

# PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

---

Prof. Luiz Carlos Querino Filho

[luiz.querino@fatec.sp.gov.br](mailto:luiz.querino@fatec.sp.gov.br)

Fatec Garça – 2018

**pdm-02**



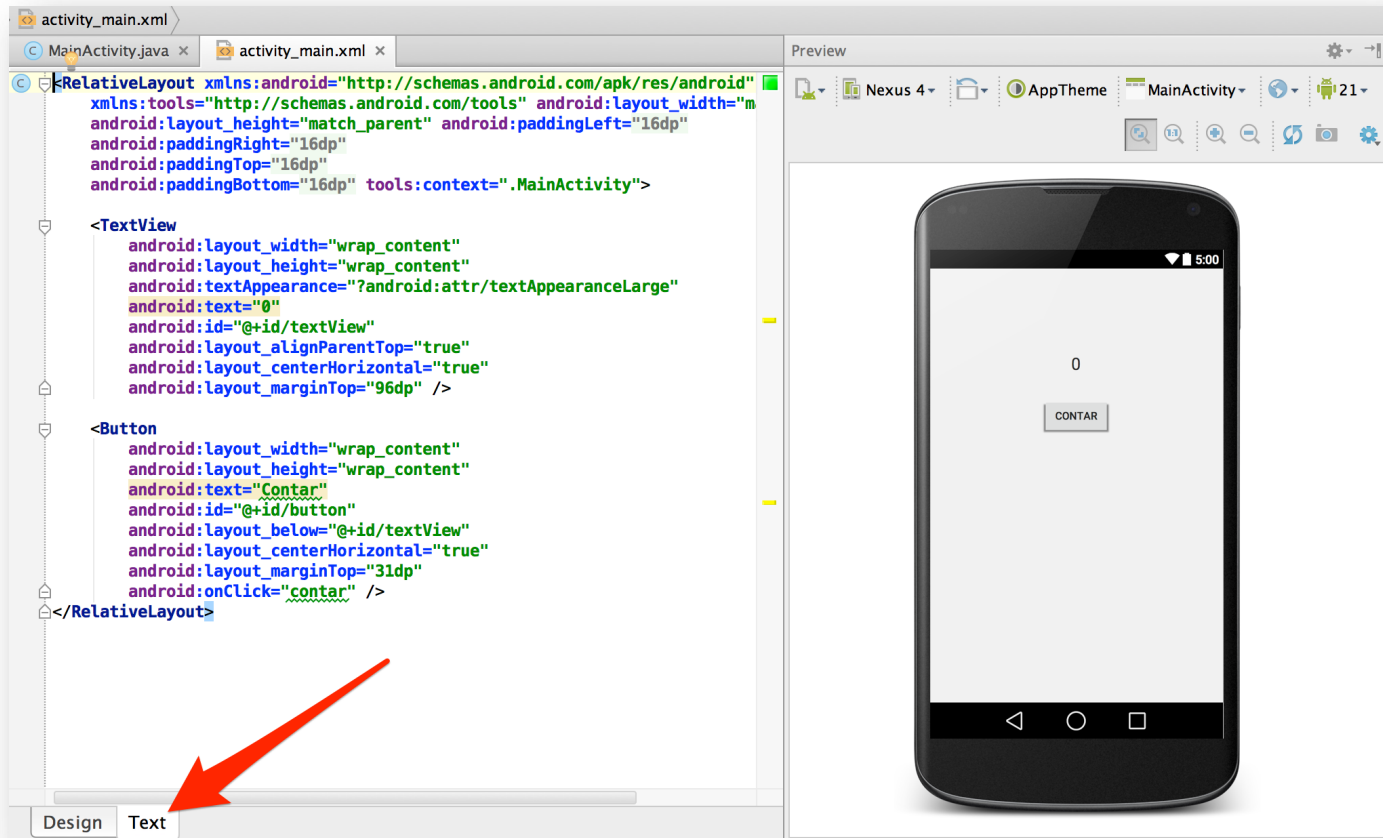
# COMPONENTES DE UM PROJETO ANDROID

---

# Arquivos de Layout

- Localizados no seu projeto em **res/layout**
- Arquivos XML que contém a definição dos elementos (**widgets**) existentes na tela e suas propriedades.
- O seu conteúdo pode ser editado diretamente em XML (**com cuidado**) ou graficamente.
- Opcionalmente, a interface também pode ser criada diretamente no código Java (o que **raramente** é recomendado)
- Os elementos ficam agrupados dentro de componentes gerenciadores de layout, como **ConstraintLayout**, **RelativeLayout** ou **LinearLayout**.

# Conteúdo do arquivo content\_main.xml



- Para alternar entre a edição gráfica e a edição XML, clique na **aba** correspondente (**Design | Text**).

# /res/values/strings.xml

- Neste arquivo estão as *strings* de texto utilizadas no projeto
- Cada *string* possui um identificador (seu nome) e um valor associado.
- Quando é necessário indicar o texto a ser exibido em um objeto de interface, vincula-se o **identificador** da string ao objeto.
- Dessa forma, o processo de **internacionalização** do aplicativo fica muito mais fácil.
- As *strings* podem ser criadas visualmente ou diretamente pelo arquivo XML.

# Classe R.java

- Classe **gerada automaticamente**
- As constantes existentes nessa classe são usadas dentro do código Java para possibilitar acesso a recursos diversos do projeto, como:
  - Ids de Widgets na tela
  - Imagens e figuras (drawables)
  - Strings
  - Cores
  - Layouts de tela completos
  - etc.
- **NUNCA** abra ou modifique esta classe manualmente.

# Activity

- A tela inicial da aplicação é representada por uma classe filha de **`android.app.AppCompatActivity`**
- O projeto deve possuir uma classe filha de **`AppCompatActivity`**, **`Activity`** ou **`ListActivity`** para cada tela.
- O método **`onCreate()`** deve ser implementado dentro da classe filha, sendo invocado pelo Android assim que a tela for criada.
- Para “desenhar” elementos na tela, são usadas classes descendentes de **`android.view.View`**.
- Widgets como botões e caixas de texto são todos “filhos” de **`View`**.

