Android - Cards de Referência

Fragments

Bem vindo ao nosso décimo primeiro card sobre Android! :-) Agora, vamos conversar sobre os Fragments. Eles facilitam bastante o trabalho de construir aplicativos para dispositivos com diferentes tamanhos de telas.

FRAGMENTS

Inicialmente, é bom entender o que os Fragments NÃO são. Eles não são widgets de tela. Muito menos tipos de Activities. Também não são containers, como LinearLayout e RelativeLayout. Então, eles são o que? Fragments agrupam containers e Widgets e são ancorados em Activities. Uma Activity pode ter nenhum ou muitos Fragments. A principal razão de existir dos Fragments é a existência de diversos tipos e tamanhos de telas de dispositivos que rodam Android.

Como você faria um aplicativo que roda em tablets de 10.1" e Smartphones? Imagine que ele tem apenas uma tela, mas essa tela terá um layout diferente a depender do tipo de dispositivo. Se aberto em um tablet, a tela se arrumará para exibir os componentes da melhor forma possível na horizontal. Em smartphones, exibirá tudo bem bonito, mas na vertical. E agora?

A maioria das pessoas resolveria isto criando dois arquivos de Layout separados, um para cada tela. Colocaria um arquivo em **/res/layout/** e o outro em **/res/layout-xlarge/**. Depois disto, faria uma única classe de Activity que trataria os eventos oriundos de ambos os layouts. Mas, e se você quiser componentes a mais na tela do tablet? Afinal de contas, tem mais espaço, você pode adicionar funcionalidades que fazem sentido apenas no tablet.

Sua Activity vai começar a ficar complexa demais, certo? Começará a tratar diversas situações diferentes a depender do Layout usado. Um monte de IF pra lá e pra cá.

MINIMIZANDO O PROBLEMA

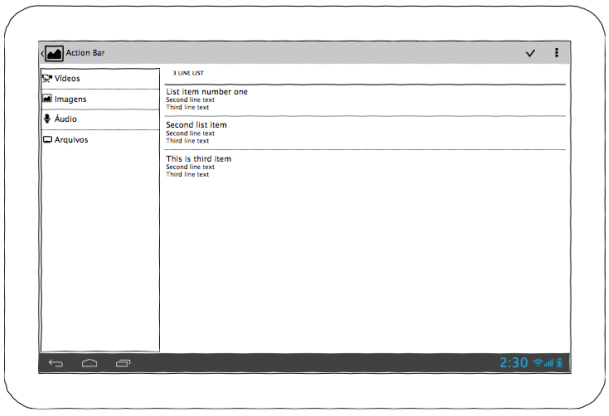
Os Fragments não trazem uma solução definitiva para o problema. Eles apenas minimizam este problema. Imagine que você pode criar u mesmo aplicativo que pode ser executado até mesmo em TVs (sim, tem Android em TVs).

Agora, cada pedaço da tela, que potencialmente pode ser usado em mais de uma situação, estará em uma Fragment. As Activities que determinam quais Fragments serão usados para compor a tela.

Fragments tem seu próprio ciclo de vida. Tem seus próprios widgets. Eles também tratam os eventos dos seus widgets e se comunicam com as Activities que usam ele. Com os Fragments, você pode reaproveitar lógica de apresentação.

A imagem abaixo é do aplicativo rodando em um smartphone. Temos um menu inicial e depois uma listagem que exibe dados conforme a seleção inicial.

A imagem abaixo é do mesmo aplicativo para tablet. Teremos o menu e a listagem em uma única tela.



Com Fragments, temos aqui duas "porções" de tela que podem ser reaproveitadas: menu e listagem. A depender do dispositivo, vamos carregar uma Activity diferente.

Teremos dois Fragments em nosso projeto. No caso do smartphone, teremos duas Activities. No caso do tablet, apenas uma Activity. Vamos ver primeiro a questão do smartphone.

Uma Activity terá o Fragment de Menu. A outra Activity, um Fragment de listagem. No caso do Tablet, teremos uma única Activity com os dois Fragments.

