FRAGMENTS

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

FRAGMENTS

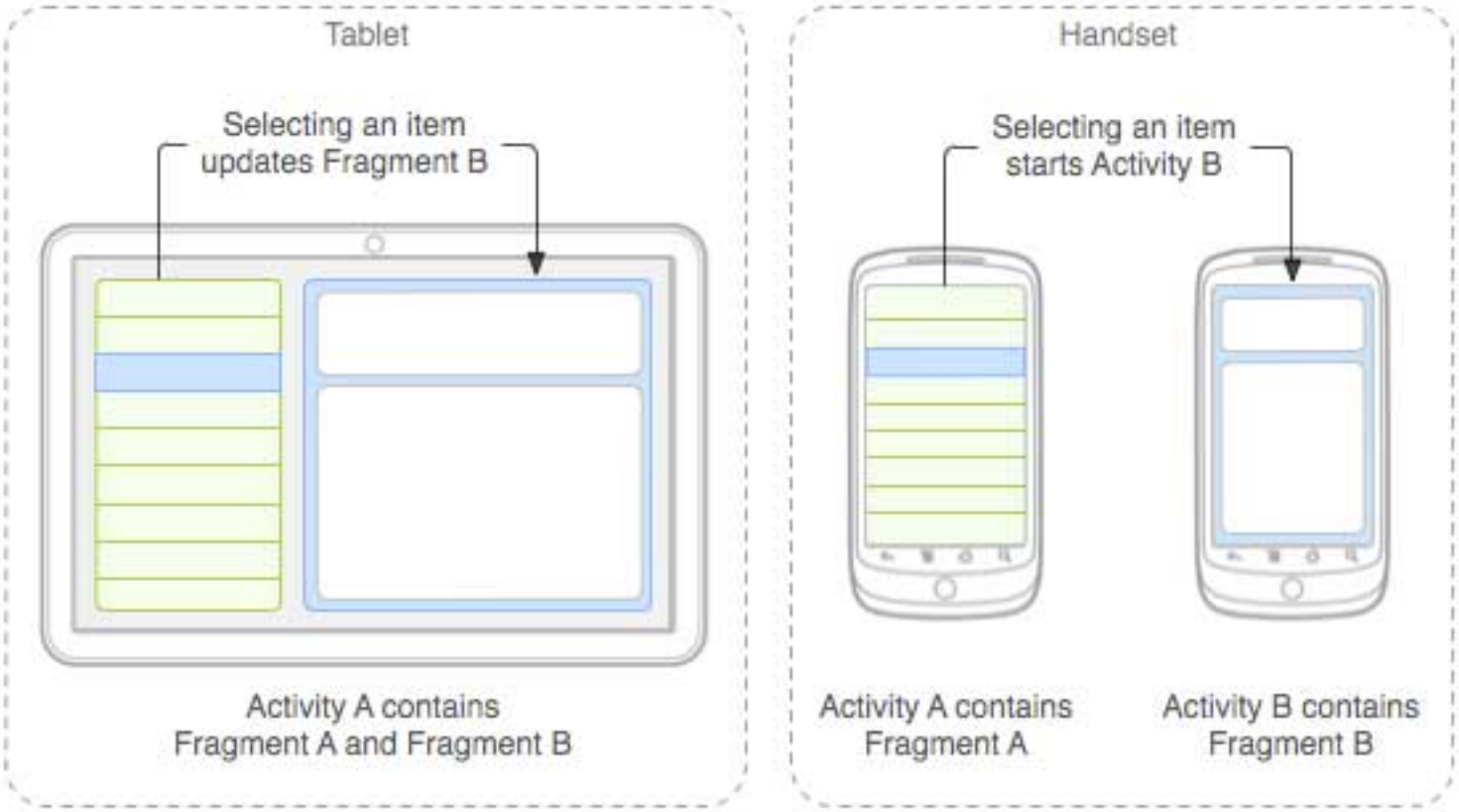
A **Fragment** represents a behavior or a portion of user interface in an **Activity**.