# Exemplo de uma subclasse de AppCompatActivity

```
package com.example.querino.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener((view) -> {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

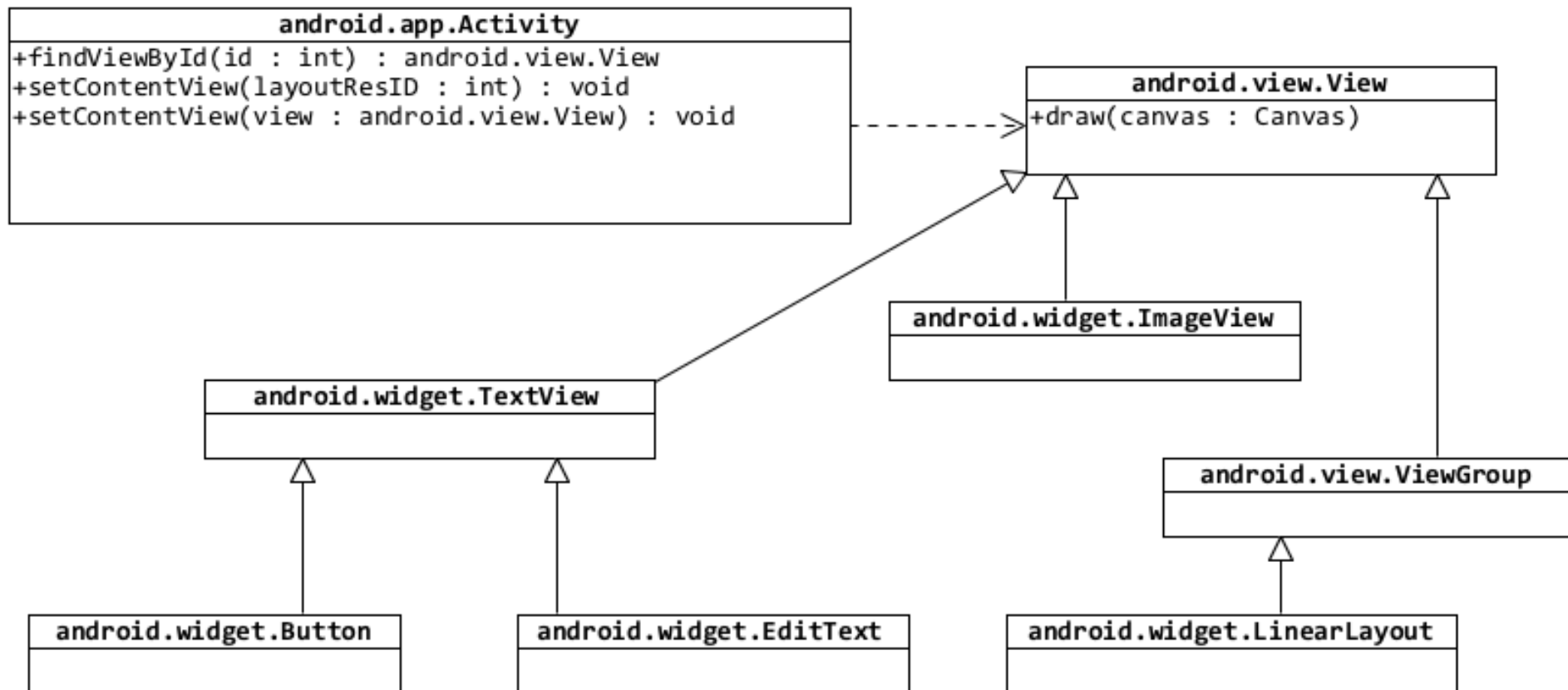
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```



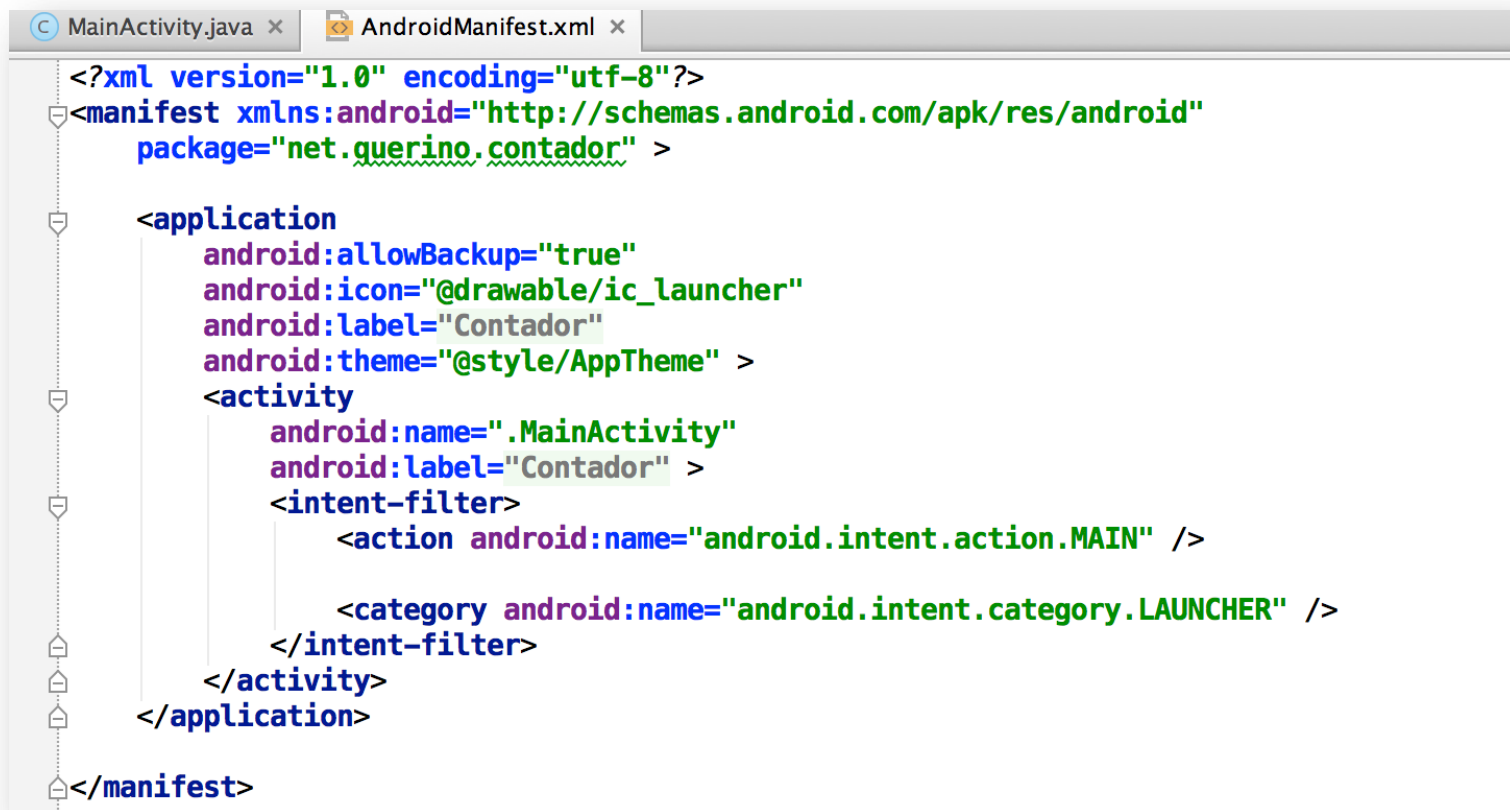
# Diagrama de Classes – Activity e View



# AndroidManifest.xml

- Localizado na pasta **manifests** do projeto
- Nele, são declaradas todas as **Activities** existentes no projeto
- Também deve possuir a indicação da existência de outros elementos, como **Services** e **Broadcast Receivers**.
- Serve também para indicação das **permissões** que a aplicação possui como, por exemplo, acesso à rede.
