Fragments

Fragments

Fragment are Android's solution to creating reusable user interfaces

Official Guide to Fragments

Fragment

An activity is a container for views

When you have a larger screen device than a phone –like a tablet it can look too simple to use phone interface here.

→ Fragments

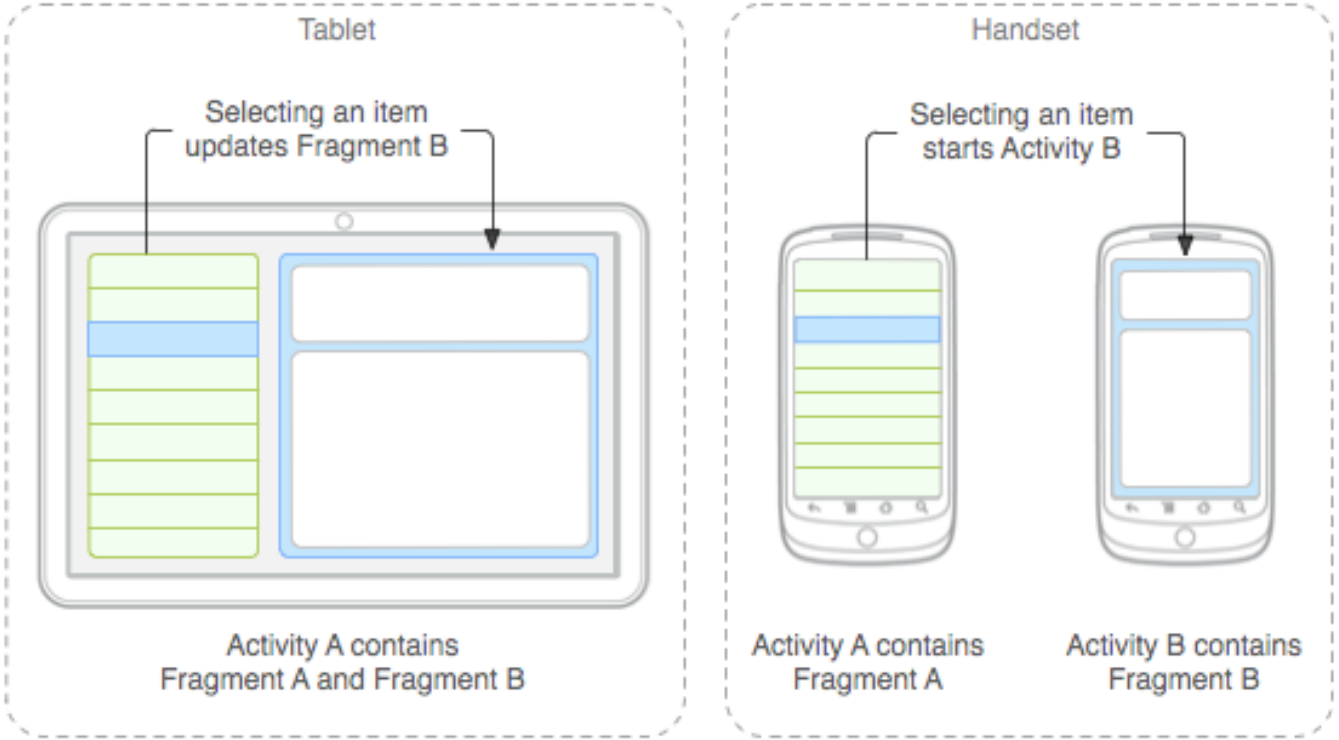
Mini-activities, each with its own set of views

One or more fragments can be embedded in an Activity

You can do this dynamically as a function of the device type (tablet or not) or orientation

ALSO, you can reuse fragments --- like reuse of mini interfaces

Use Case: Different Form Factors



Fragment

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.

You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

Use Case: Integrating third-party UIs

Rich third-party UI functionality can be shared as Fragments.

