Dispositivos Móveis

Aula 07 - Interfaces Gráficas III

Apresentação

Olá, pessoal! Continuaremos agora nossos estudos sobre interfaces gráficas no Android. Na aula de hoje, estudaremos novos elementos das interfaces gráficas Android: ListView, Menus e Fragments. Com os ListViews poderemos criar listas de itens a serem exibidos na tela. Na parte de menus, veremos alguns tipos que o Android disponibiliza, o options menu, o context menu e o menu na ActionBar. Já em relação aos Fragments, veremos como criar layouts subdivididos e que possam ser reutilizados por várias Activities.

Objetivos

Ao final desta aula, você será capaz de:

- Utilizar Menus diversos em suas aplicações.
- Implementar ListViews como forma de exibir vários itens ao usuário.
- Utilizar Fragments, dividindo o layout de uma tela

ListView

O ListView é o componente responsável por prover ao usuário listas de dados, organizadas a partir de listas presentes no código da aplicação. O ListView carrega os dados através de um Adapter, que pode ser implementado por completo, em uma nova classe, ou apenas utilizar uma das interfaces já implementadas. Nessa aula, utilizaremos a interface já implementada ArrayAdapter, que é capaz de carregar itens a partir de um array.

Outro ponto importante dos ListViews é que, além do próprio componente ListView, um layout deve ser definido para os itens que estarão presentes naquela lista, ou seja, o layout de cada linha que será exibida. É importante notar que no ArrayAdapter, a interface passada deve conter apenas um ListView onde o ArrayAdapter irá colocar as informações. Vejamos as **Listagens 1** e **2**, que indicam a declaração do ListView e do list_item.xml, respectivamente.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent"
5    android:orientation="vertical" >
6
7  <ListView
8    android:id="@+id/listView1"
9    android:layout_width="match_parent"
10   android:layout_height="wrap_content" >
11 </ListView>
12
13 </LinearLayout>
```

Listagem 1 – Declaração do arquivo activity_main.xml, contendo o ListView.

```
1  <TextView xmlns:android="http://schemas.android.com/apk/res/android"
2    android:id="@+id/listView1"
3    android:layout_width="fill_parent"
4    android:layout_height="wrap_content"
5    android:padding="10dip"
6    android:textSize="15dip"/>
```

Listagem 2 – Declaração do arquivo list_item.xml, contendo o formato dos itens do ListView.

Com os dois XMLs declarados podemos, então, criar o Adapter e colocá-lo como responsável pelo layout dos itens da lista. Para fazermos isso, devemos fazer como mostrado na **Listagem 3**. Perceba que os dados carregados nesse ListView serão carregados a partir de uma lista constante, definida no início do arquivo.