- Exemplos de configurações:
  - `<application android:icon="@>` indica uma imagem da pasta `drawables` que será usada como ícone do aplicativo.
  - `<application ... android:label="@>` é usada para indicar a string com o nome do aplicativo, exibido abaixo do ícone.
  - `<uses sdk android:minSdkVersion="X">` serve para indicar a versão mínima necessária para o aplicativo.

# AndroidManifest.xml



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.querino.contador" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="Contador"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="Contador" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Cuidado ao editar o arquivo! Erros de sintaxe no XML podem inviabilizar a compilação do seu app!

# Mais sobre a **Activity**

- **Lembrando:** cada tela da aplicação terá uma **Activity** para gerenciar seus eventos e seu estado.
- **Métodos da Activity:**
- **setContentView(View)**
  - Método pertencente à própria **Activity** (repare que ele é chamado diretamente, sem indicação do objeto/classe ao qual pertence).
  - Recebe como parâmetro a **View** principal da tela - aquela que servirá de base (como “fundação”) para as outras.
  - A referência à **View** pode ser feita usando a classe **R**, responsável por dar acesso aos recursos externos (no caso da **View** estar definida no arquivo `main.xml` – o que geralmente acontece)...
  - ...ou à uma **View** instanciada e configurada diretamente no código Java (o que é incomum e deve ser evitado, para não quebrar a divisão da aplicação em camadas)