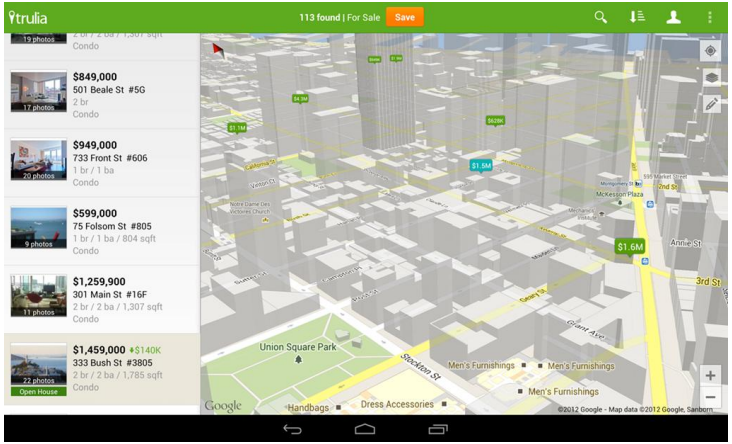


Figure 1: The Trulia Android App integrates Google Maps via [MapFragment] class.

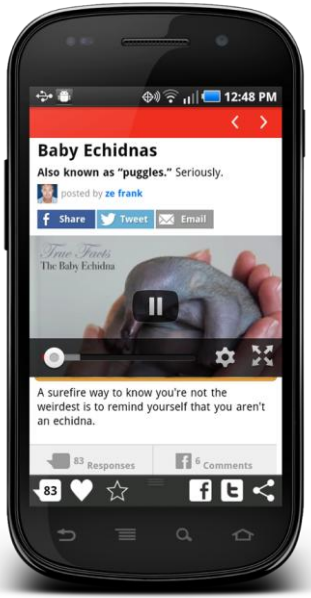


Figure 2: The BuzzFeed Android App integrates YouTube API via the [YouTubePlayerFragment] class.

Fragments vs. Activities

- A fragment is part of an activity, which contributes its own UI to that activity. Fragment can be thought of as a subactivity.
- A fragment can't exist independently. It's always part of an activity.

Fragment LifeCycle

Steps to bring up fragment:

1. onAttach(Activity) called once the fragment is associated with its activity.

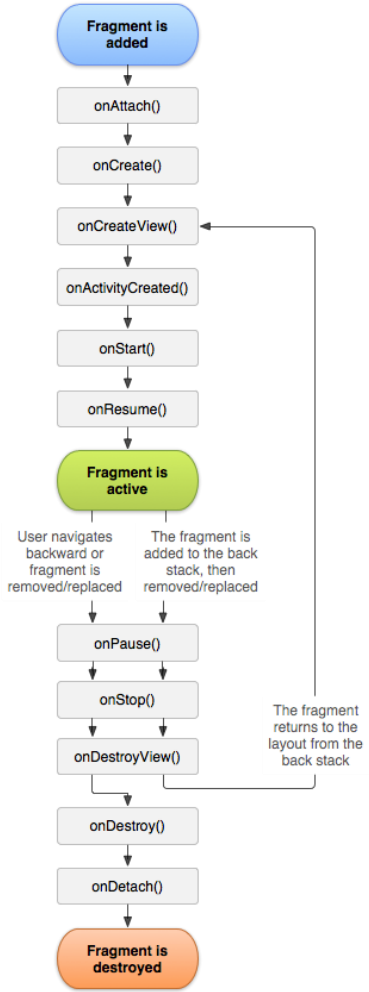
2. **onCreate(Bundle) called to do initial creation of the fragment.**

3. **onCreateView(LayoutInflater, ViewGroup, Bundle) creates and returns the view hierarchy associated with the fragment.**

4. onActivityCreated(Bundle) tells the fragment that its activity has completed its own Activity.onCreate().

5. onViewStateRestored(Bundle) tells the fragment that all of the saved state of its view hierarchy has been restored.

6. onStart() makes the fragment visible to the user (based on its containing activity being started).