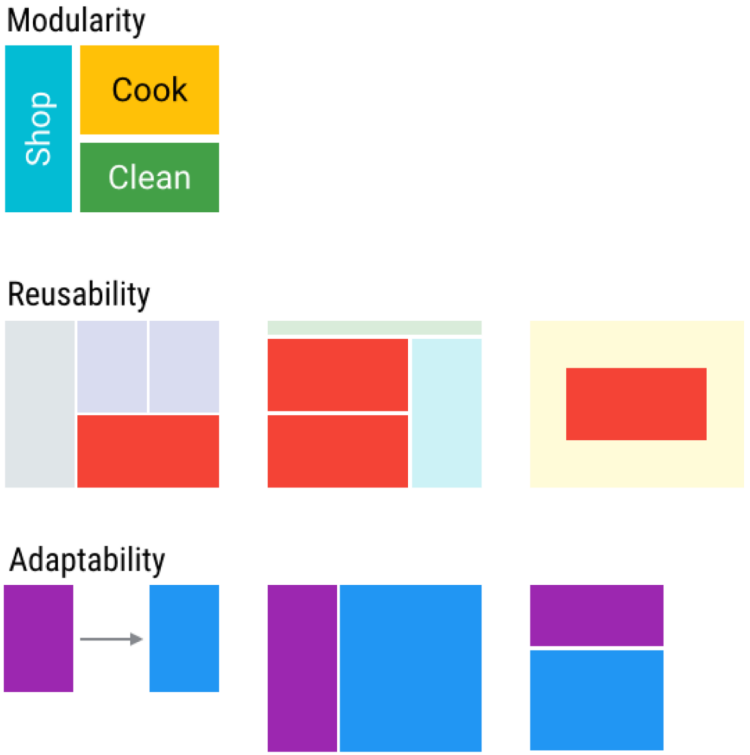
You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

Fragments were added to the Android API in Honeycomb version of Android which API version 11.

USING FRAGMENTS

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.





You don't *have* to use fragments.

However, if you do use them and use them well, they can provide:

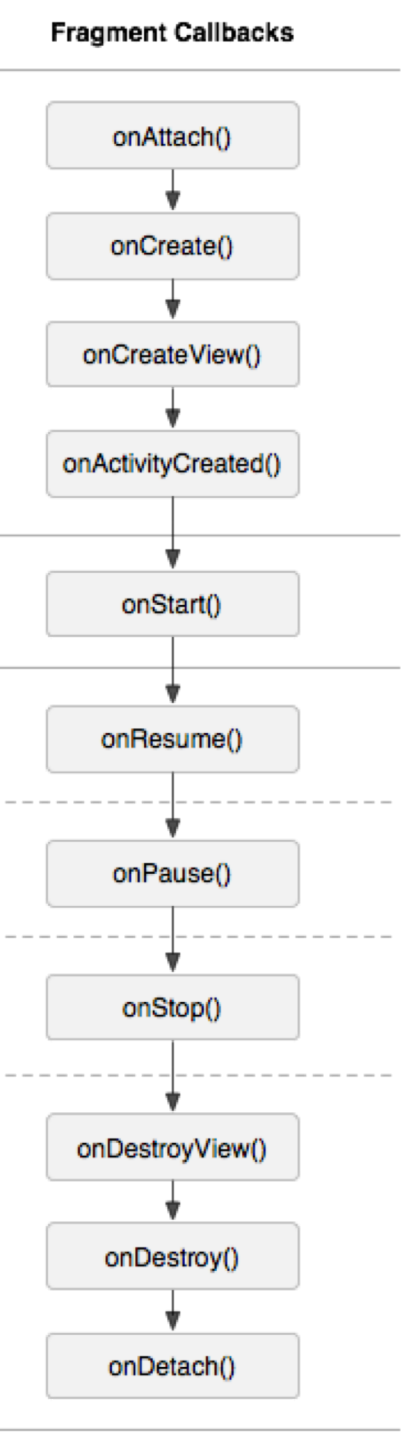
- **Modularity:** dividing complex activity code across fragments for better organization and maintenance.
- **Reusability:** placing behavior or UI parts into fragments that can be shared across multiple activities.
- **Adaptability:** representing sections of a UI as different fragments and utilizing different layouts depending on screen orientation and size.

Design for reuse.

You should design each fragment as a modular and reusable activity component.

