**The Basics of Android Fragments**

Developers can combine one or more fragments to build a single activity or even reuse fragments across multiple activities.

Fragments were introduced in Android 3.0 to improve the user experience. Classically, developers would have to build a new Activity whenever the user interacted with the application. With Fragments, developers now can update another portion of the UI on the screen that corresponds to the user selection,, without needing to move the user to another screen.

A classic example is a news application that allows a user to select the headline on the left side of the application and it then displays the details of the story on the right side corresponding to user selection.

Fragments can be used in the following scenarios:

* Fragments can be used to build flexible user interfaces across different screen sizes.
* Fragments can be used to provide Fixed/Scrolling/swipe tab displays.
* Fragments can be used where dialog boxes are needed.
* Fragments can be used to build action bar customization with the list and tab modes.

Fragment have the following characteristics

* A fragment is always embedded in an activity. It does not have existence outside of one.
* It has its own lifecycle; however, it is directly impacted by the host activity lifecycle.
* A fragment object processes its own events.
* Fragments can be dynamically added or removed from an Activity, based on user interaction with the application. This can happen when the host Activity is in the "running" state.
* Fragments support transaction with an Activity. In other words, you can click the "Back" button and undo a fragment activity when you perform a fragment transaction.

A fragment lives in the ViewGroup of the host Activity's view hierarchy.

**Creating a Fragment in Android**

Creating a fragment is simple and involves four steps:

1. Extend Fragment class.
2. Provide appearance in XML or Java.
3. Override onCreateView to link the appearance.
4. Use the Fragment in your activity.

There are three methods, at a minimum, that we need to implement for a fragment.

* **onCreate()**: This method is called when the fragment is created by Android System. Use this method to initialize necessary components of the fragment that you want to keep when the fragment is paused/stopped and then resumed.
* **onCreateView()**: This method is called when the user interface for the fragment needs to be drawn for the first time. This method returns a view.
* **onPause()**: This method is called when the user is leaving the fragment. In this method, tthe app developer needs to commit any pending changes that need to be brought back when the user resumes.

Besides the base Fragment class, the Android platform offers the following three fragment classes:

* **DialogFragment**: This class displays a floating dialog.
* **ListFragment**: This class displays a list of items managed by an adapter.
* **PreferenceFragrment**: This class offers functionality similar to the "Settings" experience.
* onAttach() is called when a fragment is connected to an activity.
* onCreate() is called to do initial creation of the fragment.
* onCreateView() is called by Android once the Fragment should inflate a view.
* onViewCreated() is called after onCreateView() and ensures that the fragment's root view is non-null. Any view setup should happen here. E.g., view lookups, attaching listeners.
* onActivityCreated() is called when host activity has completed its onCreate() method.
* onStart() is called once the fragment is ready to be displayed on screen.
* onResume() - Allocate “expensive” resources such as registering for location, sensor updates, etc.
* onPause() - Release “expensive” resources. Commit any changes.
* onDestroyView() is called when fragment's view is being destroyed, but the fragment is still kept around.
* onDestroy() is called when fragment is no longer in use.
* onDetach() is called when fragment is no longer connected to the activity.

The lifecycle execution order is mapped out below:

[](https://i.imgur.com/0EVReuq.png)

The most common ones to override are onCreateView which is in almost every fragment to setup the inflated view, onCreate for any data initialization and onActivityCreated used for setting up things that can only take place once the Activity has been fully created.

Here's an example of how you might use the various fragment lifecycle events:

public class SomeFragment extends Fragment {

ThingsAdapter adapter;

FragmentActivity listener;

// This event fires 1st, before creation of fragment or any views

// The onAttach method is called when the Fragment instance is associated with an Activity.

// This does not mean the Activity is fully initialized.

@Override

public void onAttach(Context context) {

super.onAttach(context);

if (context instanceof Activity){

this.listener = (FragmentActivity) context;

}

}

// This event fires 2nd, before views are created for the fragment

// The onCreate method is called when the Fragment instance is being created, or re-created.

