# Building a Dynamic UI with Fragments

In this lecture we will see:

* How to manage dynamically some fragments, an example.
* How fragments talk with Activity and each other.

To create a dynamic and multi-pane user interface on Android, you need to encapsulate UI components and activity behaviors into modules that you can swap into and out of your activities. You can create these modules with the [Fragment](https://developer.android.com/reference/android/app/Fragment.html) class, which behaves somewhat like a nested activity that can define its own layout and manage its own lifecycle.

When a fragment specifies its own layout, it can be configured in different combinations with other fragments inside an activity to modify your layout configuration for different screen sizes (a small screen might show one fragment at a time, but a large screen can show two or more).

When designing your application to support a wide range of screen sizes, you can reuse your fragments in different layout configurations to optimize the user experience based on the available screen space.

For example, on a handset device it might be appropriate to display just one fragment at a time for a single-pane user interface. Conversely, you may want to set fragments side-by-side on a tablet which has a wider screen size to display more information to the user.

**Example:** Let’s assume that we need to use two different layouts for a screen when the screen rotates on the left or right being in Portret and Landscape orientation.

The main activity applies a layout in the usual way, during onCreate():

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

);  
}

The layout applied is activity\_main.xml:

The following layout is an alternative to the layout that shows only one fragment at a time. In order to replace one fragment with another, the activity's layout includes an empty FrameLayout that acts as the fragment container.

<?xml version="1.0" encoding="utf-8"?>  
<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent"**>  
 <**TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerHorizontal="true"  
 android:layout\_centerVertical="true"  
 android:text="Hello World"**/>

<**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"** />

</**LinearLayout**>

To return a layout from onCreateView(), you can inflate it from a layout resource defined in XML. To help you do so, onCreateView() provides a LayoutInflater object.

For example, here are two subclasses of [Fragment](https://developer.android.com/reference/android/support/v4/app/Fragment.html) that load respectively a layout from the activity\_vertical\_fragment.xml, and activity\_horizontal\_fragment.xml files:

**public class** VerticalFragment **extends** Fragment {  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
 *// Inflate the layout for this fragment* **return** inflater.inflate(R.layout.***activity\_vertical\_fragment***,

container, **false**);  
 }  
}

**public class** HorizontalFragment **extends** Fragment {  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
 *// Inflate the layout for this fragment* **return** inflater.inflate(R.layout.***activity\_horizontal\_fragment***,

container, **false**);  
 }  
}

Here is the MainActivity, which simply embeds the VericalFragment and HorizontalFragment to display the selected screen orientation view:

**public class** MainActivity **extends** FragmentActivity {  
 **private boolean test**=**true**;  
 OrientationEventListener **mOrientationListener**;  
  
 @Override  
 **public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 portretMode();  
 }  
  
 @Override  
 **public void** onConfigurationChanged(Configuration newConfig) {  
 **super**.onConfigurationChanged(newConfig);  
 *// Checks the orientation of the screen* **if** (newConfig.**orientation** == Configuration.***ORIENTATION\_LANDSCAPE***) {  
 landscapeMode();  
 } **else if** (newConfig.**orientation** == Configuration.***ORIENTATION\_PORTRAIT***){  
 portretMode();  
 }  
 }  
 **void** portretMode(){  
 . . .

}  
 **void** landscapeMode(){  
 . . .

}  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
  
 }  
}

Le discuss the content of the method portretMode() and landscapeMode().

The procedure to replace a fragment requires the replace() method.

Keep in mind that when you perform fragment transactions, such as replace or remove one, it's often appropriate to allow the user to navigate backward and "undo" the change. To allow the user to navigate backward through the fragment transactions, you must call addToBackStack() before you commit the FragmentTransaction.

Inside your activity, call getSupportFragmentManager() to get a FragmentManager using the Support Library APIs. Then call beginTransaction() to create a FragmentTransaction and call add() to add a fragment.

You can perform multiple fragment transaction for the activity using the same FragmentTransaction. When you're ready to make the changes, you must call commit().