```
1  package br.ufrn.imd.aula07;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.AdapterView;
7  import android.widget.ArrayAdapter;
8  import android.widget.ListView;
9  import android.widget.TextView;
10 import android.widget.Toast;
11
12 public class MainActivity extends Activity {
13    static final String[ ] PAISES = new String[ ] {
14          "Argentina", "Brasil", "França", "Itália", "Alemanha", "Bélgica", "Inglaterra", "Nova Zelândia"};
15    @Override
16    public void onCreate(Bundle savedInstanceState) {
17       super.onCreate(savedInstanceState);
18       setContentView(R.layout.activity_main);
19       ListView lv = (ListView) findViewById(R.id.listView1);
20       ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.list_item, PAISES);
21       lv.setAdapter(adapter);
22
23       lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
24          @Override
25          public void onItemClick(AdapterView<?> pai, View view, int posicao, long id) {
26             Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
27                   Toast.LENGTH_SHORT).show();
28          }
29       });
30    }
31 }
```

Listagem 3 – Activity implementando a ListView

Perceba também que na **Listagem 3** há a declaração de um comportamento para o ItemClickListener. Ao definir esse método, o ListView passa a ter um comportamento para responder a cliques em seus itens. Funciona como se cada item fosse um botão e, ao ser tocado, invoca esse método. É importante saber que é possível obter qual item da lista foi tocado (através do parâmetro posição) e interagir com a View que foi tocada (através do view). Isso pode ser utilizado, como no exemplo, para obter alguma informação do item quando o mesmo é tocado. No

exemplo, um Toast é exibido com o texto referente ao elemento da linha tocada. Na **Figura 1**, podemos ver o resultado dos códigos exemplo e também um Toast sendo exibido.

Figura 01 - ListView dos países

Menus no Android

É padrão nos aparelhos rodando versões anteriores ao Android 3.0 a presença de uma tecla Menu. Ao pressionar essa tecla, os usuários Android ativam o menu da aplicação. Esse menu contém funções de configuração ou mesmo novas opções de utilização da aplicação. Em versões posteriores à versão 3.0, incluindo esta, as aplicações ganharam uma barra de ações que corresponde a esse menu. Estudaremos agora quais as opções que temos para agregar esses menus às nossas aplicações.

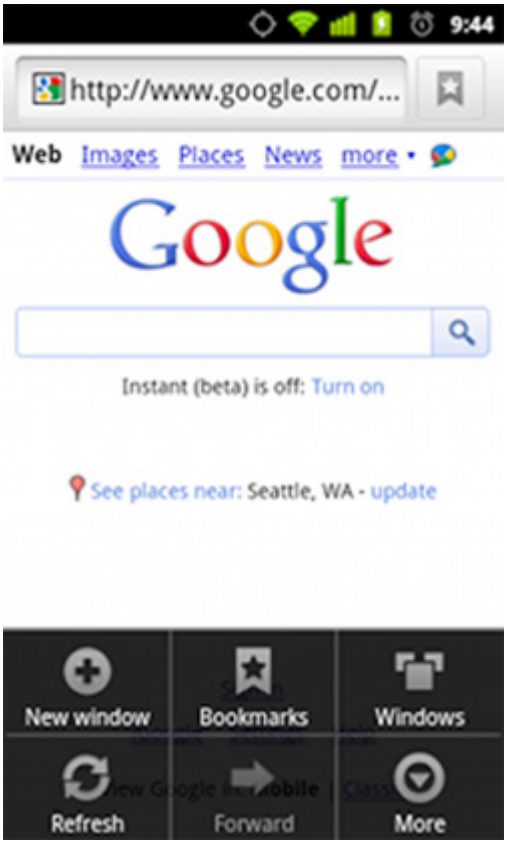
Options Menu

O menu de opções é o modelo principal de menus no Android. Cada menu desses está relacionado a uma Activity. É nesse menu que as ações que afetam toda a aplicação devem ser colocadas, como por exemplo, "escrever e-mail" em uma

aplicação de gerenciamento de e-mails. A qualquer momento você pode querer escrever um novo e-mail e essa opção pode ser invocada rapidamente, através da tecla de menu ou da opção equivalente na barra de ações do aplicativo.

Em aplicações voltadas a versão 2.3 ou anterior, ao pressionar a tecla de menu, o usuário terá o menu lançado em frente a Activity ativa, na parte inferior do aparelho. Esse menu tem capacidade de exibir até seis itens, como é visto na **Figura 2**.

Figura 02 - Menu em versões anteriores a versão 2.3.



Fonte: Android Developers

Caso mais de seis itens estejam alocados no menu, o sexto item do menu será substituído por um item padrão de *Mais,* que indica o acesso a outros elementos. Ao ser acionado, esse item lança uma lista de opções que flutua sobre a aplicação e contém todas as opções do menu. Em versões a partir da 3.0, esse menu foi substituído por um semelhante, adicionado à ActionBar. Veremos esse menu adiante.

Para adicionar um Option Menu à sua Activity, deve-se sobrescrever o método onCreateOptionsMenu(). Esse callback é invocado pelo Android quando o usuário pressiona a tecla de menu, ou toca na opção equivalente, requisitando assim a exibição do mesmo. A **Listagem 4** mostra como deve ser implementado esse método. A **Listagem 5** descreve o XML utilizado.

Para criar o diretório **menu** na pasta **res**, deve-se clicar com o botão direito na pasta **res** e utilizar a opção New -> Android resource directory, selecionando a opção **Resource Type** como **Menu** na janela que for aberta.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}
```

Listagem 4 – Implementação do onCreateOptionsMenu.

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="Configurações" android:id="@+id/config"></item>
</menu>
```

Listagem 5 – Arquivo menu.xml, localizado na pasta res/menu

O menu implementado acima inclui apenas um item, como pode ser visto no xml da **Listagem 5**. Esse item é responsável por carregar as opções de configuração do aplicativo. Veja que existem diversas opções de personalização nos itens do menu. Do mesmo jeito que nos layouts, essas opções devem ser declaradas dentro do espaço do <item>. Todas as opções para essa parte do menu podem ser vistas utilizando o recurso auto completar do Android Studio. Porém, como declarado, o programa ainda não tem ideia do funcionamento do menu e tratamento dos cliques, apenas de sua existência. Para garantir o correto funcionamento do menu, devemos implementar o método onOptionsItemSelected(), como visto na **Listagem 6**.

