Dispositivos Móveis

Aula 05 - Interfaces Gráficas II

Apresentação

Olá! Nesta aula, continuaremos nossos estudos sobre interfaces gráficas em aplicações Android. Esta aula, diferentemente da primeira, será uma aula mais prática, onde veremos exemplo de implementação das principais *Views* que compõem uma aplicação Android. Além disso, veremos exemplos visuais e o código para implementá-los, assim como as principais propriedades que podem ser utilizadas em cada um dos componentes estudados. Vamos ao trabalho!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Implementar as principais Views do Android.
- Carregar imagens em sua aplicação.
- Desenvolver as primeiras telas funcionais.

Criando uma interface

Para se criar uma interface no Android, como já vimos na aula anterior sobre interface gráfica, precisamos utilizar uma combinação de Views e ViewGroups, de modo a adicionar os componentes que gostaríamos de utilizar na nossa interface e posicioná-los da maneira correta. Inicialmente, iremos demonstrar os componentes necessários para criar uma interface de login, como a exibida na **Figura 1**.

Figura 01 - Interface de *login* a ser desenvolvida

Inicialmente, crie um novo projeto Android chamado Aula05 e que contenha uma Activity.

Para desenvolver essa interface, utilizaremos um RelativeLayout, já visto na aula anterior sobre interfaces. A partir desse **layout**, discutiremos cada um dos componentes envolvidos na tela e iremos, componente a componente, montar a interface vista na **Figura 1**, até que tenhamos a nossa primeira interface completamente funcional. Vamos começar relembrando alguns aspectos sobre o RelativeLayout.

RelativeLayout

O RelativeLayout posiciona os componentes da interface gráfica de maneira relativa ao componente pai, ou em relação a outro componente. Para utilizar esse componente como base da nossa interface, devemos fazer a declaração (vista na **Listagem 1**) no XML que será carregado em nossa Activity.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/RelativeLayout1"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent">
6 </RelativeLayout>
```

Listagem 1 - RelativeLayout

Esse código define o RelativeLayout que será utilizado em nossa interface. Perceba que apenas as propriedades básicas estão definidas. A primeira mudança que iremos fazer nessa interface gráfica é a mudança da cor de fundo. Para fazer essa mudança, utilizaremos a propriedade android:background. Perceba que essa propriedade recebe tanto cores, quanto arquivos de imagem que estejam, preferencialmente, salvos na pasta res/drawable, dentro do seu projeto Android. Como na nossa aplicação estaremos apenas mudando a cor de fundo, utilizaremos uma cor para essa propriedade. No XML, as cores são definidas como #RGB, #ARGB, #RRGGBB ou #AARRGGBB, onde o R vem de Red (vermelho), o G de Green (verde), o B de Blue (azul) e o A de Alpha, que representa o nível de transparência da cor. Quanto maior o alpha, mais da cor será utilizada, e quanto menor, mais transparente irá ficar. Todos os valores são definidos em hexadecimal, ou seja, de 0 a F, sendo A = 10, B = 11, ..., F = 15. Para a nossa aplicação, utilizaremos o sistema mais simples, o #RGB, com valores #007. A nova propriedade, então, fica:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/RelativeLayout1"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"gi
      android:background="#007">
</RelativeLayout>
```

Listagem 2 - Propriedade para alterar o background com uma cor azul

Caso queira utilizar uma imagem como plano de fundo na sua interface, o parâmetro da propriedade background deverá indicar o caminho da sua imagem. Supondo que o arquivo a ser carregado seja o icon.png, salvo na pasta drawable, o parâmetro da propriedade background seria “@drawable/icon”.

Agora que temos o XML da nossa aplicação, vamos carregá-lo na nossa Activity principal para começarmos a utilizá-lo. O arquivo se chama activity_main.xml e seu carregamento na classe MainActivity.java, através do método setContentView(R.layout.activity_main), pode ser visto na **Listagem 3**.

```
public class MainActivity extends Activity {
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
      }
}
```

Listagem 3 - Carregando a interface na Activity principal

Para suporte ideal a todos os diferentes tipos de tela, o desenvolvedor pode criar imagens específicas para cada resolução suportada e colocar essas imagens na pasta relativa àquela resolução, com o mesmo nome de arquivo em todas as pastas. Uma vez feito isso, basta referenciar a imagem, como citado anteriormente, sem o indicador de resolução. O Android se responsabilizará por encontrar automaticamente a melhor imagem a ser utilizada, durante a execução do programa.

Com o layout carregado na nossa Activity, podemos, então, começar a criar a nossa tela de login. Vamos a ela!

Atividade 01

1. Adicione uma imagem no formato PNG a sua pasta de drawables e carregue-a como background de sua aplicação.
2. Inicie um emulador e veja como fica a tela com a imagem. Rotacione o emulador (utilizando o atalho Ctrl+F11) e veja como a imagem se dimensiona.