# Mais sobre a Activity

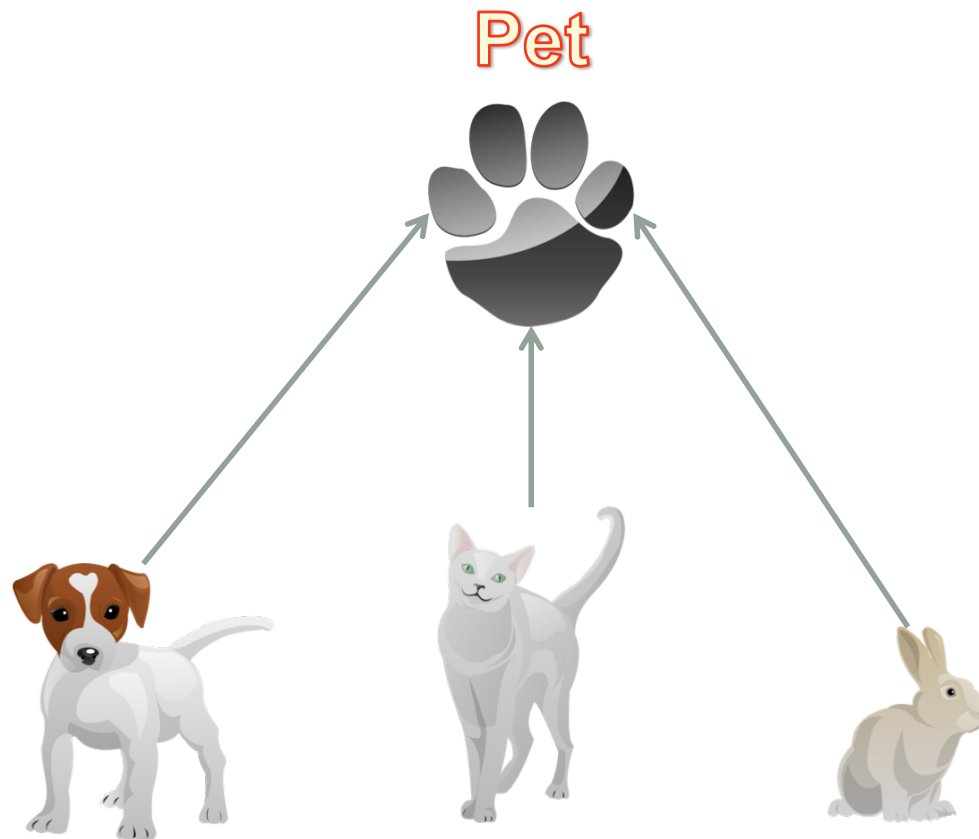
- **Métodos da Activity:**

- `View findViewById(id)`

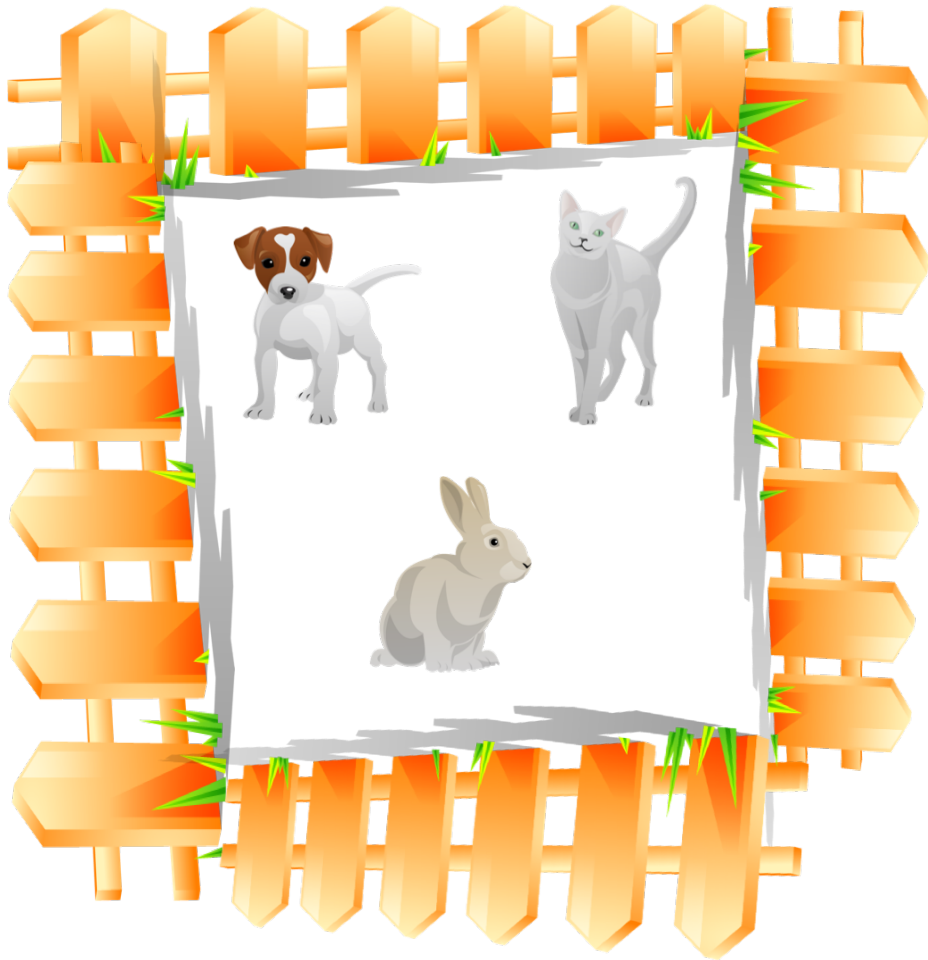
- Esse método é um dos melhores amigos do desenvolvedor Android!
- Ele “encontra” uma View dentro do layout gráfico e retorna uma referência à ela.
- Como parâmetro, deve ser passado o identificador (**ID**) da View, dentro de `R.id`. O identificador (**ID**) pode ser definido no editor de interface, clicando com o botão direito sobre o objeto e selecionando **“Edit ID...”**
- Serve para qualquer tipo descendente de **View** (Button, TextView, EditText, ...). Dessa forma, deve ser feito um “casting” no seu retorno:

