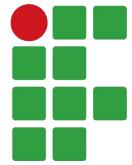


**INSTITUTO FEDERAL**  
Paraná  
Campus Paranaguá

# Interface Gráfica no Android

Dispositivos Móveis

Prof. Diego Stiehl

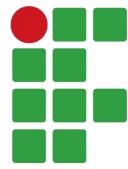


# Criação da Activity

- Quando criamos uma Activity pelo assistente do Android Studio, ganhamos:
  - Entrada no arquivo `AndroidManifest.xml`
    - Informar ao Android que a Activity existe
  - Classe Kotlin (ou Java)
    - Implementação de todas funcionalidades
  - **Arquivo app/layouts/activity\_nome.xml**
    - Criação da interface gráfica

# Estrutura do XML de Layout

- Todo **XML de layout** deve exatamente um (1) elemento raiz, podendo ser:
  - Um container para outros elementos
  - Um elemento simples
- Os elementos são adicionados por tags XML
- Cada elemento (tag) pode ter atributos
  - Afetam a si próprio
  - Podem afetar seus elementos filhos (containers)



# Estrutura do XML de Layout

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

Elemento raiz (fecha ao final)

Definição de namespaces

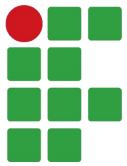
Atributos

Elemento filho do raiz

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

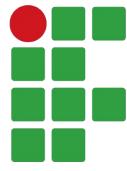
Tag autocontida

Fechamento do Elemento raiz



# Namespaces

- Usamos três namespaces XML para os atributos dos elementos
  - `xmlns:android`
    - Atributos padrão do Android
  - `xmlns:tools`
    - Atributos válidos apenas para o editor gráfico
    - Não vai para a aplicação compilada
  - `xmlns:app`
    - Atributos customizados do app

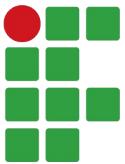


# Usando os Namespaces

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

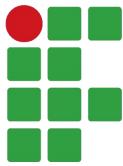
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
   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

