

**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**  
**RIO GRANDE DO NORTE**  
Campus Natal - Central

# Mais Elementos da Interface com o Usuário

Prof. Fellipe Aleixo ([fellipe.Aleixo@ifrn.edu.br](mailto:fellipe.Aleixo@ifrn.edu.br))

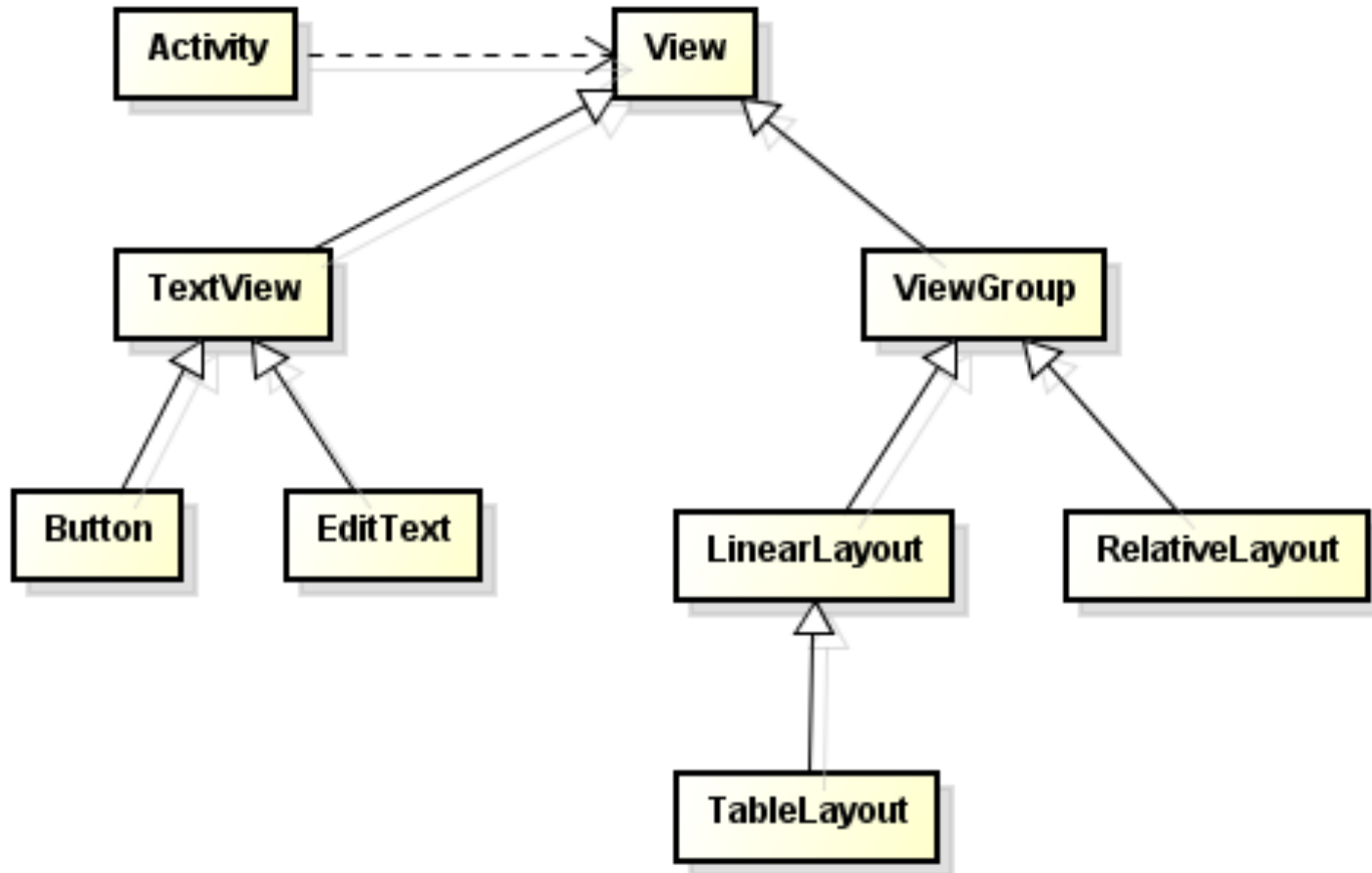
# Conteúdo

- Desenvolvimento de uma aplicação para registrar contatos e seus respectivos telefones
- Algumas Classes Importantes do Android
  - Activity, View, ViewGroup
- Layouts
  - LinearLayout, RelativeLayout
- Espaçamento e Padrões de Tamanho
  - Padding, Margin