**void** portretMode(){  
 VerticalFragment secondFragment = **new** VerticalFragment();  
 Bundle args = **new** Bundle();  
 FragmentTransaction transaction =

getSupportFragmentManager().beginTransaction();  
 transaction.replace(R.id.***fragment\_container***, secondFragment);  
 transaction.addToBackStack(**null**);  
 transaction.commit();  
 Toast.*makeText*(**this**, **"portrait"**, Toast.***LENGTH\_SHORT***).show();  
}

**void** landscapeMode(){  
 HorizontalFragment firstFragment = **new** HorizontalFragment();  
 Bundle args = **new** Bundle();  
 FragmentTransaction transaction =

getSupportFragmentManager().beginTransaction();  
 transaction.replace(R.id.***fragment\_container***, firstFragment);  
 transaction.addToBackStack(**null**);  
 transaction.commit();  
 Toast.*makeText*(**this**, **"landscape"**, Toast.***LENGTH\_SHORT***).show();  
}

## Class work: A project where the fragments talk with each other and are managed dynamically

Let try to complete the following project where MainActivity manages **three fragments**. The first has a ListView control, the second has a TextView control and the third a WebView control. We have to manage dynamically the fragments. When we push a button in the MainActivity screen, the list screen (ListFragment) will be shown, then when we click an item of the list a TextView (TextFragment) will be shown, filled with some information about the list item that we find in an array in string.xml file, and when we click on the TextView, an web page that correspond to the item of the list will be displayed. We find the web address in an array in string.xml file.

We need to organize the data of the list as an array for example

**public** String []**myarray**={**"Lecture1"**,**"Lecture2"**,**"Lecture3"**,**"Lecture4"**,**"Lecture5"**};

and regarding the data that correspond to the elements of the list, let put them in string.xml file. One array with elements that will be used to display some info in TextView of the TextFragments and one array that will be used in the WebFragment. The three arrays will communicate through the same position defined by the list item click in the ListFragment.

The array that will be used in TextFragment

<**string-array name="description"**>  
 <**item**>Introduction</**item**>  
 <**item**>Intents in Android</**item**>  
 <**item**>User Interface</**item**>  
 <**item**>Fragments</**item**>  
 <**item**>Manage Dynamically Fragments</**item**>  
</**string-array**>

The array that will be used in WebFragment

<**string-array name="urls"**>  
 <**item**>https://developer.android.com/guide/index.html</**item**>  
 <**item**><https://developer.android.com/reference/android/content/Intent.html>

</**item**>  
 <**item**><https://developer.android.com/guide/topics/ui/declaring-layout.html>

</**item**>  
 <**item**><https://developer.android.com/reference/android/app/Fragment.html>

</**item**>  
 <**item**><http://www.vogella.com/tutorials/AndroidFragments/article.html>

</**item**>  
</**string-array**>

The first and the second fragments are almost identical with these fragments in the previous lesson example. We need to change the layout of the MainActivity, because we have to work dynamically with the fragments. Therefore, the layout of the MainActivity has to be:

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent"**>  
 <**Button  
 android:id="@+id/button1"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="List"** />  
  
 <**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"** />  
</**LinearLayout**>

The layout of the fragments let be:  
**ListFragment Layout**

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:background="#0000ff"  
 android:foregroundTint="#ffffff"**>  
 <**ListView  
 android:id="@+id/list"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"** />  
</**LinearLayout**>

**TextFragment Layout**

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:background="#00ff00"**>  
 <**TextView  
 android:id="@+id/textView1"  
 android:layout\_width="150.dp"  
 android:layout\_height="wrap\_content"  
 android:textAppearance="?android:attr/textAppearanceLarge"  
 android:autoLink="web"** />  
  
</**LinearLayout**>

**WebFragment Layout**

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="com.example.user.fragmenttrancom.MainActivity"**>  
 <**WebView  
 android:id="@+id/webview"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"** />  
</**RelativeLayout**>

When we click on TextView we have to send a web address to the WebFragment, for this, we need to use the same logic that we used for the ListFragment when we clicked a list item. So, we use a method of the Communicator interface, that is implemented in MainActivity. Therefore, we add another method in Interface:

**public interface** Communicator {  
 **public void** respondList(**int** position);  
 **public void** respondLink(**int** position);  
}

the method respondList is used in ListFragment and the method respondLink is used in the TextFragment. Both these methods, implemented in MainActivity, send the position (list item position) into respective Fragments, TextFragments and WebFragments.