</android.support.constraint.ConstraintLayout>
```



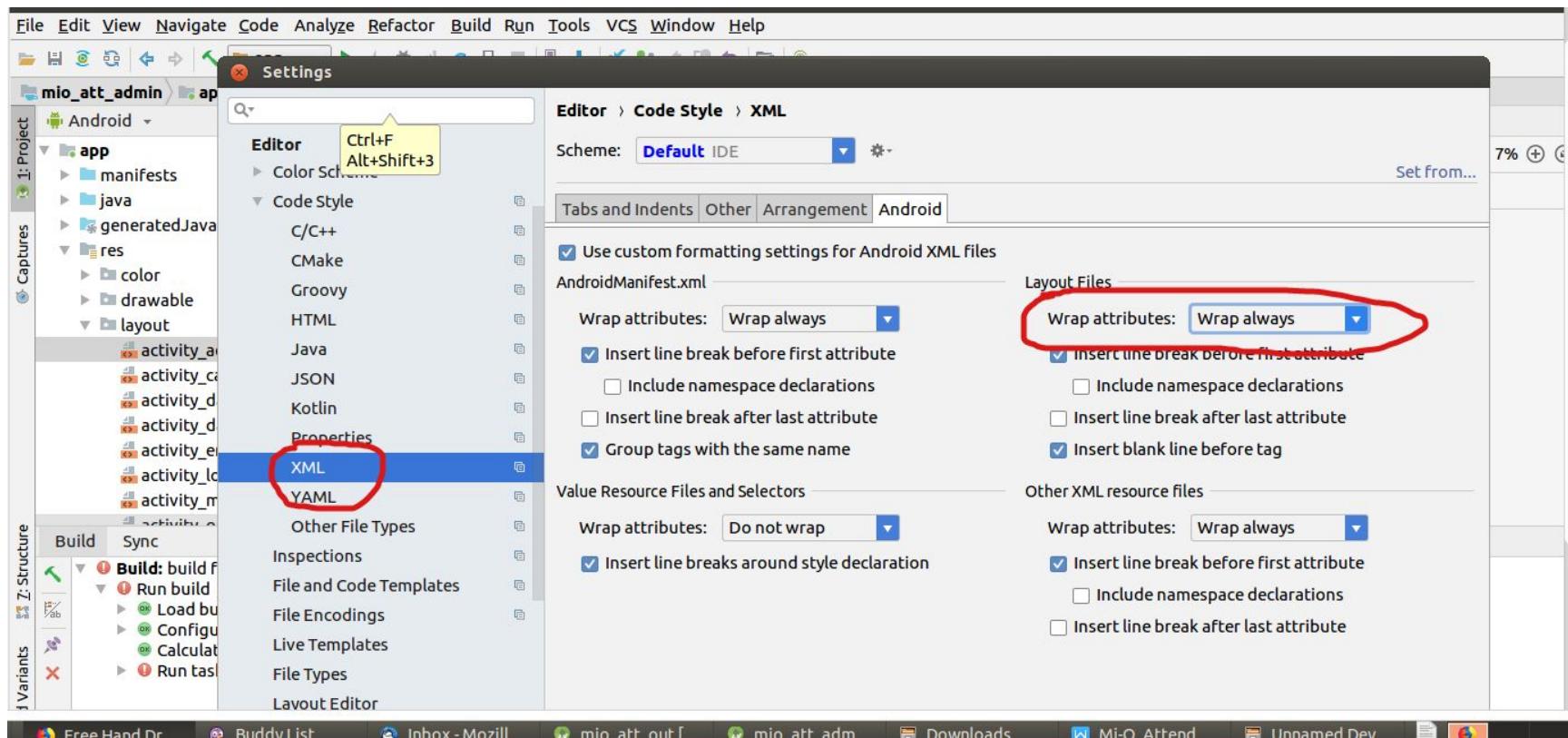
# Dica

- Sempre que possível, utilize a seguinte opção:
  - Code → Reformat Code (Ctrl+Alt+L)
    - Cuidado com o atalho no Ubuntu
- No XML ele vai:
  - Arrumar indentações
  - Reduzir quebras de linha em branco
  - Definir um atributo por linha
  - Ordenar os atributos alfabeticamente

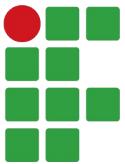


# Bug

- Arrumar “bug” do Android Studio 3.2.1

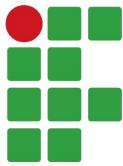


Resposta completa: <https://stackoverflow.com/a/52853796>

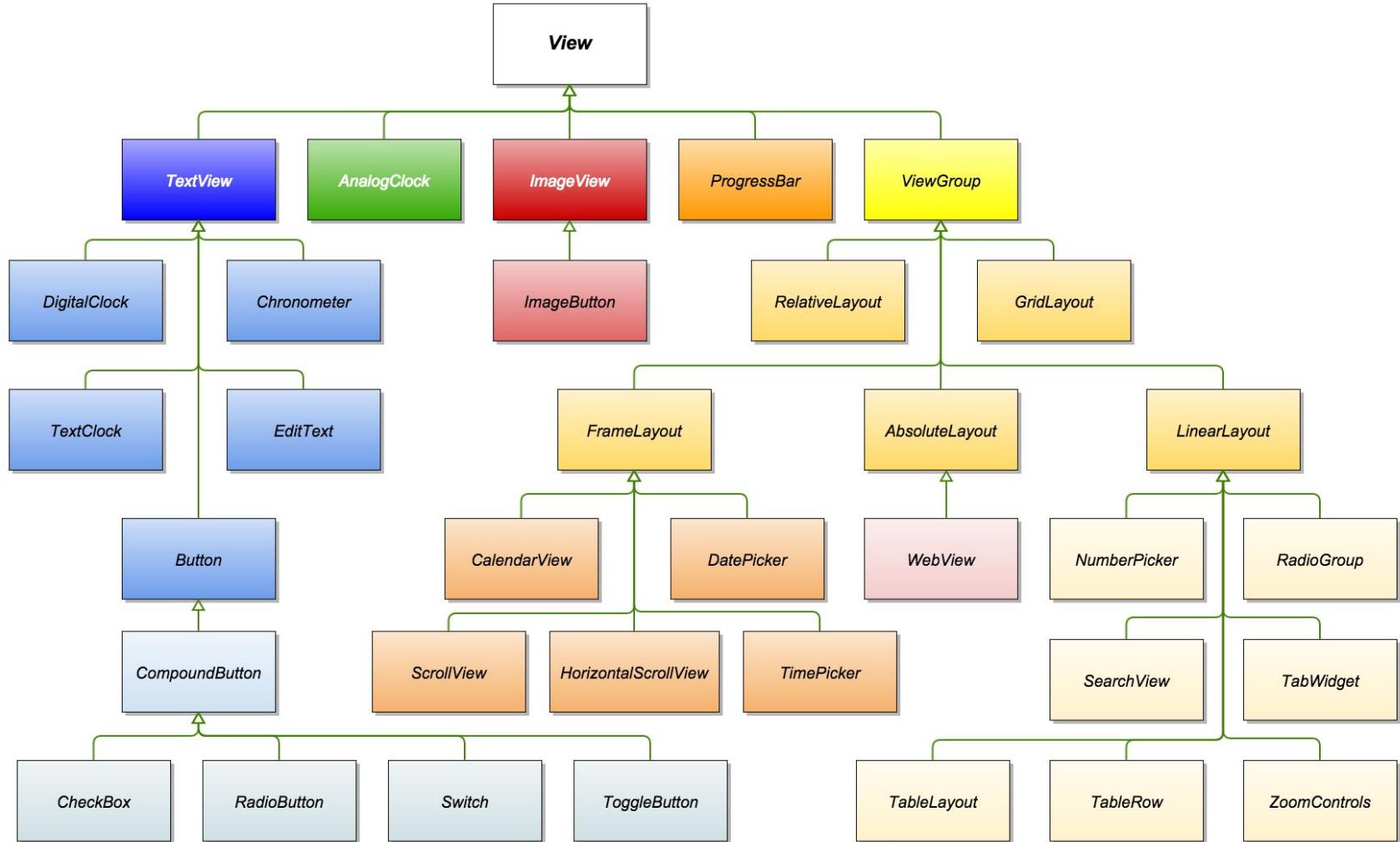


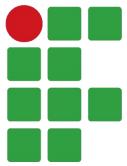
# Views

- Tudo que pode ser mostrado em uma interface gráfica do Android é uma View
  - Estende da classe android.view.View
- Pode ser de dois tipos
  - Simples (widget)
    - TextView, Button, ImageView, ProgressBar, ...
  - Gerenciador de Layout (ViewGroup)
    - LinearLayout, Frame, RelativeLayout, ...



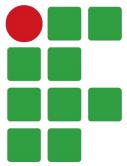
# Classe View





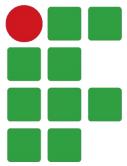
# ViewGroup

- Muitas classes estendem ViewGroup
  - Muitas para fins específicos
- Para layouts básicos, as principais são:
  - FrameLayout
  - LinearLayout
  - GridLayout
  - TableLayout
  - ScrollView
  - RelativeLayout
  - ConstraintLayout



# Layouts

- FrameLayout
  - Muito simples
  - Empilha as Views umas sobre as outras
  - Comumente usado para conter só um elemento
- LinearLayout
  - Exclusivamente unidirecional
  - Permite a criação de layouts verticais ou horizontais
