**1 What’s a Fragment?**

A fragment is a class implementing a portion of an activity.

• A fragment represents a particular operation or interface running within a larger activity.

• Fragments enable more modular activity design, making it easier to adapt an application to different screen orientations and multiple screen sizes.

• Fragments must be embedded in activities; they cannot run independent of activities.

• Most fragments define their own layout of views that live within the activity’s view hierarchy.

– However, a fragment can implement a behavior that has no user interface component.

• A fragment has its own lifecycle, closely related to the lifecycle of its host activity.

• A fragment can be a static part of an activity, instantiated automatically during the activity’s creation.

• Or, you can create, add, and remove fragments dynamically in an activity at run-time.

**2. Advantages**

• Fragments are reusable component.

• 0 or more fragments can place in single Activity.

• One Fragment can be placed in any number of Activities.

• Need not to register in manifest file.

**3. Differences between Activity and Fragment**

|  |  |
| --- | --- |
| Activity | Fragment |
| Must be register in manifest file | Need not be register in manifest file. |
| Can exist Independently | Can’t exist independently |
| Contains 0 or more fragments | Same fragment is attached to any number of activities. |
| Life cycle of Activity doesn’t depend on Fragment Life cycle. | Life cycle of Fragment depends on Activity Life cycle. |
| Can exist without fragment also. | Can’t exist without Activity. |
|  | Reusable component. |

**4 Fragments: Implemented in Honeycomb (3.0) or Later**

Fragments were added to the Android API in Honeycomb, API 11. The primary classes related to fragments are:

**android.app.Fragment**

The base class for all fragment definitions

**android.app.FragmentManager**

The class for interacting with fragment objects inside an activity

**android.app.FragmentTransaction**

The class for performing an atomic set of fragment operations

**5 Fragments: Implemented in Donut (1.6) or Later**

Google provides the Compatibility Package, a Java library that you can include in an application, implementing support for fragments and other Honeycomb features (loaders).

• You can use the Compatibility Package with applications targeting 1.6 (API 4) or later.

• For each of the classes in the Compatibility Package, the APIs work almost exactly the same as their counterparts in the latest Android platform. Therefore, you can usually refer to the online documentation for information about the supported APIs.

The primary classes related to fragments are:

**android.support.v4.app.FragmentActivity**

The base class for all activities using compatibility-based fragment (and loader) features

**android.support.v4.app.Fragment**

The base class for all fragment definitions

**android.support.v4.app.FragmentManager**

The class for interacting with fragment objects inside an activity

**android.support.v4.app.FragmentTransaction**

The class for performing an atomic set of fragment operations

To use the Compatibility Package, you must add it to your Android project and include it in your Build Path.

• Using the Eclipse ADT plugin, you can do this automatically by:

1. Selecting your Android project in the Package Explorer view

2. Bringing up the context-menu

3. Selecting Android Tools → Add Compatibility Library

**6 Fragment Lifecycle**

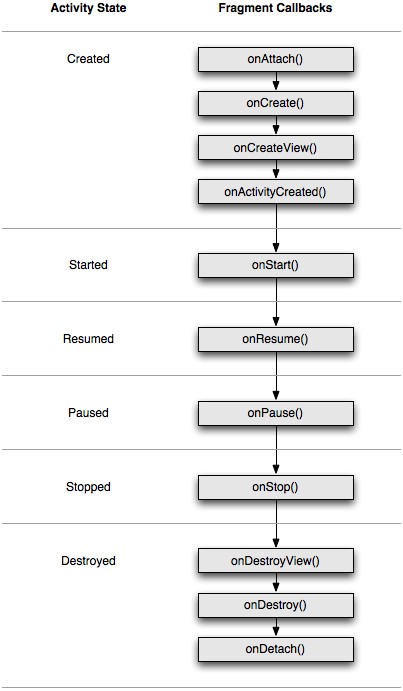


Figure 2: Fragment Lifecycle

As with the Activity class, the Fragment base class is an example of the template method design pattern.

• The Fragment base class defines a set of methods that you override to provide the custom behavior of your fragment

Implementation.

Of primary importance are the fragment lifecycle callback methods.