```
1  public boolean onOptionsItemSelected(MenuItem item) {
2     switch (item.getItemId()) {
3        case R.id.config:
4           Intent intent = new Intent(getApplicationContext(),
5           AplicationPreferences.class);
6           startActivity(intent);
7           break;
8        default:
9           return super.onOptionsItemSelected(item);
10       }
11       return true;
12 }
```

Listagem 6 – Implementação do onOptionsItemSelected()

A função onOptionsItemSelected() identifica o item que foi selecionado através de seu parâmetro e é através dele que devemos configurar as ações a serem tomadas, de acordo com o que desejarmos. No exemplo, uma nova Activity é lançada para permitir ao usuário configurar as opções que desejar. Lembre-se de criar essa Activity chamada AplicationPreferences no seu projeto, caso esteja seguindo o exemplo.

Um último ponto a ser conhecido em relação ao menu de opções é a alteração deste em tempo de execução. Durante a execução de sua aplicação pode surgir a necessidade de alterar os itens do menu. Porém, uma vez criado, o menu não é criado novamente ao ser reinicializado, ou seja, não é possível colocar essas mudanças no onCreateOptionsMenu(). Ao invés disso, a função onPrepareOptionsMenu() deve ser utilizada. Essa função é chamada toda vez que um menu precise ser alterado.

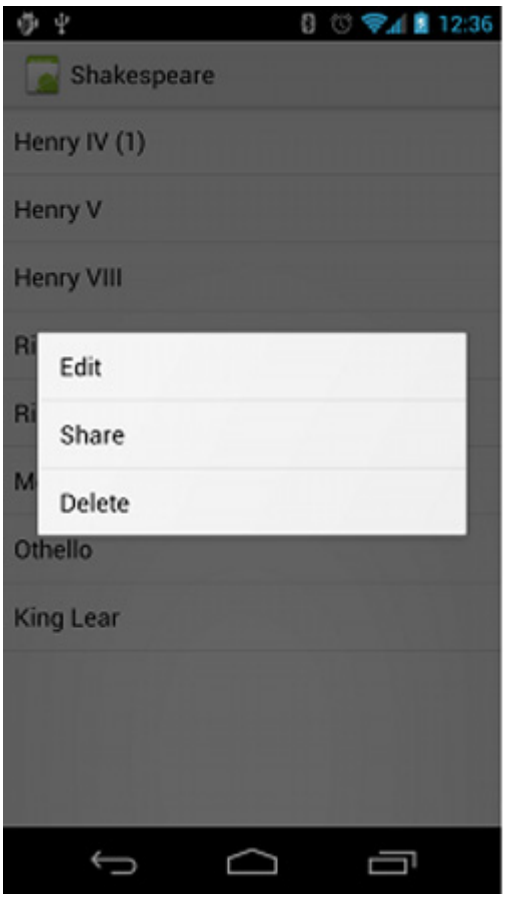
Os Option Menus são bastante utilizados em todos os tipos de aplicação, desde jogos até aplicações de gerência de informações. Por ser um comportamento padrão, espera-se dos usuários Android buscar a tecla menu ou o ícone equivalente sempre que sentem a necessidade de alteração de uma informação porém não encontram a opção em tela.

Vídeo 01 - Android 3.0: Antes e Depois

Context Menus

Os Context Menus, como o nome sugere, são menus que estão ligados a um contexto. No caso do Android, esse contexto é representado por uma View. Esses menus são bastante parecidos com menus de contexto utilizados em computadores, ao pressionar o botão direito sobre algum componente, por exemplo. No Android, esses menus costumam ser ativados através de um toque longo em cima de uma View. Ao serem lançados, os menus de contexto passam a flutuar sobre a Activity, como vemos na **Figura 3**, permitindo que o usuário escolha uma das opções disponíveis para aquele elemento.

Figura 03 - Menu de Contexto.



Fonte: Android Developers

É importante notar que, mais uma vez, há uma diferenciação entre os aparelhos na versão anterior a 3.0 e nos que tem essa versão ou superior. No caso dos aparelhos 3.0, que possuem a ActionBar substituindo o menu de opções, como vimos anteriormente, uma barra de ações contextualizadas com o componente ativo são a maneira de utilizar o menu de contexto. Ainda assim, é possível utilizar o menu

flutuante nesses aparelhos, já que os menus são invocados com toques na tela e não dependem de teclas. Para criarmos um menu de contexto devemos seguir alguns passos:

1. Registrar a View como parte do contexto do menu utilizando o método *registerForContextMenu()*, passando a View como parâmetro. Desse modo, ao ser tocada, a View torna-se capaz de invocar o menu de contexto.

2. Redefinir, na Activity, o método *onCreateContextMenu()*, responsável pela criação do menu de contexto. Ao redefinir esse método, a Activity passa a ser capaz de criar um menu de contexto ao ser requisitada pelo Android. Diferentemente do menu de opções, esse método é sempre chamado antes da criação do menu flutuante, seja a primeira vez ou não.

3. Redefinir o método *onContextItemSelected(),* responsável por passar para a Activity qual dos itens do menu foi selecionado. É nesse método que devem ser implementados os comportamentos do menu.

Vejamos agora duas listagens contendo exemplos de como fazer cada um desses passos. No caso dos Context Menus, o XML é implementado da mesma maneira do Options Menu, então, confira a **Listagem 5** para relembrar como deve-se declarar o XML para um Menu. Na **Listagem 7** vemos um exemplo de como carregar o menu de contexto, através da função onCreateContextMenu(), e, na **Listagem 8**, vemos como tratar os eventos de seleção nesse menu.