  - Fácil alocação de Views em um eixo

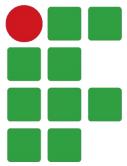


# Layouts

- **GridLayout**
  - Permite a disposição dos elementos em mosaico
- **TableLayout**
  - Disposição em formato de tabela
  - Subclasse de `LinearLayout`
- **ScrollView**
  - Cria uma barra de rolagem quando o seu elemento-filho é muito grande

# Layouts - Os Mais Importantes

- **RelativeLayout**
  - Dentro dele, os elementos têm posicionamento relativo uns aos outros
  - Layouts mais complexos e dinâmicos
  - Menor aninhamento de elementos
- **ConstraintLayout**
  - Semelhante ao RelativeLayout
    - Com mais recursos
  - Sua utilização é indicada pelo Google

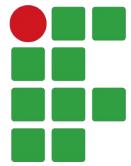


# Largura e Altura

- As duas únicas propriedades **obrigatórias** em todas Views são a Largura e Altura
- Atributos:

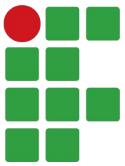
```
<Button  
    android:layout_width="xxxxxx"  
    android:layout_height="yyyyyy"  
    android:text="Me clique!" />
```

- O atributo android:text, apesar de essencial para o Button, não é obrigatório



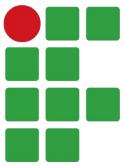
# Largura e Altura

- Os dois atributos podem ser expressos das seguintes maneiras
  - **match\_parent**
    - Expande até ocupar todo o container pai
  - **wrap\_content**
    - Contraí para ocupar apenas o necessário para que o conteúdo da View seja mostrado
  - **Tamanho fixo**
    - Tamanho especificado pelo programador
    - Utilizar unidade de medida dp
      - 5dp, 10dp, 200dp