Fragment LifeCycle

Steps to destroy fragment:

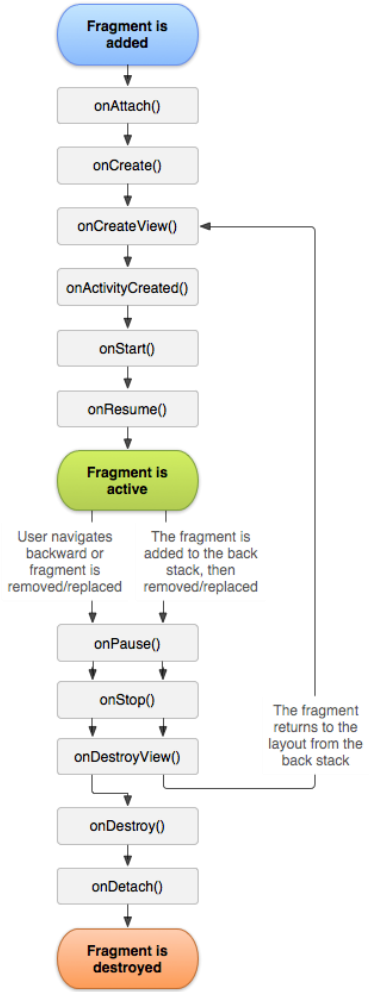
1. **onPause() fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.**

2. onStop() fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.

3. onDestroyView() allows the fragment to clean up resources associated with its View.

4. onDestroy() called to do final cleanup of the fragment's state.

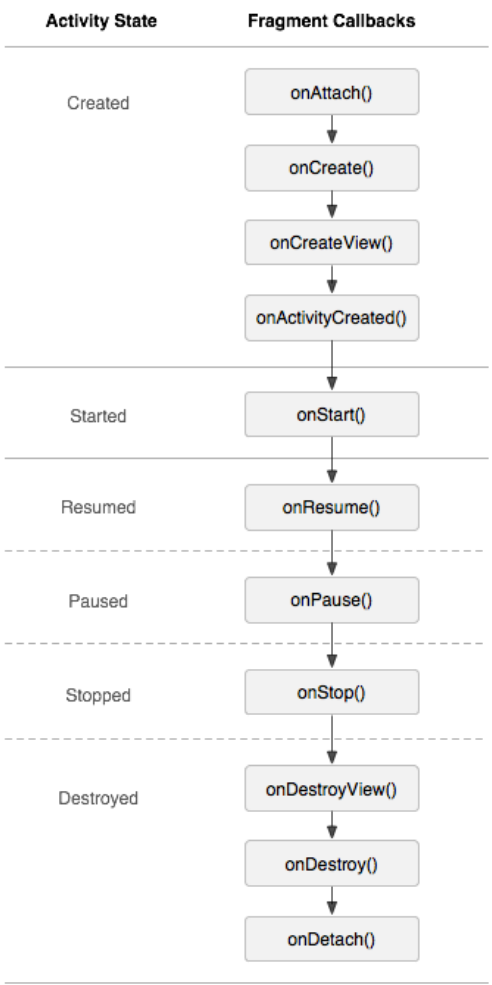
5. onDetach() called immediately prior to the fragment no longer being associated with its activity.



Fragment States

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.
- Stopped: The fragment is not visible.