```
1 @Override
2 public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
3    super.onCreateContextMenu(menu, v, menuInfo);
4    MenuInflater inflater = getMenuInflater();
5    inflater.inflate(R.menu.menu, menu);
6 }
```

Listagem 7 – Redefinição do método *onCreateContextMenu()*

```
public boolean onContextItemSelected(MenuItem item) {
   // Handle item selection
   switch (item.getItemId()) {
      case R.id.config:
         Intent intent = new Intent(getApplicationContext(),
         AplicationPreferences.class);
         startActivity(intent);
      return true;

      default:
         return super.onOptionsItemSelected(item);
   }
}
```

Listagem 8 – Redefinição do método *onContextItemSelected().*

A utilização dessas funções, como podemos ver, é bem similar a das funções utilizadas no Options Menu. Da mesma maneira que função anterior, é importante perceber que, no caso da função onCreateContextMenu(), o método super deve ser chamado logo no início da função, para permitir ao Android fazer os ajustes necessários para a exibição do menu. Já no método onContextItemSelected(), o retorno é o caso padrão, que indica a função que o evento que lhe foi passado não pertence a ela, por não ter sido consumido por nenhum dos casos anteriores.

Outro ponto importante, principalmente para aplicações que possuem muitos itens a serem colocados nos menus, é a possibilidade da utilização de submenus. No Android, não é possível aninhar os menus, ou seja, ir exibindo-os lado a lado, após seleção. As razões para isso é o espaço limitado das telas dos smartphones. A maneira encontrada é a utilização de grupos, que são carregados a medida que suas opções são selecionadas em menus anteriores. A **Listagem 9** contém o exemplo de um submenu, declarado dentro de uma opção do menu pai.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/file"
      android:title="@string/file" >
      <!-- "file" submenu -->
      <menu>
         <item android:id="@+id/create_new"
         android:title="@string/create_new" />
         <item android:id="@+id/open"
         android:title="@string/open" />
      </menu>
   </item>
</menu>
```

Listagem 9 – Submenu referente à opção File.

Com a utilização de submenus, é possível incluir opções dentro de outras, evitando assim menus muito grandes, com opções que nem sempre são pertinentes. Sendo capaz de expandir opções referentes a um item, após ele ter sido selecionado, tornando os menus bem mais compactos e eficientes para os usuários que o estão utilizando.

Vídeo 02 - Menus Avançados

ActionBar e Menus

Da versão 3.0 em diante, é mais comum a utilização de Action Bar para acesso rápido a funções e também ao menu. A **Figura 4** mostra uma Action Bar e seus elementos.

Figura 04 - Action Bar com os elementos: Nome do aplicativo [1], itens de ações [2] e menu da aplicação [3].

A Action Bar é um recurso que identifica a navegação e ações do usuário pela aplicação. A Action Bar oferece aos usuários uma interface similar em todos os aplicativos Android, sendo adaptadas para diferentes configurações de tela. Para criarmos um menu numa Action Bar, primeiro devemos ativá-la na Activity em questão e depois definir os itens do menu, de forma semelhante ao OptionMenu.

Se estamos utilizando uma versão da 3.0 em diante, devemos utilizar algum dos temas Holo para a nossa aplicação, assim, a ActionBar já será ativada automaticamente. Se formos utilizar uma versão inferior, precisamos definir que o

nosso projeto seja compatível, através da adição da biblioteca *appcompat v7*. Em seguida, criamos uma Activity estendendo a ActionBarActivity, e no AndroidManinfest devemos dizer que a nossa Activity deve usar algum tema do pacote Theme.AppCompat, como mostra a **Listagem 10**.

```
<activity
   ......
   android:theme="@style/Theme.AppCompat.Light"
   ......
</activity>
```

Listagem 10 - Definição do tema para uma Activity no AndroidManinfest

Ao ativar a ActionBar, por padrão já serão exibidos o nome do aplicativo a esquerda e o menu a direita. Para alterarmos os itens do menu, vamos criar um arquivo de layout chamado menu.xml, com as opções que deverão aparecer no menu, da mesma forma que vimos no Option Menu. Esse arquivo deve ser criado na pasta res -> menu. Caso seu projeto ainda não tenha essa pasta, adicione-a, clicando com botão direito na pasta res, escolhendo a opção New -> Android Resource Directory e então selecionando o resource type como menu. A **Listagem 11** mostra o código do menu.xml, parte dessa pasta. Nele teremos uma opção chamada Configurações e outra chamada Ajuda.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:MyApplication="http://schemas.android.com/apk/res-auto">
   <item
      android:id="@+id/action_configuracoes"
      android:orderInCategory="100"
      MyApplication:showAsAction="never"
      android:title="configurações"/>

