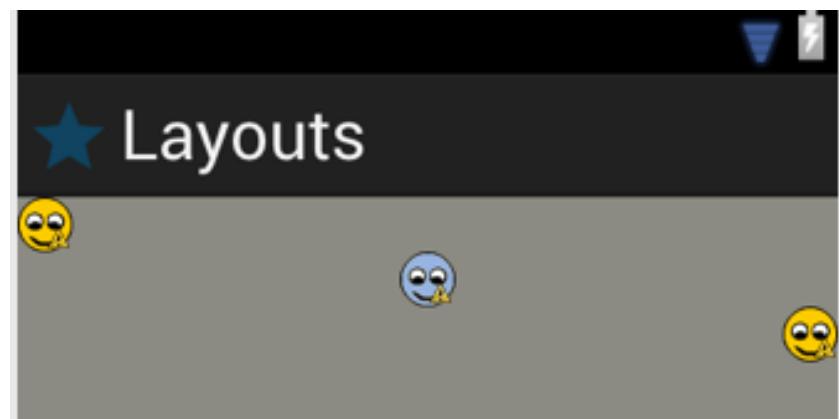
Exemplo – layout_gravity

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="#8B8883" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/smile1"
        android:layout_gravity="left"
    />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/smile2"
        android:layout_gravity="center"
    />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/smile1"
        android:layout_gravity="right"
    />
</LinearLayout>
```



Peso e Relevância de Componentes

- Outra forma de organizar os componentes na tela é atribuindo peso para cada um.
- Maior peso, maior espaço.
- Atributo - android:layout_weight.

Exemplo 1 – layout_weight

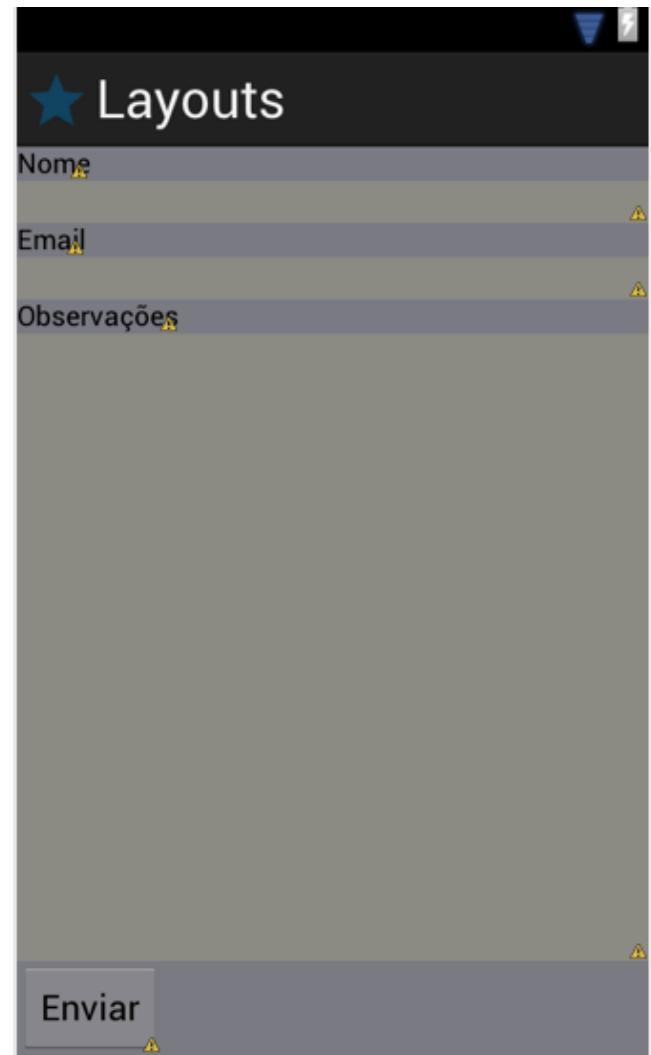
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="#7B7B83">
    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Nome" android:textColor="#000000" />
    <EditText
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:background="#8B8B83"/>
    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Email" android:textColor="#000000" />
    <EditText
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:background="#8B8B83"/>
```

Exemplo 1 – layout_weight

```
<TextView  
    android:layout_width="wrap_content" android:layout_height="wrap_content"  
    android:text="Observações" android:textColor="#000000" />  
<EditText  
    android:layout_width="fill_parent" android:layout_height="fill_parent"  
    android:background="#8B8883" android:layout_weight="1" />  
<Button  
    android:layout_width="wrap_content" android:layout_height="wrap_content"  
    android:text="Enviar" />  
</LinearLayout>
```

Exemplo 1 – layout_weight

- Campo observações é o único com maior peso, portanto ocupou o restante da tela.
- Valor padrão de peso é 0.
- Sempre que um componente precisar ocupar o restante da tela, basta definir seu peso como 1 e os outros 0.



Exemplo 2 – layout_weight

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="#7B7B83" >
    <EditText android:layout_weight="1"
        android:layout_width="fill_parent" android:layout_height="0dip"
        android:text="Texto (weight=1)" />
    <EditText android:layout_weight="2"
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:text="Texto (weight=2)" />
    <EditText android:layout_weight="3"
        android:layout_width="fill_parent" android:layout_height="0dip"
        android:text="Texto (weight=3)" />
</LinearLayout>
```

Exemplo 2 – layout_weight

- Somando os pesos dos componentes temos $1 + 2 + 3 = 6$.
- Como 6 é o peso total, o componente 3, com peso 3, ocupa 50% da tela.
- Altere o peso do primeiro componente para 3.
- Agora temos $3 + 2 + 3 = 8$, ou seja, o componente 2 ocupa 25% e os outros 2 ocupam 37,5% da tela.



RelativeLayout

- Posiciona componentes ao lado, abaixo ou acima de um outro componente já existente.
- É necessário definir um id para cada componente, para fazer o posicionamento corretamente.
- android.widget.RelativeLayout.
- Atributos para posicionamento:
 - android:layout_below, android:layout_above
 - android:layout_toRightOf, android:layout_toLeftOf
 - android:layout_alignParentTop, android:layout_alignParentBottom
 - android:layout_marginTop, android:layout_marginRight, android:layout_marginLeft

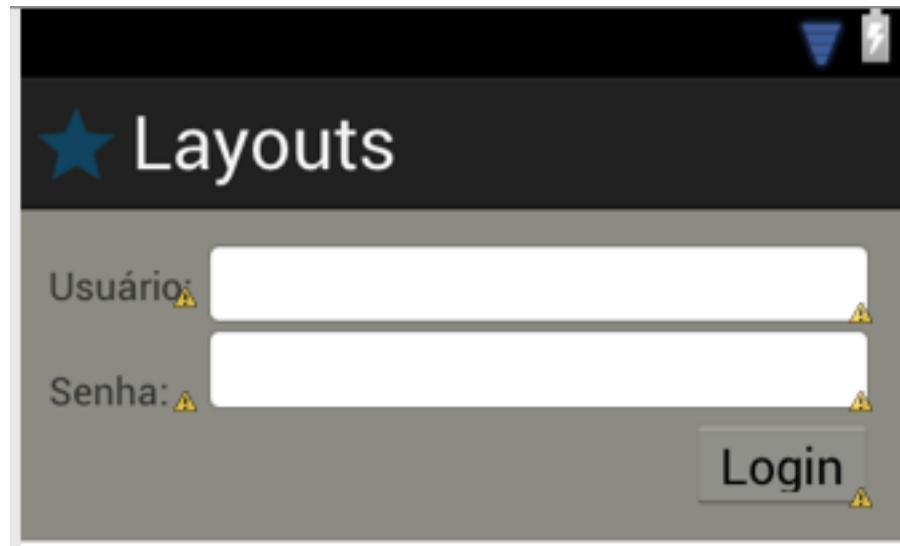
Exemplo - RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res.
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#8B8B83"
    android:padding="10dip">
    <TextView
        android:id="@+id/labelUsuario"
        android:layout_width="55dip"
        android:layout_height="wrap_content"
        android:text="Usuário: "/>
    <EditText
        android:id="@+id/campoUsuario"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_toRightOf="@+id/labelUsuario"/>
```

