Dispositivos Móveis

Aula 03 - Interfaces Gráficas I

Apresentação

Olá! Prontos para começarmos a conhecer de fato os componentes que fazem parte da programação Android?! Esta aula tem como objetivo discutir o funcionamento das interfaces gráficas no Android. Inicialmente, iremos explicar as duas maneiras de se criar interfaces gráficas, a maneira declarativa e a programática. Em seguida, discutiremos os principais componentes gráficos, como são estruturados e veremos alguns exemplos do funcionamento desses componentes. O próximo passo será explicar como utilizá-los, desde sua criação até o seu carregamento na aplicação Android, além de explicar os principais atributos que devem ser configurados nesses componentes. Por fim, veremos os principais layouts que são utilizados em aplicações Android, com exemplos de suas implementações.

Como o desenvolvimento de interfaces gráficas para o Android é uma parte importante e com bastante conteúdo, preferimos dividir o estudo desses componentes em três aulas, intercalando-as com aulas sobre Activities, para garantir que será possível ir desenvolvendo aplicações básicas, à medida que o conteúdo for sendo desenvolvido.

Objetivos

Ao final desta aula, você será capaz de:

- Entender o funcionamento das interfaces gráficas no Android e como elas são programadas.
- Reconhecer os componentes básicos que estão disponíveis no Android e como utilizá-los.
- Conhecer os principais layouts e saber como utilizá-los.

Interfaces Gráficas no Android

Existem duas maneiras de criar interfaces gráficas no Android. Ambas são muito importantes e, na grande maioria das aplicações, as duas maneiras de desenvolvimento estarão presentes, trabalhando juntas, pois elas se complementam. A primeira maneira de desenvolver interfaces que iremos estudar é a maneira declarativa, responsável pela maior parte da interface gráfica das aplicações desenvolvidas.

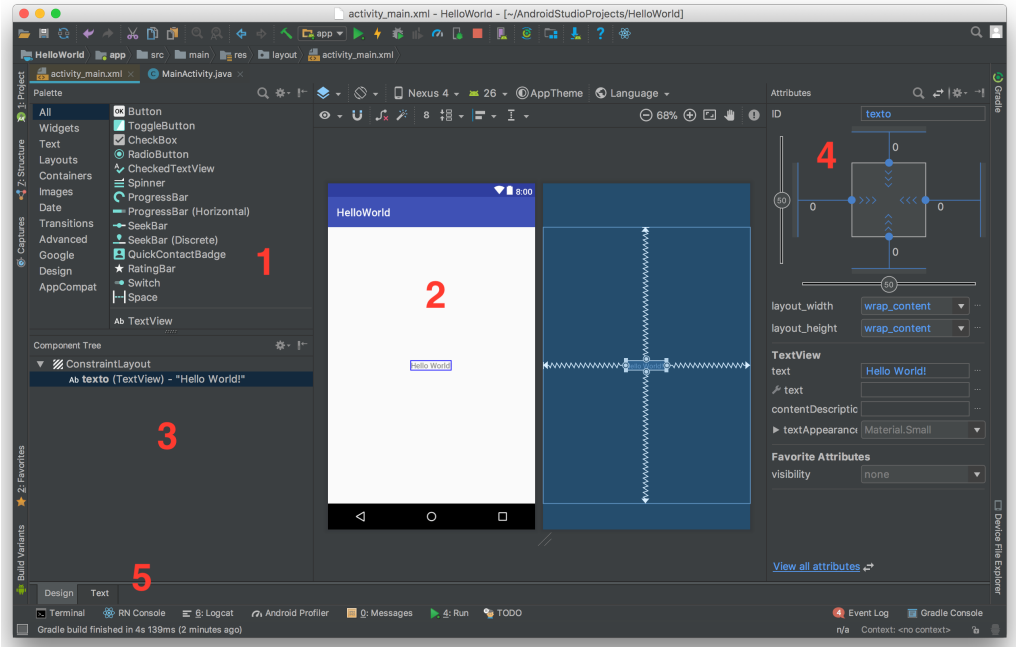
Na maneira declarativa, como o próprio nome sugere, as propriedades da interface gráfica são declaradas em um arquivo XML, seguindo um padrão próprio dos layouts Android, e então são carregadas na aplicação, a partir desse arquivo XML. Essas propriedades não são capazes de mudar ao longo da execução do programa, exceto se utilizado um pouco da segunda maneira, como veremos adiante. Uma vez carregado o XML, as propriedades se mantêm como declaradas. Essa é a maneira mais utilizada, pois, para a maioria dos componentes da aplicação, não haverá mudanças em seu estado ao longo da execução da aplicação. Um título, um campo de texto ou até mesmo um botão que seja carregado na tela durante a execução de uma aplicação tende a ficar lá por todo o seu tempo de vida visível, sem que haja a necessidade de se mover ou desaparecer. Além disso, é mais fácil criar os layouts visualmente, no Android Studio, apenas arrastando os componentes e definindo suas propriedades.

Na segunda maneira de se desenvolver interfaces gráficas, denominada maneira programática, os componentes são descritos em tempo de execução, ou seja, são definidos ao longo do código da aplicação. Isso faz com que o carregamento deles seja um pouco mais lento, além de ser um pouco mais complicado definir as propriedades através da manipulação de um objeto em código Java. Utilizando essa maneira, há tanto a possibilidade de criar novos componentes quanto de editar componentes já declarados em um XML de layout, por exemplo. Não é comum criar toda uma tela de maneira programática, esta abordagem é mais utilizada, por exemplo, para adicionar, remover ou editar componentes únicos que devem ser exibidos depois de algum evento que iremos tratar no código Java.