FRAGMENTS – KEY POINTS

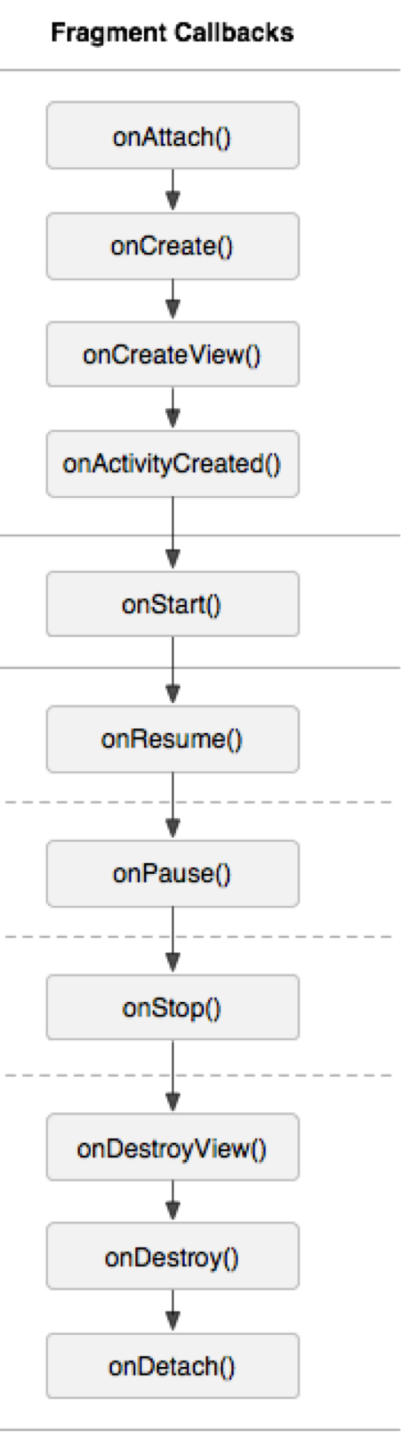
- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- Fragment life cycle is closely related to the life cycle of its host activity e.g. when the activity is paused, all the fragments available in the activity will also be stopped.
- You can add or remove fragments in an activity while the activity is running.
- Fragments and activities can communicate.



FRAGMENT LIFECYCLE

The **Fragment** class has code that looks a lot like an **Activity**.

It contains callback methods similar to an activity, such as **onCreate()**, **onStart()**, **onPause()**, and **onStop()**.



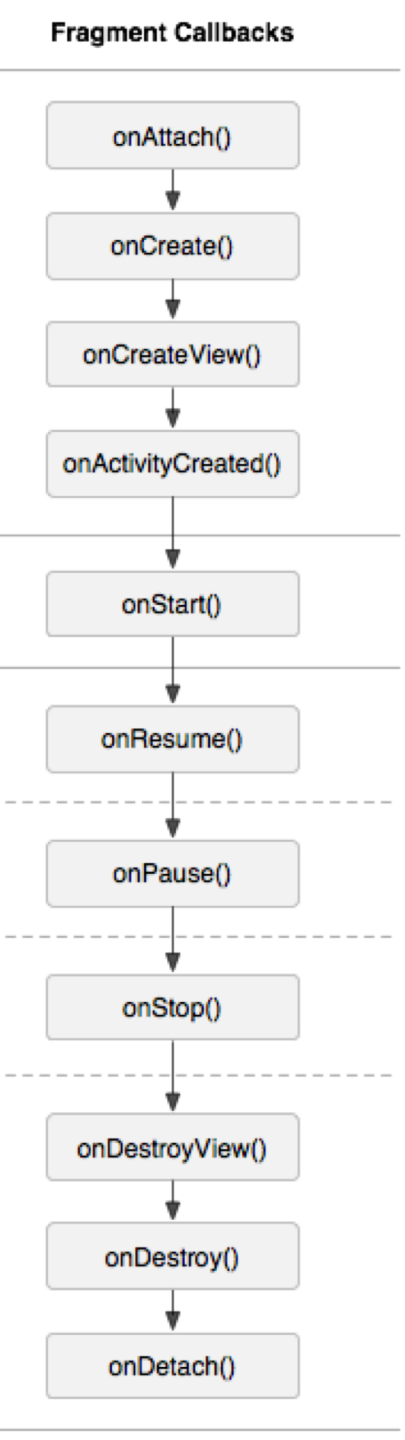
FRAGMENT LIFECYCLE

The **Fragment** class has code that looks a lot like an **Activity**.