TextView

Vídeo 02 - Principais Widgets

O primeiro componente que iremos adicionar ao nosso layout é um componente do tipo TextView. Esse componente é responsável pela exibição de textos na tela. É possível tanto definir o texto no XML, quanto alterá-lo em tempo de execução através de código Java. Basta manipular o elemento corretamente em sua Activity. Para começar nossos estudos, vamos adicionar o TextView em nosso layout XML.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/RelativeLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#007">

   <TextView
      android:id="@+id/welcome_msg"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="15dip"
      android:gravity="center"
      android:text="Olá! Por favor, insira seu login e senha"
      android:textSize="20sp"
      android:textStyle="bold"
   android:textColor="#FFF" />
</RelativeLayout>
```

Listagem 4 - TextView de boas vindas

Ufa! Quantas propriedades! Mas vamos com calma. Discutiremos cada uma delas, na sequência em que aparecem. As três primeiras propriedades que vemos são as propriedades básicas que já estudamos. Perceba que o componente está utilizando "*wrap_content*" como propriedade de altura e largura, para não utilizar mais espaço que o necessário para exibir o texto. Em seguida, vemos três componentes que dizem respeito ao posicionamento do componente no layout. O primeiro deles indica que o componente deve ter o seu topo alinhado com o topo do pai, definindo *layout_alignParentTop* como true, ou seja, alinhado com o topo do RelativeLayout. Na sequência, define-se que ele será centralizado horizontalmente em relação ao RelativeLayout pai. Isso é feito configurando *layout_centerHorizontal* para true. Para finalizar o posicionamento, definimos o espaço que ele tem em relação ao topo do layout. Esse espaço é definido por meio da propriedade *layout_marginTop*, definida como 15dip (density-independent pixels, lembra?). Com essa última configuração, a View está agora posicionada alinhada com o topo do pai, porém afastada 15 pixels e centralizada horizontalmente. Lembre-se que essas propriedades são alteráveis em qualquer direção (*alignParentBottom, marginLeft, centerVertical*, entre outros...). Para se familiarizar com todos esses comandos, lembre-se de utilizar o autocompletar do código no Android Studio (Ctrl+Espaço).

As próximas propriedades definidas dizem respeito ao texto. A primeira delas, *gravity*, como já foi visto anteriormente, define o posicionamento de um elemento de View dentro de seu espaço. No caso da View estudada, o texto está sendo definido como center, ou seja, centralizado. Outros valores são possíveis para esse atributo, assim como a combinação de valores, utilizando o caractere |. Um exemplo seria o valor "*top*|*right*" para posicionar o componente alinhado ao topo e a direita, simultaneamente.

A propriedade seguinte, *text*, é a propriedade que indica qual será o texto exibido no TextView inserido. Essa propriedade deve ser configurada preferencialmente utilizando um valor do arquivo *strings.xml*. O arquivo *strings.xml* é um arquivo padrão dos projetos Android, localizado na pasta *res**values*, e que deve ser o responsável por guardar todas as strings que são mostradas ao usuário em uma aplicação Android. Esse arquivo existe para facilitar a internacionalização de aplicações. Caso o usuário queira traduzir uma aplicação por completo, basta alterar todos os valores contidos nesse arquivo e a aplicação terá novos textos, sem precisar mudar nada no código dela. No Android Studio, você pode escrever uma mensagem em texto plano e então selecionar a mensagem e apertar Alt + Enter. Isso irá abrir um menu com a opção "Extract String Resource". Ao clicar nessa opção, você será solicitado um nome para a string a ser extraída e então esta será automaticamente colocada no arquivo strings.xml sob o nome indicado e substituída no local. Uma vez extraída a string, basta referenciá-la como já fizemos com arquivos de outro tipo. No caso da nossa aplicação, utilizamos "*@string/welcome_msg*", onde *welcome_msg* é o nome que a string foi salva dentro do arquivo *strings.xml*.

As três últimas propriedades definem o tamanho da fonte, o estilo e a cor. A propriedade *textSize* define, em pixels, o tamanho da fonte a ser utilizada no TextView, enquanto *textStyle* define se a fonte será normal, itálica ou em negrito, e, por último, *textColor* define a cor do texto. No caso do nosso TextView, a fonte é mostrada em tamanho 20sp, em negrito e na cor branca (#FFF).

Agora que já vimos todas as propriedades utilizadas no nosso TextView, vamos adiante. O próximo passo é aprender como adicionar os campos de texto para que o usuário possa inserir o seu usuário e senha para realizar o login.

EditText

O EditText é o componente responsável por receber entrada de texto do usuário. É através desse componente que o usuário vai inserir o login e a senha para se autenticar em nossa aplicação. Vejamos, na Listagem 5, o código que declara ambos os componentes e vamos inseri-lo dentro do RelativeLayout do XML.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/RelativeLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#007">

   <TextView
      android:id="@+id/welcome_msg"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="15dip"
      android:gravity="center"
      android:text="Olá! Por favor, insira seu login e senha"
      android:textSize="20sp"
      android:textStyle="bold"
   android:textColor="#FFF" />

   <EditText
      android:id="@+id/login_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/welcome_msg"
      android:layout_marginTop="50dp"
      android:hint="Login"
      android:maxLines="1" >
   </EditText>

   <EditText
      android:id="@+id/pass_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/login_text"
      android:layout_marginTop="30dp"
      android:hint="Senha"
      android:inputType="textPassword"
      android:maxLines="1"
      android:longClickable="false" />
</RelativeLayout>
```