Como foi dito anteriormente, as duas maneiras interagem muito bem e se completam. Por exemplo, imagine que um botão, declarado no XML de seu layout, deve apenas aparecer, ou ser clicável, em algumas situações, que são definidas em tempo de execução. A solução para esse problema é combinar as duas possibilidades para o desenvolvimento da interface. Utilizando a maneira declarativa, é mais simples posicionar o botão, escolher o seu estilo, definir o seu tamanho e o seu texto. Uma vez com o botão criado e posicionado, pode-se, então, programaticamente, ou seja, durante o código, mudar a propriedade de visibilidade do botão de acordo com situações detectadas durante a execução da aplicação. Isso demonstra a importância da utilização das duas maneiras de desenvolvimento.

Vamos então aos nossos primeiros códigos em Android. Para começar, vamos analisar o nosso primeiro projeto criado na aula anterior, chamado Hello World. Vamos abrir o arquivo de layout criado juntamente com nossa primeira Activity, chamado activity_main.xml. Esse arquivo está na pasta res\layout do projeto. Ao abrir o arquivo de layout será aberta a tela de edição gráfica do layout, mas podemos alterar para o código XML gerado clicando na aba Text na parte inferior esquerda, como mostra a marca 5 na **Figura 1**.

Figura 01 - Graphical Layout Editor do Android Studio



A **Listagem 1** mostra o código XML do layout desta tela. Ele contém um elemento de texto (TextView) com um campo de texto contendo o texto “Hello World”.

Outra maneira pela qual o Android Studio pode ajudar no desenvolvimento desse arquivo XML é através do recurso de autocompletar. Com esse recurso, ao apertar o atalho Ctrl+Espaço dentro do espaço de um elemento, o Android Studio mostra todas as propriedades que podem ser alteradas para aquele componente. Uma vez escolhida a propriedade, caso seja uma propriedade que tem valores constantes, o Android Studio, através do mesmo atalho, mostra a lista dos valores que aquela propriedade pode assumir.

Já na **Listagem 2**, temos um código exemplo da abordagem programática. Nessa Listagem, alteramos o texto que estava na caixa de texto declarada na **Listagem 1**. Ao alterar esse valor durante a execução do programa, temos, então, um exemplo de como as duas abordagens para o desenvolvimento das interfaces gráficas podem e devem interagir ao longo de uma aplicação.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:tools="http://schemas.android.com/tools"
5     xmlns:app="http://schemas.android.com/apk/res-auto"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context="PACOTE.DO.SEU.PROGRAMA.NOME ">
9
10    <TextView
11       android:id="@+id/texto"
12       android:layout_width="wrap_content"
13       android:layout_height="wrap_content"
14       android:text="Hello World!"
15       app:layout_constraintBottom_toBottomOf="parent"
16       app:layout_constraintLeft_toLeftOf="parent"
17       app:layout_constraintRight_toRightOf="parent"
18       app:layout_constraintTop_toTopOf="parent" />
19
20 </android.support.constraint.ConstraintLayout>
```

Listagem 1 - XML da aplicação Hello World

Como podemos ver na **Listagem 1**, existem alguns padrões a serem seguidos, como em qualquer XML. O Android Studio ajuda bastante na criação desses arquivos, gerando todo esse código XML quando usamos o Graphical Layout Editor, arrastando os elementos da interface, listados no painel da esquerda, como vemos na marca 1 da **Figura 1**, e soltando na tela da aplicação (marca 2 da **Figura 1**).

O painel da marca 3 da **Figura 1** lista todos os componentes inseridos neste layout, e o painel abaixo, indicado pela marca 4, mostra as propriedades de um determinado elemento selecionado, possibilitando sua modificação. Vamos selecionar o componente TextView na listagem de componentes e depois alterar o seu id, digitando “texto” no campo id no painel de propriedades do elemento (marcação 4). Depois salve o arquivo.

Vejamos agora, na **Listagem 2**, um código exemplo da abordagem programática para a classe MainActivity.java. Nessa Listagem, alteramos o texto que estava na caixa de texto declarada na **Listagem 1** A caixa de texto é identificada no código Java pela constante R.id.texto (como definido nas suas propriedades, conforme configuramos anteriormente), podendo assim ser recuperada através deste identificador, para que possa então ser manipulada programaticamente. Ao alterar esse valor durante a execução do programa, temos, então, um exemplo de como as duas abordagens para o desenvolvimento das interfaces gráficas podem e devem interagir ao longo de uma aplicação. Você pode testar esta mudança alterando o método onCreate da classe MainActivity.java e executando, em seguida, para verificar a mudança.