• The lifecycle of the activity in which the fragment lives directly affects the lifecycle of the fragment.

• Each lifecycle callback for the activity results in a similar callback for each fragment.

• For example, when the activity receives onPause(), each fragment in the activity receives onPause().

Fragments have a few extra lifecycle callbacks managing interaction with the activity:

**onAttach(Activity)**

Called when the fragment has been associated with the activity.

**onCreateView(LayoutInflater, ViewGroup, Bundle)**

Called to create the view hierarchy associated with the fragment.

**onActivityCreated(Bundle)**

Called when the activity’s onCreate() method has returned.

**onDestroyView()**

Called when the view hierarchy associated with the fragment is being removed.

**onDetach()**

Called when the fragment is being disassociated from the activity.

**7 Creating the Fragment Class**

Each fragment must be implemented as a subclass of Fragment.

Many of the Fragment methods are analogous to those found in Activity, and you should use them in a similar fashion.

**onCreate(Bundle)**

Initialize resources used by your fragment except those related to the user interface.

**onCreateView(LayoutInflater, ViewGroup, Bundle)**

Create and return the view hierarchy associated with the fragment.

**onResume()**

Allocate “expensive” resources (in terms of battery life, monetary cost, etc.), such as registering for location updates, sensor updates, etc.

**onPause()**

Release “expensive” resources. Commit any changes that should be persisted beyond the current user session.

**8 Creating a Fragment Layout**

To provide a layout for a fragment, your fragment’s class must implement the onCreateView() callback method.

• The Android system invokes this method when it’s time for the fragment to create its layout.

• This method must return a View that is the root of your fragment’s layout.

As with an activity, you can create your layout programmatically by directly instantiating and configuring view objects, or declaratively by providing an XML layout file and inflating the layout.

• The declarative approach usually is simpler and easier.

• In support of the declarative approach, the system provides a reference to a LayoutInflater and a ViewGroup

from the activity’s layout, which will serve as the parent of your fragment’s layout.

**Example**

res/layout/first\_fragment.xml

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

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android](http://schemas.android.com/apk/res/android)" android:orientation="vertical"

android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:id=["@+id](mailto:@+id)/first\_fragment\_root">

<TextView android:layout\_height="wrap\_content" android:text=["@string](mailto:@string)/text\_first\_fragment\_title" android:layout\_width="match\_parent" android:gravity="center\_horizontal" />

<EditText android:layout\_height="wrap\_content" android:layout\_width="match\_parent" android:id=["@+id](mailto:@+id)/edit\_first\_msg" />

<Button android:layout\_height="wrap\_content" android:layout\_gravity="center\_horizontal" android:layout\_width="wrap\_content" android:id=["@+id](mailto:@+id)/button\_first" android:text=["@string](mailto:@string)/button\_first\_text" />

</LinearLayout>

Then to initialize your fragment’s layout:

public class FirstFragment extends Fragment implements OnClickListener {

[@Override](mailto:@Override)

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

View view = inflater.inflate(R.layout.first\_fragment, container, false);

Button nextButton = (Button) view.findViewById(R.id.button\_first);

nextButton.setOnClickListener(this);

return view;

}

// ...

}

**9 “Statically” Including a Fragment in an Activity Layout**

The easiest way to incorporate a fragment into an activity is by including it directly into the activity’s layout file.

• This works well if the fragment should always be present in the layout.

• However, this approach does not allow you to dynamically remove the fragment at run-time.

In the activity’s layout file, simply use the <fragment> element (yes, that’s really lowercase) where you want to include the fragment.

• Use the android:name attribute to provide the package-qualified class name of the fragment.

• Specify the layout attributes to control the size and position of the fragment.

**Note**

Each fragment requires a unique identifier that the system can use to restore the fragment if the activity is restarted (and which you can use to capture the fragment to perform transactions, such as remove it). There are three ways to provide an ID for a fragment:

• Supply the android:id attribute with a unique ID.

• Supply the android:tag attribute with a unique string.

• If you provide neither of the previous two, the system uses the ID of the fragment’s container view.