It contains callback methods similar to an activity, such as **onCreate()**, **onStart()**, **onPause()**, and **onStop()**.

onCreate()

The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.



FRAGMENT LIFECYCLE

The **Fragment** class has code that looks a lot like an **Activity**.

It contains callback methods similar to an activity, such as **onCreate()**, **onStart()**, **onPause()**, and **onStop()**.

onPause()

The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

HANDLING THE FRAGMENT LIFECYCLE

Managing the lifecycle of a fragment is a lot like managing the lifecycle of an activity.

Like an activity, a fragment can exist in three states:

Resumed

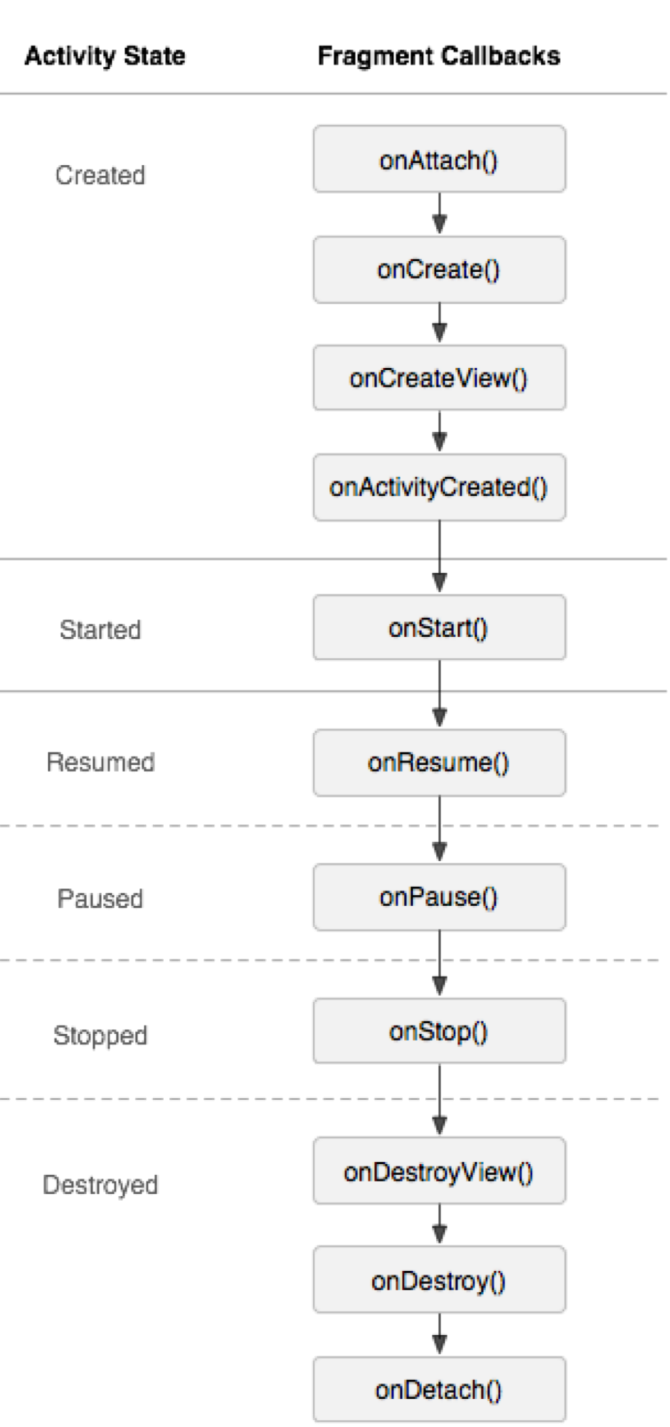
The fragment is visible in the running activity.

Paused

Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).