```
1  import android.support.v7.app.AppCompatActivity;
2  import android.os.Bundle;
3  import android.widget.TextView;
4
5  public class MainActivity extends AppCompatActivity {
6
7      @Override
8      protected void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         setContentView(R.layout.activity_main);
11
12         TextView texto = (TextView) findViewById(R.id.texto);
13         texto.setText("Olá Mundo");
14     }
15 }
```

Listagem 2 - Exemplo da alteração de uma propriedade da interface gráfica durante a execução da Aplicação

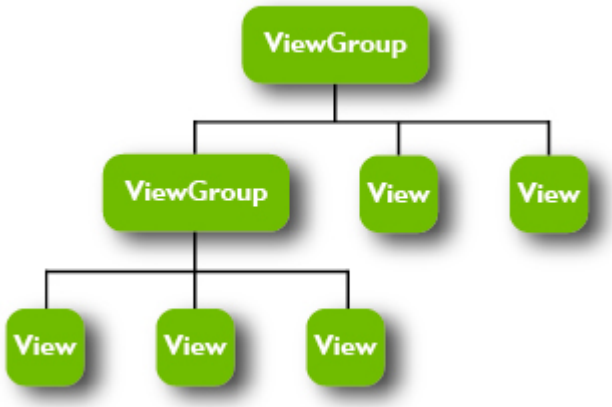
Conhecendo as duas abordagens disponíveis para o desenvolvimento das interfaces gráficas em aplicações Android, vamos então estudar mais a fundo quais são os componentes gráficos que estão disponíveis e como eles são estruturados dentro de uma aplicação Android.

Componentes Gráficos

Os componentes gráficos no Android são divididos em dois principais grupos, que são as Views e os ViewGroups. As Views são as unidades básicas de uma interface gráfica no Android. Por exemplo, um botão representa uma View, um campo de texto representa uma View, entre outros. Elas são a base de qualquer componente visual e armazenam dentro de si propriedades relevantes apenas a elas mesmas. Já os ViewGroups, como o nome sugere, são grupos de Views. Os ViewGroups costumam conter Views e até outros ViewGroups dentro deles. Eles mantêm propriedades que são relevantes a todos os elementos contidos no grupo.

Esses componentes obedecem a uma estrutura hierárquica. Toda interface começa a partir de um ViewGroup, que contém as Views da interface, ou outros ViewGroups, com propriedades distintas e que armazenam outras Views e ViewGroups e assim por diante. Alterações feitas em níveis mais altos da estrutura hierárquica alteram os componentes que estão abaixo. Um exemplo dessa estrutura pode ser visto na **Figura 2**.

Figura 02 - Estrutura dos componentes de uma interface gráfica.



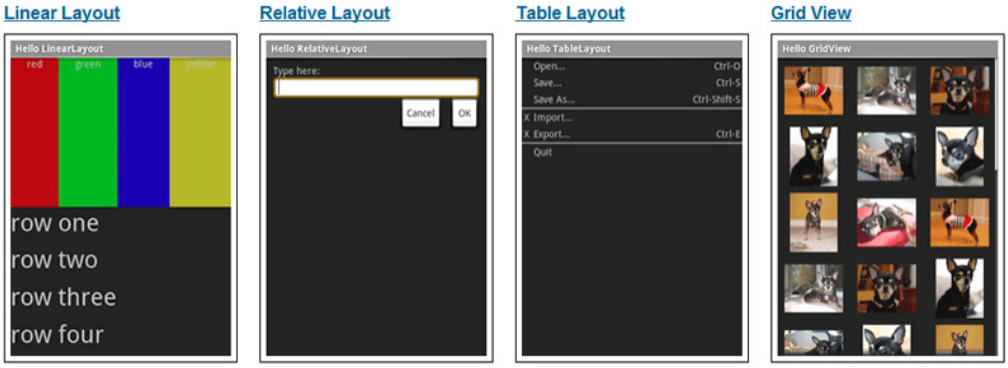
Fonte: Android Developers

O Android já tem algumas Views e ViewGroups padrões, mas, apesar disso, nada impede que o usuário crie suas próprias Views e ViewGroups, até mesmo estendendo os componentes padrões do Android, de forma a criar componentes que atendam a suas necessidades. Além disso, também é possível utilizar componentes criados por terceiros, caso o componente padrão do Android não

solucione o seu problema. Como já foi dito em outras aulas, toda essa abertura para o desenvolvimento das aplicações é uma grande vantagem que o Android trouxe para os desenvolvedores de aplicações móveis.

Agora que já sabemos a diferença de uma View para um ViewGroup, vamos então a alguns exemplos de cada um desses componentes, para melhor compreender seu funcionamento. Na **Figura 3**, vemos alguns exemplos de layouts padrões do Android. Os layouts são considerados ViewGroups, pois eles abrigam outras Views dentro deles, carregando para todas elas algumas propriedades.

Figura 03 - Exemplos de ViewGroups

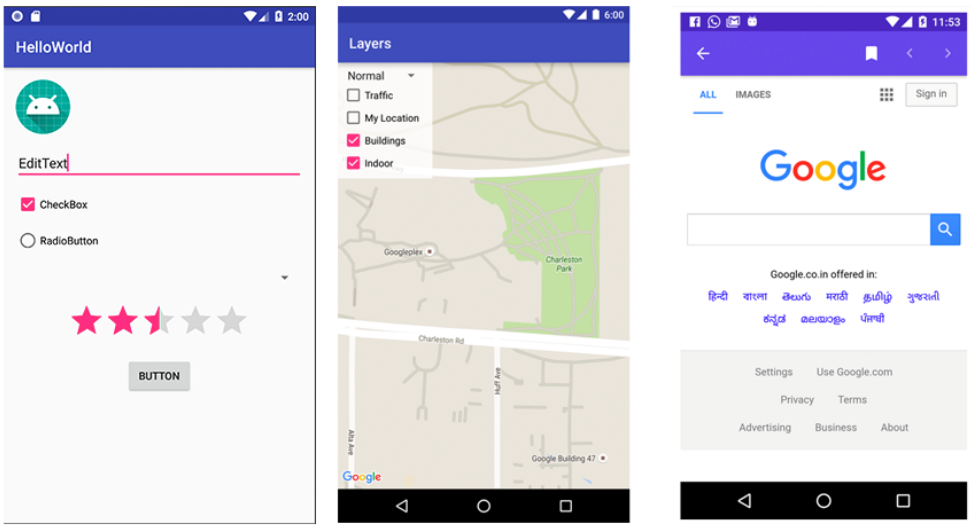


Fonte: <http://developers.android.com/>. Acesso em: 28 de abr. 2015.

Utilizando como exemplo o layout GridView, da **Figura 3**, vemos a estruturação de um ViewGroup, que define o posicionamento de diversas ImageViews, que são Views contidas no ViewGroup principal. Essas ImageViews têm propriedades próprias, como, por exemplo, qual a foto que devem carregar, mas também obedecem a propriedade de posicionamento, herdada do ViewGroup na qual está contida.

Na **Figura 4**, podemos ver alguns exemplos de widgets que são padrões do Android e que podem ser utilizados em qualquer aplicação desenvolvida. Vemos widgets de tamanhos e funcionalidades bem diversas, abrangendo diferentes interesses.

Figura 04 - Exemplos de Views



Form Stuff Mapview Webview

Vejamos, então, a imagem intitulada Form Stuff, como exemplo. Nessa imagem, alguns componentes básicos para o desenvolvimento de um formulário podem ser vistos. Na primeira linha, temos um ImageView, responsável pela exibição de imagens. Na segunda linha, temos uma EditText, que é um campo de texto responsável por receber texto do usuário. Na terceira, um CheckBox e assim por diante. Perceba que essas Views já têm os comportamentos padrões do Android predefinidos, como, por exemplo, abrir o teclado virtual caso um campo para inserção de texto seja selecionado em um dispositivo que não possui teclado físico. Outro exemplo interessante de View é a Web View. Essa View permite ao usuário navegar pela web dentro de sua aplicação, sem a necessidade de um browser externo. Agora que já conhecemos um pouco sobre os layouts e widgets disponíveis no Android, podemos, então, passar a entender como os utilizaremos.