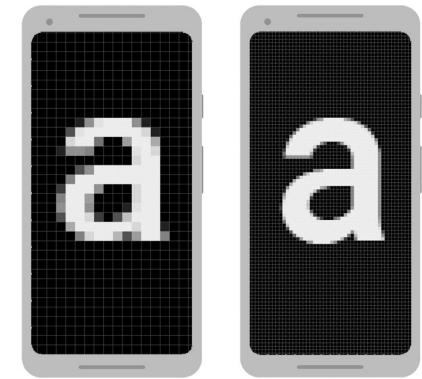
# Prática

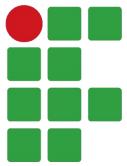
- Crie uma Activity
- No XML
  - Troque o ConstraintLayout por FrameLayout
  - Adicione um Button
  - Usando as três formas vistas
    - Manipule a altura do elemento
    - Manipule a altura do elemento
- Observação
  - Sempre verifique o resultado na aba Design



# dp (Density-independent Pixels)

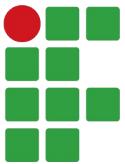
- O **dp** é a unidade de medida recomendada
- É baseada na densidade da tela
  - Quantidade de pixels por polegada<sup>2</sup>
  - Quanto maior a densidade, melhor a qualidade da imagem na tela
- O dp calcula um tamanho dinâmico de pixels para cada diferente densidade de tela
- Os elementos tendem a ter tamanhos equivalentes e razoáveis em diferentes telas





# LinearLayout

- Elementos são distribuídos em um eixo
- Podemos controlar tamanhos e alinhamentos
  - Para cada elemento
- Propriedades importantes
  - **android:orientation**
  - **android:layout\_gravity** (nos filhos)
  - **android:layout\_weight** (nos filhos)
  - Além das dimensões (obrigatórias)



# LinearLayout

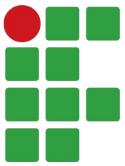
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="start" />

    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end" />

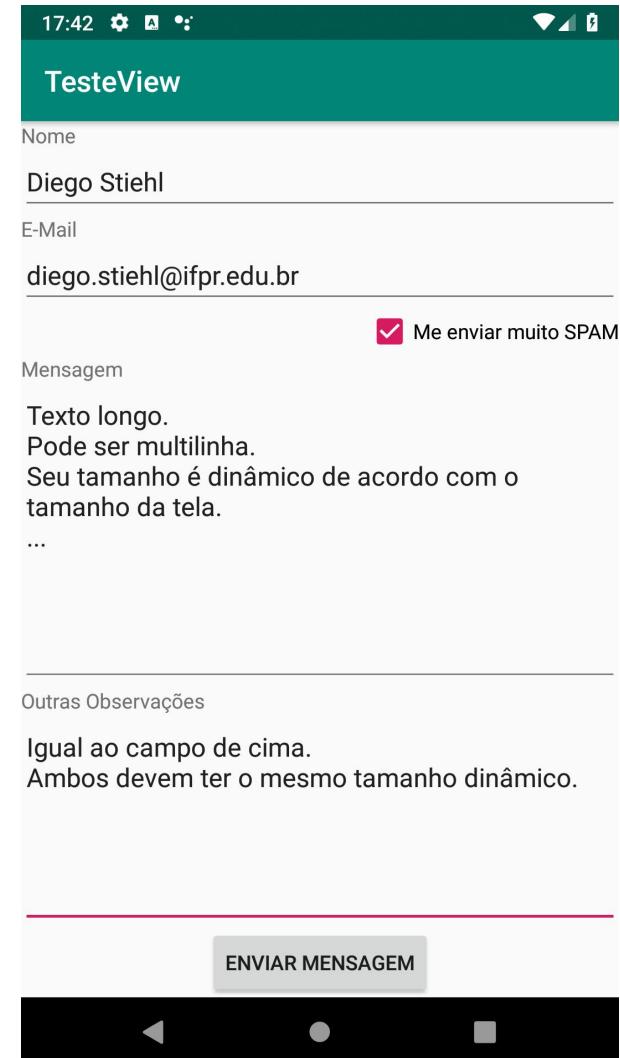
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="2" />

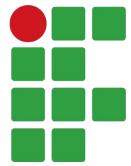
    <Button
        android:text="Button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
</LinearLayout>
```