// Use onCreate for any standard setup that does not require the activity to be fully created

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

ArrayList<Thing> things = new ArrayList<Thing>();

adapter = new ThingsAdapter(getActivity(), things);

}

// The onCreateView method is called when Fragment should create its View object hierarchy,

// either dynamically or via XML layout inflation.

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_some, parent, false);

}

// This event is triggered soon after onCreateView().

// onViewCreated() is only called if the view returned from onCreateView() is non-null.

// Any view setup should occur here. E.g., view lookups and attaching view listeners.

@Override

public void onViewCreated(View view, Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

ListView lv = (ListView) view.findViewById(R.id.lvSome);

lv.setAdapter(adapter);

}

// This method is called when the fragment is no longer connected to the Activity

// Any references saved in onAttach should be nulled out here to prevent memory leaks.

@Override

public void onDetach() {

super.onDetach();

this.listener = null;

}

// This method is called after the parent Activity's onCreate() method has completed.

// Accessing the view hierarchy of the parent activity must be done in the onActivityCreated.

// At this point, it is safe to search for activity View objects by their ID, for example.

@Override

public void onActivityCreated(Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

}

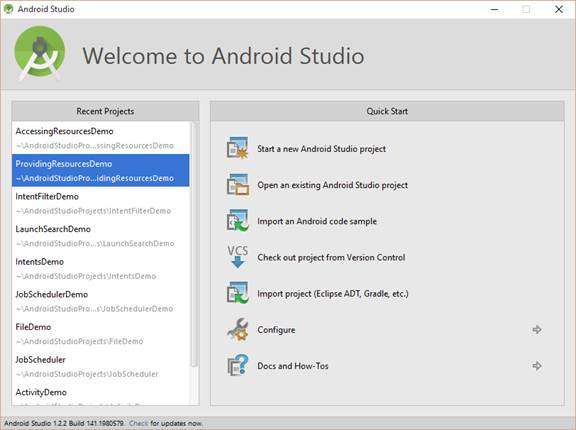
}

Let us create a simple application that demonstrates working with Fragment.

## Hands On with Android Fragments

Let's create a simple Android application that demonstrates working with Activities.

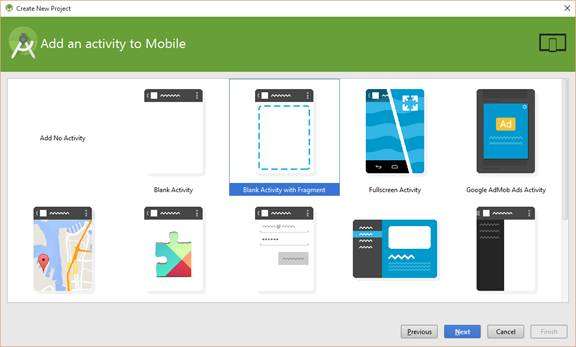
Fire up Android Studio and Start a new Android Studio Project.

  
**Figure 1:** Starting a new Android Studio project

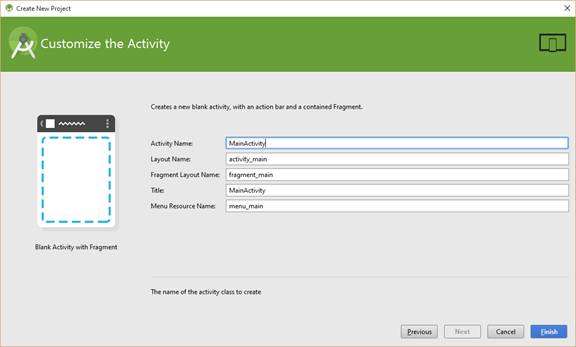
Provide FragmentDemo as the Application Name and click Next.

**e 3:** Leaving the default values in place

On the "Add an activity to Mobile" page, choose "Blank Activity with Fragment." This creates an application with a single activity and a Fragment.