Atividade 01

---

1. Qual a diferença de um View para um ViewGroup?
2. Um ViewGroup pode conter outros ViewGroups? E um View pode conter outros View?

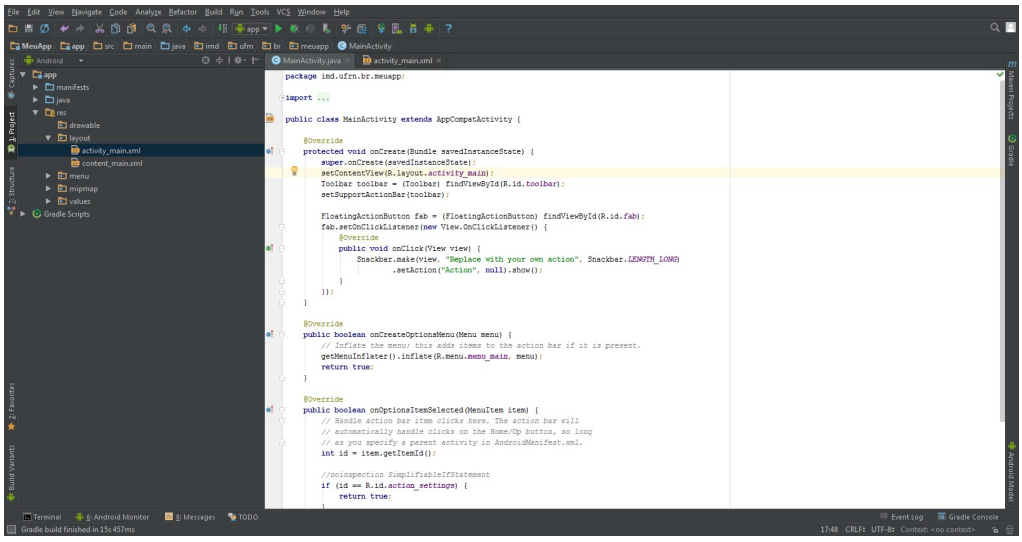
Utilizando layout e widgets

A maneira mais comum de se desenvolver layouts para sua aplicação é através de arquivos XML. Esses arquivos contêm a declaração dos objetos que vão fazer parte da interface, bem como as propriedades desses objetos. Ou seja, dentro do XML, nós vamos ter a declaração de todos os Views e ViewGroups que farão parte da aplicação que está sendo desenvolvida.

Quando se utiliza o carregamento de um arquivo desse tipo como método para montar a interface, o Android internamente cria um objeto Java para cada elemento que foi descrito no XML e então atualiza as propriedades desses objetos de acordo com as propriedades que foram descritas no XML. Com esses objetos em memória, a aplicação pode então exibi-los corretamente, de acordo com o que foi definido no XML. Também, a partir do momento que temos esses objetos carregados, podemos então alterar as propriedades deles programaticamente, como vimos na **Listagem 2**. Para saber como carregar esse layout XML na aplicação, precisamos antes ver como o Android faz a referência de objetos que não são arquivos .java dentro da aplicação.

Como vimos na aula anterior, todo projeto de aplicação Android vem com uma pasta chamada res, que é a abreviação de Resources, ou recursos, em português. Dentro dessa pasta res, várias outras pastas são responsáveis por armazenar os arquivos que não são .java e que serão utilizados dentro da aplicação. Os arquivos que estão dentro dessas pastas serão referenciados em uma classe gerada automaticamente pelo Android, chamada R. Utilizando o objeto dessa classe R, o Android consegue carregar um objeto que não seja .java à aplicação. Como o layout é um arquivo XML, ele também não foge a essa regra. Vejamos então a **Figura 5**.

Figura 05 - À esquerda, as pastas de recursos. À direita, o carregamento de um XML na aplicação Android



Como podemos ver na **Figura 5**, dentro da pasta Res, já existe uma pasta chamada layout, onde deveremos colocar todos os layouts que estarão em nossas aplicações. Uma vez com o arquivo nessa pasta, podemos carregá-lo para nossa aplicação através do comando setContentView(R.layout.NOMEdoLAYOUT), dentro de nossa Activity principal, sendo R.layout.activity_main o caminho desde a pasta Res, referenciada por R, até o layout activity_main.xml, que está na pasta layout. Perceba que não é necessário colocar a extensão do arquivo na hora de referenciá-lo. Veja novamente a **Listagem 2** para ver um exemplo disso no código.

Sabendo como criar e carregar um layout XML para a nossa aplicação, vamos agora ver como podemos editar os componentes desse layout, de modo a ficarem configurados da maneira que gostaríamos que fossem exibidos.

Atributos dos Views

Todo elemento declarado no XML possui um conjunto de atributos. Alguns desses devem ser obrigatoriamente configurados, outros são opcionais para a correta exibição daquela View ou Widget. Alguns atributos são exclusivos de algum View ou ViewGroup específico, outros são padrão e podem ser configurados em qualquer componente. Dentre todos esses atributos, estudaremos agora os três

básicos, que são importantes para todas as Views. Mais adiante, quando virmos os principais layouts do Android, discutiremos alguns atributos que são específicos de cada um dos layouts.

O primeiro desses atributos a serem destacados é o atributo ID. Esse atributo identifica o objeto dentro do XML no qual ele está sendo descrito, e é através dessa propriedade que poderemos acessar esse elemento a partir do código Java. Ou seja, todo objeto que precisa ser acessado no código Java ou mesmo ser utilizado por outro objeto dentro do arquivo XML, deve possuir um ID. Para adicionar um ID ao objeto, a propriedade android:id deve ser configurada. O valor desse atributo é definido na forma "@+id/nomeDoObjeto", indicando que o nome do objeto será uma inclusão ao grupo ID. Na **Listagem 2**, temos um exemplo de como encontrar uma View pelo seu ID e então alterar alguma propriedade em tempo de execução. Na linha em destaque da **Listagem 3**, vemos essa referência sendo feita no XML.