Listagem 5 - EditText de login e senha

Vemos que ambos os componentes tem as propriedades de layout bem semelhantes às que já vimos anteriormente, sendo posicionando na tela como necessário. A grande diferença que vemos nas propriedades de layout desses componentes, em relação aos já vistos anteriormente, é a utilização de referências à outras Views para o posicionamento dos componentes. Em ambos, podemos ver a propriedade layout_below indicando o posicionamento do EditText em relação a outra View previamente identificada. O EditText login está posicionado abaixo do componente com ID *welcome_msg*, que é o TextView, enquanto o EditText de senha está localizado abaixo do componente com ID *login_text*, que é o campo de login.

Passando dos atributos de layout, vemos, então, os atributos que são pertinentes a um EditText. Em ambos os componentes, temos o atributo *hint* configurado com uma string. Esse atributo é responsável por colocar uma string dentro do EditText para auxiliar o usuário quanto ao objetivo daquele componente. Essa palavra fica em uma cor mais clara que o texto e some assim que o usuário insere dados no EditText. Ela aparece apenas caso o EditText esteja vazio. É importante utilizar esse atributo para facilitar a compreensão da aplicação por parte do usuário final.

Outro atributo utilizado em ambos os campos é o atributo maxLines. Como o nome diz, esse atributo indica a quantidade máxima de linhas que devem compor aquele EditText. Com esse atributo configurado para 1, o Android trata para que não seja possível adicionar quebras de linha ou mesmo textos de múltiplas linhas dentro do campo marcado com essa configuração. Há também algumas propriedades que podem deixar o campo com múltiplas linhas, configurando quantas linhas ele terá de tamanho mínimo ou máximo, por exemplo. Para ver mais sobre essas propriedades, utilize o autocompletar do Android Studio dentro do componente EditText e navegue pela lista completa, ou acesse a documentação oficial do Android.

No EditText de senha, a propriedade *longClickable* colocada como false indica que o componente não vai responder caso o usuário toque nele por um período maior de tempo, ou seja, não será possível abrir um menu de opções para aquele campo. Com isso, evitamos uma das maneiras de copiar os dados do campo, não permitindo que a senha possa ser copiada e/ou colada.

Por fim, a última propriedade importante que veremos para os EditTexts é o *inputType*. Essa propriedade define qual será o tipo de dado que pode ser digitado no campo do EditText. No caso do EditText de senha, o *inputType* é configurado para *textPassword*, indicando que aquele campo é um campo de senha e os caracteres ali digitados devem ser ocultados. Agora que já somos capazes de adquirir informações do usuário, vamos estudar como declarar botões em nosso layout.

Buttons

O Button é o componente que representa um botão textual no Android. Diversos atributos podem ser configurados em um botão, mas aqui comentaremos apenas os principais, que tornam possível a criação de um botão simples e funcional. Vejamos a **Listagem 6**, demonstrado a declaração dos dois botões utilizados na nossa tela de *login*.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/RelativeLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#007">

   <TextView
      android:id="@+id/welcome_msg"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="15dip"
      android:gravity="center"
      android:text="Olá! Por favor, insira seu login e senha"
      android:textSize="20sp"
      android:textStyle="bold"
   android:textColor="#FFF" />

   <EditText
      android:id="@+id/login_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/welcome_msg"
      android:layout_marginTop="50dp"
      android:hint="Login"
      android:maxLines="1" >
   </EditText>

   <EditText
      android:id="@+id/pass_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/login_text"
      android:layout_marginTop="30dp"
      android:hint="Senha"
      android:inputType="textPassword"
      android:maxLines="1"
      android:longClickable="false" />

   <Button
      android:id="@+id/confirmar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
```