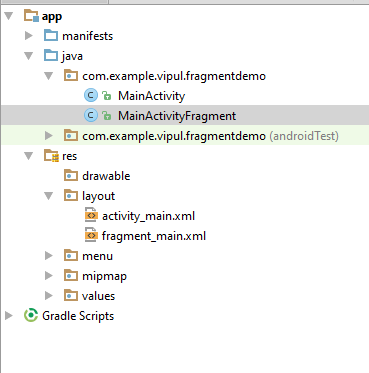
  
**Figure 4:** Adding a new blank activity

We then are prompted to customize the activity. We will leave the default values unchanged.

  
**Figure 5:** Again, leaving the default values in place

Click Finish to creating the project files.

Let's look at how Android Studio creates the default application. Open up Project Explorer and expand the /java node and /res node.

  
**Figure 6:** Expanding the /java and /res nodes

We can see the following:

* **There are two Java files:** One for the activity and one for the fragment.
* **There are two layout files:** One for the activity and one for the fragment.

The fragment\_main.xml file (layout file for the fragment) contains the following. Update the layout file to include the highlighted text. This text will make our fragment stand out when it's loaded into the MainActivity.

<**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"**

**android:paddingLeft="@dimen/activity\_horizontal\_margin"**

**android:paddingRight="@dimen/activity\_horizontal\_margin"**

**android:paddingTop="@dimen/activity\_vertical\_margin"**

**android:paddingBottom="@dimen/activity\_vertical\_margin"**

**tools:context=".MainActivityFragment"**

**android:background="#33FF00"**>

<**TextView android:text="@string/hello\_world"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**/>

</**RelativeLayout**>

We can see that it has a very similar content compared to the activity file.

Next, we observe the Activity file, activity\_main.xml.

<**fragment xmlns:android=**

**"http://schemas.android.com/apk/res/android"**

**xmlns:tools="http://schemas.android.com/tools"**

**android:id="@+id/fragment"**

**android:name="com.example.vipul.fragmentdemo.MainActivityFragment"**

**tools:layout="@layout/fragment\_main"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"** />

You can see from the activity layout file that it refers to the fragment layout.

The Java file for Fragment (MainActivityFragment.java) contains the following template code:

**package** com.example.vipul.fragmentdemo;

**import** android.support.v4.app.Fragment;

**import** android.os.Bundle;

**import** android.view.LayoutInflater;

**import** android.view.View;

**import** android.view.ViewGroup;

/\*\*

\* A placeholder fragment containing a simple view.

\*/

**public class** MainActivityFragment **extends** Fragment {

**public** MainActivityFragment() {

}

@Override

**public** View onCreateView(LayoutInflater inflater,

ViewGroup container,

Bundle savedInstanceState) {

**return** inflater.inflate(R.layout.fragment\_main,

container, **false**);

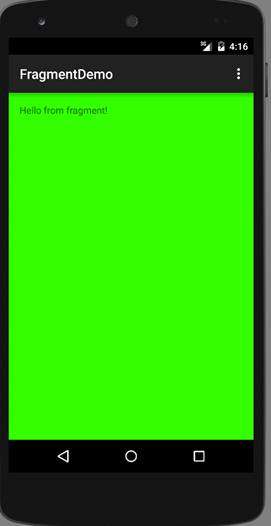
}

}

We can see that the onCreateView method has been implemented as an override and it calls the helper inflater class to create the view.

Let's make one change in /res/values/strings.xml to update "Hello World!" to "Hello from fragment!" because this text is being used in the fragment.

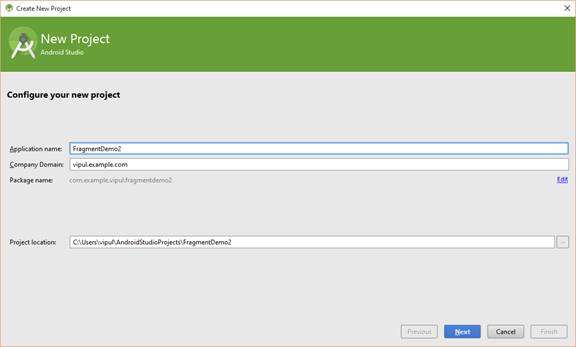
When we run our application, we will see the following when the application loads:

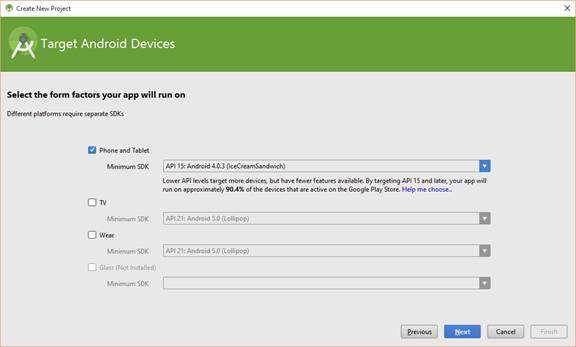
  
**Figure 7:** The application's output

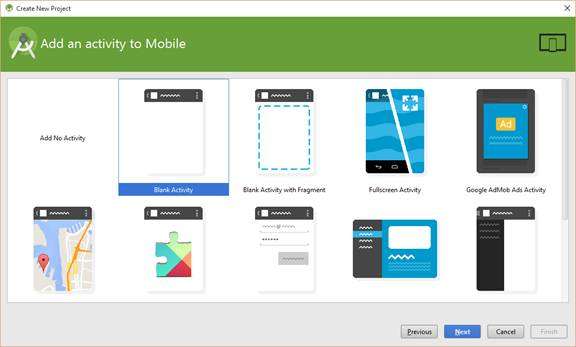
We can see that the fragment was loaded (the string helps us correlate).

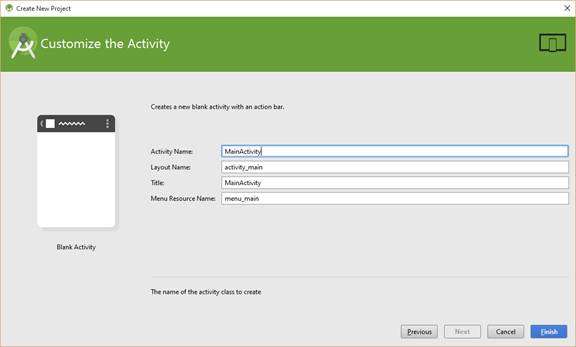
Now, we will create another simple application where we will demo creating a Fragment in a different manner (without using Android Studio support for Fragments).

Create a new project called FragmentDemo2 and this time choose the template for a blank activity.

  
**Figure 8:** Creating a second new project

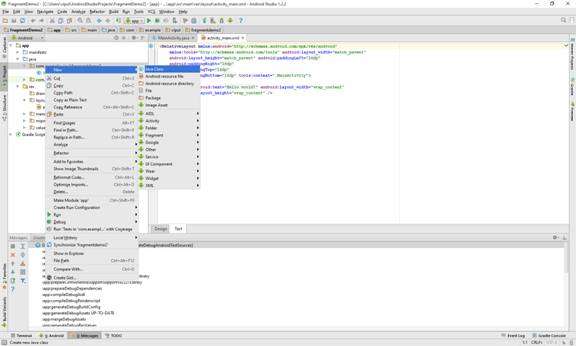
  
**Figure 9:** Choosing the "Phone and Tablet" option

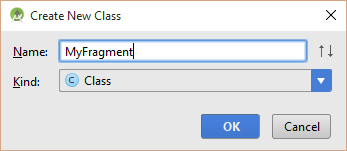
  
**Figure 10:** Selecting a blank activity

  
**Figure 11:** Naming this "MainActivity"

Click Finish to create the project.

Now, on the Project Explorer, add a new class.

  
**Figure 12:** Adding a new class from Project Explorer

  
**Figure 13:** Naming the new class "MyFragment"

We will see that a new Java file, MyFragment.java, is created with the following content:

**package** com.example.vipul.fragmentdemo2;

**import** android.app.Fragment;

/\*\*

\* Created by Vipul on 11/21/2015.

\*/

**public class** MyFragment {

}

As you recall from the basics mentioned before, we need to extend the Fragment class to create an Android fragment.