Stopped

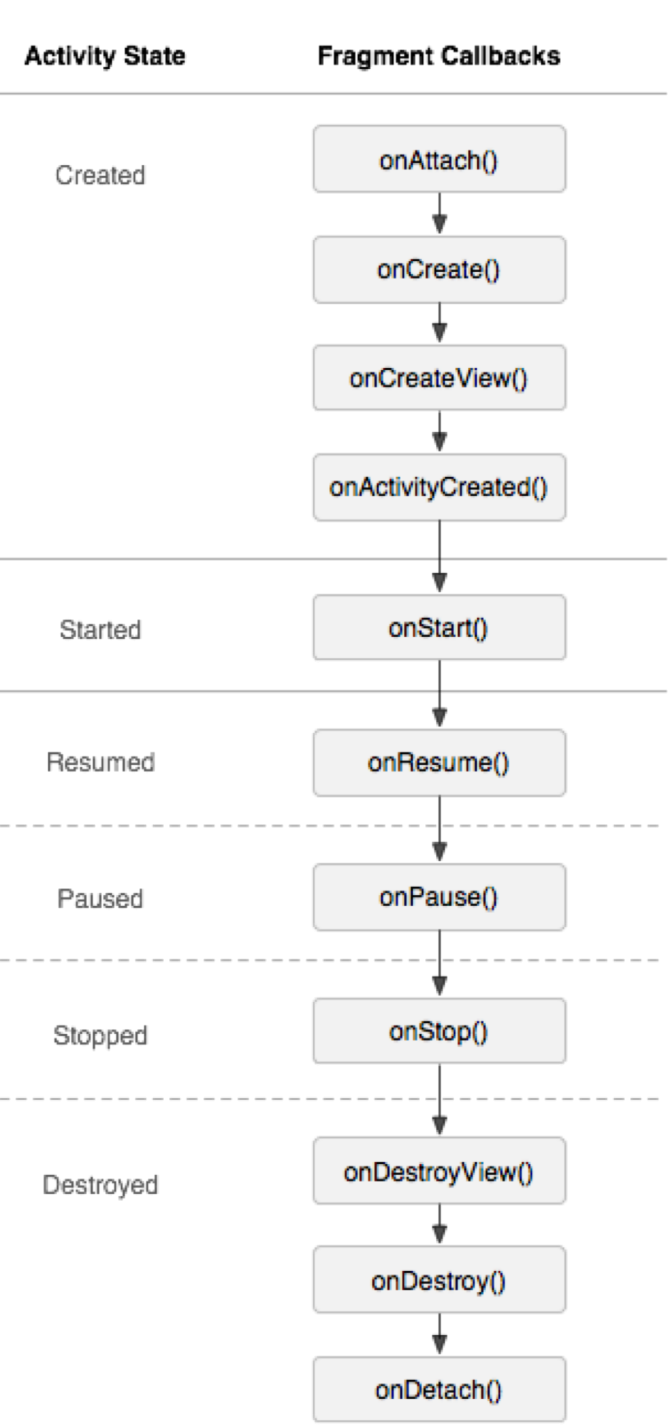
The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed.



COORDINATING WITH THE ACTIVITY

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

Some callbacks correspond to the activity lifecycle, e.g. **onPause()**.