# Classes Importantes do Android

- Classe `android.app.Activity`
  - Representa uma atividade (tela) da aplicação
  - Responsável pelo estado e eventos da tela
- Classe `android.view.View`
  - Representa os componentes visuais
  - Responsável pelo desenho dos componentes na tela
- Classe `android.view.ViewGroup`
  - Classe base para os *layouts*
  - Container para outras *views* (inclusive outros *viewGroups*)

# Hierarquia de Classes

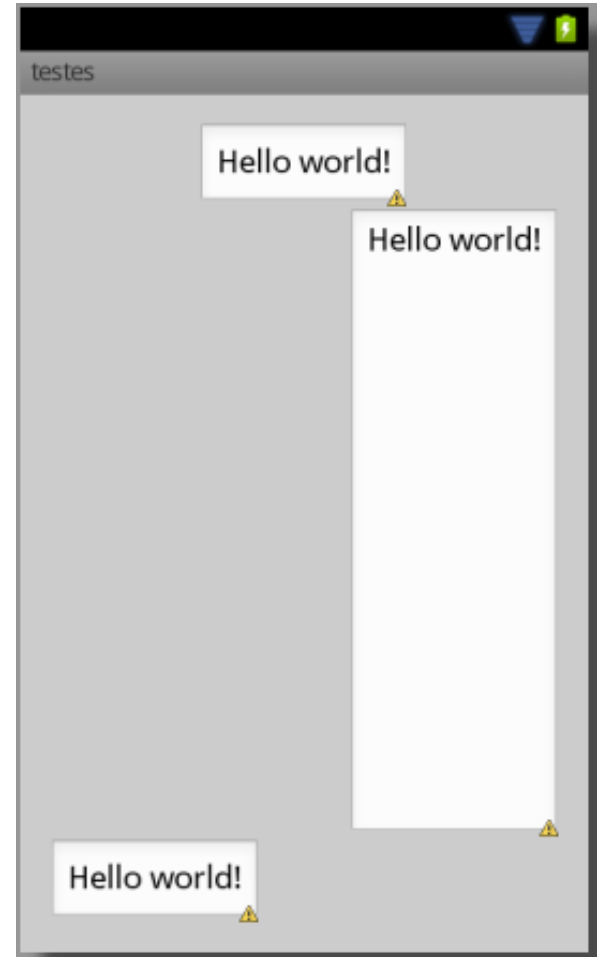


# LinearLayout

- **LinearLayout**
  - Arranja os elementos filhos em uma única linha ou coluna
- **Orientation**
  - Define a direção ou orientação dos elementos no layout: vertical ou horizontal. O padrão é horizontal.
- **Gravity**
  - Define o alinhamento dos elementos dentro do layout
- **Weight**
  - Define o preenchimento dos elementos no espaço restante disponível do layout

# LinearLayout – Exemplo

```
<LinearLayout ...
  android:orientation="vertical">
  <EditText ...
    android:text="@string/hello_world"
    android:layout_gravity="center" />
  <EditText ...
    android:text="@string/hello_world"
    android:layout_gravity="right"
    android:layout_weight="1"
    android:gravity="top" />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```



# RelativeLayout

- RelativeLayout
  - Arranja os elementos filhos utilizando posicionamento relativo
  - O posicionamento pode ser entre elementos filhos
  - Ou entre um elemento filho e seu pai (o próprio *layout*)
- Propriedades
  - Várias propriedades de posicionamento podem ser utilizadas
  - `Layout_alignParentTop`: alinha a borda superior do elemento com o layout
  - `Layout_centerVertical`: centraliza o elemento verticalmente no layout
  - `Layout_toRightOf`: alinha a borda esquerda de um elemento à direita de um segundo elemento informado

# RelativeLayout – Exemplo