```
EditText txtIdade = (EditText)findViewById(R.id.txtIdade);
```

# Entendendo o findViewById()



# Entendendo o findViewById()



# Fazendo uma experiência...

- Como experiência, vamos criar uma **View** (no caso, um botão) diretamente no código e defini-la como **View** principal. Para isso, modifique a **Activity** de acordo com o exemplo abaixo:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //setContentView(R.layout.activity_main);
    Button umBotao = new Button(this);
    umBotao.setText("Botão criado no código Java");
    setContentView(umBotao);
}
```



# Gerenciadores de Layout

- Dentro de um arquivo de layout, os widgets ficam agrupados em “containers” de widgets.
- Estes containers são os **Gerenciadores de Layout**
- São “espaços retangulares” onde colocamos os **widgets** obedecendo algumas regras de posicionamento.
- O gerenciador de layout padrão do Android é o **ConstraintLayout**.
- Com ele, os widgets são posicionados com restrições de posicionamento (por exemplo: centralizado, com uma margem de x pontos, etc.).
- O uso de um gerenciador como o **ConstraintLayout** possibilita uma melhor adaptação da tela do programa aos diferentes tamanhos de aparelhos.

# Outro tipo de Layout: **LinearLayout**

- Layouts são instâncias da classe `android.view.ViewGroups`
- O tipo de Layout mais básico é o **LinearLayout**. Com ele, os elementos são agrupados um após o outro, horizontalmente ou verticalmente.
- A definição da **orientação** (vertical ou horizontal) do **LinearLayout** é feita pela propriedade **orientation**.
  - Orientação vertical: widgets são “empilhados” (um abaixo do outro)
  - Orientação horizontal: widgets são colocados um ao lado do outro. Este é o padrão para o **LinearLayout**.
- Para criar layouts com várias linhas e colunas, mescle **LinearLayouts** com diferentes orientações.

# LinearLayout como layout raiz

- Para mudar o **layout raiz** de uma tela para **LinearLayout**, você precisará editar diretamente o arquivo em XML.
- Como o padrão do **LinearLayout** é a orientação horizontal, é preciso também incluir a configuração específica para vertical.



