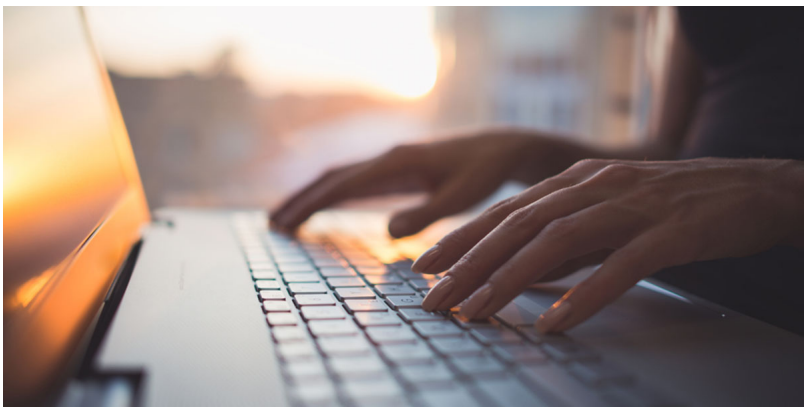
Programação para dispositivos móveis

Aula 3 - Interface Gráfica do Usuário

INTRODUÇÃO



A criação de uma interface gráfica mobile é uma das grandes dificuldades no desenvolvimento voltado para telas de pequeno tamanho. Para tanto, o Android oferece um conjunto de objetos que facilitam este tipo de desenvolvimento.

Esta aula visa apresentar a classe View, View Group, bem como os principais Widgets e gerenciadores de layout Android.

OBJETIVOS

Descrever a classe View, View Group e seus tipos;

Explicar os principais gerenciadores de layouts;

Explicar a implementação de componentes gráficos e gerenciadores de layouts em aplicativo Android.

COMPONENTES GRÁFICOS

Fonte da Imagem:

A interface com o usuário é um ponto fundamental para o desenvolvimento de aplicativos Android, pois é através desta que o usuário interage.

A construção desse canal de comunicação demanda bastante tempo para desenvolvimento, principalmente no cenário mobile. Nele existe uma variedade muito grande de tipos dispositivos móveis, que, muitas vezes, possuem uma capacidade de tela superior até mesmo às televisões de última geração.

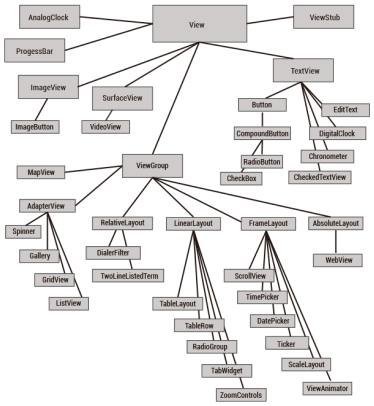
Para tanto, o Android oferece suporte nativo para o desenvolvimento de interfaces gráficas sofisticadas.

Podemos destacar a View, que é a classe base de todo e qualquer componente gráfico. Responsável por desenhar e controlar os eventos.

TIPOS DE COMPONENTES

O Android fornece um conjunto muito grande de componentes, mas podemos agrupá-los em dois tipos:

A imagem abaixo ilustra a hierarquia da classe View, demonstrando que o Android oferece um sofisticado e poderoso modelo baseado em Widgets e Layouts.



TIPOS DE COMPONENTES GRÁFICOS

Existem vários tipos de componentes. Entre eles, podemos destacar:

TEXTVIEW

Considerado um dos componentes mais usados em Android. Define um texto que será exibido na tela.

Por default, não pode ser editado, embora possamos alterar essa configuração.

```
< TextView
    android:id="@+id/resultado"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:scrollHorizontally="true"
    android:textSize="40dp"/>
```

BUTTON

Nada mais é do que um simples botão.

Após clicarmos nele, podemos executar uma ação.

```
< Button
    android:id="@+id/button1"
    android:layout_width="80dp"
    android:layout_height="70dp"
    android:text="7"
    android:textSize="40dp" />
```

IMAGEVIEW

Responsável por inserir imagens.

```
< ImageView
    android:id="@+id/imageView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:src="@drawable/ic_launcher" />
```

EDITTEXT

Uma caixa onde é digitada alguma informação, que poderá ser usada para interagir com a aplicação Android.

```
< EditText
    android:id="@+id/editText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:ems="10" >
```

CHECKBOX

Um botão do tipo “caixa de seleção”, onde existe dois estados: verdadeiro ou falso.

```
< CheckBox
    android:id="@+id/cbbTeste"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Exemplo />
```

RADIOBUTTON

Similar ao CheckBox, porém não deve permitir a seleção de mais de uma opção.