   <item
      android:id="@+id/action_ajuda"
      android:orderInCategory="100"
      MyApplication:showAsAction="never"
      android:title="Ajuda"/>
</menu>
```

Listagem 11 – Código do layout menu.xml

Perceba que a propriedade showAsAction é adicionada utilizando o nome de sua aplicação. Lembre-se de alterar quando for utilizar o exemplo!

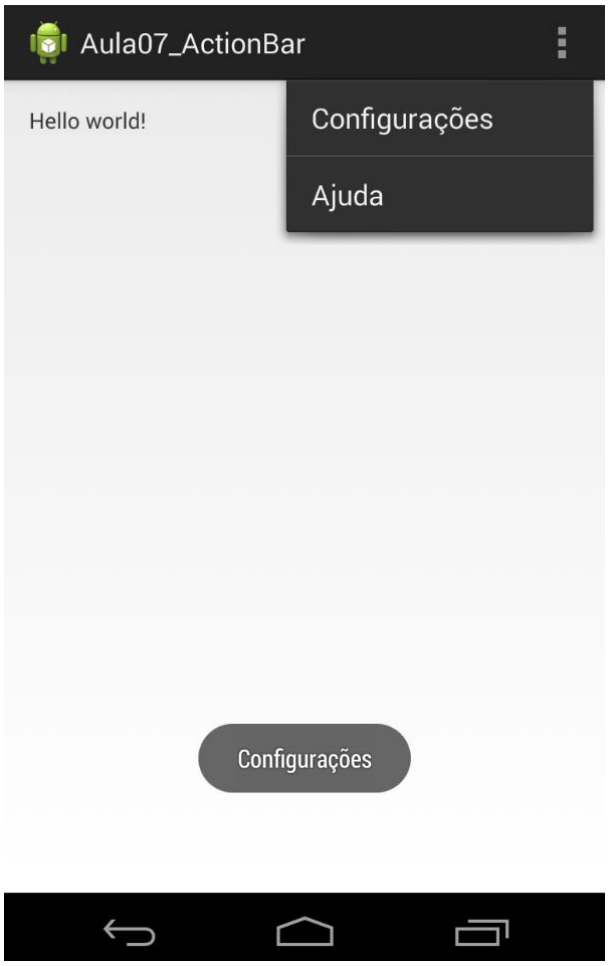
Na nossa Activity também devemos definir os métodos onCreateOptionMenu() e onOptionsMenuSelected(), também da mesma forma que no exemplo do Option Menu. A **Listagem 12** mostra o código desses dois métodos. Ao clicar em alguma das opções do menu, será exibido um Toast com o texto da opção selecionada. A **Figura 5** mostra o exemplo sendo executado.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_configuracoes) {
        Toast.makeText(getApplicationContext(), "Configurações", Toast.LENGTH_LONG).show();
        return true;
    }
    if (id == R.id.action_ajuda) {
        Toast.makeText(getApplicationContext(), "Ajuda", Toast.LENGTH_LONG).show();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Listagem 12 – Activity implementando um menu na ActionBar

Figura 05 - Exemplo de utilização de menu na ActionBar.



Atividade 01

1. Descreva a utilização de um Options Menu, citando as funções importantes a sua implementação.
2. Crie um Context Menu para, a View Cores, em que as opções são Azul, Verde e Amarelo. Caso azul seja escolhido, exiba as opções Azul Claro e Azul Escuro.

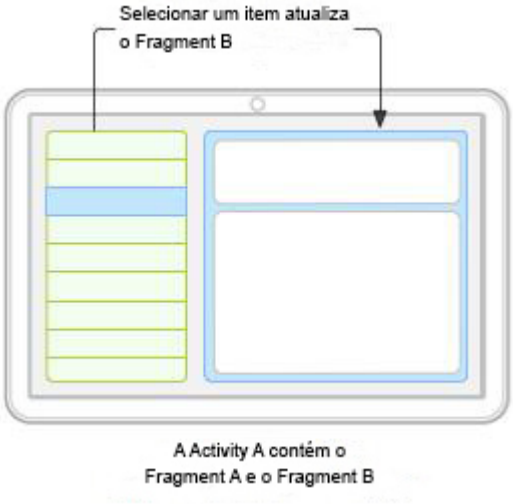
Fragments

Os Fragments representam uma parte da interface com o usuário numa Activity. Podemos combinar vários Fragments numa única Activity para criar uma interface com múltiplos painéis ou reutilizar um Fragment noutras Activities. Você pode

pensar num Fragment como uma seção de uma Activity, que tem seu próprio ciclo de vida, recebe e trata seus próprios eventos de entrada e interação, e que podemos adicioná-lo ou removê-lo enquanto a Activity está sendo executada, como se fosse uma Sub-Activity.