```
<RelativeLayout ... >
  <EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:text="@string/hello_world" />
  <EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:text="@string/hello_world"
    android:layout_below="@id/editText1" />
  <EditText
    android:layout_width="match_parent"
    android:text="@string/hello_world"
    android:layout_below="@id/editText1"
    android:layout_toRightOf="@id/editText2" />
```





# Espaçamento e Tamanhos

- **Padding**
  - É o espaço para dentro da borda do elemento, ou seja, a distância entre o conteúdo interno do elemento e sua borda
- **Margin**
  - É o espaço para fora da borda do elemento, ou seja, a distância entre a borda do elemento e os demais elementos na vizinhança
- **Padrões de Tamanho**
  - **px** – Pixel – Evitar ser utilizado
  - **dp** – Pixel independente de densidade – Utilizar para dimensões de componentes e espaçamentos
  - **sp** – Pixel independente de escala – Utilizar para o tamanho de fontes

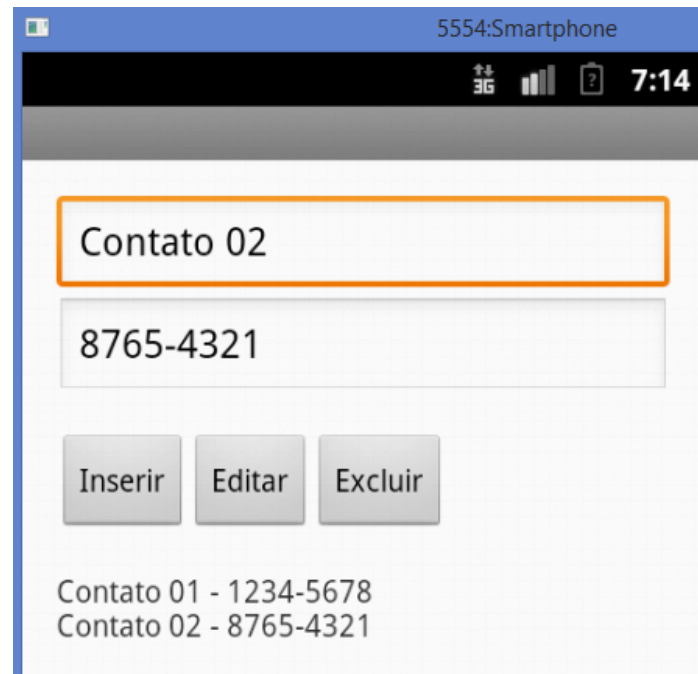
# Padding e Margin – Exemplo

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
<EditText ...
    android:text="@string/hello_world" />
<EditText ...
    android:text="@string/hello_world"
    android:paddingTop="20dp"
    android:paddingBottom="20dp" />
<EditText ...
    android:text="@string/hello_world"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="20dp" />
<EditText ...
    android:text="@string/hello_world" />
```