```
1  <TextView
2      android:id="@+id/texto"
3      android:layout_width="fill_parent"
4      android:layout_height="wrap_content"
5      android:text="Hello World" />
6
7  <Button
8      android:id="@+id/button"
9      android:layout_width="wrap_content"
10     android:layout_height="wrap_content"
11     android:layout_below="@+id/texto"
12     android:text="Button"/>
```

Listagem 3 - Referenciando um objeto da interface no próprio XML

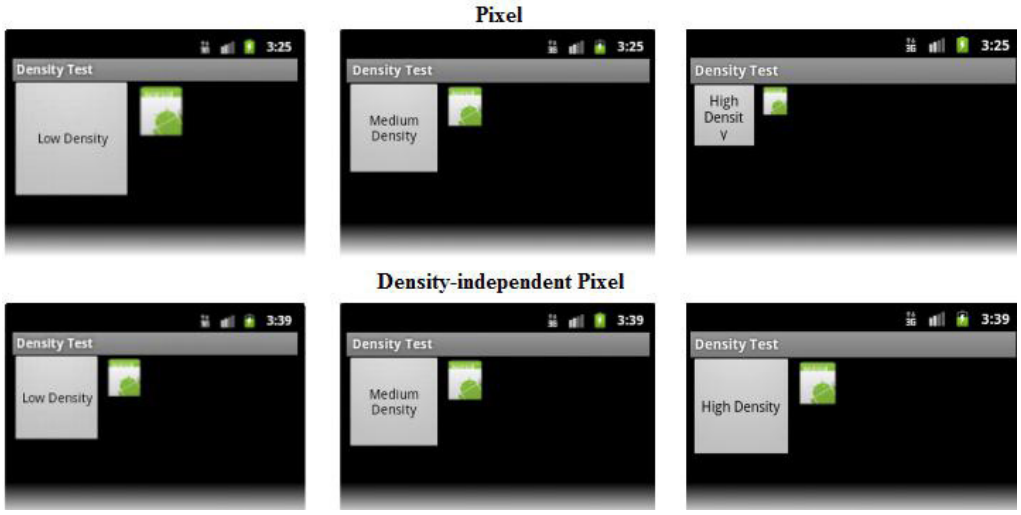
Uma vez definido o ID do objeto, vamos aos atributos de layout que devem ser definidos para todas as Views e ViewGroups. O atributo layout_width é o atributo que define o tamanho horizontal do componente, enquanto o atributo layout_height define o tamanho vertical. Esses dois atributos devem obrigatoriamente ser configurados para todos os componentes. É possível configurar esses atributos com medidas exatas, utilizando, por exemplo, o tamanho em pixels(px); porém, graças ao tamanho variável que as telas de dispositivos Android possuem, duas constantes costumam ser utilizadas para esses atributos. São elas:

- **fill_parent (versões anteriores a 2.2) ou match_parent (utilizada a partir da versão 2.2):** Esse valor indica que a View deve ocupar todo o tamanho disponibilizado a ela pelo seu elemento pai.

- **wrap_content:** Indica a View que ela deve se redimensionar de modo a ocupar apenas o tamanho que seus elementos precisam para serem exibidos.

Apesar das diferentes telas, caso ainda assim deseje utilizar o tamanho em pixel do elemento, deve-se utilizar a unidade dip, abreviação de Density-independent pixels. Essa unidade define o tamanho do elemento em uma unidade que não varia nas diferentes telas suportadas pela plataforma Android, criando, como vemos na **Figura 6**, um elemento de tamanho fixo, seja qual for a densidade de pixels tela.

Figura 06 - Variação do tamanho dos elementos ao utilizar px (acima) ou dip (abaixo)



Esses três atributos são os principais de toda view definida no XML de sua aplicação Android. Obviamente, existem outros atributos que podem ser configurados para personalizar a view como o desenvolvedor desejar. A lista completa de atributos pode ser encontrada utilizando o recurso autocompletar do Android Studio, verificada no painel de propriedades dos elementos na tela do editor gráfico, bem como ser consultada no site de referências do Android. Para se familiarizar com cada um desses atributos, nada melhor do que praticar!

Atividade 02

1. Crie um novo projeto no Android Studio. Dentro do layout activity_main.xml, padrão de um novo projeto, crie um botão posicionado abaixo do campo de texto, com ID "botão_nome", ocupando todo o espaço horizontal que lhe é fornecido, o mínimo espaço vertical possível e mude o

texto do botão para seu nome. Execute o novo projeto em um emulador e veja o resultado.

2. Altere o tamanho da fonte do texto do botão para 20 dip. O que acontece com o tamanho horizontal do botão? E com o vertical?

Principais Layouts

O Android disponibiliza vários layouts para seus desenvolvedores, mas, dentre eles, três se destacam no desenvolvimento de aplicações. Esses três layouts possuem características distintas e a combinação deles possibilita a criação de aplicações diversas, com interfaces tão complexas quanto necessário, conforme veremos a seguir.