```
52      android:layout_below="@+id/pass_text"
53      android:text="Confirmar" />
54
55
56    <Button
57      android:id="@+id/limpar"
58      android:layout_width="wrap_content"
59      android:layout_height="wrap_content"
60      android:layout_alignParentRight="true"
61      android:layout_below="@+id/pass_text"
62      android:text="Limpar" />
63  </RelativeLayout>
```

Listagem 6 - Declaração dos botões do layout

Apenas com os atributos que já estudamos anteriormente, é possível criar os botões que precisamos para tornar a nossa tela de login funcional. O principal ponto em relação aos botões não é a declaração deles no XML, mas sim a programação e tratamento dos cliques na Activity, afinal, é isso que os torna funcionais. Vejamos, então, na **Listagem 7**, a implementação do comportamento dos botões em nossa Activity. O botão Confirmar irá indicar sucesso apenas se o usuário e a senha digitados forem iguais a “aluno”. Já o botão limpar vai, simplesmente, limpar o valor dos campos.

```
1  package br.ufrn.imd.aula05f;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.Button;
7  import android.widget.EditText;
8
9  public class Aula05Activity extends Activity {
10   @Override
11   protected void onCreate(Bundle savedInstanceState) {
12      super.onCreate(savedInstanceState);
13      setContentView(R.layout.activity_main);
14
15      Button confirmar = (Button) findViewById (R.id.confirmar);
16      Button limpar = (Button) findViewById (R.id.limpar);
17
18      final EditText login = (EditText) findViewById(R.id.login_text);
19      final EditText senha = (EditText) findViewById(R.id.pass_text);
20
21      confirmar.setOnClickListener(new View.OnClickListener() {
22
23         @Override
24         public void onClick(View v) {
25            String loginStr = login.getText().toString();
26            String senhaStr = senha.getText().toString();
27            if (loginStr != null && loginStr.equals("aluno")) {
28               if (senhaStr != null && senhaStr.equals("aluno")) {
29                  //Sucesso
30               } else {
31                  //Falha senha
32               }
33            } else {
34               //Falha login
35            }
36         }
37      });
38
39      limpar.setOnClickListener(new View.OnClickListener() {
40
41         @Override
42         public void onClick(View v) {
43            login.setText("");
44            senha.setText("");
45         }
46      });
47   }
48 }
```

Listagem 7 - Implementando o comportamento dos botões

Como se pode ver na **Listagem 7**, para implementar o comportamento dos botões após um clique, é necessário, primeiramente, referenciar os componentes Button no código Java por meio do método *findViewById*, utilizando o ID configurado no XML de layout. Uma vez que tenhamos a referência desses botões, é possível configurar o *onClickListener* deles para executar alguma ação que queiramos. Como precisaríamos dos valores dos EditTexts para tomarmos as decisões após o clique no botão de confirmar, utilizamos novamente o método *findViewById* para encontrar e referenciar os componentes e, então, utilizando o método *getText()*, podemos pegar o valor contido no campo do EditText e, após convertê-lo para string utilizando o *toString()*, verificá-lo.

Agora que já temos uma tela capaz de simular um login, vamos adicionar uma imagem que possa representar o sucesso ou não do login para que o usuário tenha uma resposta após clicar no botão Confirmar. Em outras aulas, estudaremos a comunicação entre Activities e poderemos, então, passar para outra Activity, caso o login esteja correto.

ImageView

O ImageView, como o nome sugere, é o componente responsável por mostrar imagens ao usuário. Ele é semelhante ao TextView, porém, trabalha com propriedades de imagens e não de texto. Vejamos, na **Listagem 8**, como é feita a declaração do componente em nossa aplicação. Vamos utilizar uma imagem padrão Android para o exemplo.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/RelativeLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="#007">

   <TextView
      android:id="@+id/welcome_msg"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="15dip"
      android:gravity="center"
      android:text="Olá! Por favor, insira seu login e senha"
      android:textSize="20sp"
      android:textStyle="bold"
   android:textColor="#FFF" />

   <EditText
      android:id="@+id/login_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/welcome_msg"
      android:layout_marginTop="50dp"
      android:hint="Login"
      android:maxLines="1" >
   </EditText>

   <EditText
      android:id="@+id/pass_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentRight="true"
      android:layout_below="@+id/login_text"
      android:layout_marginTop="30dp"
      android:hint="Senha"
      android:inputType="textPassword"
      android:maxLines="1"
      android:longClickable="false" />

   <Button
      android:id="@+id/confirmar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
```

```
52      android:layout_below="@+id/pass_text"
53      android:text="Confirmar" />
54
55
56    <Button
57      android:id="@+id/limpar"
58      android:layout_width="wrap_content"
59      android:layout_height="wrap_content"
60      android:layout_alignParentRight="true"
61      android:layout_below="@+id/pass_text"
62      android:text="Limpar" />
63
64    <ImageView
65      android:id="@+id/imageView1"
66      android:layout_width="wrap_content"
67      android:layout_height="wrap_content"
68      android:layout_alignBottom="@+id/confirmar"
69      android:layout_centerHorizontal="true"
70      android:src="@android:drawable/presence_offline" />
71  </RelativeLayout>
```