# android.widget.EditText

- Este *widget* é usada para entrada de texto.
- No construtor de interface, ele está localizado dentro da categoria ***Text Fields***.
- O Android traz dentro desta categoria variações do **EditText** apropriadas para entrada de diversos tipos de informação, como:
  - Texto Simples
  - Senhas
  - Datas
  - Números Inteiros com ou sem sinal
  - Números decimais
- **IMPORTANTE:** cabe ao usuário validar os dados!

# android.widget.EditText

- Métodos importantes:

**Editable** `getText()`;

- Retorna o texto existente no **EditText** como um tipo `android.text.Editable`. **Editable** é um tipo específico do Android – é basicamente uma **String** que pode ser modificada. Para um **Editable** se tornar uma **String**, use o método **toString()**:

```
String minhaString = editText1.getText().toString();
```

**void** `setText(CharSequence text)`

- Define o texto dentro do **EditText** como o **CharSequence** passado como parâmetro. **CharSequence** é uma *interface* que a classe **String** implementa. Então, uma **String** é um **CharSequence**. Dessa forma, você pode passar uma **String** diretamente como parâmetro:

```
String soma = String.valueOf(numero);  
editText.setText(soma);
```

# android.widget.Toast

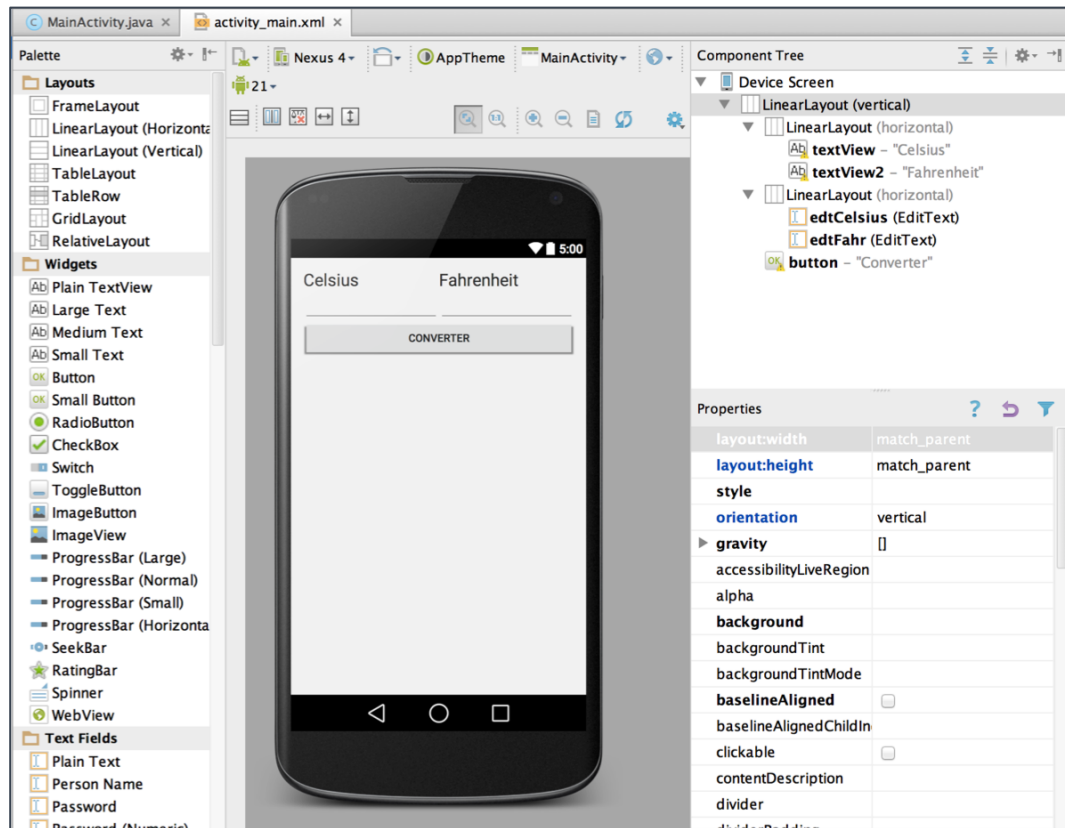
- A classe **Toast** é usada para exibir uma mensagem rápida para o usuário.
- A forma mais básica de se usar uma **Toast** é criar uma mensagem com o método estático `makeText()`, e exibi-la com `show()`.
- `Toast.makeText(Context contexto, CharSequence texto, int duracao)`
  - **contexto**: geralmente, a Activity atual (`this`)
  - **texto**: a String a ser exibida
  - **duracao**: uma constante determinando o tempo que a mensagem ficará na tela. Pode ser:
    - `Toast.LENGTH_SHORT` para exibição rápida
    - `Toast.LENGTH_LONG` para maior duração