FRAGMENTS NA PRÁTICA

Vamos para a prática. Como um Fragment corresponde a um "pedaço" de tela, temos que definir como esse "pedaço" se parecerá. Enfim, precisamos criar um arquivo XML de Layout para nossa Fragment. Obviamente, você pode construir sua Fragment só com código Java, mas esta não é a melhor forma.

Este Layout não tem limitações, faço-o da mesma forma como já fazia com as Activities. No trecho de código abaixo colocamos apenas uma ListView no nosso layout.

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <ListView android:id="@+id/lista" >
   </ListView>

</LinearLayout>
```

Agora, precisamos de uma classe que tratará os eventos. Precisamos criar uma nova classe que deve herdar de **android.support.v4.app.Fragment** ou qualquer uma de suas especializações. Para este nosso primeiro exemplo, precisamos apenas sobrescrever o método **onCreateView()**.

Este método é chamado sempre que for o momento de exibir o layout da Fragment. Observe que ele retorna um objeto do tipo View. Podemos usar o LayoutInflater que recebemos como parâmetro para "inflar" o arquivo XML.

```
public class ListaFragment extends Fragment {

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

Apenas este código já é suficiente para exibir nossa View. Mas, e se quisermos tratar algum elemento da tela? E para preencher nossa ListView com alguns valores? Precisamos fazer algumas poucas mudanças.

```
public class ListaFragment extends Fragment {
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container, Bundle savedInstanceState) {

      View view = inflater.inflate(R.layout.fragment_lista, container);

      ListView listView = (ListView) view.findViewById(R.id.lista);
      ArrayAdapter<String> adapter =
            new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_1, android.R.id.text1);

      adapter.add("Item 1");
      adapter.add("Item 2");
      adapter.add("Item 3");

      listView.setAdapter(adapter);
      return view;
   }
}
```

Podemos obter uma referência para os objetos usando a própria View que "inflamos", usando o método **findViewById()**. Pronto? Basta isto? Não. Agora, precisamos criar o nosso arquivo de Layout da Activity.

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

  <fragment
   android:id="@+id/list"
   android:name="com.alienlabz.exemplo.fragments.ListaFragment"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" />