Listagem 8 - Declaração do ImageView

A única novidade que encontramos entre as propriedades descritas para esse elemento é a propriedade *src*. Essa propriedade é responsável por indicar qual a imagem que deverá ser carregada na ImageView declarada. Veja que a imagem referenciada é uma imagem padrão do Android, que está contida no pacote @android. Esse pacote é padrão e está disponível em todas as aplicações. Você pode usar uma imagem própria, colocando-a na pasta *res\drawable* e usando uma referência a essa imagem, como, por exemplo, *android:src="@drawable/minha_imagem".*

Uma vez declarado o componente, vamos voltar à Activity e fazer com que a imagem mude, caso o login seja bem sucedido, e voltar ao seu estado inicial após a limpeza da tela. Vejamos, na **Listagem 9**, a declaração da ImageView na classe MainActivity.java, e as mudanças feitas nos métodos onClick() dos botões.

```
package br.ufrn.imd.aula05f;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;

public class Aula05Activity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Button confirmar = (Button) findViewById (R.id.confirmar);
      Button limpar = (Button) findViewById (R.id.limpar);

      final EditText login = (EditText) findViewById(R.id.login_text);
      final EditText senha = (EditText) findViewById(R.id.pass_text);
      final ImageView imgView = (ImageView) findViewById(R.id.imageView1);

      confirmar.setOnClickListener(new View.OnClickListener() {
         @Override

         public void onClick(View v) {
            String loginStr = login.getText().toString();
            String senhaStr = senha.getText().toString();
            if (loginStr != null && loginStr.equals("aluno")) {
               if (senhaStr != null && senhaStr.equals("aluno")) {
                  //Sucesso
                  imgView.setImageResource(android.R.drawable.presence_online);
               } else {
                  //Falha senha
               }
            } else {
               //Falha login
            }
         }
      });

      limpar.setOnClickListener(new View.OnClickListener() {
         @Override

         public void onClick(View v) {
            login.setText("");
            senha.setText("");
            imgView.setImageResource(android.R.drawable.presence_offline);
         }
      });
   }
```

```
52 }
```

Listagem 9 - Mudando a imagem na Acitivity

A utilização do método *setImageResource* torna possível mudar a imagem de um ImageView. Com ele mudamos a imagem, quando há sucesso no login, ou mudamos novamente para a imagem inicial, quando o botão de limpar é clicado. Veja que o pacote R.drawable, que é referenciado nesses métodos, não é o pacote normal, mas sim o *android.R.drawable*. Como estamos utilizando um *drawable*, que é padrão do Android e não da nossa aplicação, precisamos adicionar a referência ao pacote *android.R* e não ao R local.

Na próxima sessão, veremos como exibir mensagens utilizando o componentes como o Toast e Dialogs. Como exemplo, mostraremos uma mensagem caso o usuário insira login e/ou senha inválidos.

A seguir, temos um vídeo exemplificando a utilização de ImageViews. Aqueles que desejarem acompanhar o desenvolvimento do vídeo utilizando o Android Studio poderão o fazer com o mesmo código exibido e obedecendo a mesma estrutura de pastas no projeto.

Vídeo 03 - Recursos Avançados do ImageView

Atividade 02

1. Substitua o TextView de boas vindas por um ImageView de boas vindas.
2. Passe a dar o feedback em relação ao Login/Senha em um TextView. O TextView deve indicar "Sucesso", em verde, no caso de um login bem sucedido, "Erro de login", em amarelo, no caso de login errado e "Erro de senha", em vermelho, no caso da senha errada.

Toast e Dialogs

O Toast é um componente simples que exibe uma mensagem durante um breve período de tempo durante a execução de uma Activity. Geralmente ele é disparado ao ocorrer algum evento na aplicação, informando o usuário do ocorrido, e desaparecendo pouco tempo depois.

Já os Dialogs são componentes responsáveis, como em sistemas Desktop, pela criação de **popups** que flutuam na frente da Activity atual, contendo ou requisitando algum tipo de informação. São esses componentes, por exemplo, que podem enviar uma mensagem de falha ao usuário quando seu sistema se comportar de maneira inesperada.

O Android possui alguns Dialogs predefinidos, que podem ser criados pelo usuário sem maiores dificuldades. Esses Dialogs são responsáveis por mostrar o progresso de alguma atividade que está acontecendo em segundo plano, ou enviar um alerta ao usuário, por exemplo. Na próxima seção, estudaremos a AlertDialog e a ProgressDialog, como exemplos, além de discutir um pouco sobre a criação de Dialogs personalizadas, criadas a partir de um layout desenvolvido por nós e inflado num componente Dialog.