Fragments são bastante utilizados para a criação de interfaces mais robustas, que possam ser divididas em vários grupos pela tela. A utilização destes é muito comum em tablets, onde temos menus ao lado esquerdo e painéis de detalhamento ao lado direito. Por exemplo, se selecionamos um menu diferente no painel esquerdo, o painel direito deve mostrar outro layout; para que não tenhamos que ficar sempre trocando as Activities, podemos usar um Fragment que será o painel esquerdo, e outros Fragments para o painel direito, sendo alterados de acordo com a seleção do painel esquerdo. A **Figura 6** ilustra o uso de Fragments.

Figura 06 - Uma Activity que contém dois Fragments, um na esquerda, outro na direita.



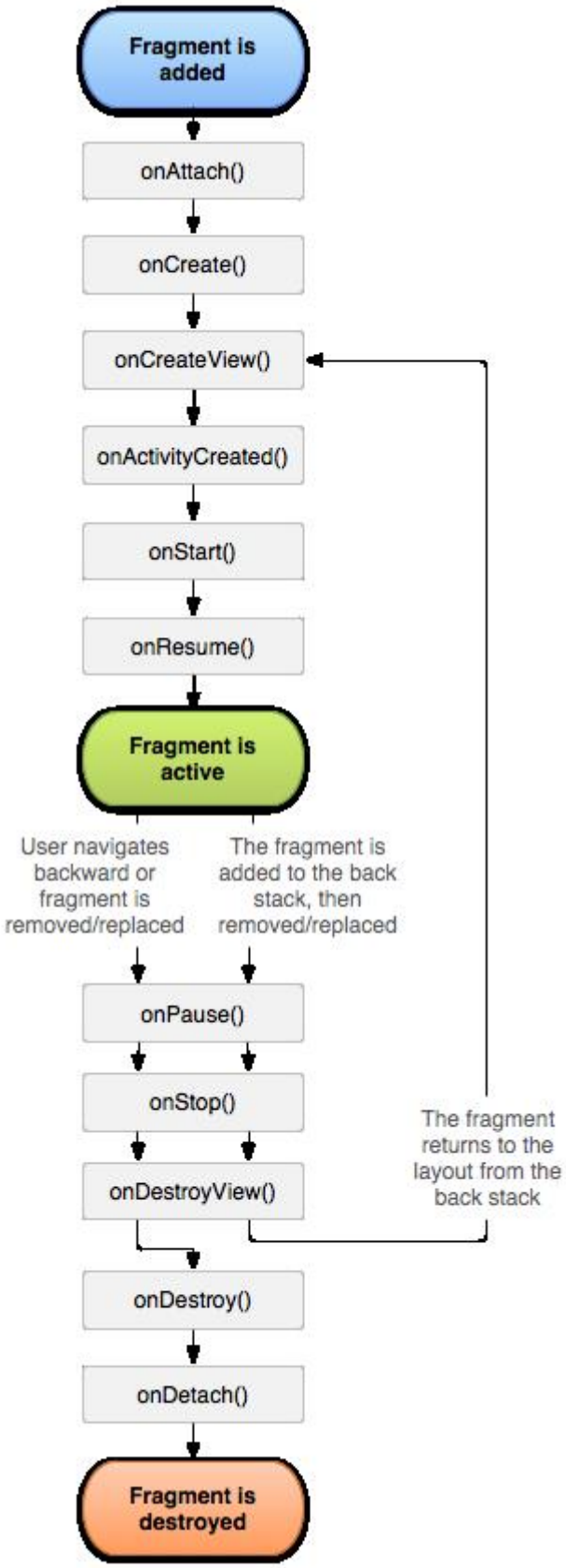
Um Fragment sempre deve ser incorporado numa Activity e seu ciclo de vida é diretamente afetado pelo ciclo de vida da Activity host. Por exemplo, quando a Activity é interrompida ou destruída, todos os Fragments inseridos também o são. No entanto, enquanto uma Activity está sendo executada, podemos manipular cada Fragment de forma independente. Eles funcionam de forma semelhante às Activities, sendo que também possuem uma pilha de execução, permitindo que possamos avançar, abrindo um novo Fragment, ou voltar para o Fragment exibido anteriormente.

Quando adicionamos um Fragment como parte de um layout, ele funciona como um ViewGroup dentro da Activity, definindo também seu próprio layout de exibição. Podemos inserir um Fragment num layout declarando-o com um elemento < *fragment* > ou também através do código Java. No entanto, um Fragment não precisa necessariamente fazer parte do layout da Activity. Podemos usá-lo sem uma interface própria, sendo um componente invisível, mas que trabalha dentro da Activity.

Ciclo de Vida dos Fragments

Para usar Fragments, devemos criar uma subclasse da classe Fragment, ou de uma das subclasses que a estendem, como DialogFragment, ListFragment, PreferenceFragment, etc. A classe Fragment é bem parecida com uma Activity, contendo callbacks executados quando um Fragment passa de um estado para outro, como onCreate(), onStart(), onPause() e onStop(), entre outros. Os Fragments possuem um ciclo de vida próprio, mas diferente do ciclo de vida das Activities, como é mostrado na **Figura 7**.

Figura 07 - Ciclo de vida dos Fragments.