The way how we communicate with fragments from MainActivity is different now. For example if we consider the fragment WebFragment, from MainActivity we have :

**public void** respondLink(**int** position){  
 WebFragment webFragment = **new** WebFragment();  
 Bundle bundle = **new** Bundle();  
 bundle.putInt(**"url"**,position);  
 webFragment.setArguments(bundle);  
 . . .   
}

WebFragment webFragment = **new** WebFragment(); we get the fragment reference

and if we would like to send some information into WebFragment we use the lines

Bundle bundle = **new** Bundle();  
bundle.putInt(**"url"**,position);  
WebFragment.setArguments(bundle);

And to get the same information inside the fragment, we use the lines

@Override  
**public void** onActivityCreated(Bundle savedInstanceState) {  
 **super**.onActivityCreated(savedInstanceState);  
 Bundle bundle = **this**.getArguments();  
 **position** = bundle.getInt(**"url"**);  
 . . .   
}

For both fragments we could use the same method to activate one fragment.

**void** callFragment(Fragment fragment)  
{  
 FragmentTransaction transaction =

getSupportFragmentManager().beginTransaction();  
 transaction.replace(R.id.***fragment\_container***, fragment);  
 transaction.addToBackStack(**null**);  
 transaction.commit();  
  
}

**The new element for us is the code of WebFragment.**

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turn your application to a web application.

In order to add WebView to your application, you have to add **<WebView>** element to your xml layout file.

<WebView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/webview"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

/>

In order to use it, you have to get a reference of this view in Java file. To get a reference, create an object of the class WebView.

WebView mWebView = (WebView) findViewById(R.id.webview);

In order to load a web url into the WebView, you need to call a method loadUrl(String url) of the WebView class, specifying the required url. Its syntax is:

mWebView.loadUrl("http://www.google.com");

Apart from just loading url, you can have more control over your WebView by using the methods defined in WebView class. See the comments in the code:

**public class** WebFragment **extends** Fragment {  
 WebView **mWebView**;  
 **int position**;  
 String **url**;  
 @Nullable  
 @Override  
 **public** View onCreateView(LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 View v=inflater.inflate(R.layout.***web\_layout***, container, **false**);  
 **mWebView** = (WebView) v.findViewById(R.id.***webview***);  
 **return** v;  
  
 }  
  
 @Override  
 **public void** onActivityCreated(Bundle savedInstanceState) {  
 **super**.onActivityCreated(savedInstanceState);  
 Bundle bundle = **this**.getArguments();  
 **position** = bundle.getInt(**"url"**);  
 Log.*d*(**"position web "**,**"position "**+**position**);  
 **url**=getContent(**position**);  
 *//Sets whether the WebView should load image resources.* **mWebView**.getSettings().setLoadsImagesAutomatically(**true**);  
 *//If the web page you plan to load in your WebView use  
 // JavaScript, you must enable JavaScript for your WebView.* **mWebView**.getSettings().setJavaScriptEnabled(**true**);  
 **mWebView**.setScrollBarStyle(View.***SCROLLBARS\_INSIDE\_OVERLAY***);  
  
 **mWebView**.loadUrl(**url**);  
 *// to open links clicked by the user, simply provide a   
 // WebViewClient for your WebView, using setWebViewClient().* **mWebView**.setWebViewClient(**new** WebViewClient());  
  
  
  
 }  
  
 **public** String getContent(**int** data) {  
 Resources res = getResources();  
 String[] desc = res.getStringArray(R.array.***urls***);  
 **return** desc[data];  
 }  
  
}

Modify the AndroidManifest.xml to add the permission

<**uses-permission android:name="android.permission.INTERNET"** />

**References**

https://developer.android.com/training/basics/fragments/fragment-ui.html

<https://developer.android.com/guide/components/fragments.html#Transactions>

https://developer.android.com/guide/webapps/webview.html