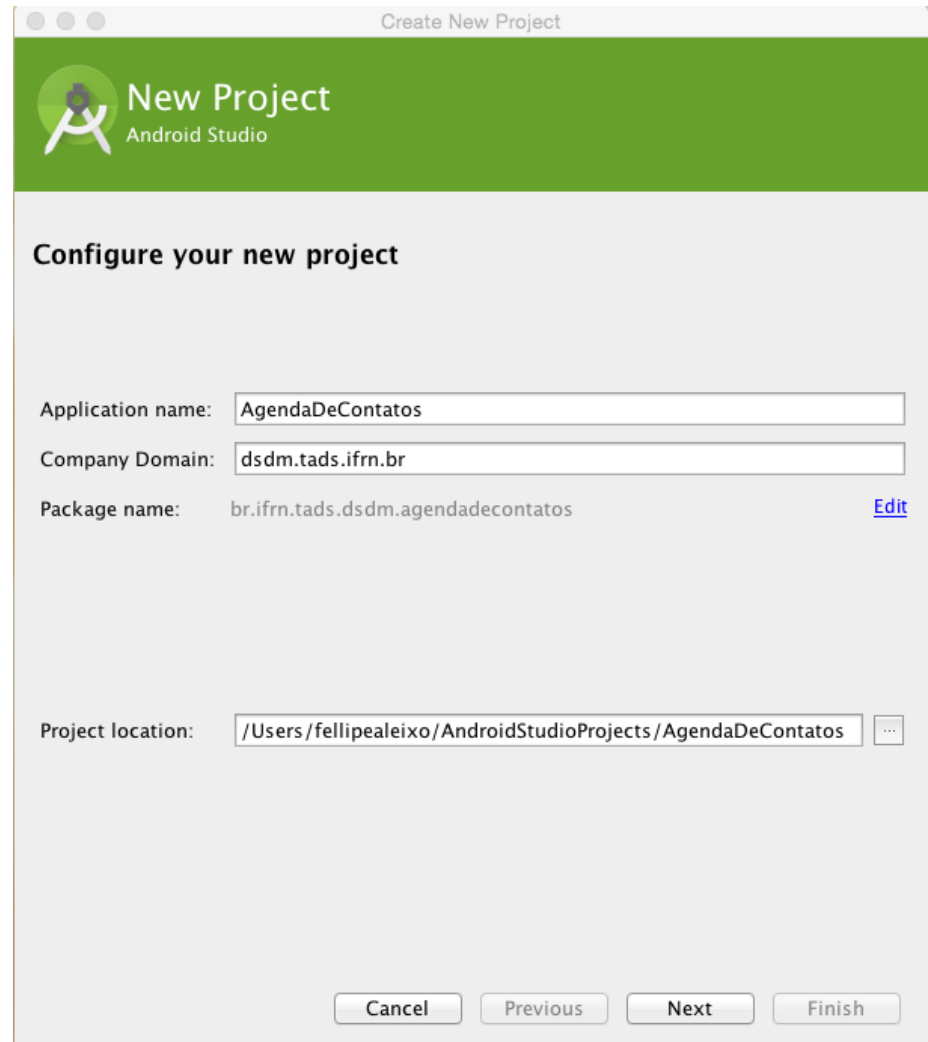
# Exemplo – Agenda de Contatos

- Agenda de Contatos
  - CRUD em uma lista de contatos (nome e telefone)
  - Inicialmente não trabalharemos com persistência



# Criação do Projeto

- Siga os passos do exemplo anterior e adicione o projeto conforme mostrado ao lado
- A interface padrão (**Blank Activity**) e demais arquivos do projeto são criados.



Create New Project

New Project  
Android Studio

Configure your new project

Application name:

Company Domain:

Package name:  [Edit](#)

Project location:  ...

Cancel Previous Next Finish

# strings.xml

- No arquivo strings.xml, defina as strings a serem utilizadas na aplicação:

```
<resources>  
  <string name="app_name">Agenda de Contatos</string>  
  
  <string name="nome">Digite o nome aqui</string>  
  <string name="fone">Digite o fone aqui</string>  
  <string name="inserir">Inserir</string>  
  <string name="editar">Editar</string>  
  <string name="excluir">Excluir</string>  
  <string name="contatos">Contatos</string>  
</resources>
```

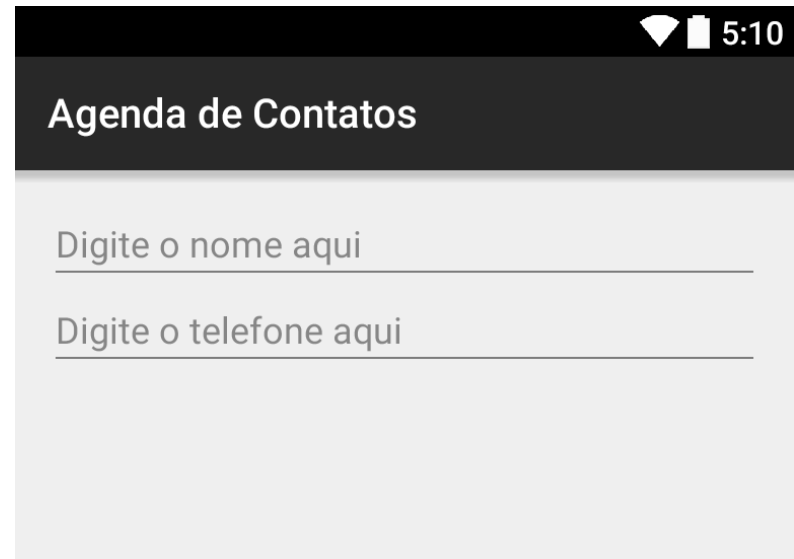
# Interface: LinearLayout

- No arquivo `activity_main.xml`, substitua o `RelativeLayout` por um `LinearLayout` e sete a propriedade *orientation* para `vertical`