Exemplo - RelativeLayout

```
<TextView  
    android:id="@+id/labelSenha"  
    android:layout_width="55dip"  
    android:layout_height="wrap_content"  
    android:layout_below="@id/campoUsuario"  
    android:gravity="left"  
    android:text="Senha: "/>  
<EditText  
    android:id="@+id/campoSenha"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:background="@android:drawable/editbox_background"  
    android:layout_toRightOf="@id/labelSenha"  
    android:layout_alignTop="@id/labelSenha"  
    android:password="true" />  
<Button  
    android:id="@+id/btLogin"  
    android:layout_width="wrap_content"  
    android:layout_height="35dip"  
    android:layout_below="@id/campoSenha"  
    android:layout_marginTop="10dip"  
    android:layout_alignParentRight="true"  
    android:text="Login" />  
</RelativeLayout>
```

Exemplo - RelativeLayout

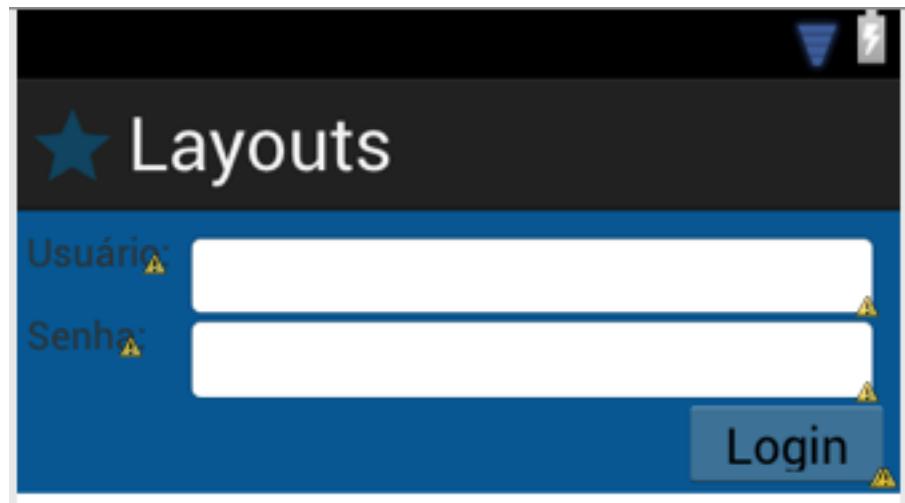


AbsoluteLayout

- Permite controlar exatamente a posição dos componentes na tela.
- Devem ser fornecidas as coordenadas x e y, utilizando-se os atributos android:layout_x e android:layout_y.
- android.widget.AbsoluteLayout.
- Problema deste layout são os diversos celulares e telas de resoluções diferentes.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:background="#005793">
    <TextView android:layout_x="5px" android:layout_y="10px"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Usuário:"/>
    <EditText android:layout_x="90px" android:layout_y="10px"
        android:layout_width="380px" android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"/>
    <TextView android:layout_x="5px" android:layout_y="55px"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Senha:"/>
    <EditText android:layout_x="90px" android:layout_y="55px"
        android:layout_width="380px" android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:password="true"/>
    <Button
        android:layout_x="360px" android:layout_y="100px"
        android:layout_width="wrap_content" android:layout_height="35dip"
        android:text="Login..."/>
</AbsoluteLayout>
```

Exemplo - AbsoluteLayout



TableLayout

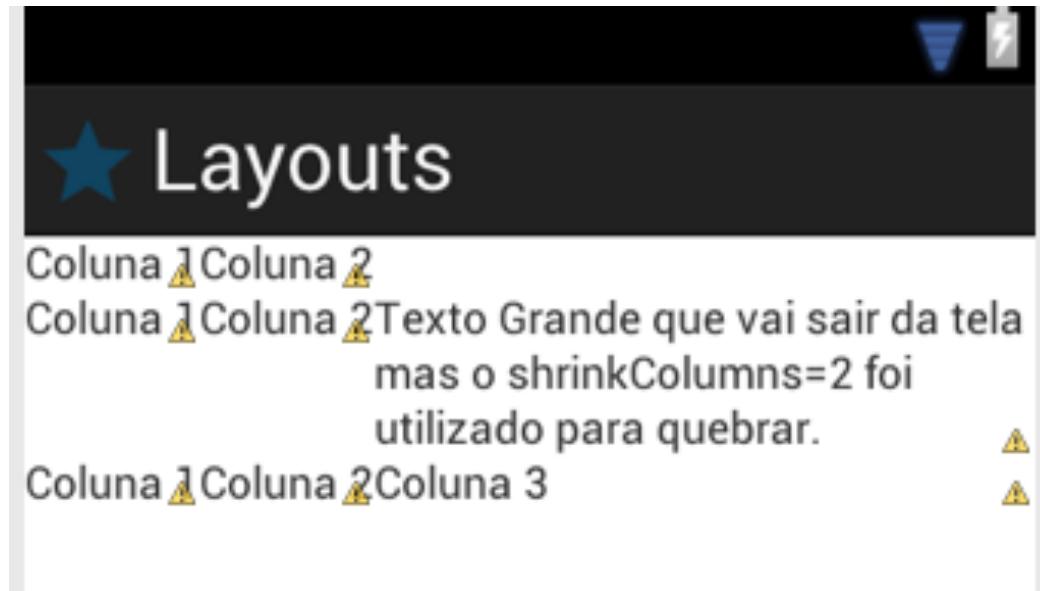
- Útil para construir formulários.
- Cada linha da tabela é formada por uma TableRow.
- A TableRow é uma subclasse de LinearLayout.
- Pode conter outros componentes, onde cada um representa uma coluna na tabela.
- android.widget.TableLayout.
- android.widget.TableRow.
- Atributos android:stretchColumns e android:shrinkColumns.
- android:stretchColumns faz com que colunas ocupem o espaço disponível na tela.
- android:shrinkColumns faz com que colunas sejam sempre exibidas na tela.

TableLayout e shrinkColumns

- Contração de colunas para que as mesmas caibam na tela.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:shrinkColumns="2">
    <TableRow>
        <TextView android:text="Coluna 1" />
        <TextView android:text="Coluna 2" />
    </TableRow>
    <TableRow>
        <TextView android:text="Coluna 1" />
        <TextView android:text="Coluna 2" />
        <TextView android:text="Texto Grande que vai sair da tela mas o shrinkColumns=2 foi utilizado para quebrar. " />
    </TableRow>
    <TableRow>
        <TextView android:text="Coluna 1" />
        <TextView android:text="Coluna 2" />
        <TextView android:text="Coluna 3" />
    </TableRow>
</TableLayout>
```