Make the following highlighted change in your MyFragment class.

**package** com.example.vipul.fragmentdemo2;

**import** android.app.Fragment;

/\*\*

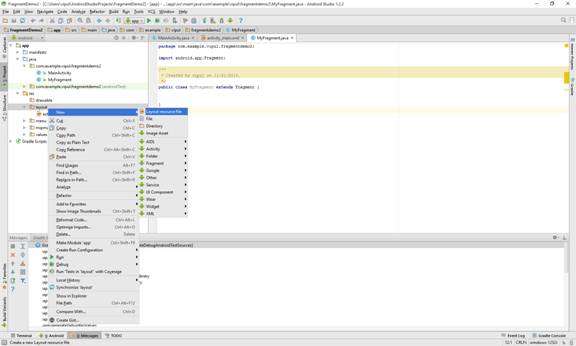
\* Created by Vipul on 11/21/2015.

\*/

**public class** MyFragment **extends**Fragment {

}

Next, we need to create a layout file for the fragment. Right-click the /res/layout node and select New -> Layout resource file.

  
**Figure 14:** Creating a new Layout resource file

Let's name it my\_fragment\_layout and click Ok.

The layout is generated. Here are the contents of the layout file.

<?**xml version="1.0" encoding="utf-8"**?>

<**LinearLayout xmlns:android=**

**"http://schemas.android.com/apk/res/android"**

**android:orientation="vertical"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**>

</**LinearLayout**>

To help with identifying the fragment clearly, we will add an EditText and have a background color for our fragment.

<?**xml version="1.0" encoding="utf-8"**?>

<**LinearLayout xmlns:android=**

**"http://schemas.android.com/apk/res/android"**

**android:orientation="vertical"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:background="#FFBB45"**>

<**EditText**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:id="@+id/editText"**

**android:layout\_gravity="center\_horizontal"**

**android:text="This is a fragment"**/>

</**LinearLayout**>

Next, we need associate the fragment layout with the fragment class file. Open the MyFragmet.java class and implement the onCreateView method:

//myfragment.java

**public class** MyFragment **extends** Fragment {

@Override

**public** View onCreateView(LayoutInflater inflater,

ViewGroup container, Bundle savedInstanceState) {

**return** inflater.inflate(R.layout.my\_fragment\_layout,

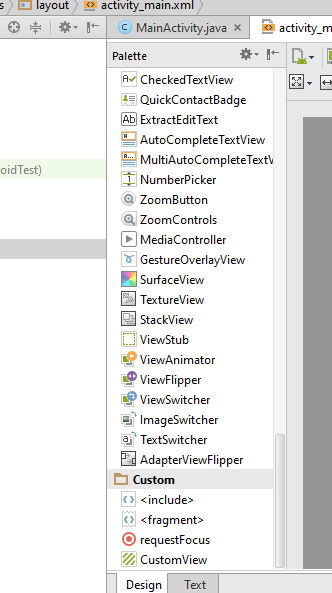
container, **false**);

}

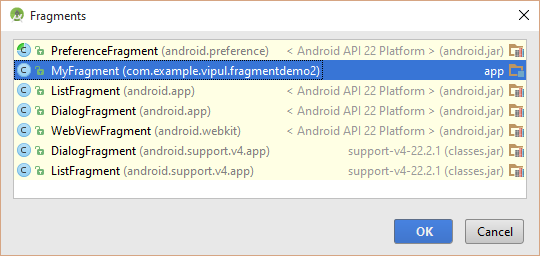
}

Lastly, we need to imbibe the fragment into the Activity.

We can use Android Studio support in the Design view for the MainActivity layout file to select a fragment from inside the Custom choices.

  
**Figure 15:** Selecting a fragment from the Design view

Open the activity\_main layout file in design view and, inside the Palete, click Fragment under the "Custom" section. You will be prompted to select a fragment.

  
**Figure 16:** You're prompted to select a fragment

Now, our application is complete. When we run our application, you will see what's depicted in Figure 17.

  
**Figure 17:** The application, running and displaying its text

We now have fragments in action.