Toast

O Toast fornece um feedback simples e rápido ao usuário em uma pequena janela. Essa janela é exibida na quantidade de espaço necessário de acordo com a mensagem que lhe foi determinada, permanecendo visível por um tempo limite. Por exemplo, ao apagar um contato, uma mensagem “Contato apagado” será exibida por alguns instantes e depois desaparecerá. Este recurso não possibilita interação com o usuário, apenas exibe informações rápidas, diferentemente dos Dialogs, que fornecem interação com o usuário e serão estudados na próxima sessão.

Utilizando nosso exemplo da tela de login, vamos exibir uma mensagem de login ou de senha inválido caso o usuário digite alguma coisa diferente de “aluno” nos campos, como mostra a **Figura 2**. A **Listagem 10** mostra o código com a utilização do Toast na classe MainActivity.

Figura 02 - Exibição de um Toast ao digitar um login inválido

```
1  package br.ufrn.imd.aula05f;
2  import android.app.Activity;
3  import android.os.Bundle;
4  import android.view.View;
5  import android.widget.Button;
6  import android.widget.EditText;
7  import android.widget.ImageView;
8  import android.widget.Toast;
9
10 public class Aula05Activity extends Activity {
11    @Override
12    protected void onCreate(Bundle savedInstanceState) {
13       super.onCreate(savedInstanceState);
14       setContentView(R.layout.activity_main);
15       Button confirmar = (Button) findViewById (R.id.confirmar);
16       Button limpar = (Button) findViewById (R.id.limpar);
17       final EditText login = (EditText) findViewById(R.id.login_text);
18       final EditText senha = (EditText) findViewById(R.id.pass_text);
19       final ImageView imgView = (ImageView) findViewById(R.id.imageView1);
20
21       confirmar.setOnClickListener(new View.OnClickListener() {
22          @Override
23          public void onClick(View v) {
24             String loginStr = login.getText().toString();
25             String senhaStr = senha.getText().toString();
26             if (loginStr != null && loginStr.equals("aluno")) {
27                if (senhaStr != null && senhaStr.equals("aluno")) {
28                   // Sucesso
29                   imgView.setImageResource(android.R.drawable.presence_online);
30                } else {
31                   // Falha senha
32                   Toast.makeText(getApplicationContext(), "Senha inválida", Toast.LENGTH_SHORT).sho
33                }
34             } else {
35                // Falha login
36                Toast.makeText(getApplicationContext(), "login inválido", Toast.LENGTH_LONG).show();
37             }
38          }
39       });
40       limpar.setOnClickListener(new View.OnClickListener() {
41          @Override
42
43          public void onClick(View v) {
44             login.setText("");
45             senha.setText("");
46             imgView.setImageResource(android.R.drawable.presence_offline);
47          }
48       });
49    }
50 }
```

Listagem 10 - Código com exibição do Toast

Note que para usarmos esse componente, fazemos uso do método *makeToast* com três parâmetros: contexto, texto da mensagem e duração de exibição. A duração é uma constante existente na classe Toast, tendo como opção *LENGTH_SHORT* para um tempo curto de exibição ou *LENGHT_LONG* para um tempo longo. Note que colocamos o tempo longo para "login inválido" e o tempo curto para "Senha inválida". Para exibir o Toast após criá-lo, precisamos chamar o método *show()*.

AlertDialog

Um AlertDialog é um Dialog padrão do Android para o envio de alertas para o usuário. Diferentemente do Toast, esse componente é capaz de criar um Dialog que pode conter de zero a três botões, agregados ou não a uma lista de itens selecionáveis, permitindo ao desenvolvedor prover, num único Dialog, as mais diversas opções de interação para o usuário. Esse é o Dialog mais flexível entre os padrões utilizados pelo Android e, por esse motivo, é o mais utilizado pelos desenvolvedores. Graças a sua capacidade de conter um número diverso de botões ou até mesmo uma lista de opções, essa componente supre a maioria das necessidades dos desenvolvedores.

Como qualquer outro Dialog, o AlertDialog é uma extensão da classe Dialog. Ele é capaz de mostrar ao usuário diversas interfaces, como uma mensagem de texto, uma lista de opções (que pode ser formada de *checkboxes* ou *radio buttons*), botões para permitir a interação do usuário, e também pode conter um título. É importante notar também que esses Dialogs tem a opção de ser cancelável ou não. Ser cancelável é dar a possibilidade ao usuário de sair do Dialog através da tecla voltar sem necessariamente escolher uma das respostas.

