1 Antes de Começar

- Neste curso, desenvolveremos aplicativos utilizando as linguagens **XML** e **Java**.
- Usaremos a linguagem XML para criação da interface gráfica com o usuário.
- Usaremos a linguagem Java para implementar o comportamento dos aplicativos. Assume-se que você já possui conhecimentos da sintaxe da linguagem.

2 Activities

- Uma **Activity** é uma classe especial que fornece uma tela na qual é possível adicionar botões, caixas de textos, imagens e outros elementos. Observe a declaração de uma activity.

```
public class MainActivity extends Activity
```

- Em Java convencional, existe a função **main** que inicia a execução do programa.

```
public static void main(String[] args)
```

- Em programação para dispositivos móveis, a função **main** é substituída por métodos do ciclo de vida de uma Activity. Neste curso, utilizamos o método **onCreate**.

```
protected void onCreate(Bundle savedInstanceState)
```

- O método **onCreate** é o primeiro a ser invocado no ciclo de vida de uma activity.

3 Interface Gráfica com o Usuário

- Em um programa convencional, a interface gráfica é gerada principalmente usando **Swing** ou **AWT**, que são classes Java. Em um código para Android, é muito comum usarmos a linguagem XML para criar a Interface Gráfica como no exemplo abaixo.

```
1  <LinearLayout
2    xmlns:android="http://schemas..."
3    xmlns:app="http://schemas.android..."
4    xmlns:tools="http://schemas.android.com/tools"
5    android:orientation="vertical"
6    android:id="@+id/content_main"
7    android:layout_width="match_parent"
8    android:layout_height="match_parent"
9    android:paddingBottom="16dp"
10   android:paddingLeft="16dp"
11   android:paddingRight="16dp"
12   android:paddingTop="16dp">
13   <TextView
14    android:id="@+id/textView"
15     android:layout_width="wrap_content"
16     android:layout_height="wrap_content"
17     android:text="Hello World!" />
18 </LinearLayout>
```

- Note, a partir do exemplo, que o XML organiza as informações usando blocos delimitados por parênteses angulares, chamados de **tags**. Por exemplo, a tag **TextView** (Linha 13) indica que uma caixa de texto será adicionada na interface gráfica.
- Uma tag pode possuir outras encadeadas, como é o caso de **LinearLayout** (Linha 1), que possui **TextView** como uma tag interna. Nesse caso, o alcance da tag é delimitado por **<LinearLayout ... >** e **</LinearLayout>**. Observe o posicionamento da barra vertical **/**.
- Quando a tag não possui outras encadeadas, como é o caso de **TextView**,o posicionamento da barra vertical muda: **<TextView ... />**.
- Uma tag pode possuir atributos como, por exemplo, **layout_width** e **layout_height** (Linhas 7–8 e 15–16). Note que o valor atribuído deve estar entre aspas.
- Na programação para android, você não pode inventar qualquer nome para as tags e atributos, você terá que memorizar os principais identificadores.

3.1 Principais Identificadores

- **id**: atributo que permite acessar uma **view** a partir de um código em Java.
- **layout_width**: atributo que informa a largura de uma **view**. É possível fornecer o valor (i) **match_parent** para a **view** ter o mesmo tamanho daquela que a contém, (ii) **wrap_content** para a **view** ser grande o suficiente para suportar todo o seu conteúdo, ou (iii) fornecer numericamente um tamanho apropriado com a unidade **dp**.
- **layout_height**: atributo que informa a altura de uma **view**. É possível fornecer os mesmos valores explicados acima para **layout_width**.
- **paddingTop**: margem superior.
- **paddintBottom**: margem inferior.
- **paddingLeft**: margem esquerda.
- **paddingRight**: margem direita.

4 Acessando a Interface Gráfica no Código Java

- O código a seguir mostra o método **onCreate** de uma activity acessando uma TextView de id **textView** declarada conforme o código XML da seção anterior (Linhas 10–16).

```
1  package br.unicamp.ft.ulisses.myapplication;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.widget.TextView;
6
7  public class MainActivity extends AppCompatActivity {
8      private TextView textView;
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13
14         textView = (TextView)findViewById(R.id.textView);
15         textView.setText("Hello Android!");
16     }
17 }
```

- O método **onCreate** inicia com uma chamada para **super.onCreate** seguida de uma chamada para **setContentView** para definir qual arquivo XML deverá ser usado por esta activity.
- O arquivo XML está dentro da pasta **res/layout** e se chama **activity_main.xml**. Nesse caso, o argumento passado para **setContentView** é **R.layout.activity_main** (Linha 12).
- O método **findViewById** é o mais importante neste trecho de código (Linha 14). Ele instancia um objeto Java que representa um elemento da interface gráfica. No código em XML, o valor atribuído ao id da TextView é **@+id/textView**. Nesse caso, o argumento passado ao **findViewById** deve ser **R.id.textView**.
- Uma prática comum é obter no método **onCreate** todas as referências para os elementos da Interface Gráfica usados em algum lugar no código e colocá-las em atributos da classe. Observe que isso foi feito atribuindo a referência obtida com **findViewById** no atributo **textView** declarado na linha 8.