LinearLayout

O LinearLayout, como o nome sugere, organiza os elementos internos a ele de maneira linear, horizontalmente ou verticalmente. Em outras palavras, um LinearLayout vertical posicionará todos os filhos um abaixo do outro, enquanto um LinearLayout horizontal os posicionará um ao lado do outro. Vejamos os atributos importantes a serem definidos em um LinearLayout:

- *android:orientation* – esse atributo recebe os valores horizontal ou vertical e é o atributo responsável por definir em que eixo os atributos serão enfileirados.

- *android:id* – atributo de identificação, estudado na seção 3.

- *android:layout_height e android:layout_width* – atributos de altura e largura, também estudados anteriormente.

- *android:gravity* – atributo responsável por definir o posicionamento dos filhos desse layout. Um atributo gravity configurado com o valor top, por exemplo, indica que o filho estará alinhado com o topo do espaço que lhe foi cedido.

- *android:padding* – o padding é expressado em um valor fixo, definido em pixels, ou preferencialmente dip, para esquerda, direita, cima e baixo. Ele pode ser especificado para uma direção específica

(utilizando *android:paddingRight*, *android:paddingTop*, etc...) ou para todas as direções, com o *android:padding*. O valor do padding é adicionado à view, passando a ser parte de seu tamanho. Ou seja, caso uma view tenha o tamanho horizontal de 2 dip e tenha um paddingRight de 2dip adicionado como atributo, seu tamanho horizontal passa a ser 4 dip, sendo os 2 dips de seu tamanho original deslocados para a esquerda 2 dip, por causa do paddingRight.

- *android:layout_weight* – atributo definido para os elementos internos ao LinearLayout, indicando qual o peso que aquele componente irá receber para ser exibido. Quanto maior o peso, maior o espaço que uma View irá receber, caso haja espaço sobrando no Layout. Se o peso de uma View for 0 (padrão), ela não entrará na divisão dos pixels extra.

RelativeLayout

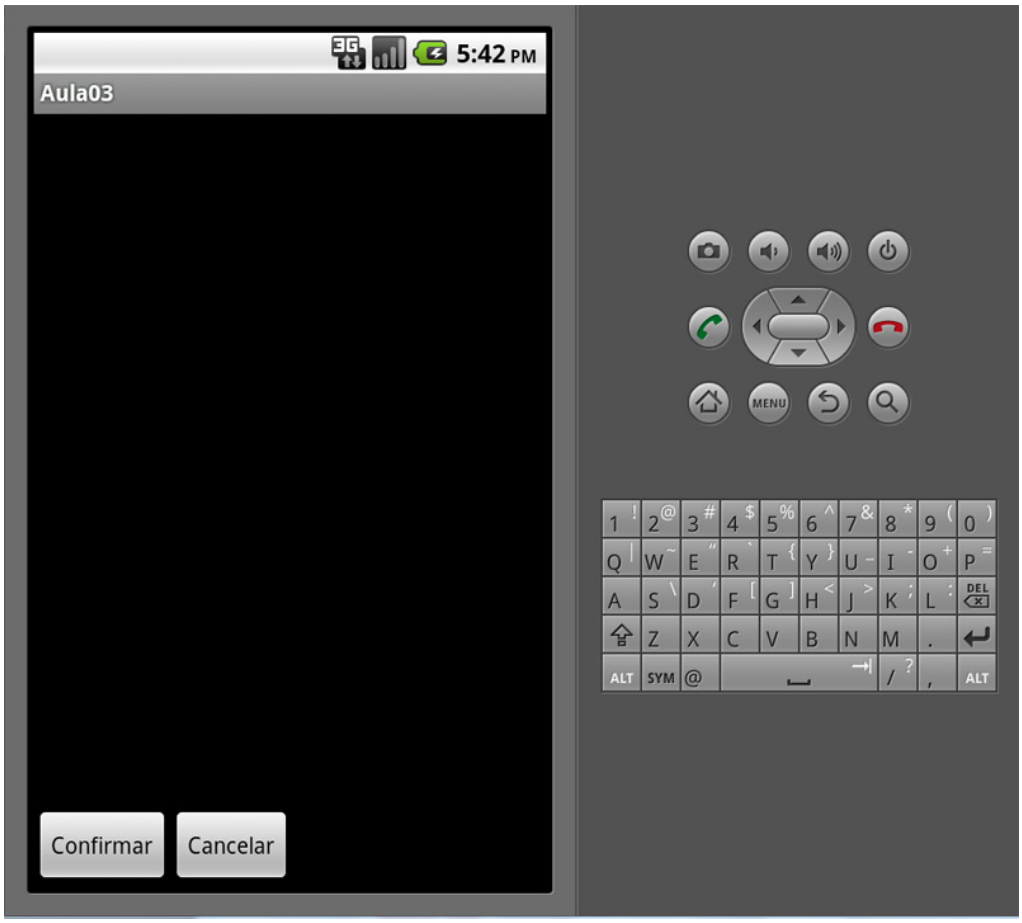
O segundo layout que estudaremos nessa seção é o RelativeLayout. Esse layout organiza seus filhos relacionando uns aos outros, ou relacionando-os ao pai. A utilização desse layout cria diversas novas opções de organização que influenciam no posicionamento dos Views e ViewGroups nele contidos. Por exemplo, vamos supor que queremos criar um botão_confirmar, que ficará posicionado no canto inferior esquerdo do RelativeLayout, que é a visão principal da tela, e um botão_cancelar, que ficará à direita do botão posicionado inicialmente. O código do XML, utilizando um RelativeLayout seria:

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/RelativeLayout"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button
      android:id="@+id/botao_confirmar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_alignParentLeft="true"
      android:text="Confirmar" />

   <Button
      android:id="@+id/botao_cancelar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_toRightOf="@+id/botao_confirmar"
      android:text="Cancelar" />
</RelativeLayout>
```