Para criar um AlertDialog, precisamos, inicialmente, pegar uma instância do Builder dessa classe. Em seguida, escolhemos as propriedades que queremos configurar para aquele AlertDialog e, por fim, mandamos o sistema criar o Dialog baseado no Builder que foi configurado. Isso tudo geralmente é feito diretamente no código Java da Activity onde será exibido o Dialog. Vamos criar um novo projeto Android para testar esse componente. A **Listagem 11** mostra o código utilizado para criar um AlertDialog como o vista na **Figura 3**.

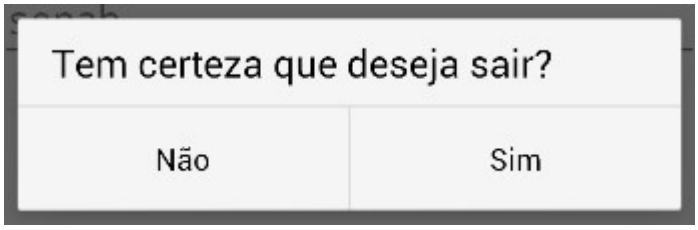
```
public class MainActivity extends Activity {

@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setMessage("Tem certeza que deseja sair?")
         .setCancelable(false)
         .setPositiveButton("Sim", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
            MainActivity.this.finish();
            }
         })
         .setNegativeButton("Não", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
               dialog.cancel();
            }
         });
      AlertDialog alert = builder.create();
   alert.show();
}
```

Listagem 11 - Criando uma AlertDialog

Figura 03 - AlertDialog com um texto e dois botões



Na **Listagem 11**, vemos alguns métodos que podem ser utilizados para a criação do AlertDialog. Mais uma vez, lembramos que o autocompletar do Android Studio pode mostrar todos. Nesse exemplo, definimos primeiramente uma mensagem, através do método *setMessage()*, do Builder. A propriedade *cancelable* é configurada para falso, indicando que o Dialog não poderá ser cancelado pelo usuário, utilizando o botão *back* do aparelho.

Na sequência, definimos a configuração do botão positivo, que indicará a confirmação do Dialog. No caso desse Dialog, o texto do botão positivo é configurado para “Sim” e o comportamento do clique é definido como *MainActivity.this.finish()*, indicando que a Activity deve ser finalizada. Já o botão negativo, configurado pelo *setNegativeButton()*, é configurado com a palavra “Não” e

o comportamento indica o fechamento da Dialog. Por fim, instanciamos um objeto AlertDialog e mandamos o Builder criar o Dialog, salvando-a nesse objeto e então utilizando o método *show()* para exibir o Dialog para o usuário.

Essa configuração de Dialog é apenas uma das várias possíveis para os AlertDialogs, nesse caso, criada com dois botões e uma mensagem. Para criar outras configurações, o Builder pode ser configurado utilizando outros métodos. Vimos aqui apenas como funciona o conceito para a criação desses Dialogs.

ProgressDialog

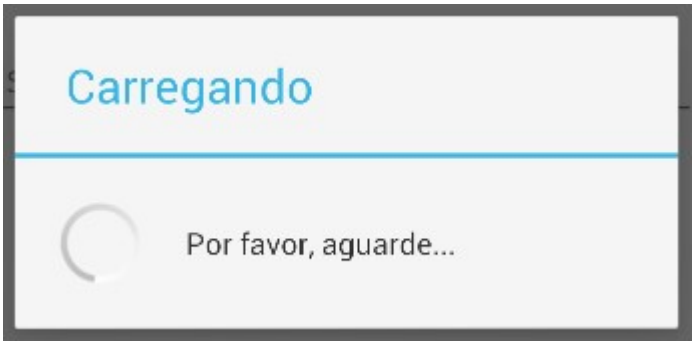
O segundo tipo de Dialog que iremos estudar é o ProgressDialog. Esse tipo de Dialog, que extende AlertDialog, permite ao desenvolvedor criar uma Dialog de carregamento, utilizando uma barra de progresso ou uma roda giratória para indicar processamento. Por ser uma extensão de AlertDialog, o ProgressDialog também pode ter os componentes do anterior.

A criação de um ProgressDialog é bastante simples. A **Listagem 12** exibe o código necessário para a criação do ProgressDialog que pode ser visto na **Figura 4**. Você pode inseri-lo no método onCreate para ver o seu funcionamento!

```
1 // progressDialog de processamento indefinido
2 ProgressDialog dialog = ProgressDialog.show(this, "Carregando", "Por favor, aguarde...", true);
```