</RelativeLayout>
```

Existem duas formas de "anexar" um Fragment a uma Activity. A primeira é mais estática e estamos usando no trecho de código acima. Usamos a tag <fragment> com o atributo **andorid:name** apontando para o caminho completo da classe de Fragment. A outra forma é dinâmica e veremos ela mais adiante.

Ainda falta mais um detalhe. Para que uma Activity possa gerenciar os Fragments, ela precisa estender de FragmentActivity, em vez de Activity. Isto já é suficiente, mas não tudo, para se ter um Fragment executando! :-)

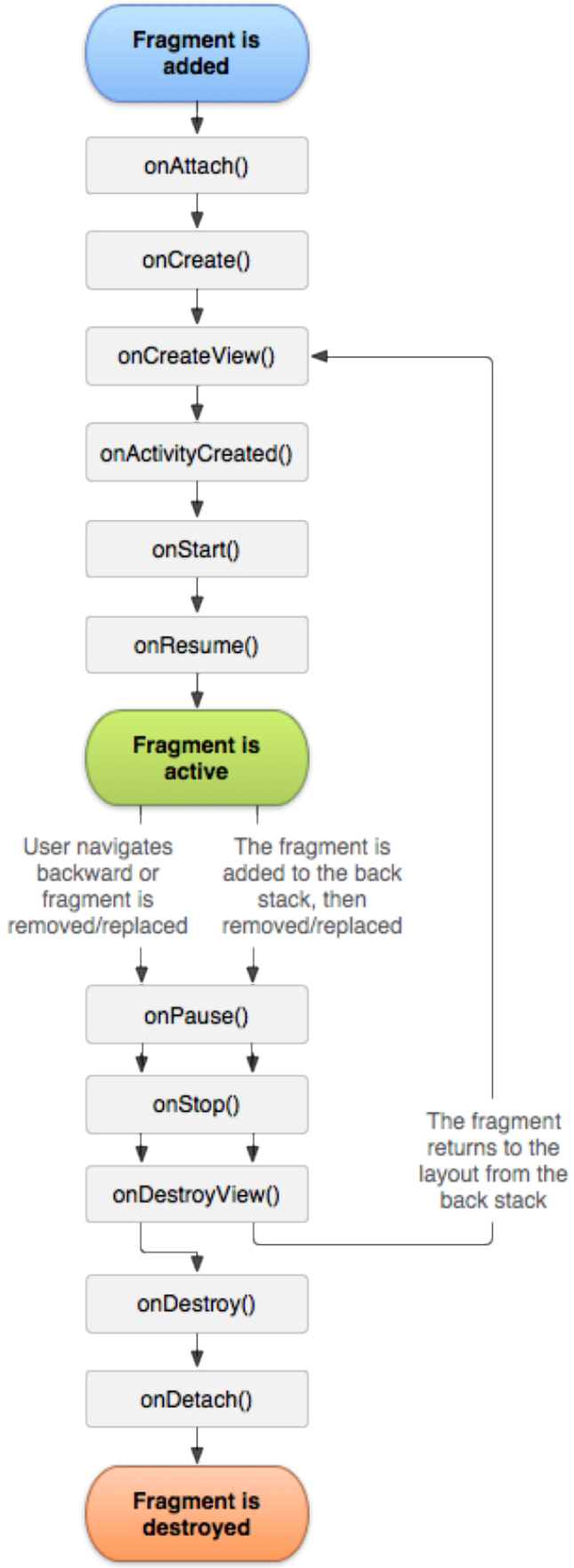
```
public class MainActivity extends FragmentActivity {

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

}
```

CICLO DE VIDA

Assim como no caso das Activities, os Fragments também possuem seu ciclo de vida particular. Vamos analisar a imagem abaixo, retirada da documentação do Android.



Observe que você possui métodos "gancho" para cada momento do ciclo de vida. Basicamente, um Fragment tem os mesmos estados de uma Activity.

Resumido (Resumed) - O Fragment está visível na Activity.

Pausado (Paused) - Outra Activity está na frente da Activity que contém este Fragment.

Parado (Stopped) - O Fragment não está mais visível. Isto ocorre por um de dois motivos: a Activity que a contém foi parada também ou o próprio Fragment foi removido da Activity. Neste último caso, o Fragment continua "vivo". Seu estado continua salvo e guardado para uso futuro.

Igual a uma Activity, você também tem o método onSaveInstanceState() para guardar o estado dela e recuperar posteriormente nos métodos **onCreate()**, **onCreateView()** ou **onActivityCreated()**. O gerenciamento do ciclo de vida da Fragment é bastante similar com uma Activity.

A única questão que precisamos nos preocupar é que agora o ciclo de vida da Activity influencia no ciclo de vida do Fragment. Como você pode observar na imagem, temos métodos "hook" similares entre a Activity e o Fragment. Toda vez que um método deste é chamado na Activity, uma chamada também é feita no Fragment (ou fragments, se a Activity tiver mais de um).

Contudo, temos alguns métodos que não são "espelhados" dessa forma. Por exemplo, **onAttach()** é chamado quando o Fragment é anexado a uma Activity. **onCreateView()** é chamado para você montar a View. **onActivityCreated()** é chamado assim que o método onCreate() da Activity é finalizado.

O **onDestroyView()** é chamado quando a View associada ao Fragment é removida. Por último, **onDetach()** é chamado quando a Fragment não estiver mais anexada à Activity.

TRANSAÇÕES COM FRAGMENTS

Conforme já tínhamos conversado, também podemos adicionar e remover Fragments de uma Activity de forma dinâmica. Mas existem muitas outras operações que podemos executar. Vamos ver agora algumas transações com Fragments.

Dentro de uma transação você pode fazer diversas alterações nos Fragments: remover, adicionar e alternar. Após isto, é interessante adicionar essa transação para o Back Stack. Motivo? Caso o usuário aperte o botão Back do dispositivo, os Fragments podem retornar para a transação ou estado anterior.

Para adicionar e alternar entre Fragments, precisamos criar um novo Layout e Fragment. Vamos fazer um layout simples, apenas com um campo de texto e usar o Fragment que criamos anteriormente. Vamos alternar entre eles dois.

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <EditText
      android:id="@+id/texto"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />

</LinearLayout>
```