Listagem 4 - XML utilizando RelativeLayout

Como vemos no exemplo acima, utilizamos atributos de referência de posicionamento nos dois botões. No primeiro botão, que deve ser posicionado no canto inferior esquerdo, há a utilização dos atributos *android:layout_alignParentBottom* e *android:layout_alignParentLeft*, indicando o posicionamento do botão no canto inferior esquerdo. Já o outro botão, posicionado ao lado do primeiro, utiliza-se da propriedade *android_alignParentBottom* para se posicionar em relação ao pai e da propriedade *android:layout_toRightOf*, passando o ID do objeto, como vimos na seção 3, para se posicionar em relação a uma outra View. O resultado pode ser visto na **Figura 7**.

Figura 07 - Botões posicionados utilizando RelativeLayout



TableLayout

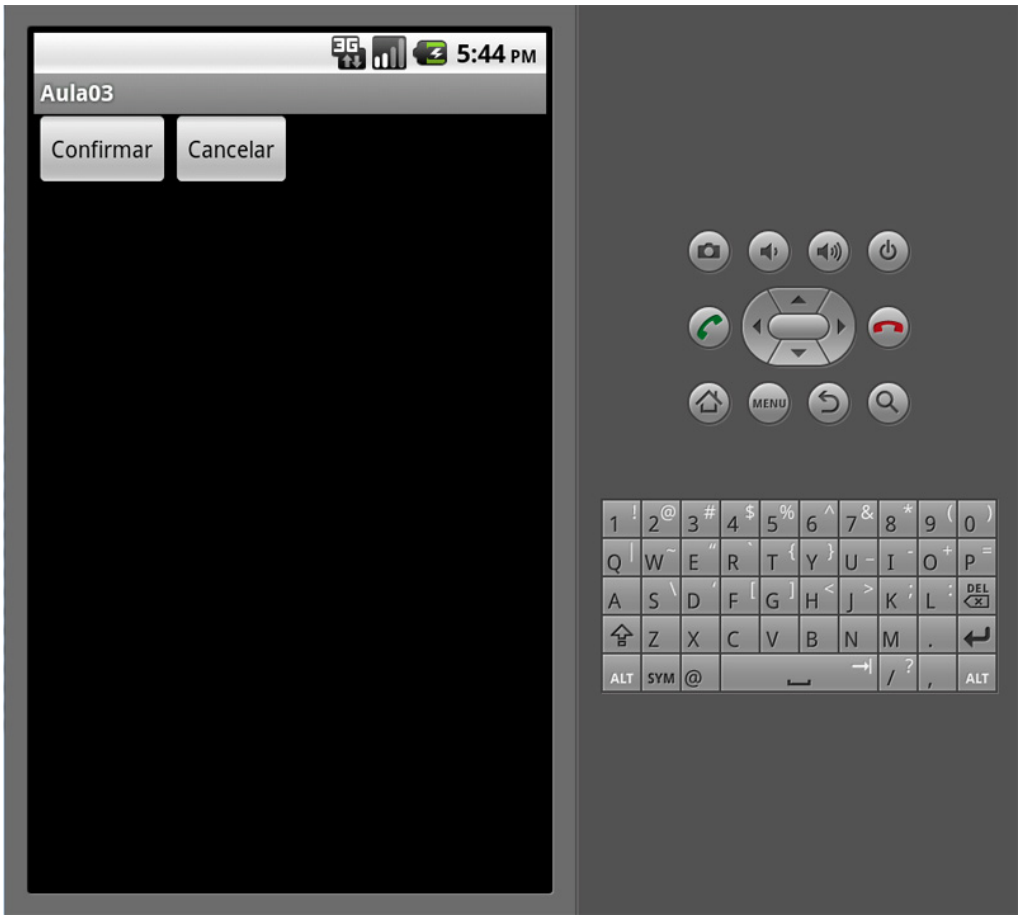
Como o nome sugere, o TableLayout organiza os elementos em forma de tabela, ou seja, respeitando linhas e colunas. Nesse Layout, você define as linhas do layout e quais são os elementos que irão fazer parte de cada linha, formando assim as colunas. Vejamos o exemplo de código a seguir, para melhor entender:

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/TableLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TableRow
      android:id="@+id/tableRow1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" >

   <Button
      android:id="@+id/botao_confirmar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Confirmar" />

   <Button
      android:id="@+id/botao_cancelar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
         android:text="Cancelar" />

   </TableRow>

</TableLayout>
```

Listagem 5 - XML utilizando TableLayout

Como veremos a seguir na **Figura 8**, o resultado do código apresentado na Listagem 5 produz um resultado bem parecido com o layout anterior, porém, alinhado com o topo da tela, já que o layout começa ocupando a tela por cima. Perceba que é possível produzir telas semelhantes com layouts diferentes. Cabe então ao programador decidir qual layout será o mais fácil de implementar para atingir o objetivo que se deseja. Com tempo e estudo, essa tarefa se tornará simples.

Figura 08 - Botões posicionados utilizando TableLayout



Outros Layouts Importantes

Além desses 3 principais layouts do Android, podemos citar ainda outros 3: o GridLayout, o FrameLayout e os Fragments.

O GridLayout torna a tela numa espécie de tabela, onde podemos separar os componentes por linhas e colunas, definir espaços, e organizá-los de forma relativa.

O FrameLayout é um layout utilizado para um só componente ocupar todo seu espaço. Se vários componentes forem adicionados a um FrameLayout, eles ficarão sobrepostos. Geralmente ele é utilizado como layout base de uma tela, ocupando toda a largura e altura da tela, sendo que dentro dele é colocado algum outro layout para exibir e organizar outros componentes, caso seja necessário, ocupando assim toda a área da tela.

Iremos aprender mais sobre os Fragments na Aula 7 – Interfaces gráficas III.

Com o conhecimento dos layouts mais utilizados no desenvolvimento de aplicações Android, encerramos a nossa primeira aula sobre interfaces gráficas no Android. Esta aula é de grande importância para o desenvolvimento do curso, então tenha uma dedicação especial com os exercícios e o entendimento do conteúdo.

Até a próxima!

Atividade 03

Apresente o código para criar a mesma tela da **Figura 8** utilizando LinearLayout.

Leitura Complementar

Layouts no Android. Disponível em: <http://www.thecodebakers.org/2011/04/layouts-no-android.html>. Acesso em: 01 dez. 2015.

Resumo

Nesta aula, vimos como são declaradas as interfaces gráficas no Android, assim como as diferenças encontradas na utilização de cada um dos métodos. Em seguida, vimos como são divididos os componentes gráficos e como as Views se estruturam de forma hierárquica em uma aplicação Android. Aprendemos então como utilizar layouts e widgets, tanto como criá-los, quanto como carregá-los na aplicação. Vimos ainda quais os principais atributos dessas Views. Por fim, vimos os principais layouts utilizados no Android e suas principais características.

Autoavaliação

1. Defina a maneira declarativa de definição de interfaces e explique como ela deve interagir com a maneira programática para criar aplicações.
2. Como se estruturam as Views e ViewGroups dentro da interface de uma aplicação Android? Como elas interagem dentro dessa estrutura?
3. Em que pasta devem ser colocados os arquivos XML que serão carregados como layouts? Qual o comando para carregá-los?
4. Quais as duas principais propriedades de layout em um View e quais valores elas podem assumir? Descreva esses valores.
5. Quais os três principais layouts do Android? Descreva brevemente cada um deles.

Referências

ANDROID Developers. 2012. Disponível em: <http://developers.android.com>. Acesso em: 31 abr. 2012.

DIMARZIO, J. **Android:** a programmer's guide. São Paulo:McGraw-Hill, 2008. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 16 abr. 2012.

LECHETA, Ricardo R. **Google android:** aprenda a criar aplicaçoes. 2. ed. São Paulo: Novatec, 2010.

LECHETA, Ricardo R. **Google android para tablets**. São Paulo: Novatec, 2012.

MEIER, R. **Professional Android 2 application development**. New York: John Wiley & Sons, 2010. Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC>. Acesso em: 16 abr. 2012.