For example:

Activity Layout

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

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android](http://schemas.android.com/apk/res/android)" android:orientation="horizontal"

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

<fragment android:name="com.example.news.ArticleListFragment" android:id=["@+id](mailto:@+id)/list"

android:layout\_weight="1" android:layout\_width="0dp" android:layout\_height="match\_parent" />

<fragment android:name="com.example.news.ArticleReaderFragment" android:id=["@+id](mailto:@+id)/viewer"

android:layout\_weight="2" android:layout\_width="0dp" android:layout\_height="match\_parent" />

</LinearLayout>

**10 Dynamically Adding a Fragment to an Activity**

At any time while your activity is running, you can add fragments to your activity layout.

1. First, use Activity.getFragmentManager() to get a reference to the FragmentManager.

2. Invoke FragmentManager.beginTransaction() to get an instance of FragmentTransaction.

3. Instantiate an instance of your fragment.

4. Use the FragmentTransaction.add() to add the fragment to a ViewGroup in the activity, specified by its

ID. Optionally, you can also provide a String tag to identify the fragment.

5. Commit the transaction using FragmentTransaction.commit().

For example:

FragmentManager fragmentManager = getFragmentManager()

// Or: FragmentManager fragmentManager = getSupportFragmentManager() FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction(); ExampleFragment fragment = new ExampleFragment(); fragmentTransaction.add(R.id.fragment\_container, fragment); fragmentTransaction.commit();

**11 Finding Fragments**

The FragmentManager class has methods for finding a fragment contained within an activity:

**findFragmentById(int id)**

Finds a fragment with the specified ID.

**findFragmentByTag(String tag)**

Finds a fragment with the specified tag.

Both of these methods return a reference to the fragment, or null if no matching fragment is found.

**12 Fragment Operations**

You can perform many other operations on dynamic fragments other than adding them to an activity, such as removing them and changing their visibility.

• Each set of changes that you commit to the activity is called a transaction.

• You perform fragment operations using the methods in the FragmentTransaction class. Methods include:

**add()**

Add a fragment to the activity.

**remove()**

Remove a fragment from the activity. This operation destroys the fragment instance unless the transaction is added to the transaction back stack, described later.

**replace()**

Remove one fragment from the UI and replace it with another.

**hide()**

Hide a fragment in the UI (set its visibility to hidden without destroying the view hierarchy).

**show()**

Show a previously hidden fragment.

**detach()** (API 13)

Detach a fragment from the UI, destroying its view hierarchy but retaining the fragment instance.

**attach()** (API 13)

Reattach a fragment that has previously been detached from the UI, re-creating its view hierarchy.

**13 Performing Fragment Transactions**

To perform a fragment transaction:

1. Obtain an instance of FragmentTransaction by calling FragmentManager.beginTransaction().

2. Perform any number of fragment operations using the transaction instance.

1. Call commit() to apply the transaction to the activity.

For example:

FragmentManager fragmentManager = getFragmentManager()

// Or: FragmentManager fragmentManager = getSupportFragmentManager()

fragmentManager.beginTransaction()

.remove(fragment1)

.add(R.id.fragment\_container, fragment2)

.show(fragment3)

.hide(fragment4)

.commit();

The order in which you add changes to a FragmentTransaction doesn’t matter, except:

• You must call commit() last.

• If you’re adding multiple fragments to the same container, then the order in which you add them determines the order they appear in the view hierarchy.

**14 Communication Between the Activity and the Fragment**

A given instance of a fragment is directly tied to the activity that contains it.

We can send data from Activity to Fragment using bundle.

In Activity:

Bundle bundle=new Bundle();

bundle.putString(“key”,”sending data”);

FragmentOne frag=new FragmentOne();

frag.setArguments(bundle);

In Fragment:

String text=getArguments().getString(“key”);

**15 Additional Fragment Subclasses**

There are some additional subclasses of Fragment designed for common uses:

**ListFragment**

A fragment that automatically manages a ListView. Analogous to the ListActivity class.

**PreferenceFragment**

A fragment that automatically manages a set of Preference objects. Using an XML preference resource, the fragment can automatically create an interface for displaying and editing a set of preferences. Analogous to the PreferenceActivity class.

**WebFragment**

A fragment that automatically creates and manages a WebView.