# Prática

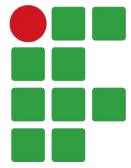
- Fazer este layout →
- Usar LinearLayout





# RelativeLayout

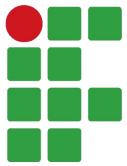
- Elementos-filho têm suas posições relativas aos demais elementos-filho ou ao elemento-pai (RelativeLayout)
  - View pode estar acima, abaixo, à direita ou à esquerda de outra
  - View pode estar alinhada à margem superior, inferior, direita ou esquerda do container
- Podemos definir espaçamentos (margens)
- Importante a definição do atributo `id`  
`android:id="@+id/idDaMinhaView"`



# RelativeLayout - Propriedades

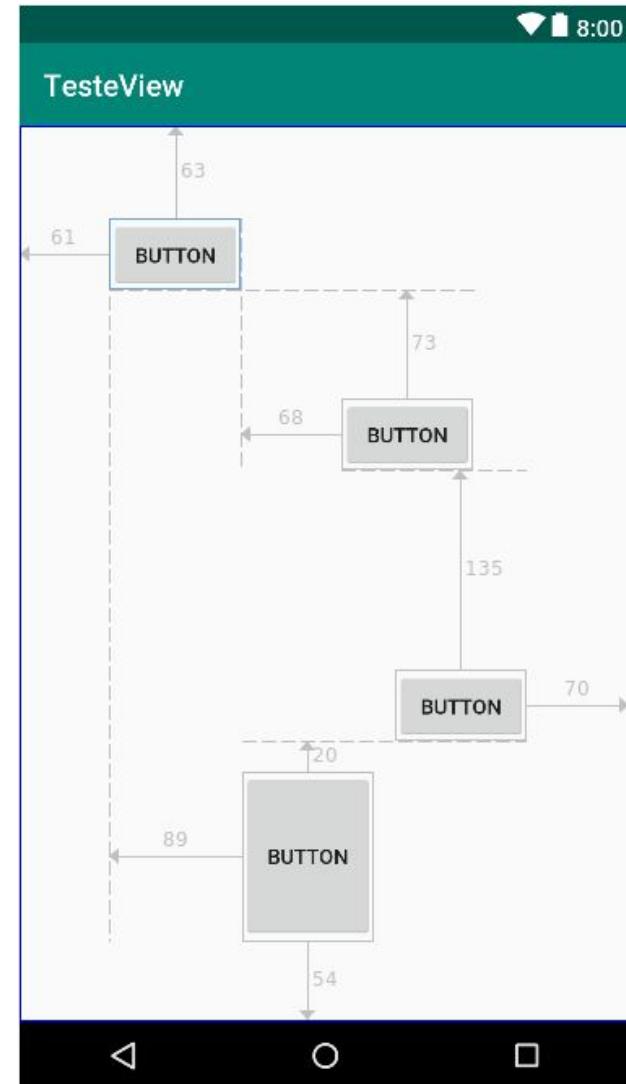
<Button

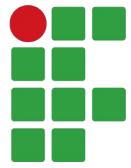
```
    android:id="@+id/umElemento"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@id/outroElemento"
    android:layout_below="@id/outroElemento"
    android:layout_toStartOf="@id/outroElemento"
    android:layout_toEndOf="@id/outroElemento"
    android:layout_alignTop="@id/outroElemento"
    android:layout_alignBottom="@id/outroElemento"
    android:layout_alignStart="@id/outroElemento"
    android:layout_alignEnd="@id/outroElemento"
    android:layout_alignBaseline="@id/outroElemento"
    android:layout_alignParentTop="true"
    android:layout_alignParentBottom="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentEnd="true"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:layout_centerInParent="true"
    android:layout_marginTop="50dp"
    android:layout_marginBottom="50dp"
    android:layout_marginStart="50dp"
    android:layout_marginEnd="50dp" />
```



# RelativeLayout

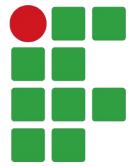
- Use o modo Design
- Algumas lapidações terão que ser feitas via XML