5 Coletânea de Views

- As **Views** são os elementos que você irá adicionar na Interface Gráfica. É interessante que você conheça um grupo delas para criar os seus primeiros aplicativos. Em geral, você precisa saber como adicionar elementos na interface gráfica (código XML) e como dar função para esses elementos (código Java).

5.1 Button

- **Código XML**: observe o atributo **onClick** (Linha 7). Ele informa o nome do método que será disparado no código da activity quando o botão for pressionado.

```
1  <LinearLayout ... >
2      <Button
3          android:layout_width="wrap_content"
4          android:layout_height="wrap_content"
5          android:background="@mipmap/ic_launcher"
6          android:text="@string/clickme"
7          android:onClick="onClick"
8          />
9  </LinearLayout>
```

- **Código Java**: observe a assinatura do método **onClick** na linha 7. O retorno e o tipo do parâmetro seguem o padrão mostrado no trecho de código. A chamada para **Toast.makeText** cria uma janela popup temporária com uma mensagem ao usuário.

```
1  public class MainActivity extends AppCompatActivity {
2
3      protected void onCreate(Bundle savedInstanceState) {
4          super.onCreate(savedInstanceState);
5          setContentView(R.layout.activity_main);
6      }
7      public void onClick(View view){
8          Toast toast = Toast.makeText(this, "Você
               ↪  pressionou o botão", Toast.LENGTH_SHORT);
9          toast.show();
10     }
11 }
```

5.2 EditText

- **Código XML**: uma EditText é uma caixa de texto que permite que o usuário escreva alguma coisa. Isso difere da TextView que apenas fornece informações ao usuário.

```
1  <LinearLayout ...>
2
3      <EditText
4          android:id="@+id/editText"
5          android:layout_width="match_parent"
6          android:layout_height="wrap_content"
7          android:hint="Dica para o usuário" />
8
9      <Button
10             ...
11         android:onClick="onClick"
12         />
13 </LinearLayout>
```

- **Código Java**: o método **getText** é invocado para receber o que foi escrito na caixa de texto.

```
public class MainActivity extends AppCompatActivity {

    private EditText editText;

    protected void onCreate(Bundle savedInstanceState) {
        ...
        editText = (EditText)findViewById(R.id.editText);
    }
    public void onClick(View view){
        String txt = editText.getText().toString();
        Toast toast = Toast.makeText(this, "Você
        ↪  escreveu: "+ txt, Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

5.3 CheckBox

Código XML: observe que não há nada que ligue uma **checkbox** a outra. Isso ocorre porque podem ser selecionadas ou desselecionadas de forma independente.

- **Checkbox** herda de **Button**, o que permite usar o atributo **onClick** para disparar um gatilho quando a mesma for pressionada (Linhas 7 e 13). Note que no exemplo disparamos o gatilho para o mesmo método **onCheckBoxClicked**, este deverá distinguir entre uma e outra **Checkbox**.

```
<LinearLayout ... >
    <CheckBox
        android:id="@+id/checkbox_1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/checkbox1"
        android:onClick="onCheckboxClicked" />
    <CheckBox
        android:id="@+id/checkbox_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/checkbox2"
        android:onClick="onCheckboxClicked" />
    <Button
        ...
        android:onClick="onClick"
        />
</LinearLayout>
```

- **Código Java**: temos um método que será invocado no momento da seleção (**onCheckBoxClicked**, Linhas 13–31) e um método que será invocado ao pressionar o botão (**onClick**, Linhas 34–47).
- **onCheckBoxClicked** recebe um objeto do tipo **View** por parâmetro, que é uma instância da **checkBox** que disparou o gatilho. Nesse caso, verificamos qual das duas **checkBoxes** disparou o gatilho e executamos a ação apropriada.
- O objeto recebido por parâmetro em **onClick** é uma instância de botão, o que não nos ajuda a identificar as **checkBoxes**. Assim, temos que recorrer aos atributos da classe para saber qual delas está selecionada e qual não está.