RADIOGROUP

Tem por objetivo agrupar estes radiobuttons, permitindo a seleção de apenas um RadioButton dentro deste Radiogroup.

```
< RadioGroup
    android:id="@+id/radioSex"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

< RadioButton
    android:id="@+id/radioMale"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/radio_male"
    android:checked="true" />

< RadioButton
    android:id="@+id/radioFemale"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/radio_female" />
```

IMPLEMENTAÇÃO DE UIS

Podemos implementar as Uis do usuário de forma declarativa e programática. Na declarativa, fazemos uso do XML para declarar a aparência da interface.

Já na programática, implementamos através da linguagem Java.
Podemos fazer uma analogia de declarativa com HTML e programática com a Swing e AWT do Java.

Devemos dar preferência sempre a forma declarativa, pois tende a ser mais fácil de entender, desenvolver e até mesmo manutenir.

Já imaginou o trabalho que seria para organizar os componentes em cada tipo de tela?

Nesse ponto, o Android nos ajuda muito através de um conjunto de classes, pré-definidas, denominadas Gerenciadores de Layout, que tem por finalidade organizar os componentes de nossas telas.

Observe a figura abaixo:

Como apresentado anteriormente, a classe View é a base para todo e qualquer componente visual no Android. Já a classe ViewGroup tem como responsabilidade organizar os componentes de uma tela.

São conhecidos como Gerenciadores de Layouts. Funcionam como container para outros tipos de Views, podendo ser Widgets ou até mesmo outros ViewGroups.

Principais layouts

• **AbsoluteLayout (android.widget.AbsoluteLayout):**
Os componentes são posicionados na tela em função das coordenadas X e Y fornecidas.

• **FrameLayout (android.widget.FrameLayout):**
O componente ocupa a tela inteira. Funciona como uma pilha sendo que uma view fica por cima da outra.

• **LinearLayout (android.widget.LinearLayout):**
Os componentes são organizados na vertical ou horizontal.

• **TableLayout (android.widget.TableLayout):**
Assim como em uma tabela, os componente são organizados em colunas e linhas.

• **RelativeLayout (android.widget.RelativeLayout):**
A organização do componente é implementada relativa a outro componente.

AbsoluteLayout(deprecated)

Nesse tipo de gerenciador de layout, todos os componentes são posicionados em função de abcissa e ordenada, tendo como base o canto superior esquerdo.

É bastante oportuno lembrar que esse é menos flexível e mais difícil de manter que os demais gerenciadores. É importante também saber que esse gerenciador está obsoleto (Deprecated).

Exemplo:

FrameLayout

Serve para reservar um espaço na tela que será utilizado por um ou mais componentes.

Quando mais de um componente for inserido dentro de um único FrameLayout, haverá uma sobreposição dos mesmos. A menos, é claro, que você divida a tela e coloque cada um na sua posição.

Exemplo:

Como no exemplo anterior, criaremos o nosso projeto com o nome ***Aula5FrameLayout***.

Assim como em todos os nossos exemplos nesta aula, usaremos a API 23, conforme podemos verificar na tela:

Para fins pedagógicos, selecionamos uma Empty Activity.

Precisamos dominar esse conceito melhor, não é mesmo? Por isso, depois trabalharemos com os demais tipos.

Para facilitar, manteremos o nome sugerido pelo Android Studio para atividade e layout.

Veja a tela abaixo:

Este é o coração de nosso layout. No caso, FrameLayout.

Altere o conteúdo do arquivo, conforme tela abaixo:

Alguns parâmetros precisam ser discutidos para entendermos melhor esse tipo de layout. Criamos, então, os componentes textView, textView1, textView2, textView3, textView4.

Já verificamos, no exemplo referente à AbsoluteLayout, os parâmetros android: layout_height e android:layout_width.

Contudo, existem novos. Veja:

• **android:layout_marginTop:**
Propriedade que define o tamanho da margem superior.

Embora não tenhamos implementado em nosso exemplo, existem suas correlatas, como:
• **android:layout_marginBottom:**
Margem inferior;

• **android:layout_marginLeft:**
Margem esquerda;

• **android:layout_marginRight:**
Margem direita;

• **android:layout_margin:**
As quatro margens.

Exemplo:

android:layout_margin="10dp"

• **android:layout_gravity:**
Define a posição dos componentes em relação ao gerenciador de layout que os contém.