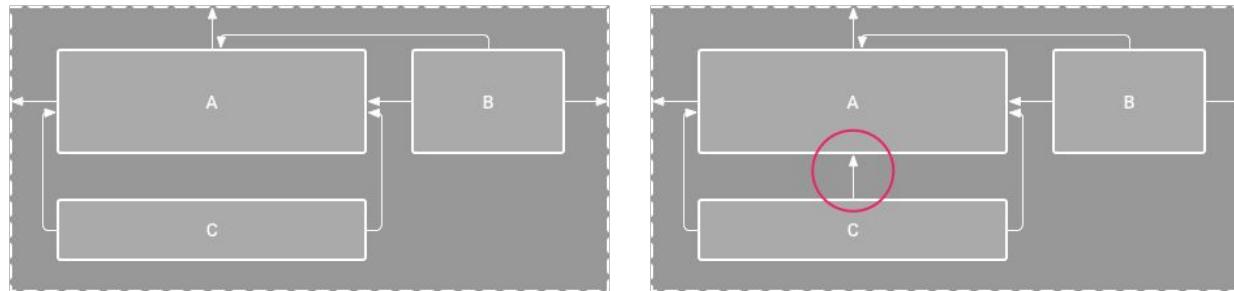
# ConstraintLayout

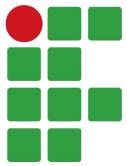
- Permite criar layouts grandes e completos mantendo uma estrutura “flat”
  - Sem aninhar containers
- Muito parecido com o RelativeLayout
  - Porém mais flexível
- Funciona muito melhor com o modo Design
  - Dificilmente precisaremos mexer no XML
- Trabalha com a definição de Constraints (**Restrições**) nos eixos X e Y



# ConstraintLayout

- Toda View precisa ter, ao menos, uma Constraint definida para os eixos X e Y
- Constraint é uma ligação com o mesmo eixo de outra View, do container ou linha-guia
- É criada uma “cerca” virtual, restrinindo a movimentação e tamanho da View

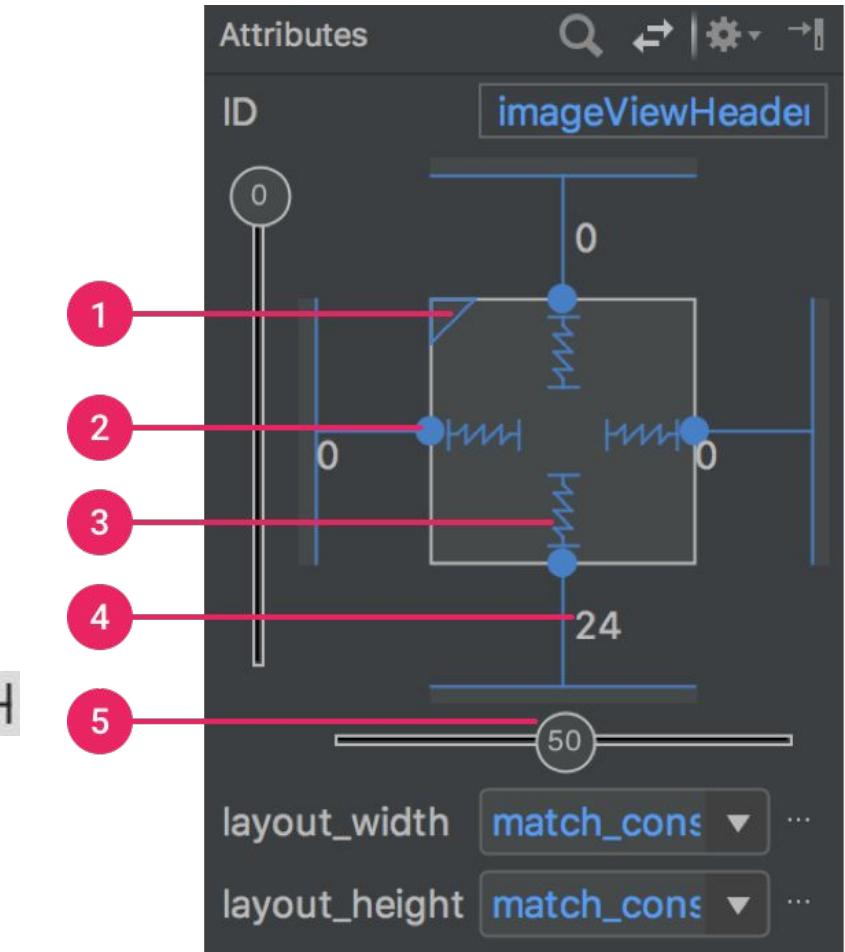


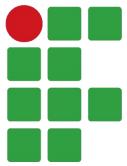


# Painel da View

## Selecione uma View

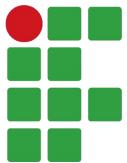
1. Proporção
2. Apagar Constraint
3. Modo de medição
  - a. Fixed
  - b. Wrap Content
  - c. Match Constraints
4. Margens
5. Tendência