TableLayout e shrinkColumns



TableLayout e stretchColumns

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:stretchColumns="1" >
    <TextView android:text="Lista de Produtos" />
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:text="Produto A" />
        <TextView android:text="R$100,00" android:layout_gravity="right" />
    </TableRow>
    <TableRow>
        <TextView android:text="Produto B" />
        <TextView android:text="R$200,00" android:layout_gravity="right" />
    </TableRow>
    <TableRow>
        <TextView android:text="Produto C" />
        <TextView android:text="R$300,00" android:layout_gravity="right" />
    </TableRow>
    <View android:layout_height="2dip" android:background="#FF909090" />
    <LinearLayout
        android:layout_width="wrap_content" android:layout_height="fill_parent"
        android:gravity="bottom" >
        <Button      android:layout_width="wrap_content"
                    android:layout_height="35dip" android:text=" Cancelar " />
        <Button      android:layout_width="wrap_content"
                    android:layout_height="35dip" android:text=" Comprar " />
    </LinearLayout>
</TableLayout>
```

TableLayout e stretchColumns

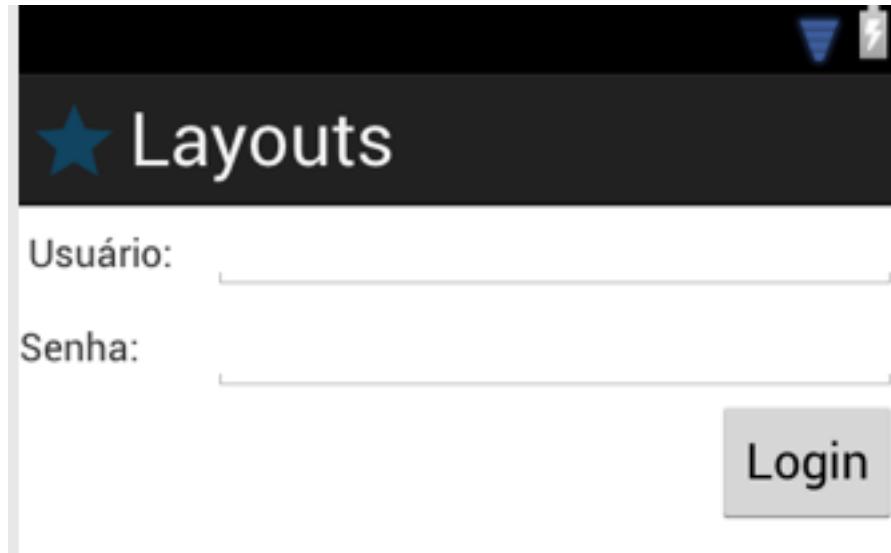
- Faz a expansão da coluna 2.
- É possível adicionar outros componentes que não sejam TableRow.
- Estes componentes também ocupam uma linha.
- Possui Views dentro da tabela.
- É um layout dentro de outro.



TableLayout e Formulários

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView android:text="Usuário: " android:padding="3dip" />
        <EditText android:id="@+id/campoLogin" android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView android:text="Senha: " />
        <EditText android:id="@+id/campoSenha" android:password="true" />
    </TableRow>
    <TableRow android:gravity="right">
        <Button android:id="@+id/login" android:text="Login" />
    </TableRow>
</TableLayout>
```

