Fragments

Jestin James M
Assistant Professor, Dept of Computer Science
Little Flower College, Guruvayoor

Fragments

- Introduced In Android 3.0
- One way to think of a fragment is as a sub-activity
- A fragment can have a view hierarchy associated with it
- it has a lifecycle much like an activity's lifecycle
- "If only I could put multiple activities together on a tablet's screen at the same time,"

When to Use Fragments

- you can reuse a chunk of user interface and functionality across devices and screen sizes.
- you can have a list and a detail view of the selected item on screen at the same time.
- This is easy to picture in a landscape orientation with the list on the left and the details on the right
- But what if the user rotates the device to portrait mode

When to Use Fragments

- list to be in the top portion of the screen and the details in the bottom portion

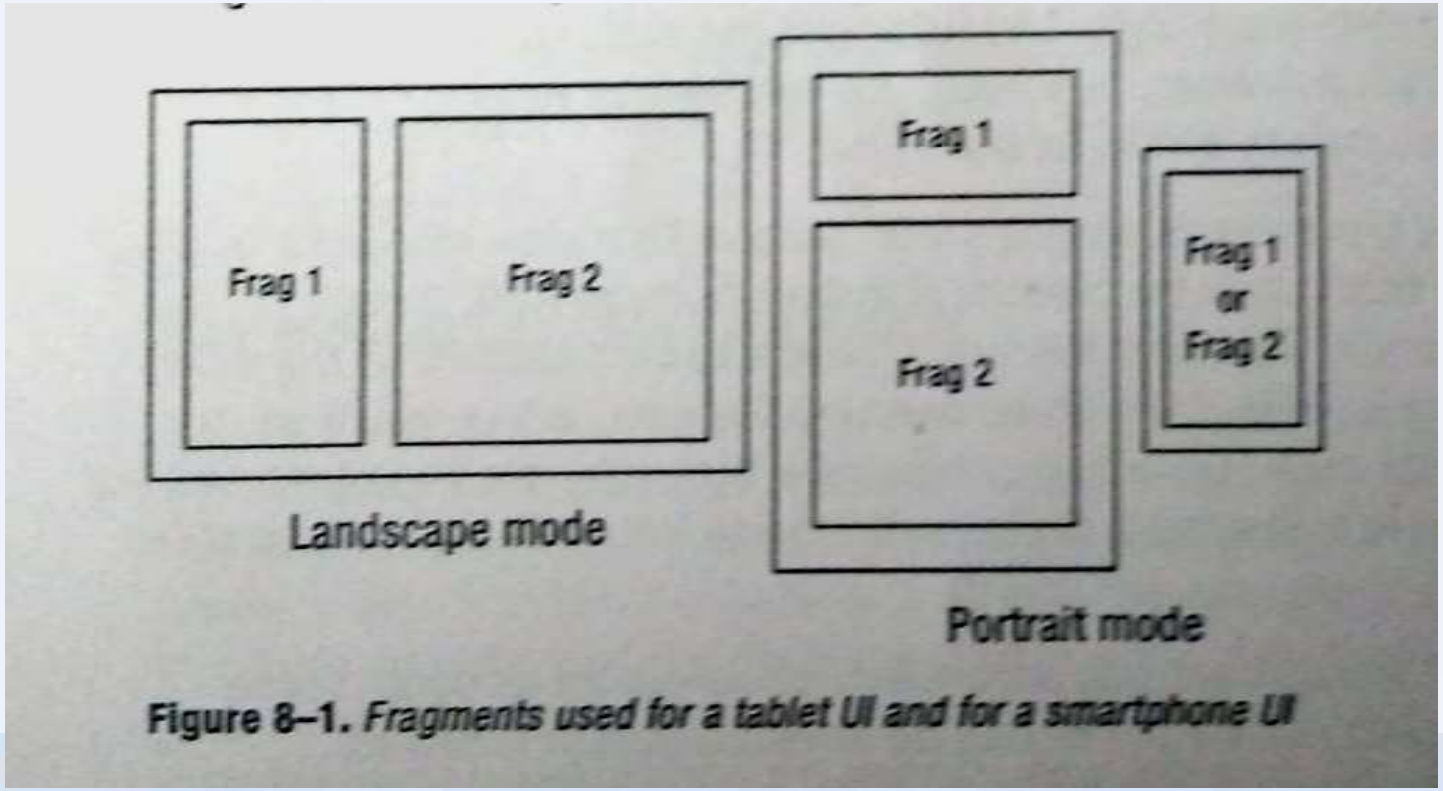


Figure 8–1. *Fragments used for a tablet UI and for a smartphone UI*

When to Use Fragments

- ach fragment will have its own layout that can be reused across many configurations.
- imagine that the user interface has changed within the same activity, and the user wants to go back a step, or two, or three
- activity, pressing the Back button will take the user out of the activity entirely
- With fragments, the Back button can step backward through a stack of fragments while staying inside the current activity