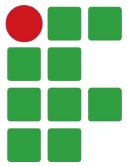
# ScrollView

- Algumas vezes uma View pode extrapolar alguma altura limite
  - Seja uma View simples ou um ViewGroup
- Podemos encapsular ela em uma ScrollView
- O Android irá criar uma barra de rolagem
- Observação
  - O ScrollView aceita apenas um elemento-filho



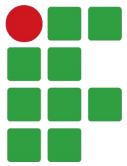
# Prática

- Reconstrua a tela de Login utilizando ConstraintLayout
  - Pensa nos diversos tamanhos de tela
  - Pensa na experiência do usuário
    - No celular com tela pequena
    - No tablet com tela grande
    - Na TV?



# Strings

- Não é boa prática inserir as strings da aplicação diretamente no layout ou nos arquivos de código-fonte
  - Difícil manutenção
  - Não propicia o reaproveitamento
  - Impossibilidade de internacionalização



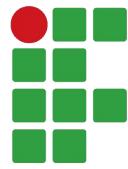
# Strings

- Todas Strings de interface gráfica devem ser definidas em um arquivo de recursos
  - res/values(strings.xml)

```
<resources>
    <string name="app_name">Olá Mundo</string>
    <string name="welcome">Bem vindo</string>
    <string name="name">Nome</string>
</resources>
```