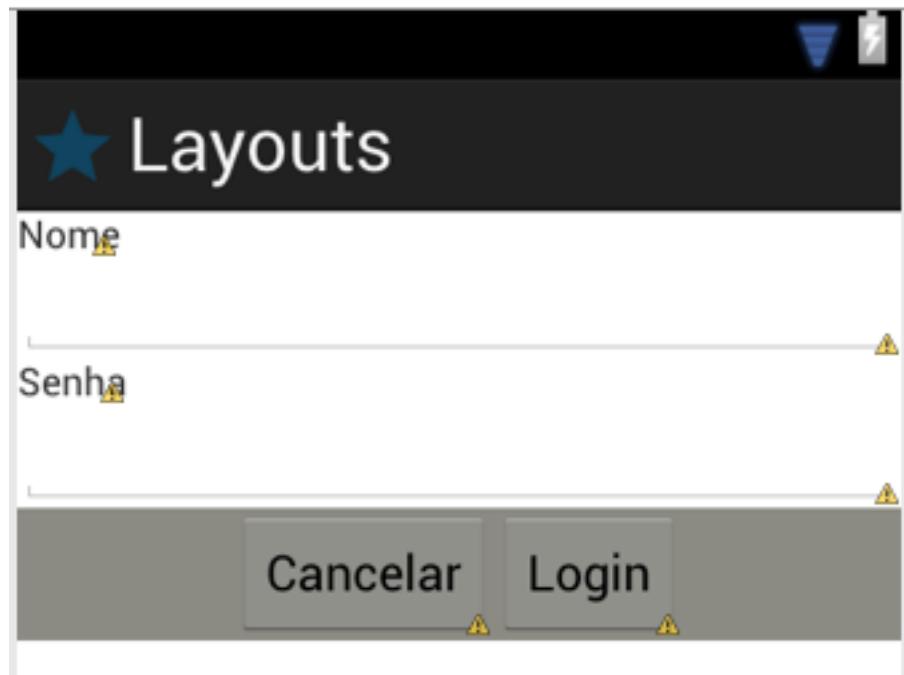
TableLayout e Formulários



Mais de um Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent" >
    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Nome" />
    <EditText
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:textColor="#ff0000" />
    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Senha" />
    <EditText
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:password="true" />
    <LinearLayout
        android:orientation="horizontal" android:gravity="center"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:background="#8B8BB83">
        <Button
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="Cancelar" />
        <Button
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="Login" />
    </LinearLayout>
</LinearLayout>
```

Mais de um Layout



ConstraintLayout

- Permite que criar layouts grandes e complexos com uma hierarquia flat no seu XML (sem a necessidade de ficar aninhando um monte de layouts uns dentro dos outros)
- Similar ao RelativeLayout, todas suas views se relacionam entre si e com o parent, mas é mais flexível e mais fácil de usar com o Layout Editor.
- Também possui as melhores características do LinearLayout que possui sua responsividade baseada em pesos.
- A ideia é usar somente o Layout Editor, sem necessidade de usar o XML.

* Adaptado de <https://developer.android.com/training/constraint-layout/index.html>

ConstraintLayout

- Disponível a partir da API 9.
- Constraints significa restrições ou limitações.
 - Essas restrições são importantes para o funcionamento deste layout.
- Cada constraint define a posição da view a partir de seus eixos vertical e horizontal.
- Portanto deve-se adicionar ao menos uma constraint horizontal e uma vertical para a view.
- Cada constraint representa uma conexão ou alinhamento em relação à outra view, o layout parent ou mesmo uma linha-guia invisível

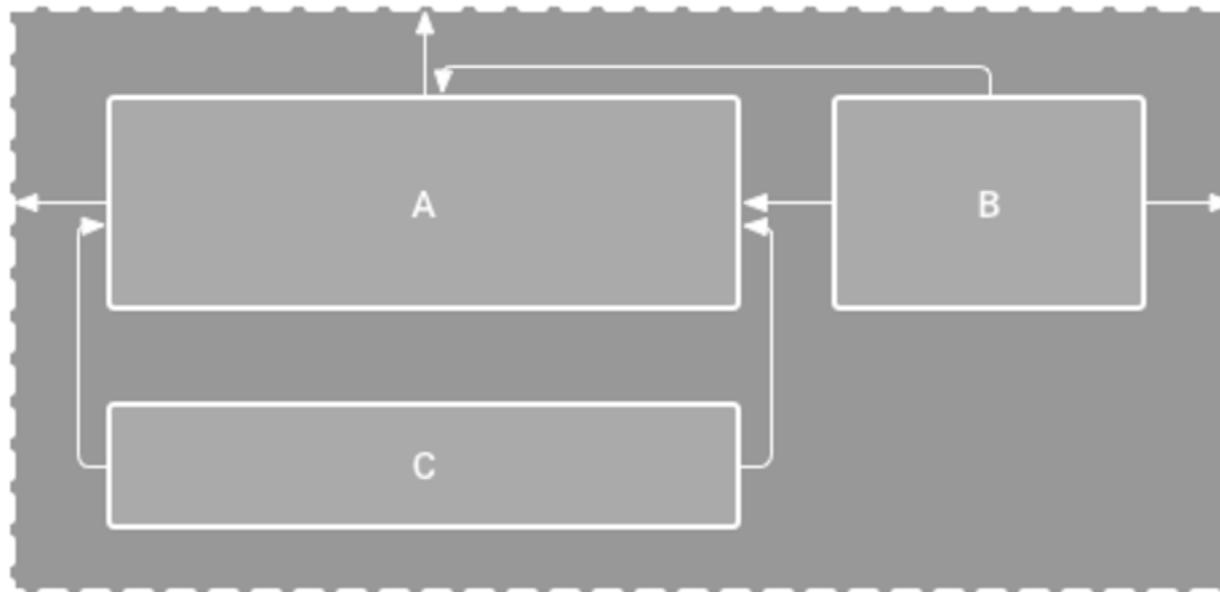
ConstraintLayout

- Usando o Layout Editor:
 - Quando se arrasta e solta uma view no Layout Editor, ela fica exatamente onde você a deixou, mesmo que não possua constraint alguma.
 - No entanto, isso é apenas para tornar o seu trabalho mais fácil quando estiver posicionando os elementos.
 - Se uma view não possui constraints, ela ficará no canto superior esquerdo da tela automaticamente (0,0).

ConstraintLayout

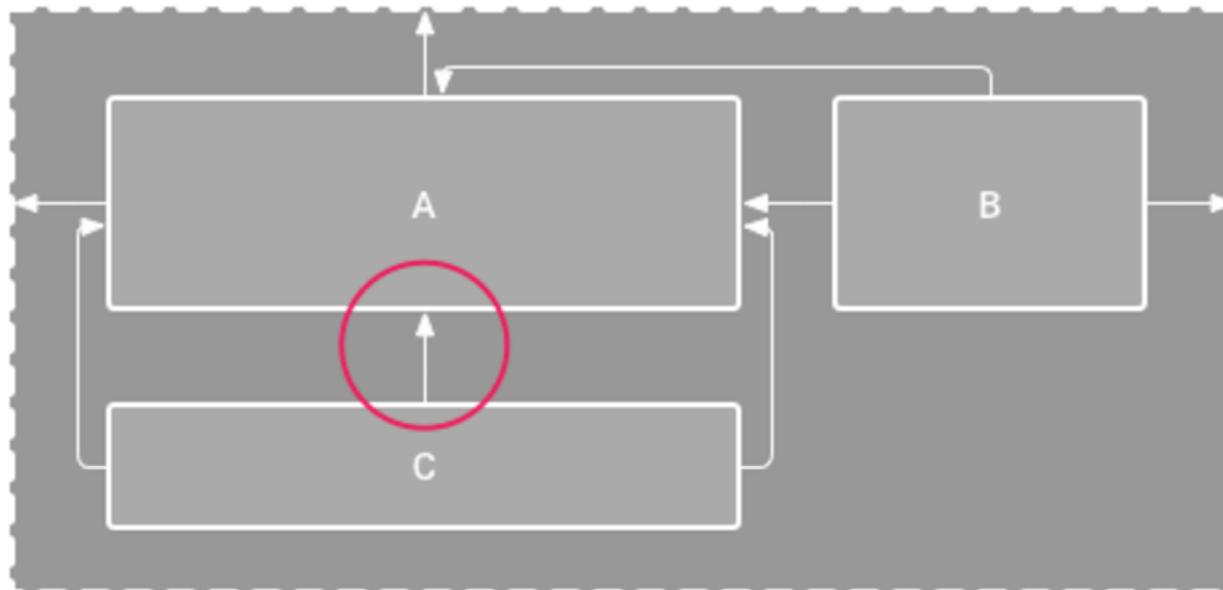
- Usando o Layout Editor:

- Na figura abaixo, o layout parece ok no editor, mas não há constraint vertical na view C.
- Quando este layout renderizar em um dispositivo, a view C se alinhará horizontalmente com as faces esquerda e direita da view A, mas irá aparecer no topo da tela porque não há constraint vertical.



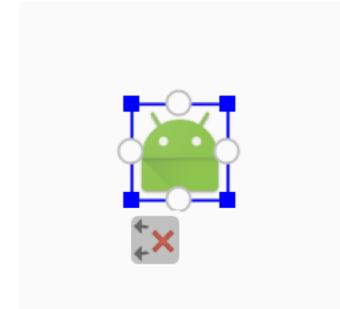
ConstraintLayout