Normalmente, devemos implementar três desses métodos para cada Fragment: onCreate(), onCreateView(), onPause(). Os outros métodos também podem ser implementados normalmente, como nas Activities.

- **onCreate()** - o sistema chama este método ao criar o Fragment. Dentro desse método devemos inicializar componentes essenciais ao Fragment e que desejamos manter quando o Fragment for pausado ou parado e, em seguida, reiniciado.

- **onCreateView()** - o sistema chama este método para que o Fragment desenhe sua interface. Para desenhar uma interface, devemos retornar uma View deste método. Podemos retornar null se o fragmento não fornece uma interface.

- **onPause()** - O sistema chama esse método como a primeira indicação de que o usuário está saindo do fragmento (embora isso nem sempre significa que o fragmento está sendo destruído, podendo ser reiniciado). É neste método onde devemos salvar quaisquer propriedades que devem ser mantidas pela aplicação, pois o usuário pode não voltar e o Fragment não será reiniciado.

- **onAttach()** - Chamado quando o Fragment é associado à Activity.

- **onActivityCreated()** - Chamado quando o método onCreate() da Activity é finalizado indicando que a Activity foi criada.

- **onDestroyView()** - Chamado quando a hierarquia de exibição associada com o Fragment está sendo removida.

- **onDetach()** - Chamado quando o Fragment é desassociado da Activity.

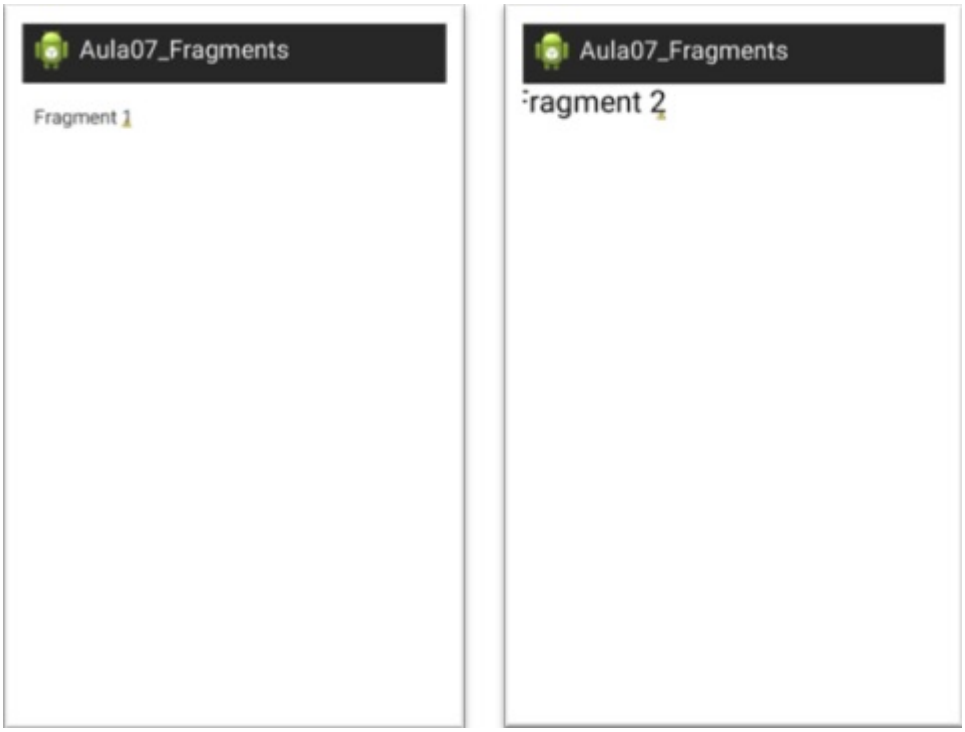
- **onStart(), onResume(), onPause() e onStop()** funcionam da mesma forma que numa Activity.

Exemplo com Fragments

Para vermos o funcionamento de Fragments dentro de uma Activity, vamos criar um novo projeto contendo uma Activity em branco. Em seguida, vamos modificar o layout dessa Activity para conter um FrameLayout e um botão. A ideia é que o

FrameLayout seja o container dos nossos Fragments e que eles sejam trocados através de um clique no botão. A **Figura 8** mostra o layout da nossa Activity principal e a **Listagem 13** contém o código do arquivo content_main.xml.

Figura 08 - Layout principal contendo o botão e os Fragments.




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

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Mudar Fragment"
   android:id="@+id/button"
   android:layout_gravity="center_horizontal|bottom" />
</FrameLayout>
```