- E reutilizadas nos arquivos de layout

```
<TextView
    android:text="@string/welcome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

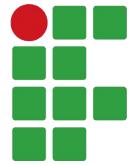


# Internacionalização (i18n)

- Usando o arquivo de strings, podemos facilmente traduzir nossa aplicação

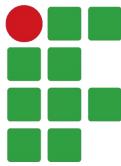
```
<resources>
    <string name="app_name">Olá Mundo</string>
    <string name="welcome">Bem vindo</string>
```

Key	Resource Folder	Untranslatable	Default Value	English (en)
app_name	app/src/main/res	<input checked="" type="checkbox"/>	Olá Mundo	Olá Mundo
name	app/src/main/res	<input type="checkbox"/>	Nome	Name
welcome	app/src/main/res	<input type="checkbox"/>	Bem vindo	Welcome



# Internacionalização

- Boa prática:
  - Criar todas as strings padrão (Default Value) em inglês
  - Criar as traduções para os idiomas desejados
- Resultado:
  - Se o usuário quiser uma tradução que não exista, por padrão mostra em inglês (idioma global)
    - Imagine um chinês vendo mensagens em português (ou o contrário)



# O que acontece?

## Diretórios

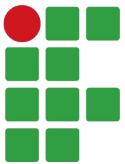
- ▶ values
- ▶ values-af-rNA
- ▶ values-en
- ▶ values-es
- ▶ values-fr
- ▶ values-pt-rBR

## No Android Studio

- ▼ res
  - ▶ drawable
  - ▶ layout
  - ▶ mipmap
  - ▼ values
    - colors.xml
    - ▼ strings (6)
      - strings.xml
      - strings.xml (af-rNA)
      - strings.xml (en)
      - strings.xml (es)
      - strings.xml (fr)
      - strings.xml (pt-rBR)
    - styles.xml

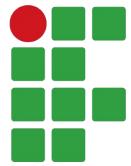
# Diferentes Configurações

- Criamos variações do **mesmo recurso** para diferentes configurações
  - Neste caso, a configuração de idioma e local
  - Vai considerar o idioma e local configurado no dispositivo Android do usuário final
- Podemos fazer variações para muitas configs
  - Tamanho, orientação e densidade da tela, versão do Android, com/sem touch, modo noturno, ...
- O Android sempre escolhe o mais adequado



# Prática

- Permita a internacionalização da tela de Login
  - Padrão: inglês
  - Crie mais 3 idiomas
  - Teste mudando as configurações do dispositivo



# Criando Variações

- Podemos criar novas variações de recursos
  - File → New → Android Resource File
- Escolher o **Resource type**
  - layout, values, drawable, ...
- Informar **File name**
  - Deve bater com um existente
- Note o **Directory name** gerado

Documentação completa:

<https://developer.android.com/guide/topics/resources/providing-resources>

## New Resource File



File name:	activity_main	↑↓
Resource type:	Layout	▼
Root element:	LinearLayout	
Source set:	main	▼
Directory name:	layout-land	

Available qualifiers:

- Country Code
- Network Code
- Locale
- Layout Direction
- Smallest Screen Width
- Screen Width
- Screen Height
- Size
- Ratio
- UI Mode
- Night Mode
- Density
- Touch Screen
- Keyboard
- Text Input
- Navigation State
- Navigation Method
- Dimension
- Version

Chosen qualifiers:

- Landscape

Screen orientation:

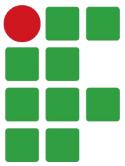
- Landscape

>> <<

OK

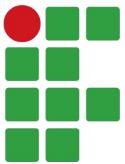
Cancel

Help



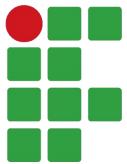
# Prática

- Faça uma variação do layout da prática do slide 21
- Característica:
  - Orientação em paisagem
- Construa um layout com os mesmos campos, mas utilizando ConstraintLayout e mudando o posicionamento dos campos
- Teste no dispositivo, com ambas orientações



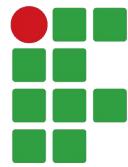
# Imagens

- No Android, as imagens são tratadas como **Desenháveis** (Drawables)
  - Podem ser de diversos tipos
    - Arquivos de imagem (jpg, png, gif)
    - Arquivos Nine-Patch (ajustáveis)
    - **Vetoriais (XML)** ← Mais indicado
    - Animações
    - Diversos outros ([ver documentação](#))
- Necessidade de criar variações
  - Ler: [Suporte a diferentes densidades | Android Developers](#)



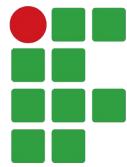
# Drawables

- Para termos desenháveis, bata criá-los na pasta **res/drawables/ (ou variações)**
- Para um arquivo de imagem simples, basta copiar e colar (ou arrastar) do computador
  - res/drawables/logotipo.png
- O Android Studio cria as referências
  - R.drawable.logotipo ← Uso em código
  - @drawable/logotipo ← Uso em XML de layout



# Imagens Vetoriais

- Na maioria dos casos, é sugerida a utilização de imagens vetoriais
  - Sem perda de qualidade
  - Suporte a todas resoluções e densidades
  - Menor tamanho final do pacote do aplicativo
- Se você tem uma imagem em SVG em mãos, faça o seguinte:
  - File → New → Vector Asset
    - Escolha Local File (SVG, PSD)



# Imagens Vetoriais

Asset Studio

Configure Vector Asset

Android Studio

Asset Type:  Clip Art  Local file (SVG, PSD)

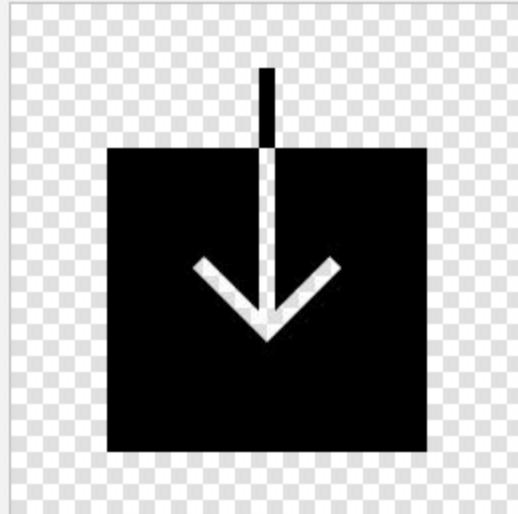
Name: ic\_download

Path: /home/diego/Downloads/download.svg

Size: 24 dp X 24 dp  Override

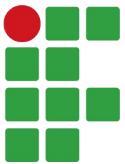
Opacity: 100 %

Enable auto mirroring for RTL layout



Vector Drawable Preview

Previous Next Cancel Finish Help



# Prática

- Coloque uma imagem de logotipo na variação em paisagem da tela da prática do slide 39
  - Importar a imagem com variações de densidade:
    - mdpi, hdpi, xhdpi e xxhdpi
- Coloque ícones vetoriais ao lado dos campos de texto