- Usando o Layout Editor:
 - O correto seria adicionar uma constraint vertical entre as views A e C.



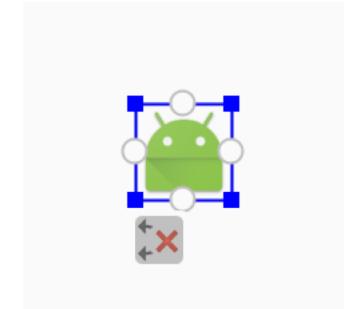
ConstraintLayout

- Crie um projeto qualquer para iniciarmos um exemplo com ConstraintLayout.
- Vá para o XML da tela principal e abra o editor visual.
- Remova qualquer view se o layout possuir alguma por padrão.
- Arraste um ImageView e escolha uma imagem qualquer.
- Quando você solta o componente ele exibe bordas ao seu redor com ícones quadrados para redimensionamento nos cantos do componente e ícones circulares para constraints nas laterais dele.



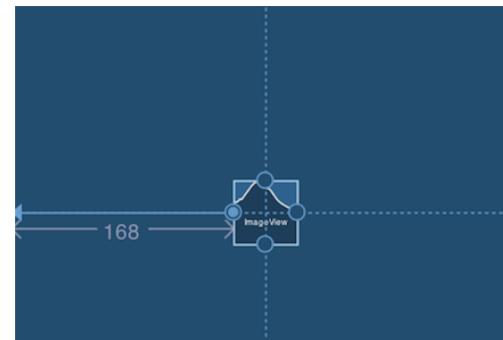
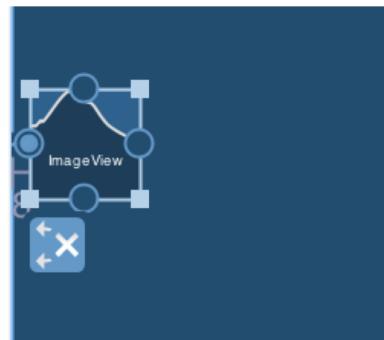
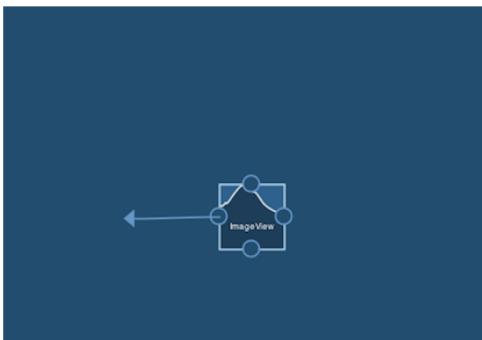
ConstraintLayout

- Crie um projeto qualquer para iniciarmos um exemplo com ConstraintLayout.
- Vá para o XML da tela principal e abra o editor visual.
- Remova qualquer view se o layout possuir alguma por padrão.
- Arraste um ImageView e escolha uma imagem qualquer.
- Quando você solta o componente ele exibe bordas ao seu redor com ícones quadrados para redimensionamento nos cantos do componente e ícones circulares para constraints nas laterais dele.



ConstraintLayout

- Clique no componente para selecioná-lo.
- Então clique e segure um dos ícones circulares (manipuladores de constraints) arrastando até um ponto de ancoragem disponível (a face de outro componente, do parent layout ou uma linha-guia).
- Quando você soltar, a constraint será criada, com uma margem default separando os dois componentes.

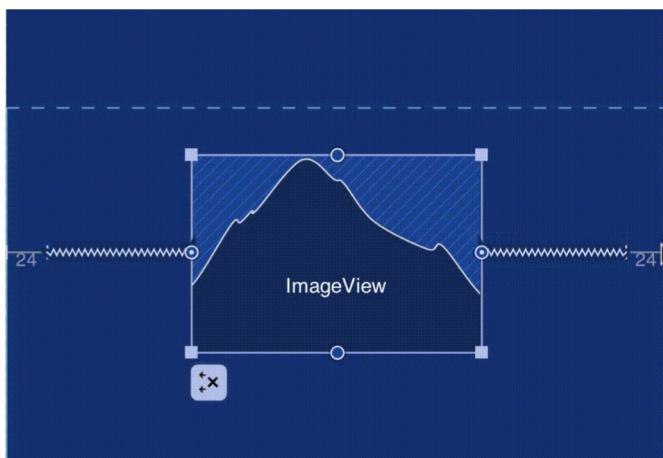


ConstraintLayout

- Regras:
 - Cada componente deve ter ao menos duas constraints, uma vertical e uma horizontal.
 - Você pode criar constraints somente entre um constraint handle (o ícone circular) e um ponto de ancoragem que compartilhem o mesmo plano, ou seja, vertical com vertical e horizontal com horizontal.
 - Cada constraint handle pode ser usado para apenas uma constraint, mas você pode criar várias constraints (de diferentes componentes) para um mesmo ponto de ancoragem.

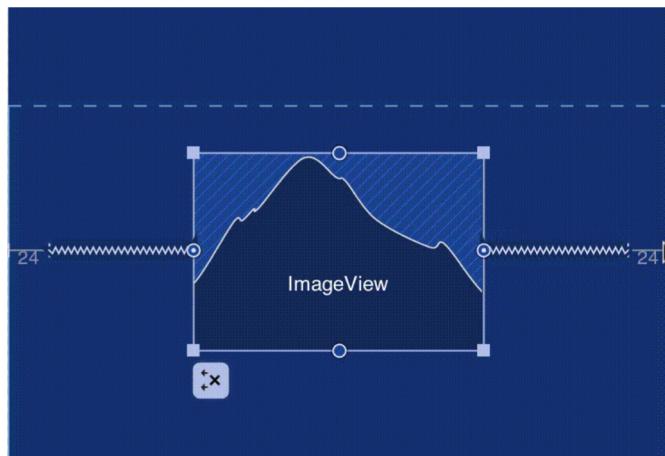
ConstraintLayout

- Para remover uma constraint, seleciona a view e então clique no constraint handle, ou, remova todas constraints selecionando a view e depois clicando em “Clear Constraints”.
- Se você adicionar constraints opostas em um componente, as linhas da constraint se tornam serrilhadas como na imagem abaixo, indicando forças opostas.



ConstraintLayout

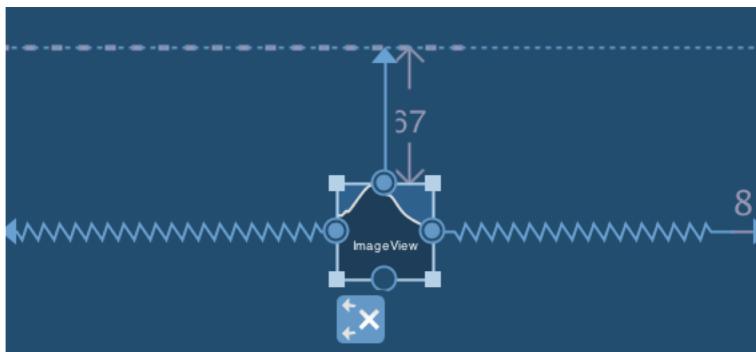
- Para remover uma constraint, seleciona a view e então clique no constraint handle, ou, remova todas constraints selecionando a view e depois clicando em “Clear Constraints”.
- Se você adicionar constraints opostas em um componente, as linhas da constraint se tornam serrilhadas como na imagem abaixo, indicando forças opostas.