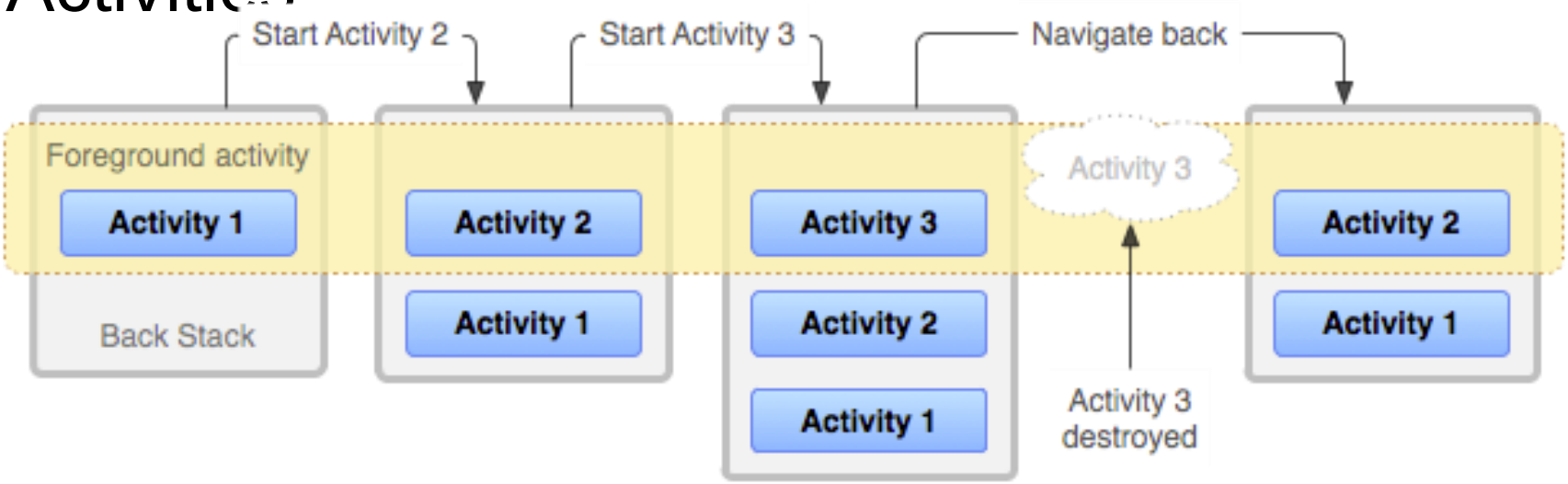
Boilerplate Fragment Code

Android Studio's "New Activity" wizards contain boilerplate code for the following:

- BlankActivity with Fragment
- Google Maps Fragment
- TabbedActivity
- Master/Detail Flow

Back Stack

Recall: Standard back button behavior with Activities



Back Stack and Fragments

- Unlike Activities, Fragment transitions are not automatically put on the back stack.
- Fragment transitions can manually be put on back stack.
- When the user presses back in the activity, any transactions on the back stack are popped off before the activity itself is finished.
- Code Example

Concept: Adding a Fragment

You can add a Fragment either:

- Declare the fragment inside the activity's layout file.
- Programmatically add the fragment using a FragmentManager

Concept: Constructor

Every fragment must have an **empty** constructor, so it can be instantiated when restoring its activity's state.

Arguments can be supplied by the caller with `setArguments(Bundle)` and later retrieved by the Fragment with `getArguments()`.

Concept: Fragment Tags

A Fragment's ID specifies the ID of a container to insert the Fragment into.

A container may host many different Fragments over time.

Use tags to uniquely identify the fragment.

Concept: Fragment Communication

- Define custom callback interfaces
- In Fragment's onAttach, save reference to callback
- Hosting Activity implements callback interface

Communicating with Other Fragments

Concept: FragmentManager

Allows you to:

- Open a FragmentTransaction to add or remove fragments
- Get fragments in activity (via findFragmentById() or findFragmentByTag())

Fragment inside Activity

- it lives in a ViewGroup inside the activity's view hierarchy
- fragment has its own view layout.
- via XML: Insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element,
- via CODE: from your application code by adding it to an existing ViewGroup.
- you may also use a fragment without its own UI as an invisible worker for the activity.

Fragment – extend a Fragment class

- via CODE: extend android.app.Fragment OR one of its subclasses (DialogFragment, ListFragment, PreferenceFragment, WebViewFragment)

- IMPORTANT: must include a public empty constructor. The framework will often re-instantiate a fragment class when needed, in particular during state restore, and needs to be able to find this constructor to instantiate it. If the empty constructor is not available, a runtime exception will occur in some cases during state restore.

- CALL Back functions (like Activity) : examples onCreate(), onStart(), onPause(), and onStop().