```
public class MainActivity extends AppCompatActivity {

  CheckBox checkBox1;
  CheckBox checkBox2;

  protected void onCreate(Bundle savedInstanceState) {
      ...
    checkBox1 = (CheckBox) findViewById(R.id.checkbox_1);
    checkBox2 = (CheckBox) findViewById(R.id.checkbox_2);
  }

  /** Chamado quando algum dos CheckBoxes é clicado */
  public void onCheckboxClicked(View view) {
    boolean checked = ((CheckBox) view).isChecked();
    switch(view.getId()) {
      case R.id.checkbox_1:
        if (checked) {
          // Primeira opção selecionada.
        }else {
          // Primeira opção cancelada.
        }
        break;
      case R.id.checkbox_2:
        if (checked) {
                  // Segunda opção selecionada
        }
        else {
          // Segunda opção cancelada.
        }
    }
  }

  /** Chamado quando o button é clicado */
  public void onClick(View view) {
    boolean checked1 = checkBox1.isChecked();
    boolean checked2 = checkBox2.isChecked();

    if (checked1 && checked2) {
      // Duas opções selecionadas.
    } else if (checked1) {
      // Apenas a primeira opção.
    } else if (checked2){
          // Apenas a segunda opção.
    } else {
          // Nenhuma opção selecionada.
    }
  }
}
```

5.4 RadioButton

- **Código XML**: observe no código que os **RadioButtons** estão agrupados em um mesmo **RadioGroup**. Isso ocorre porque eles não são independentes, dado que o comportamento padrão é que apenas um **RadioButton** de cada **RadioGroup** esteja selecionado.
- **RadioButtons** também herdam de botões, o que permite usar o atributo **onClick**.

```
<LinearLayout ... >
  <RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
      android:id="@+id/radio_1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/radiobox1"
      android:onClick="onRadioButtonClick" />
    <RadioButton
      android:id="@+id/radio_2"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/radiobox2"
      android:onClick="onRadioButtonClick" />
  </RadioGroup>

  <Button
      ...
    android:onClick="onClick"
    />
</LinearLayout>
```

- **Código Java**: no método **onRadioButtonClick**, disparado quando algum **RadioButton** é pressionado, estamos informando ao usuário o rótulo desse elemento pressionado.
- No método **onClick**, disparado quando o botão é pressionado, invocamos o método **getCheckedRadioButtonId** na instância do **RadioGroup**. Isso nos permite identificar qual dos **RadioButtons** está selecionado em um dado momento.

```
public class MainActivity extends AppCompatActivity {

  private RadioGroup radioGroup;

  protected void onCreate(Bundle savedInstanceState) {
      ...
    radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
  }

  /** Chamado quando o RadioButton é clicado */
  public void onRadioButtonClick(View view) {
    String text=((RadioButton)view).getText().toString();
    Toast.makeText(this, "Você selecionou a "+text,
                  Toast.LENGTH_SHORT).show();
  }

  /** Chamado quando o button é clicado */
  public void onClick(View view) {
    int id = radioGroup.getCheckedRadioButtonId();
    switch(id) {
      case R.id.radio_1:
        Toast.makeText(this, "Você selecionou a primeira
        ↪  opção", Toast.LENGTH_SHORT).show();
        break;
      case R.id.radio_2:
        Toast.makeText(this, "Você selecionou a segunda
        ↪  opção", Toast.LENGTH_SHORT).show();
        break;
      }
    }
}
```

6 Gerenciadores de Layout

Temos usado em nossos exemplos o gerenciador de layout **LinearLayout** sem explicar o que ele faz e sem informar quais outros gerenciadores de layout são comuns. Abaixo, fornecemos uma lista não exaustiva dos gerenciadores.

- **RelativeLayout**: os elementos da interface gráfica são posicionados tendo os outros elementos como referência. Ao adicionar um elemento, devemos posicioná-lo em relação à janela (usando atributos como **layout_alignParentTop**, **layout_alignParentRight**, **layout_centerInParent**) ou em relação aos outros elementos (usando atributos como **layout_alignRight**, **layout_below**, **layout_toRightOf**.
- **LinearLayout**: nesse gerenciador de layout, os elementos são posicionados na ordem em que aparecem no arquivo XML, sendo que podem ser colocados horizontalmente ou verticalmente de acordo com o atributo **android:orientation**, obrigatório quando estamos usando um LinearLayout.
- **GridLayout**: nesse gerenciador, os elementos da interface gráfica são posicionados em uma tabela contendo linhas e colunas. Um atributo obrigatório é **ColumnCount**, que informa o número de colunas.