Precisamos alterar nosso arquivo de Layout da Activity. O motivo é apenas um: não podemos alterar Fragments estáticos, declarados no XML de layout. Em seu lugar, vamos colocar um FrameLayout apenas para "guardar o lugar".

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <FrameLayout
      android:id="@+id/fragment_container"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >
   </FrameLayout>

   <Button
      android:text="Mudar Fragment"
      android:id="@+id/button_proximo"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />

</LinearLayout>
```

Observe que agora temos um botão que sempre estará na tela. Ao clicar nele, vamos alternar entre os Fragments. Vamos ao código da Activity, que agora está um pouco maior.

Inicialmente, temos que ter uma referência para o **FragmentManager** e o **FragmentTransaction**. Observe que chamamos o método **getSupportFragmentManager()** em vez de **getFragmentManager()**. Motivo? Simples. Estamos usando tudo da biblioteca de suporte, pois estamos supondo que este nosso projeto rodará em Androids com versão anterior à 3.X.

Caso você tenha certeza que seu projeto rodará apenas a partir da versão 3.X, então, use **getFragmentManager()**. Feito isto, vamos iniciar a transação e adicionar o Fragment de Listagem com **transaction.add()**. O primeiro parâmetro é o ID do FrameLayout que criamos, o segundo o Fragment.

```
public class MainActivity extends FragmentActivity {
   private Button button;
   private Fragment nextFragment;

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

      final FragmentManager manager =
            getSupportFragmentManager();

      FragmentTransaction transaction =
            manager.beginTransaction();

      transaction.add(R.id.fragment_container,
            new ListaFragment());

      nextFragment = new MenuFragment();
      transaction.commit();

      button = (Button) findViewById(R.id.button_proximo);
      button.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            FragmentTransaction transaction =
                  manager.beginTransaction();