Fragment methods (callback functions)

onAttach(Activity) called once the fragment is associated with its activity.

onCreate(Bundle) called to do initial creation of the fragment.

onCreateView(LayoutInflater, ViewGroup, Bundle) creates and returns the view hierarchy associated with the fragment.

onActivityCreated(Bundle) tells the fragment that its activity has completed its own Activity.onCreaate.

onStart() makes the fragment visible to the user (based on its containing activity being started).

onResume() makes the fragment interacting with the user (based on its containing activity being resumed).

Fragment methods (callback functions)

As a fragment is no longer being used, it goes through a reverse series of callbacks:

onPause() fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.

onStop() fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.

onDestroyView() allows the fragment to clean up resources associated with its View.

onDestroy() called to do final cleanup of the fragment's state.

onDetach() called immediately prior to the fragment no longer being associated with its activity.

Create your own Fragment class or use known sub-classes

- DialogFragment Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the Activity class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.
- ListFragment Displays a list of items that are managed by an adapter (such as a SimpleCursorAdapter), similar to ListActivity. It provides several methods for managing a list view, such as the onListItemClick() callback to handle click events.
- PreferenceFragment Displays a hierarchy of Preference objects as a list, similar to PreferenceActivity. This is useful when creating a "settings" activity for your application.

Fragments and their UI

- Most fragments will have a UI
- Will have its own layout
- you must implement the onCreateView() callback method, which the Android system calls when it's time for the fragment to draw its layout. Your implementation of this method must return a View that is the root of your fragment's layout.

•Note some subclasses of Fragment like ListFragment have already implemented this method and you don’t need to override.

Fragments and their UI – onCreateView() using XML

Can implement onCreateView using XML

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }

}
```

Activity parent's ViewGroup

Bundle that provides data about the previous instance of the fragment, if the fragment is being resumed

Have *example_fragment.xml* file that contains the layout
This will be contained in resource layout folder.

OPTION1 –adding to an Activity via Activity layout XML.

```
<?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>
```

Need unique ids for each so system can restore the fragment if the activity is restarted

OPTION2 –creating and adding to an Activity via CODE.

```
/*Inside Activity Code where you want to add Fragment (dynamically anywhere or in  onCreate() callback)
*/
//get FragmentTransaction associated with this Activity
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

//Create instance of your Fragment
ExampleFragment fragment = new ExampleFragment();

//Add Fragment instance to your Activity
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

This points to the Activity ViewGroup in which the fragment should be placed, specified by resource ID

OPTION 3- Adding Fragment that has NO UI using Code

use a fragment to provide a background behavior for the activity without presenting additional UI.

use add(Fragment, String) (supplying a unique string "tag" for the fragment, rather than a view ID).

it's not associated with a view in the activity layout, it does not receive a call to onCreateView(). So you don't need to implement that method.

If you want to get the fragment from the activity later, you need to use findFragmentByTag().

For an example activity that uses a fragment as a background worker, without a UI, see the FragmentRetainInstance.java sample.

Managing Fragments

FragmentManager methods:

Get fragments that exist in Activity =

findFragmentById() (for fragments that provide a UI in the activity layout)

findFragmentByTag() (for fragments that do or don't provide a UI).

Pop fragments off the back stack,

popBackStack() (simulating a *Back* command by the user).

Register a listener for changes to the back stack,

addOnBackStackChangedListener().

Fragment Transactions – adding, removing and replacing dynamically

```
// Create new fragment and transaction
 Fragment newFragment = new ExampleFragment();
 FragmentTransaction transaction getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment
// and add the transaction to the back stack
  transaction.replace(R.id.fragment_container, newFragment);
  transaction.addToBackStack(null);

// Commit the transaction
   transaction.commit();
```

newFragment replaces whatever fragment (if any) is currently in the layout container identified by the R.id.fragment_container

If you do not call addToBackStack() when you perform a transaction that removes a fragment, then that fragment is destroyed when the transaction is committed and the user cannot navigate back to it. Whereas, if you do call addToBackStack() when removing a fragment, then the fragment is *stopped* and will be resumed if the user navigates back.

Example ---ListFragment

Taken from http://www.techrepublic.com/blog/app-builder/get-started-with-android-fragments/1050

Questions?