The Structure of a Fragment

- A fragment can have a view hierarchy to engage with a user
- any other view hierarchy in that it can be created (inflated) from an XML layout specification or created in code
- everything you know about views applies to fragments as well

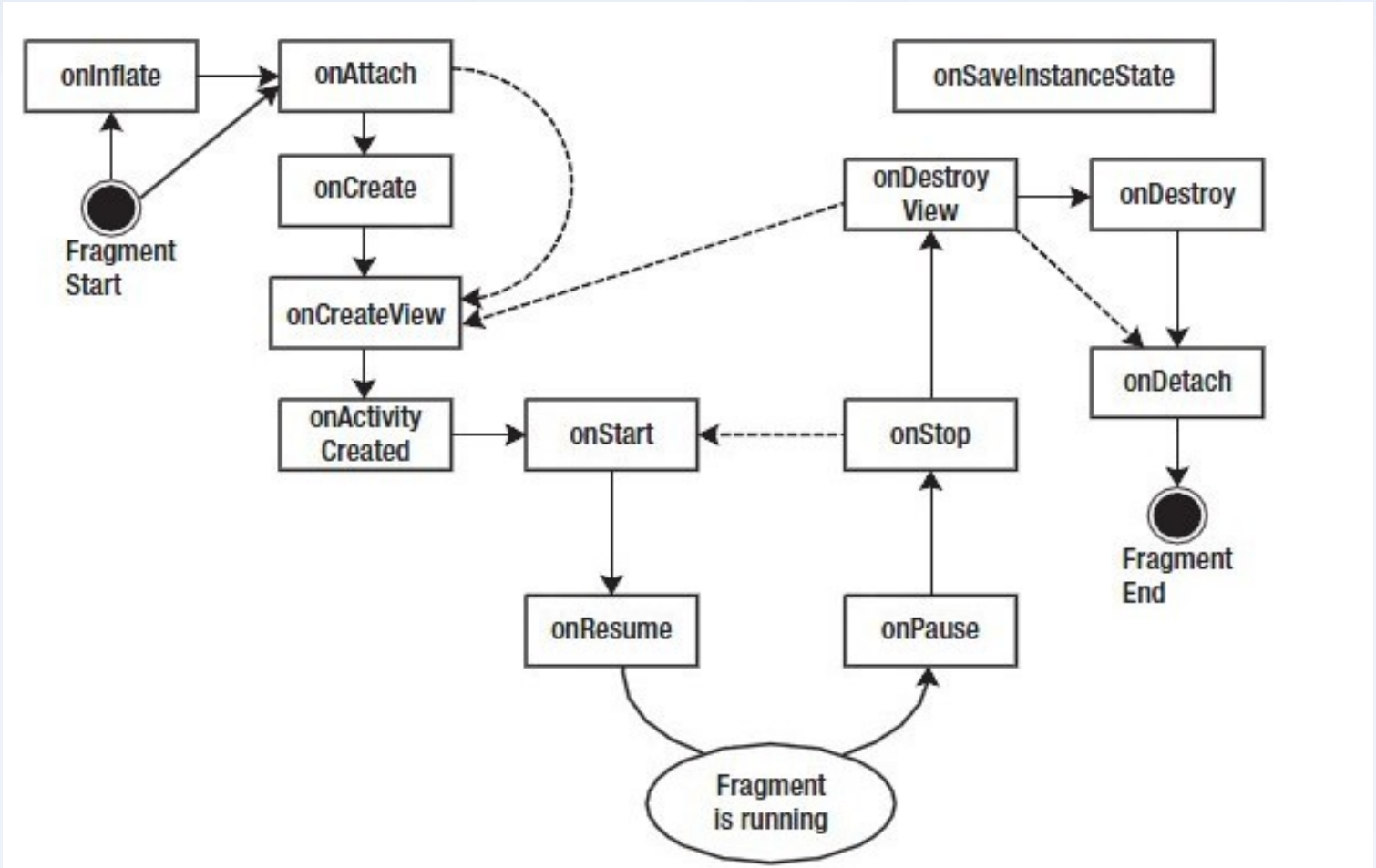
The Structure of a Fragment

- Similar to an activity, a fragment can be saved and later restored automatically by the system
- When the system restores a fragment, it calls the default constructor (with no arguments)
- and then restores this bundle of arguments to the newly created fragment
- An activity can have multiple fragments in play at one time