Listagem 13 – Listagem referente ao arquivo content_main.xml.

Em seguida, devemos criar mais dois arquivos contendo o layout dos Fragments. Para diferenciar bem os Fragments, vamos adicionar uma cor de fundo em um deles e deixar o padrão branco no outro. As **Listagens 14** e **15** trazem os códigos dos arquivos fragment1.xml e fragment2.xml, respectivamente.

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/article"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:textSize="18sp"
    android:background="@color/colorPrimary"
    android:text="Fragment 1"/>
```

Listagem 14 – Código do arquivo fragment1.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/article2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:textSize="18sp"
    android:text="Fragment 2"/>
```

Listagem 15 – Código do arquivo fragment2.xml

Com os layouts todos criados, podemos criar agora as nossas classes. Primeiramente, vamos criar a TelaFragment e a TelaFragment2, responsáveis por carregar os layouts definidos acima. O código de ambas é muito similar, alterando-se apenas o layout que devem inflar e o nome da classe. O código da TelaFragment pode ser visto na **Listagem 16**. O do TelaFragment2 fica sob sua responsabilidade! É importante notar apenas que ambas as classes devem estender de Fragment, uma vez que representam elementos desse tipo. O método a ser implementado para inflar o layout respectivo é o onCreateView, que é chamado ao se instanciar o Fragment, como visto anteriormente.

```
public class TelaFragment extends Fragment {

    public TelaFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

Listagem 16 – Código referente ao arquivo TelaFragment.java

Por fim, precisamos apenas criar a nossa Activity principal, implementando o clique do botão para fazer a troca entre Fragments. Essa troca deve ser feita utilizando o FragmentManager, que pode ser obtido através do método getFragmentManager(). A Activity deve estender a classe FragmentActivity. O código da Activity principal está descrito na **Listagem 17**.

```
1  public class MainActivity extends FragmentActivity {
2
3     int i = 1;
4
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7        super.onCreate(savedInstanceState);
8        setContentView(R.layout.content_main);
9
10       if (savedInstanceState == null) {
11          getFragmentManager().beginTransaction()
12          .add(R.id.fragment_container, new TelaFragment())
13          .commit();
14       }
15
16       Button b = (Button) findViewById(R.id.button);
17       b.setOnClickListener(new View.OnClickListener() {
18
19          @Override
20          public void onClick(View v) {
21             if (i == 0) {
22                getFragmentManager().beginTransaction()
23                .replace(R.id.fragment_container, new TelaFragment())
24                .commit();
25                i = 1;
26             } else {
27                getFragmentManager().beginTransaction()
28                .replace(R.id.fragment_container, new TelaFragment2())
29                .commit();
30                i = 0;
31             }
32          }
33       });
34    }
35 }
```

Listagem 17 – Código da classe MainActivity.java.

A utilização de Fragments pode trazer diversos benefícios à sua aplicação. O conceito pode não ser tão trivial, mas é bem importante! Para facilitar o acompanhamento, caso tenha tido algum problema, baixe aqui o projeto completo e faça os seus testes!

Bons estudos e até a próxima!

Atividade 02

1. Implemente um terceiro Fragment no exemplo da **Figura 8**.

Leitura Complementar

DEVELOPERS. Dialogs. Disponível em: <http://developer.android.com/guide/topics/ui/dialogs.html>. Acesso em: 25 abr. 2012.

_____. Menus. Disponível em: <http://developer.android.com/guide/topics/ui/menus.html>. Acesso em: 25 de abril de 2012.

_____. ListView. Disponível em: <http://developer.android.com/guide/components/fragments.html>. Acesso em: 25 de abril de 2012.

Resumo

Vimos nesta aula como podemos usar Listas, Menus e Fragments em nossas aplicações Android. Na primeira parte, vimos como criar uma lista com várias linhas, onde cada uma exibe uma informação diferente. Na parte de Menu, estudamos três tipos de menu. O primeiro, o Options Menu, que aparece na parte de baixo da aplicação e deve ser referente a aspectos gerais da aplicação. Esse menu foi substituído pela ActionBar em versões do Android posteriores a versão 3.0, sendo o nosso terceiro menu estudado. Já o Context Menu é responsável por fornecer um menu flutuante relacionado a uma View e deve ser ativado através da interação com essa View a que ele se refere. Na última parte da aula, estudamos os Fragments e como implementá-los, possibilitando-nos a criar interfaces subdivididas e reaproveitáveis de uma tela.

Autoavaliação

1. Utilizando a Activity de Login desenvolvida em aulas anteriores, crie um Options Menu que dê aos usuários a possibilidade de fechar a aplicação.

2. Dê um toque longo no EditView de Login. O que acontece? Como é conhecido esse tipo de menu? Descreva brevemente para que ele é utilizado.

3. Crie um ListView que exiba o nome de 10 países diferentes.

4. Qual a relação entre Fragments e os layouts para Tablets e Smartphones?

Referências

DEVELOPERS. **Fragments**. Disponível em: <http://developer.android.com/guide/components/fragments.html>. Acesso em: 30 setembro de 2014.

_____. **Menus**. Disponível em: <http://developer.android.com/guide/topics/ui/menus.html>. Acesso em: 30 setembro de 2014.

_____. **ListView e Adapters**. Disponível em: <http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews>. Acesso em: 30 setembro de 2014.