COORDINATING WITH THE ACTIVITY

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

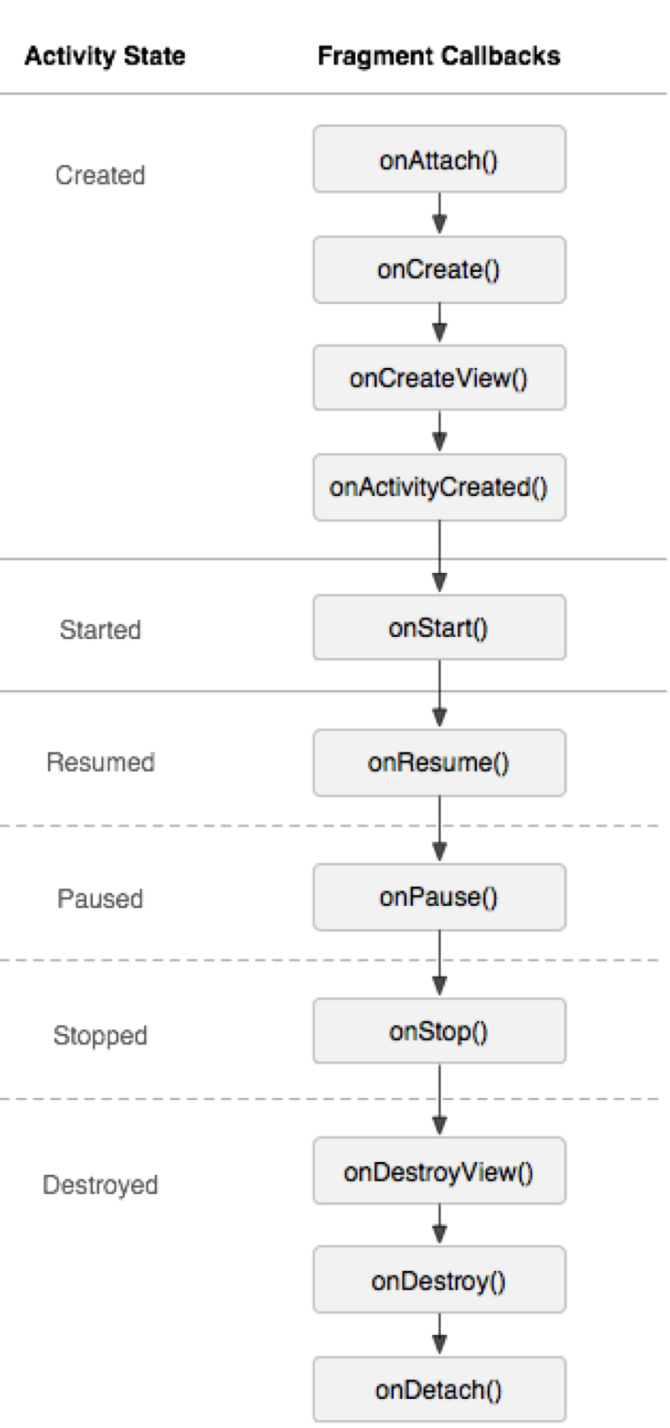
Fragments have some additional callbacks:

onAttach() Called when the fragment has been associated with the activity (the Activity is passed in here).

onActivityCreated() 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.



COORDINATING WITH THE ACTIVITY

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

onCreateView()

Called to create the view hierarchy associated with the fragment.

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout.

You can return null if the fragment does not provide a UI.

CREATING FRAGMENTS

You create fragments by extending **Fragment** class.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

The inflate() method takes three arguments:

1. The resource ID of the layout you want to inflate.

CREATING FRAGMENTS

You create fragments by extending **Fragment** class.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

The inflate() method takes three arguments:

2. The ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going.

CREATING FRAGMENTS

You create fragments by extending **Fragment** class.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

The inflate() method takes three arguments:

1. A boolean indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.

ADD A FRAGMENT TO AN ACTIVITY

While fragments are reusable, modular UI components, each instance of a **Fragment** class must be associated with a parent **FragmentActivity** (or a regular **Activity** for more recent implementations).

You can insert a fragment into your activity layout by:

- declaring the fragment in the activity's layout file, as a **<fragment>** element.
- or from your application code by adding it to an existing **ViewGroup**.

When you add a fragment as a part of your activity layout, it lives in a **ViewGroup** inside the activity's view hierarchy and the fragment defines its own view layout.

Declare the fragment inside the activity's layout file.

In this case, you can specify layout properties for the fragment as if it were a view. For example, here's the layout file for an activity with two fragments:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns: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/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Declare the fragment inside the activity's layout file.

In this case, you can specify layout properties for the fragment as if it were a view. For example, here's the layout file for an activity with two fragments:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.a
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_pare
    <fragment android:name="com.example.news.
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_pare
</LinearLayout>
```

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 container view.

Programmatically add the fragment to an existing ViewGroup.

At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment.

To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction. You can get an instance of FragmentTransaction from your Activity like this:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Then you add a fragment using the **add()** method, specifying the fragment to add and the view in which to insert it.

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

COMMUNICATING WITH THE ACTIVITY

Although a Fragment is implemented as an object that's independent from an Activity and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that contains it.

Specifically, the fragment can access the Activity instance with getActivity() and easily perform tasks such as find a view in the activity layout:

```
View listView = getActivity().findViewById(R.id.list);
```

Likewise, your activity can call methods in the fragment by acquiring a reference to the **Fragment** from **FragmentManager**,
using **findFragmentById()** or**findFragmentByTag()**.

```
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fragment);
```

COMMUNICATING WITH THE ACTIVITY

You can also create a custom listener interface.

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

FURTHER DETAILS

Android Developer guide to fragments:

https://developer.android.com/guide/components/fragments.html#Lifecycle

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.