The Structure of a Fragment

- if a fragment has been switched out with another fragment,
- the fragment-switching transaction can be saved on a back stack.
- The back stack is managed by the fragment manager tied to the activity.
- The back stack is how the Back button behavior is managed

A Fragment's Lifecycle



A Fragment's Lifecycle

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 creates and returns the view hierarchy associated with the fragment.
4. onActivityCreated tells the fragment that its activity has completed its own Activity onCreate().

A Fragment's Lifecycle

5. onStart() makes the fragment visible to the user (based on its containing activity being started
6. onResume() makes the fragment begin interacting with the user
7. 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.

8. 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.
9. onDestroyView() allows the fragment to clean up resources associated with its View.
10. onDestroy() called to do final cleanup of the fragment's state.
11. onDetach() called immediately prior to the fragment no longer being associated with its activity.

A Fragment’s Lifecycle

A Fragment's Lifecycle

- At the very beginning, a fragment is instantiated.
- It now exists as an object in memory
- The first thing that is likely to happen is that initialization arguments will be added to your fragment object
- When the system is restoring a fragment from a saved state,
- the default constructor is invoked, followed by the attachment of the initialization arguments bundle

The onInflate() Callback

- If your fragment is defined by a <fragment> tag in a layout that is being inflated
- your fragment's onInflate() callback is called
- when an activity has called setContentView() for its main layout
- This passes in the activity an AttributeSet with the attributes from the <fragment> tag, and a saved bundle.

The onAttach() Callback

- The onAttach() callback is invoked after your fragment is associated with its activity.
- One thing to note is that the Fragment class has a getActivity() method
- It will return the attached activity for your fragment should you need it.

The onCreate() Callback

- similar to the activity's onCreate(),
- you should not put code in here that relies on the existence of the activity's view hierarchy
- This callback gets the saved state bundle passed in, if there is one.
- Your fragment code is running on the UI thread

The onCreateView() Callback

- you will return a view hierarchy for this fragment.
- The arguments passed in to this callback include a LayoutInflater
- The parent is provided so you can use it with the inflate() method of the LayoutInflater.

The onActivityCreated() Callback

- This is called after the activity has completed its onCreate() callback
- You can now trust that the activity's view hierarchy, including your own view hierarchy
- It's also where you can be sure that any other fragment for this activity has been attached to your activity.

The onStart() Callback

- Now your fragment is visible to the user
- you haven’t started interacting with the user just yet
- This callback is tied to the activity’s onStart().

The onResume() Callback

- The last callback before the user can interact with your fragment is onResume()
- This callback is tied to the activity's onResume().
- When this callback returns, the user is free to interact with this fragment
- For example, if you have a camera preview in your fragment, you would probably enable it in the fragment's onResume().

The onPause() Callback

- The first undo callback on a fragment is onPause().
- This callback is tied to the activity's onPause()
- you don't want to be playing audio if the user is taking a phone call.

The onSaveInstanceState() Callback

- fragments have an opportunity to save state for later reconstruction.
- This callback passes in a Bundle object to be used as the container for whatever state information you want to hang onto
- To prevent memory problems, be careful about what you save into this bundle
- Only save what you need.
- If you need to keep a reference to another fragment, save its tag instead of trying to save the other fragment.

The onStop() Callback

- This one is tied to the activity's onStop()
- a purpose similar to an activity's onStop().
- A fragment that has been stopped could go straight back to the onStart() callback, which then leads to onResume().

The onDestroyView() Callback

- If your fragment is on its way to being killed off or saved, the next callback in the undo direction is onDestroyView()
- This will be called after the view hierarchy you created on your onCreateView() callback

The onDestroy() Callback

- This is called when the fragment is no longer in use.
- that it is still attached to the activity and is still findable, but it can’t do much.

The onDetach() Callback

- The final callback in a fragment's lifecycle is onDetach().
- Once this is invoked, the fragment is not tied to its activity,
- it does not have a view hierarchy anymore, and all its resources should have been released.

Using setRetainInstance()

- being re-created and therefore your fragments will be coming back also.
- Therefore, fragment comes with a method called setRetainInstance(),