Listagem 12 - Criação de um ProgressDialog de progresso indefinido

Figura 04 - ProgressDialog criado na Listagem 12



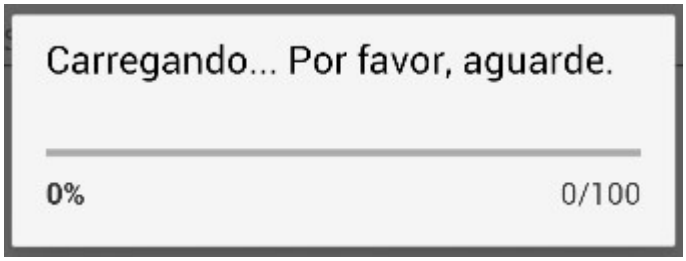
Como vimos na **Listagem 12**, para criar um ProgressDialog de progresso indefinido, como mostrado na **Figura 4**, precisamos apenas de um método. Esse método possui 4 parâmetros. O primeiro deles é o contexto sobre o qual o Dialog deve ser criada. Normalmente pode-se utilizar referência a *Activity*, com o *this*, ou utilizar o método *getApplicationContext()* para obter o contexto dentro de uma *Activity*. O segundo parâmetro é o título do Dialog e o terceiro é o texto que será exibido em seu corpo. Por fim, a variável booleana indica se o progresso é indefinido ou não.

Caso o progresso relacionado ao Dialog seja definido, como por exemplo, o download de um arquivo de tamanho conhecido, podemos utilizar um ProgressDialog com uma barra de progresso indicando o passo do processamento que está acontecendo. Vejamos na **Listagem 13** um exemplo de como implementar o Dialog mostrado na **Figura 5**.

```
1  ProgressDialog progressDialog;
2      progressDialog = new ProgressDialog(this);
3      progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
4      progressDialog.setMessage("Carregando... Por favor, aguarde.");
5      progressDialog.setCancelable(false);
6      progressDialog.show();
```

Listagem 13 - Implementação de uma ProgressDialog de progresso definido

Figura 05 - ProgressDialog com barra de carregamento



O grande ponto em relação a esses Dialogs com barras de carregamento é como atualizar o progresso da barra. O método *setProgress()* é responsável por alterar a porcentagem que está sendo exibida no Dialog. É através desse método que o desenvolvedor deve atualizar o progresso para informar corretamente o usuário. Normalmente, utiliza-se uma *thread* separada, que comunica o progresso a *thread* principal, utilizando um *handler* para passar as mensagens. Esses conceitos são um pouco avançados para o nosso curso, mas alguns tutoriais sobre *thread* estão disponíveis na página do Android Developers para aqueles que queiram se aprofundar um pouco mais no assunto.

Conclusão

Encerramos assim a nossa segunda aula sobre interface gráfica, mas ainda veremos mais alguns importantes componentes gráficos na aula 7, possibilitando o desenvolvimento de aplicações cada vez mais robustas. Até lá, amigos!

Leitura complementar

ANDROID Design. Disponível em: <http://developer.android.com/design/index.html>. Acesso em: 04 dez. 2015.

Resumo

Hoje, colocamos a mão na massa e implementamos diferentes componentes de uma interface gráfica. Estudamos como funcionam os Buttons, TextViews, ImageViews, EditTexts e relembramos como implementar um RelativeLayout, como tínhamos visto na aula passada sobre interfaces gráficas. Esses estudos foram feitos através do desenvolvimento de uma tela de login. Em seguida, estudamos outros componentes importantes para exibição de mensagens na tela, o Toast e os Dialogs. Esses componentes são utilizados em diversas aplicações Android e servem como base para o estudo de diversos outros que, infelizmente, não teremos tempo de ver durante o curso.

Autoavaliação

1. Queremos criar um novo layout onde possamos posicionar os componentes um em relação ao outro. Qual layout deveremos utilizar? E se quisermos os componentes em sequência, um abaixo do outro?

2. Suponha que nesse primeiro layout queiramos criar um título, posicionado no centro do layout, alinhado ao topo e com o texto "Editar Informações". Quais atributos deverão ser configurados e qual componente deverá ser utilizado?

3. Agora queremos adicionar a esse Layout dois campos de texto, um com apenas uma linha para se colocar o título e outro com múltiplas linhas, mas não menos que 3 e não mais que 7. Quais os componentes que deveremos utilizar e quais atributos configurar?

4. Por fim, precisamos de botões nesse componente. Um botão, alinhado a esquerda e embaixo, deve confirmar o envio, mostrando ao usuário um Toast escrito "Sucesso". O outro botão, posicionado ao lado do anterior, deve limpar os dois campos. Qual o método que poderá responder ao clique do usuário em um desses botões? Quais as propriedades de layout necessárias para o posicionamento desses botões como citado?

Referências

ANDROID Developers. 2012. Disponível em: <http://developer.android.com>. Acesso em: 6 mai. 2015.

DIMARZIO, J. **Android:** a programmer's guide. São Paulo:McGraw-Hill, 2008. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 6 maio 2015.

HASEMAN, C. **Android essentials**. Berkeley, CA. USA: Apress, 2008.

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

_____. **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: 6 maio 2012.