- Se “wrap content”, centralizado entre as constraints.
- Se “match parent”, componente é esticado.

ConstraintLayout

- É possível criar linhas invisíveis para guiar o posicionamento no layout.
- Tanto vertical, quanto horizontal.
- Clique no ícone de adicionar guidelines na toolbar.



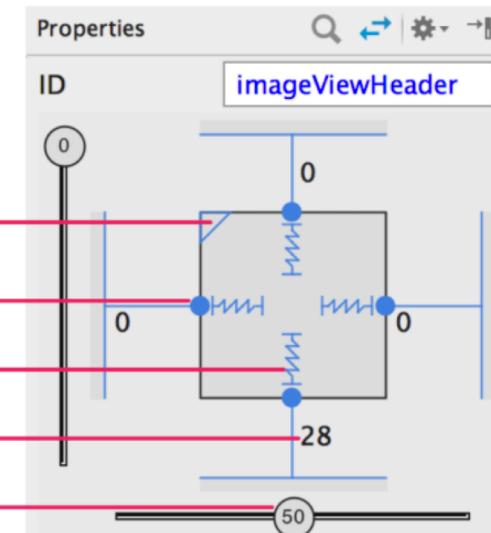
- Adicione uma guia horizontal e posicione a ImageView em relação a ela.

ConstraintLayout

- Você pode usar os cantos do seu componente para redimensioná-lo, mas isso não é recomendado uma vez que adiciona um valor de largura e altura literal à ele, impedindo-o de se ajustar às diferentes resoluções de telas.
- Para selecionar modos de redimensionamento recomendados, clique no componente e abra a janela de propriedades dele no lado direito do editor.
- Próximo ao topo da janela de propriedades está o View Inspector, que inclui controles para muitas propriedades de layout.

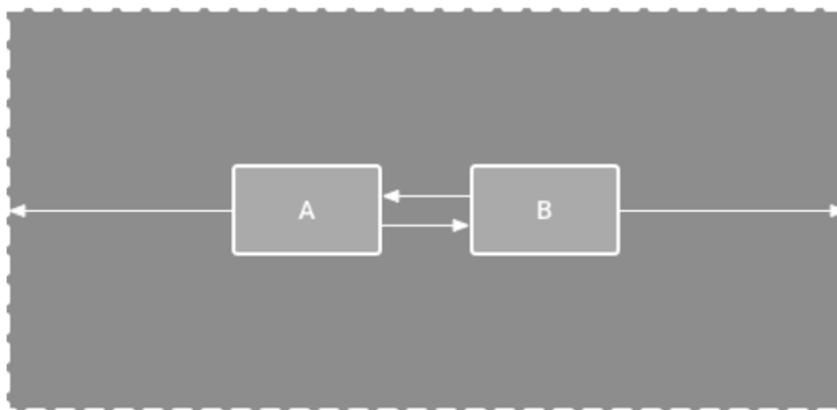
ConstraintLayout

- Esta janela inclui controles para
 - 1) proporção do tamanho
 - 2) excluir constraints
 - 3) modo de altura e largura
 - 4) margens
 - 5) viés das constraints (tipo um offset)
- O item 3 define como a altura e largura do componente será calculada.
 - **Wrap Content (ícone de setas)**: clássico, o tamanho é ajustado conforme o conteúdo.
 - **Match Constraints (ícone serrilhado)**: o componente se estica para preencher as constraints, excluindo margens. Usar esse modo permite definir uma proporção de tamanho (16:9, por exemplo).
 - **Fixed (ícone reto)**: valores literais, não recomendado.



ConstraintLayout

- Um grupo de componentes podem ser ligados usando constraints bi-direcionais uns com os outros, formando uma corrente.



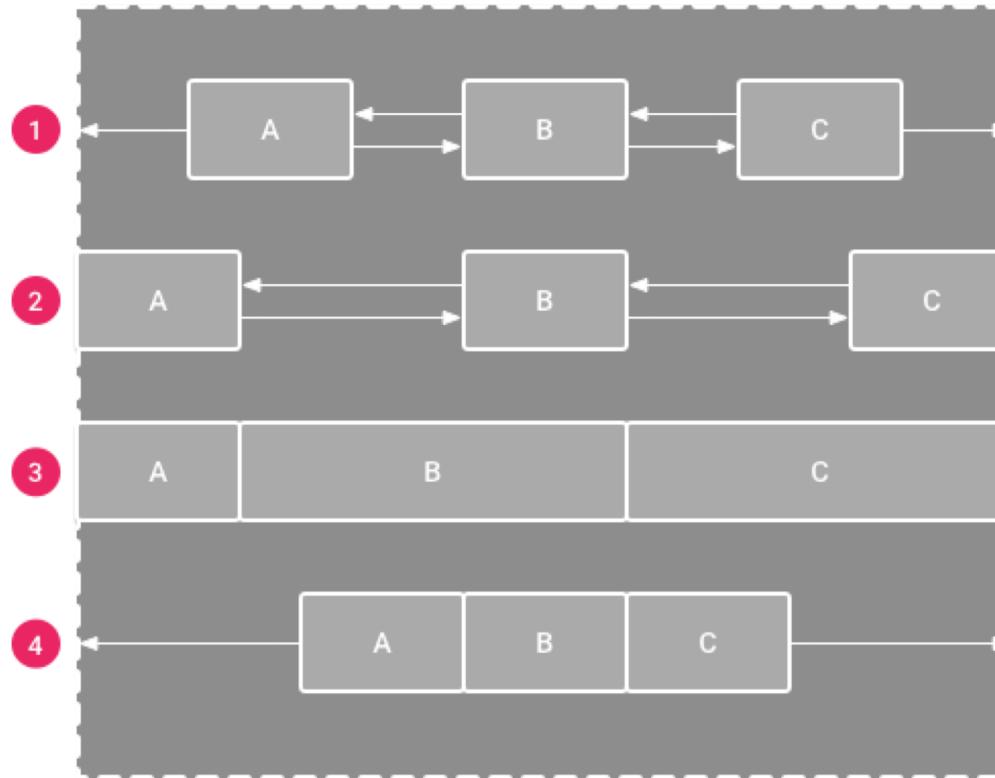
- Uma corrente permite que você distribua os componentes horizontalmente ou verticalmente.

ConstraintLayout

- Spread
 - Default, os componentes são distribuídos de maneira uniforme, após suas margens serem calculadas.
- Spread inside
 - São respeitadas as constraints de cada extremidade e o restante do espaço é distribuído uniformemente.
- Weighted
 - Quando a corrente é definida como Spread ou Spread Inside, você pode colocar o tamanho de seus componentes como “match constraint” para que eles ocupem todo o espaço disponível uniformemente.
 - Caso deseje que um componente ocupe mais espaço que o outro, você pode definir pesos diferenciados para eles usando as propriedades `layout_constraintHorizontal_weight` e `layout_constraintVertical_weight`, assim como funcionava no LinearLayout clássico.

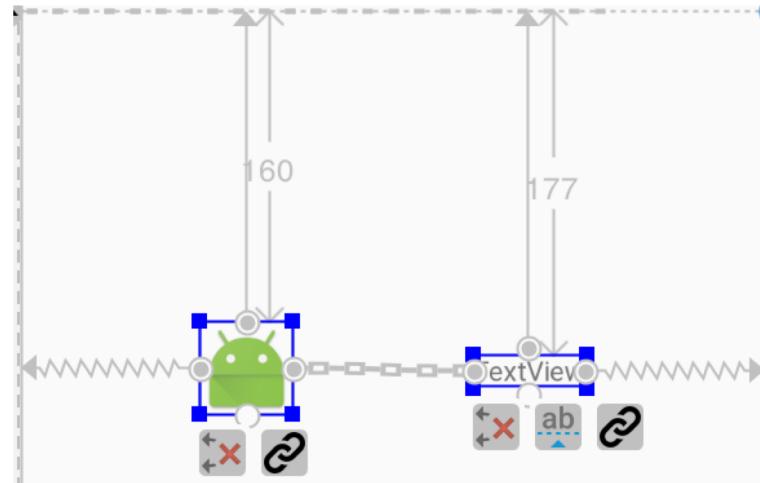
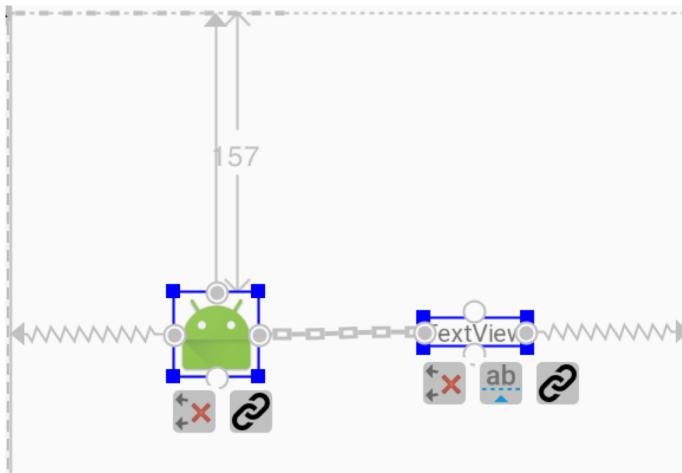
ConstraintLayout

- Packed
 - Os componentes ficam grudados uns nos outros.



ConstraintLayout