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context=".MainActivity"
  android:orientation="vertical">
</LinearLayout>
```

# Interface: EditTexts

- Adicione dois `EditTexts` no *layout*, os componentes seguem a disposição na vertical, de acordo com a propriedade *orientation*
- Sete a propriedade `layout_width` para `match_parent`, para utilizar toda a largura da linha, e a propriedade *hint* para exibir a dica



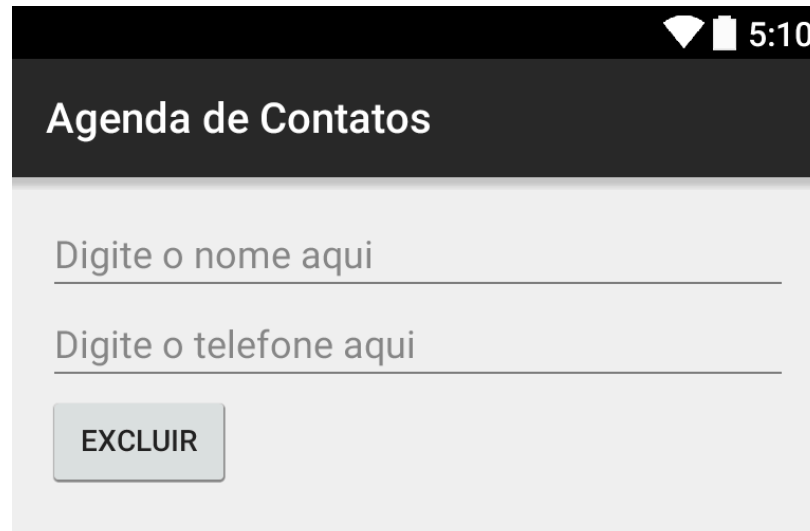
# Interface: EditTexts

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android" ... >
  <EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="text"
    android:hint="@string/nome"/>
  <EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="text"
    android:hint="@string/fone"/>
```



# Interface: RelativeLayout

- Abaixo do editText2, insira um **RelativeLayout**
  - Os botões com as operações da aplicação serão incluídos nele
- Insira três botões com os textos: Inserir, Editar e Excluir

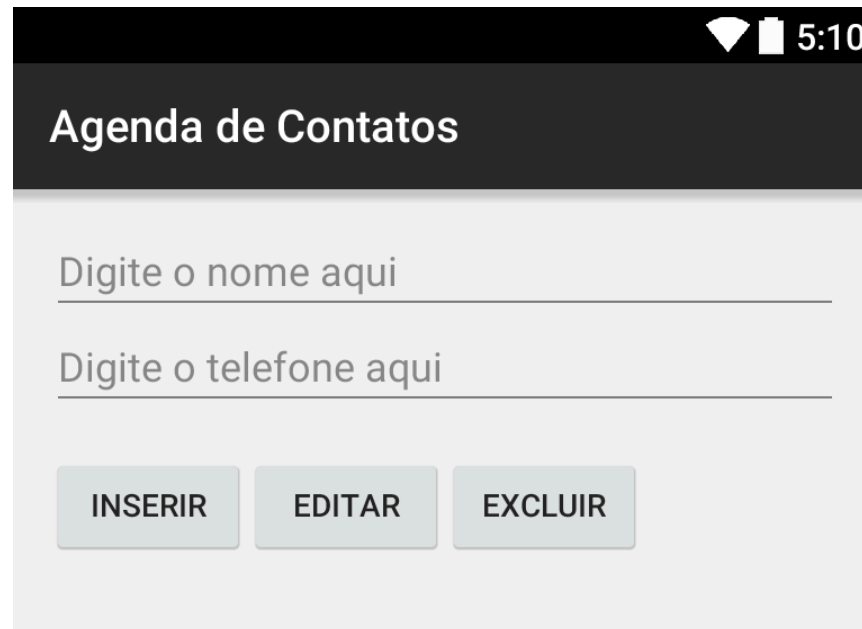


# Interface: RelativeLayout

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android" ... >
  <EditText
    android:id="@+id/editText2"
  ...
  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/inserir" />
  ...
</RelativeLayout>
```

# Interface: Buttons

- Ajuste o *padding* do *layout* e a localização dos botões inseridos nele, informações os posicionamentos relativos



# Interface: Buttons

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >
...
<Button
    android:id="@+id/button2"
    android:layout_toRightOf="@id/button1"
...
<Button
    android:id="@+id/button3"
    android:layout_toRightOf="@id/button2"
...
```