• **android:gravity:**
Define o posicionamento do conteúdo do componente;

• **android:padding:**
Define a distância entre o conteúdo interno do componente e sua borda. Nesse caso, todos os 4 espaçamentos serão iguais.

Poderíamos também escolher o espaçamento que desejássemos usando:

• **android:paddingTop:**
Espaçamento superior;

• **android:paddingBottom:**
Espaçamento inferior;

• **android:paddingRight:**
Espaçamento lado direito;

• **android:paddingLeft:**
Espaçamento lado esquerdo;

• **android:background:**
Define uma cor (em hexadecimal) ou imagem referente ao background de nosso layout;

• **android:textColor:**
Define a cor de nossa fonte;

• **android:textSize:**
Define o tamanho de nossa fonte;

• **android:textStyle:**
Define o estilo do texto (bold, italic, normal).

LinearLayout

Esse tipo de gerenciador de layout tem por característica a capacidade de alinhar verticalmente ou horizontalmente os componentes. Isso depende, obviamente, da orientação escolhida.

Veja os exemplos abaixo:

LinearLayout Horizontal:

Como já criamos vários projetos passo a passo anteriormente, vamos direto ao ponto:

Na tela abaixo, estão as alterações que efetuamos no arquivo **activity_main.xml** para demonstrar o **LinearLayout Horizontal**.

Ao executar, você poderá constatar, conforme exibido na tela abaixo, que os widgets estão alinhados na horizontal.

Até o momento, não usamos nenhum atributo desconhecido, exceto, é claro, o LinearLayout.

LinearLayout Vertical:

Agora vamos transformar o exemplo anterior em vertical.

Para tanto, só precisamos alterar o atributo **android:orientation="vertical"**, conforme demonstrado na tela.

Repare que esse atributo não foi modificado no exemplo anterior, pois o default é horizontal.

RelativeLayout

O RelativeLayout organiza os componentes do layout de uma tela em relação uns aos outros.

Isso é uma grande vantagem deste gerenciador de layout, no que tange à flexibilidade de posicionamento dos componentes.

Na teoria, é muito fácil de entender, mas como seria em um código fonte?

Vamos ver o exemplo abaixo:

Altere o arquivo **activity_main.xml**, conforme a tela:

Principais atributos:

• **android:layout_below:**
Posiciona abaixo do componente indicado;

• **android:layout_above:**
Posiciona acima do componente indicado;

• **android:layout_toRightOf:**
Posiciona à direita do componente indicado;

• **android:layout_toLeftOf:**
Posiciona à esquerda do componente indicado;

• **android:layout_alignParentTop:**
Alinha no topo do layout-pai;

• **android:layout_alignParentBottom:**
Alinha abaixo do layout-pai;

• **android:layout_alignParentRight:**
Alinha à direita do layout-pai;

• **android:layout_alignParentLeft:**
Alinha à esquerda do layout-pai;

• **android:layout_alignTop:**
Alinha no topo do componente indicado;

• **android:layout_alignBottom:**
Alinha abaixo do componente indicado;

• **android:layout_alignRight:**
Alinha à direita do componente indicado;

• **android:layout_alignLeft:**
Alinha à esquerda do componente indicado;

• **android:layout_marginTop:**
Define a margem superior do componente;

• **android:layout_marginBottom:**
Define a margem inferior do componente;

• **android:layout_marginRight:**
Define a margem direita do componente;

• **android:layout_marginLeft:**
Define a margem esquerda do componente;

• **android:layout_centerHorizontal:**
Centraliza o componente horizontalmente;

• **android:layout_centerVertical:**
Centraliza o componente verticalmente.

TableLayout

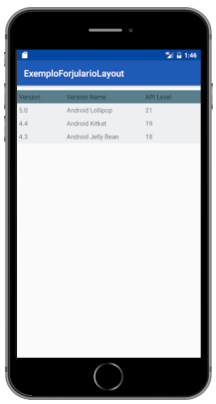
Esse gerenciador de layout organiza seus componentes em colunas e linhas, ou seja, como uma tabela.

Cada **< tableRow >< /tableRow >** corresponde a uma linha de nossa tabela. As colunas são criadas à medida que incluímos os componentes em cada linha.

Veja o exemplo:
Altere o arquivo **activity_main.xml**, conforme tela:

ATIVIDADE

Observe a tela abaixo:



Usando XML, implemente o layout demonstrado na tela acima.

Resposta Correta

Glossário