- Exemplo de uso:

```
Toast.makeText(this, "Sumindo rapidinho...",  
              Toast.LENGTH_SHORT).show();
```

# Conversor de Temperatura

- Crie o layout com **LinearLayouts** “aninhados” (um vertical como raiz e, dentro deste, um horizontal para cada linha).
- Siga o “**Component Tree**” para não se perder!
- Para colocar um widget exatamente dentro do seu layout correto, arraste-o para a posição desejada dentro do **Component Tree**.



# Conversor de Temperatura

- Crie a classe **Temperatura**. Ela possuirá os seguintes métodos:
  - **double getFahrenheitFromCelsius(double celsius)**
    - Retorna um double com os graus Fahrenheit correspondentes aos graus Celsius passados como parâmetro.
  - **double getCelsiusFromFahrenheit(double fahrenheit)**
    - Retorna um double com os graus Celsius correspondentes aos graus Fahrenheit passados como parâmetro.
- Quando o usuário clicar no botão, **verificar se existe conteúdo no primeiro EditText** (edtCelsius). Se houver, mostrar os graus convertidos para Fahrenheit em etdFahrenheit.



# Conversor de Temperatura

- **Se não houver conteúdo**, verificar se existe valor no segundo **EditText** (`edtFahrenheit`). Se houver, mostrar os graus convertidos para Celsius em `edtCelsius`.
- **Se ambos estiverem vazios**, exibir uma mensagem com um **Toast** pedindo ao usuário informar um valor.
- Fórmulas:
  - $\text{Celsius} = ((\text{Fahrenheit} - 32) * 5 / 9)$
  - $\text{Fahrenheit} = ((\text{Celsius} * 9) / 5) + 32$

# Revisão de Java!

- Se você não se lembra, estas são algumas formas de se realizar conversões em Java:
  - `String String.valueOf(valor)`
    - Onde valor pode ser double, float, int, char, boolean.
  - `double Double.parseDouble(String valor)`
  - `int Integer.parseInt(String valor)`
- DICA EXTRA: É possível saber se uma String está vazia se seu método `equals("")` retornar `true`.

# Dicas de layout: propriedades úteis

- **android:orientation="horizontal|vertical"**
  - Define a orientação do layout linear como horizontal ou vertical.
- **android:layout\_width="match\_parent|wrap\_content"**
  - **match\_parent**: o widget vai ocupar a largura máxima disponível dentro do layout onde está inserido
  - **wrap\_content**: a largura do widget será definida pelo seu conteúdo
- **android:layout\_height="match\_parent|wrap\_content"**
  - **match\_parent**: o widget vai ocupar a altura máxima disponível dentro do layout onde está inserido
  - **wrap\_content**: a altura do widget será definida pelo seu conteúdo
- **android:layout\_weight="1"**
  - Define o “peso” do widget dentro do layout. Colocando o peso 1 em todos os widgets de um mesmo layout fará com que eles fiquem com o mesmo tamanho.

# BIBLIOGRAFIA

- QUERINO FILHO, L. C. Desenvolvendo seu Primeiro Aplicativo Android. Novatec Editora. 2013
- DEITEL, H. et al. Android for Programmers: An App-Driven Approach. Pearson Education. 2012.