- Para criar uma corrente de componentes rapidamente, selecione todos eles e depois com o clique direito do mouse escolha Center Horizontally ou Center Vertically, para criar a corrente na respectiva orientação.



ConstraintLayout

- Ao invés de adicionar constraints manualmente em cada um dos componentes, é possível adicionar todos eles nas posições que desejar e então clicar no botão Infer Constraints para criar constraints automáticas.



- Infer Constraints escaneia o layout para determinar o conjunto de constraints mais eficiente para todos os componentes.
- Ele faz o melhor possível para manter todos os componentes nas posições atuais ao mesmo tempo que tenta deixar o layout flexível.
- Alguns ajustes posteriores podem ser necessários para garantir que o layout realmente ficará responsivo.

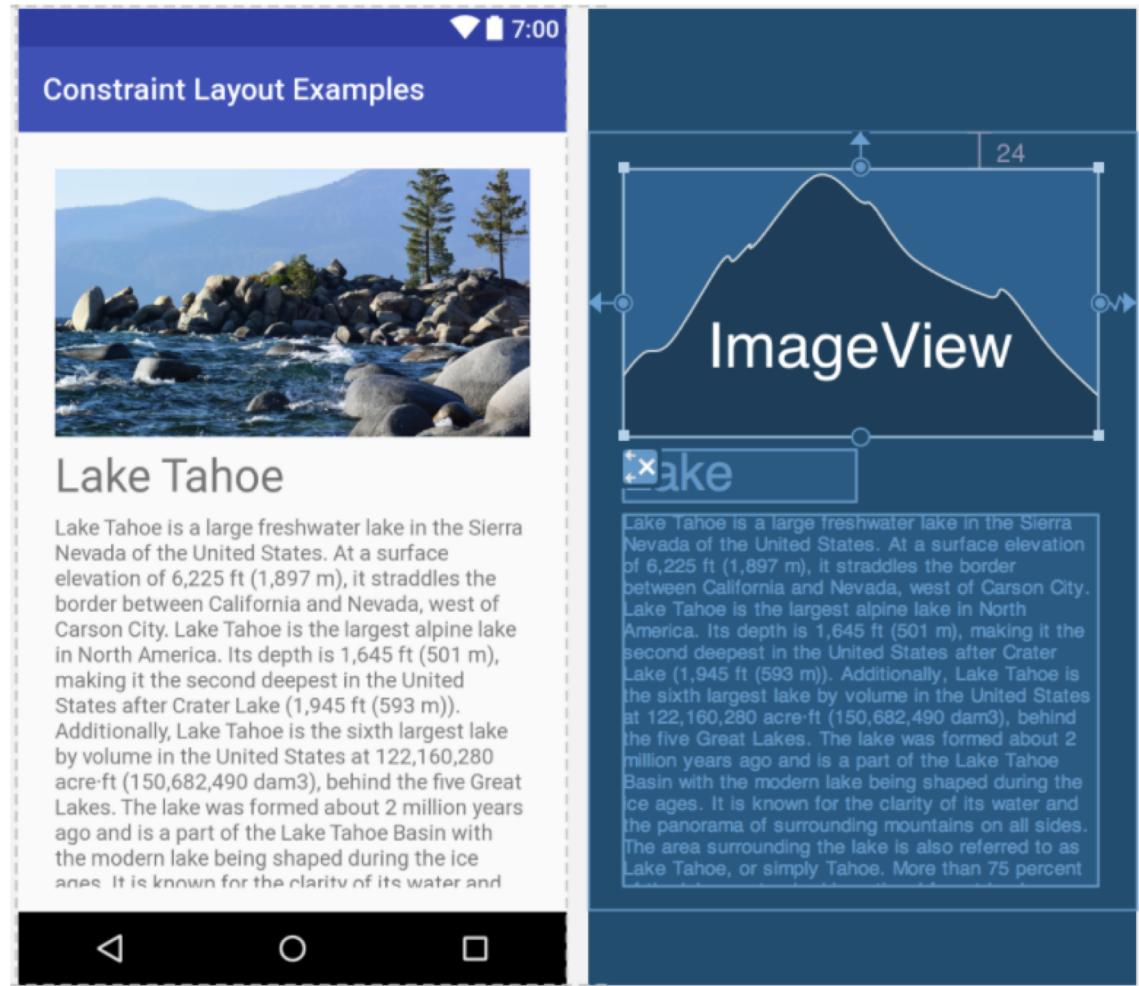
ConstraintLayout

- Autoconnect é outra funcionalidade separada que pode ser ligada e desligada.
- Quando ligada, ela automaticamente cria duas ou mais constraints para cada componente que você arrastar para o layout, mas somente considerando constraints ao parent layout, nunca a outros componentes.
- Por padrão esta opção vem desabilitada, mas você pode habilitá-la apenas clicando em seu ícone na toolbar.



ConstraintLayout

- Use o ConstraintLayout para elaborar uma tela similar ao exemplo abaixo.



ConstraintLayout

- Use o ConstraintLayout para elaborar uma tela similar ao exemplo ao lado.

Feature Article



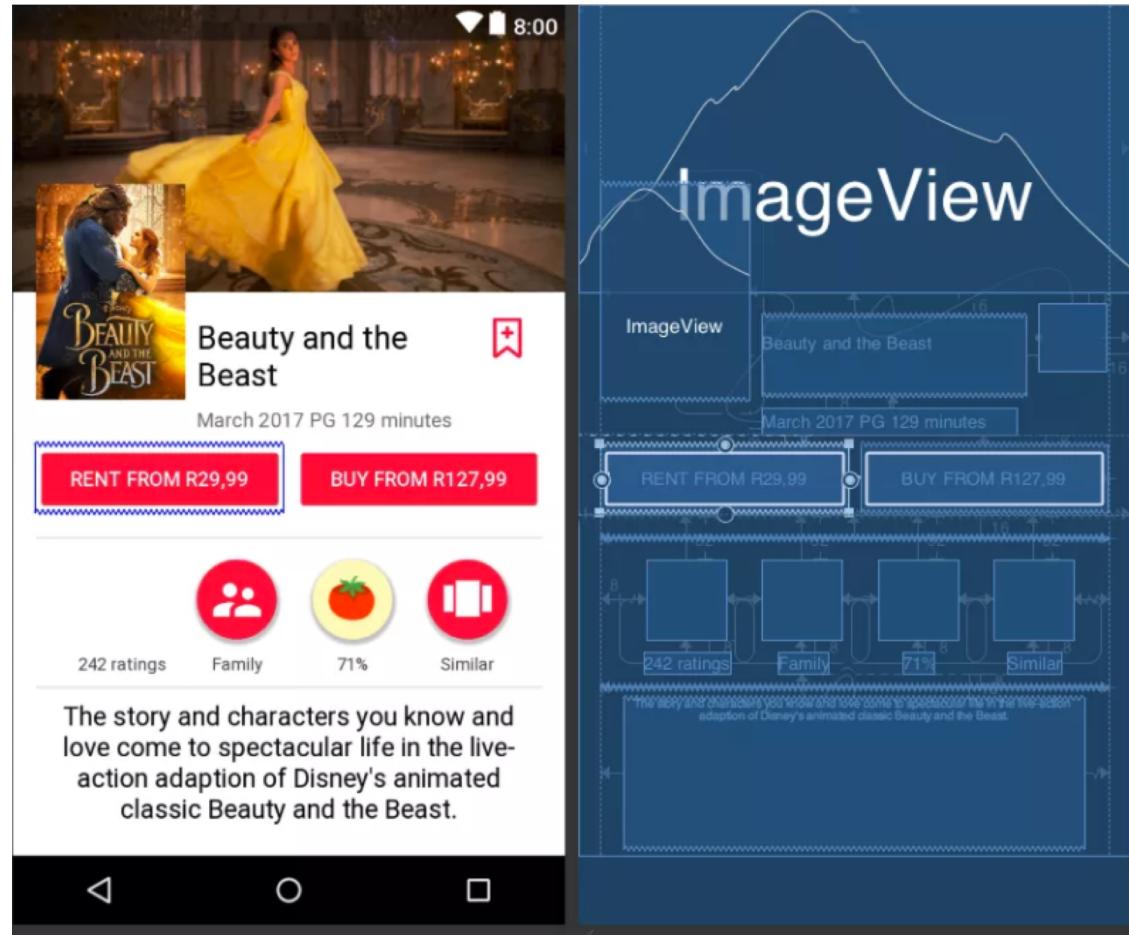
Smart and Connected Communities

A new emerging concept called smart and connected communities address preservation, revitalization, livability, and sustainability. With the integration of IoT technologies, such as mobile crowdsourcing and cyber-physical cloud computing, we can anticipate a number of benefits, including enhanced connectivity, improved living conditions, and overall "smarter" communities.

[Read more at IEEE Transmitter.](#)

ConstraintLayout

- Use o ConstraintLayout para elaborar uma tela similar ao exemplo abaixo.



ScrollView

- Telas que contém muitos elementos e que seja necessário fazer rolagem.
- A classe ScrollView é subclasse da FrameLayout.
- Portanto, pode receber apenas um componente que vai ocupar o tamanho inteiro da tela.
- Normalmente, é colocado outro layout dentro do ScrollView, desta forma pode-se colocar diversos componentes.

Exemplo - ScrollView

- Faça um novo projeto ExemploScrollView.
- Acerte o layout.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" >  
    <LinearLayout  
        android:id="@+id/layout1"  
        android:layout_width="fill_parent" android:layout_height="wrap_content"  
        android:orientation="vertical">  
        <!--  
            100 TextViews serão adicionados dinamicamente aqui  
        -->  
    </LinearLayout>  
</ScrollView>
```