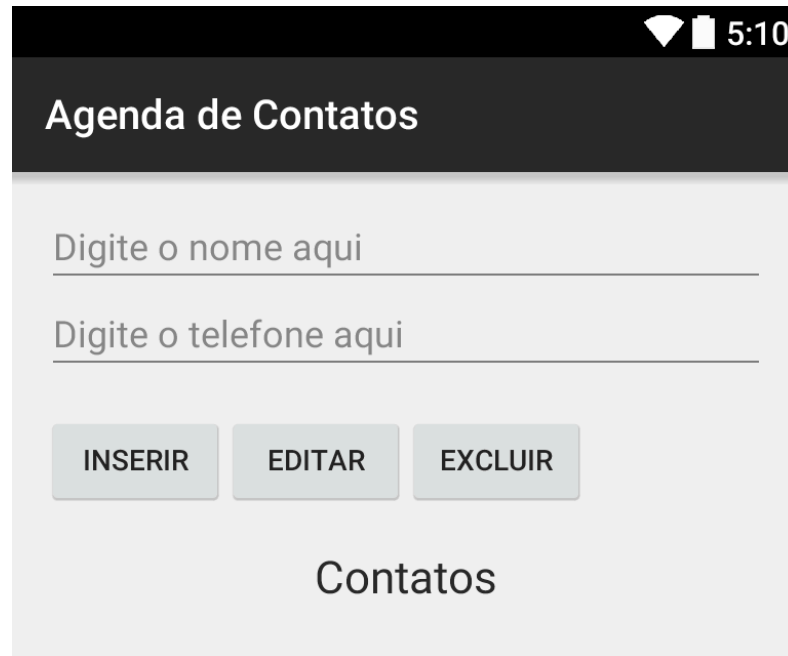
# Interface: Evento OnClick

- Em cada botão, sete o evento **OnClick** informando o método **buttonClick**
- Todos os botões ficam associados ao mesmo método que será definido na classe **MainActivity**

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/inserir"  
    android:onClick="buttonClick" />
```

# Interface: TextView

- Abaixo do [RelativeLayout](#), insira um [TextView](#) para listar os contatos e os respectivos telefones (como tal *layout* foi conseguido?)



# Interface: TextView

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceLarge"  
    android:text="@string/contatos"  
    android:id="@+id/textView"  
    android:layout_marginTop="20dp"  
    android:gravity="center"  
    android:layout_below="@id/button1"/>
```

# Classe Contato

- Em [br.ifrn.tads.dsdm.agendadecontatos](#), insira a classe **Contato**

```
public class Contato {  
    private String nome;  
    private String fone;  
    public Contato(String aNome, String aFone) {  
        nome = aNome;  
        fone = aFone;  
    }  
    public String getNome() { return nome; } ...  
    public void setFone(String aFone) { fone = aFone; } ...  
    @Override  
    public String toString() { return nome + " - " + fone; }  
}
```



# Classe Agenda

- Insira a classe `Agenda` que será utilizada para manter uma lista de objetos da classe `Contato`
  - Defina os métodos: `inserir`, `editar` e `excluir`
  - Sobrescreva o método: `toString`

# Classe Agenda

```
public class Agenda extends ArrayList<Contato> {
    private static final long serialVersionUID = 1L;

    public void inserir(String aNome, String aFone) {
        this.add(new Contato(aNome, aFone));
    }

    public boolean editar(String aNome, String aFone) {
        for (Contato c : this) {
            if(c.getNome().equals(aNome)) {
                c.setFone(aFone);
                return true;
            }
        }
        return false;
    }
}
```

# Classe Agenda

```
public boolean excluir(String aNome, String aFone) {
    for (Contato c : this)
        if(c.getNome().equals(aNome)) {
            this.remove(c);
            return true;
        }
    return false;
}

public String toString() {
    String result = "";
    for (Contato c : this)
        result += c.toString() + "\n";
    return result;
}
}
```

# Programação da Activity

- A programação da atividade é realizada na classe `MainActivity.java`
- Para referenciar os componentes, importar os pacotes que definem as suas classes
  - `import android.widget.Button;`
  - `import android.widget.EditText;`
  - `import android.widget.TextView;`

# Referenciando os Componentes

- Definir variáveis (atributos de MainActivity) para referenciar os componentes
  - `private EditText editText1;`
  - `private EditText editText2;`
  - `private Button button1;`
  - `private Button button2;`
  - `private Button button3;`
  - `private TextView textView1;`
- Definir um atributo para controlar a lista de contatos
  - `private Agenda agenda;`

# Referenciando os Componentes

- No método `onCreate`, instanciar o objeto agenda, que controla a lista de contatos, e recuperar as referências dos componentes