            transaction.replace(R.id.fragment_container,
                  nextFragment);
            transaction.addToBackStack(null);
            transaction.commit();
            if (nextFragment instanceof MenuFragment) {
               nextFragment = new ListaFragment();
            } else {
               nextFragment = new MenuFragment();
            }
         }
      });
   }
}
```

Concluímos com um **transaction.commit()**. Agora vamos definir o comportamento do clique do botão. É quase tudo igual, mas usamos agora o método **transaction.replace()**, pois já temos um Fragment lá e queremos substituir pelo **MenuFragment**.

Também usamos **transaction.addToBackStack()** para colocar esta transação na pilha. Em seguida, fazemos uma lógica bem simples para poder alternar entre os Fragments.

COMUNICANDO COM A ACTIVITY

Para finalizar este assunto, precisamos ver a melhor maneira de realizar a comunicação entre Activity e Fragment.

Atenção em um detalhe: se você faz a comunicação direta entre Fragments, está quebrando a principal vantagem de se usar Fragments. Com eles, você tem pedaços de tela independentes e que podem ser usados em diferentes Activities.

Se você acopla duas Fragments, terá que usar as duas em conjunto sempre. O ideal é que a comunicação ocorra somente no sentido Fragment -> Activity e Activity -> Fragment. Mas, qual a melhor forma de fazer isso? Criando seus próprios Listeners. Também não é uma boa ideia acoplar Fragments a implementações específicas de Activities. O motivo é o mesmo que usamos para não acoplar Fragments entre elas.

Os Listeners são uma forma mais "desacoplada" de se fazer isto. Neste caso, vamos criar uma Interface que define um Listener no nosso Fragment. Vamos garantir que toda Activity que anexar nosso Fragment nela terá que implementar esta interface e estar preparado para receber os eventos deste Fragment. Entendeu? Vamos analisar a listagem de código da coluna a seguir.

O que temos de diferente é que estamos criando a interface **OnItemListSelectedListener** e definimos ela com o método **onItemSelected()**. Feito isto, sobrescrevemos o método **onAttach()**, do ciclo de vida. Pegamos a Activity na qual o Fragment foi anexado e checamos se ela implementa a interface que acabamos de criar. Caso não implemente, lançamos uma exceção.

Também definimos um Listener para checar quando ocorre um clique na ListView. Quando isto ocorrer, repassamos este evento para a Activity. Esta é a forma mais recomendada para se fazer a comunicação na direção Fragment -> Activity.

No caso de querer se comunicar de uma Activity para um Fragment, pode-se obter diretamente a Fragment e chamar seus métodos. Por exemplo, você pode definir um método no Fragment para preencher a lista e chamar ele na Activity. Você obtém uma Activity a partir de uma Activity chamando **FragmentManager.findFragmentById()**.

OPTIONS MENU

Você também pode interagir com o menu da Activity que contém o Fragment. Você faz isto para adicionar opções específicas deste Fragment. Como fazer isso? Primeiro, você precisa chamar o método **setHasOptionsMenu()** da classe Fragment.

Depois, sobrescrever o método **onCreateOptionsMenu()**, igual a como você faz na Activity. Você também pode implementar o método **onOptionsItemSelected()** para tratar o clique nos itens do Menu.

```
public class ListaFragment extends Fragment {
   private OnItemListSelectedListener mListener;

   @Override
   public void onAttach(Activity activity) {
      super.onAttach(activity);
      try {
         mListener = (OnItemListSelectedListener) activity;
      } catch (ClassCastException e) {
         throw new ClassCastException(activity.toString() +
            " must implement OnItemListSelectedListener");
      }
   }

   @Override
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container, Bundle savedInstanceState) {

      View view = inflater.inflate(R.layout.fragment_lista,
            container, false);

      ListView listView = (ListView) view.findViewById(R.id.lista);
      listView.setOnItemClickListener(new OnItemClickListener() {

         @Override
         public void onItemClick(AdapterView<?> adapterView,
               View view, int arg2, long arg3) {
            mListener.onItemSelected();
         }

      });
      return view;
   }

   public interface OnItemListSelectedListener {
      public void onItemSelected();
   }
}
```

Sobre o Autor

Marlon Silva Carvalho

É um programador de longa data. Já peregrinou por diversas linguagens e hoje se considera um programador agnóstico. Atualmente, está fascinado pelo mundo mobile e suas consequências para a humanidade. Está mais focado no desenvolvimento para Android, embora também goste de criar aplicativos para iOS.

Trabalha em projetos sobre mobilidade no SERPRO, tendo atuado no projeto Pessoa Física, para a Receita Federal do Brasil. Você pode encontrar ele no Twitter, no perfil @marlonscarvalho, em seu blog, no endereço http://marlon.silvacarvalho.net e através do e-mail marlon.carvalho@gmail.com.

Sobre os Cards

Caso você já tenha visto os excelentes RefCards da DZone, deve ter percebido a semelhança. E é proposital. A ideia surgiu após acompanhar estes RefCards por um bom tempo. Contudo, eles são mais gerais. O objetivo destes Cards é tratar assuntos mais específicos. Trazendo informações relevantes sobre eles.

Caso tenha gostado, continue nos acompanhando! Tentaremos trazer sempre novos cards sobre os mais variados assuntos. Inicialmente, focaremos no Android.

O ícone principal deste card é de autoria de Wallec e foi obtido em seu profile no DeviantArt.
http://wwalczyszyn.deviantart.com/