Exemplo - ScrollView

- Código .java.

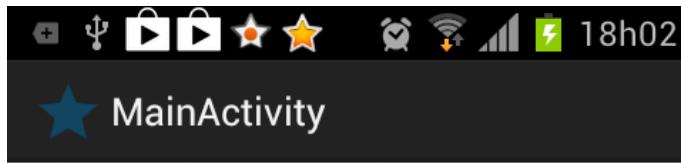
```
package br.ufpr.layouts;

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

public class ExemploScrollView extends Activity {
    @Override
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.activity_main);

        LinearLayout layout = (LinearLayout) findViewById(R.id.layout1);
        for (int i = 0; i < 100; i++) {
            TextView text = new TextView(this);
            text.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
            text.setText("Texto: " + i);
            layout.addView(text);
        }
    }
}
```

Exemplo - ScrollView



Texto: 6

Texto: 7

Texto: 8

Texto: 9

Texto: 10

Texto: 11

Texto: 12

Texto: 13

Texto: 14

Texto: 15

Texto: 16

Texto: 17

Texto: 18

Texto: 19

Texto: 20

Texto: 21

Texto: 22

Texto: 23

Texto: 24

Texto: 25

Texto: 26

Texto: 27

Texto: 28

Texto: 29

GridView

- Utilizada para exibir componentes em uma tabela.
- Seu uso clássico é para exibir várias imagens como em um álbum de fotos.
- android.widget.GridView.

Exemplo - GridView

- Faça um projeto ExemploGridView.
- Acerte o layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Exemplo de GridView:" />
    <GridView android:id="@+id/grid1"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:padding="10dip"
        android:verticalSpacing="10dip"
        android:horizontalSpacing="10dip"
        android:numColumns="auto_fit"
        android:columnWidth="40dip"
        android:gravity="center" />
</LinearLayout>
```

Exemplo - GridView

★ Layouts					
Exemplo de GridView					
Ite	Ite	Ite	Ite	Ite	Ite
m	m	m	m	m	m
1	2	3	4	5	6
Sub Item	Sub Item	Sub Item	Sub Item	Sub Item	Sub Item
1	2	3	4	5	6
Ite	Ite	Ite	Ite	Ite	Ite
m	m	m	m	m	m
7	8	9	10	11	12
Sub Item	Sub Item	Sub Item	Sub Item	Sub Item	Sub Item
7	8	9	10	11	12
Ite	Ite	Ite	Ite	Ite	Ite
m	m	m	m	m	m
13	14	15	16	17	18
Sub Item	Sub Item	Sub Item	Sub Item	Sub Item	Sub Item

Exemplo - GridView

- Para exibir as imagens no GridView é necessário criar um ListAdapter que retorne uma lista com as imagens.
- O GridView deve ser recuperado no código para podermos passar então as imagens.
- A classe BaseAdapter é útil para implementar um ListAdapter utilizando um array de itens.

Exemplo - GridView

- Classe AdaptadorImg

```
package br.ufpr.layouts;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;

public class AdaptadorImg extends BaseAdapter {
    private Context ctx;
    private final int[] imagens;
    private final android.view.ViewGroup.LayoutParams params;
    public AdaptadorImg(Context c, int[] imagens,
            android.view.ViewGroup.LayoutParams params) {
        this.ctx = c;
        this.imagens = imagens;
        this.params = params;
    }
    public int getCount() {
        return imagens.length;
    }
    ...
}
```

Exemplo - GridView

```
public Object getItem(int posicao) {
    return posicao;
}
public long getItemId(int posicao) {
    return posicao;
}
public View getView(int posicao, View convertView, ViewGroup parent) {
    ImageView img = new ImageView(ctx);
    img.setImageResource(imagens[posicao]);
    img.setAdjustViewBounds(true);
    img.setLayoutParams(params);
    return img;
}
```

- O construtor recebe o array de imagens e os parâmetros de layout para definir o tamanho de cada imagem do grid.
- getView retorna uma imagem.

Exemplo - GridView

- Classe principal.

```
package br.ufpr.layouts;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

public class ExemploGridView extends Activity {
    // array com os ids das imagens
    private int[] imagens = { R.drawable.smile1, R.drawable.smile2,
        R.drawable.smile1, R.drawable.smile2, R.drawable.smile1,
        R.drawable.smile2, R.drawable.smile1, R.drawable.smile2 };
```

Exemplo - GridView

```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.activity_main);

    GridView grid = (GridView) findViewById(R.id.grid1);
    grid.setAdapter(new AdaptadorImg(this, imagens,new GridView.LayoutParams(30, 30)));

    grid.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView parent, View v, int posicao, long id) {
            Toast.makeText(ExemploGridView.this,"Imagen selecionada: " + posicao, Toast.LENGTH_SHORT).show();
        }
    });
}
```

- O método setOnItemClickListener() pode ser utilizado para tratar os eventos gerados caso o usuário pressione alguma imagem.
- No método onItemClick é possível recuperar qual imagem foi selecionada.

Exemplo - GridView