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    agenda = new Agenda();  
    editText1 = (EditText) findViewById(R.id.editText1);  
    editText2 = (EditText) findViewById(R.id.editText2);  
    button1 = (Button) findViewById(R.id.button1);  
    button2 = (Button) findViewById(R.id.button2);  
    button3 = (Button) findViewById(R.id.button3);  
    textView1 = (TextView) findViewById(R.id.textView1);  
}
```

# Manipulação de Eventos

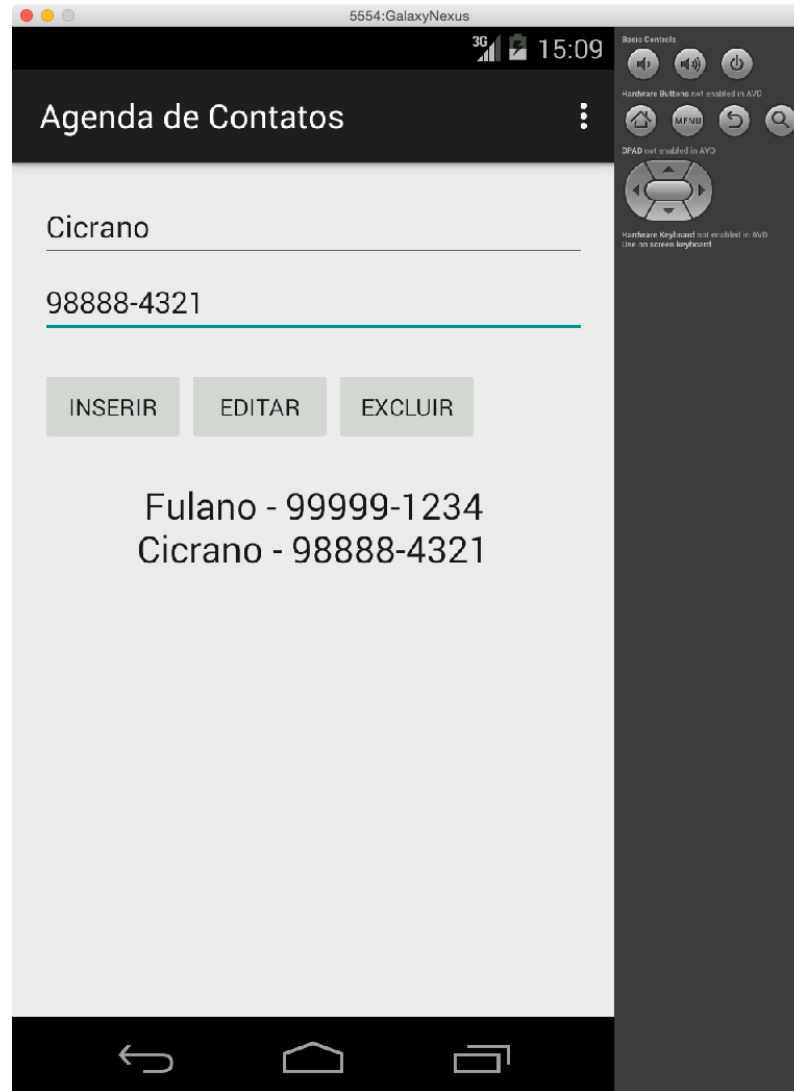
- Neste exemplo, os eventos de clique em um botão são capturados no método público `buttonClick` da Activity
- Esse método deve verificar qual botão foi clicado, e realizar a operação correspondente na lista de contatos

# Manipulação de Eventos

```
public void buttonClick(View v) {  
    if (v == button1) {  
        agenda.inserir(editText1.getText().toString(),  
                        editText2.getText().toString());  
        textView1.setText(agenda.toString());  
    }  
    if (v == button2) {  
        if (agenda.editar(editText1.getText().toString(),  
                          editText2.getText().toString()))  
            textView1.setText(agenda.toString());  
    }  
    if (v == button3) {  
        if (agenda.excluir(editText1.getText().toString(),  
                            editText2.getText().toString()))  
            textView1.setText(agenda.toString());  
    }  
}
```



# Execução da Aplicação



# Referências

- Android para Programadores – Uma abordagem baseada em aplicativos. Paul Deitel ... [et al.]. Bookman, 2013
- Google Android – Aprenda a criar aplicações para dispositivos móveis com o Android SDK. Ricardo R. Lecheta. Novatec, 2013