UI
Fragment

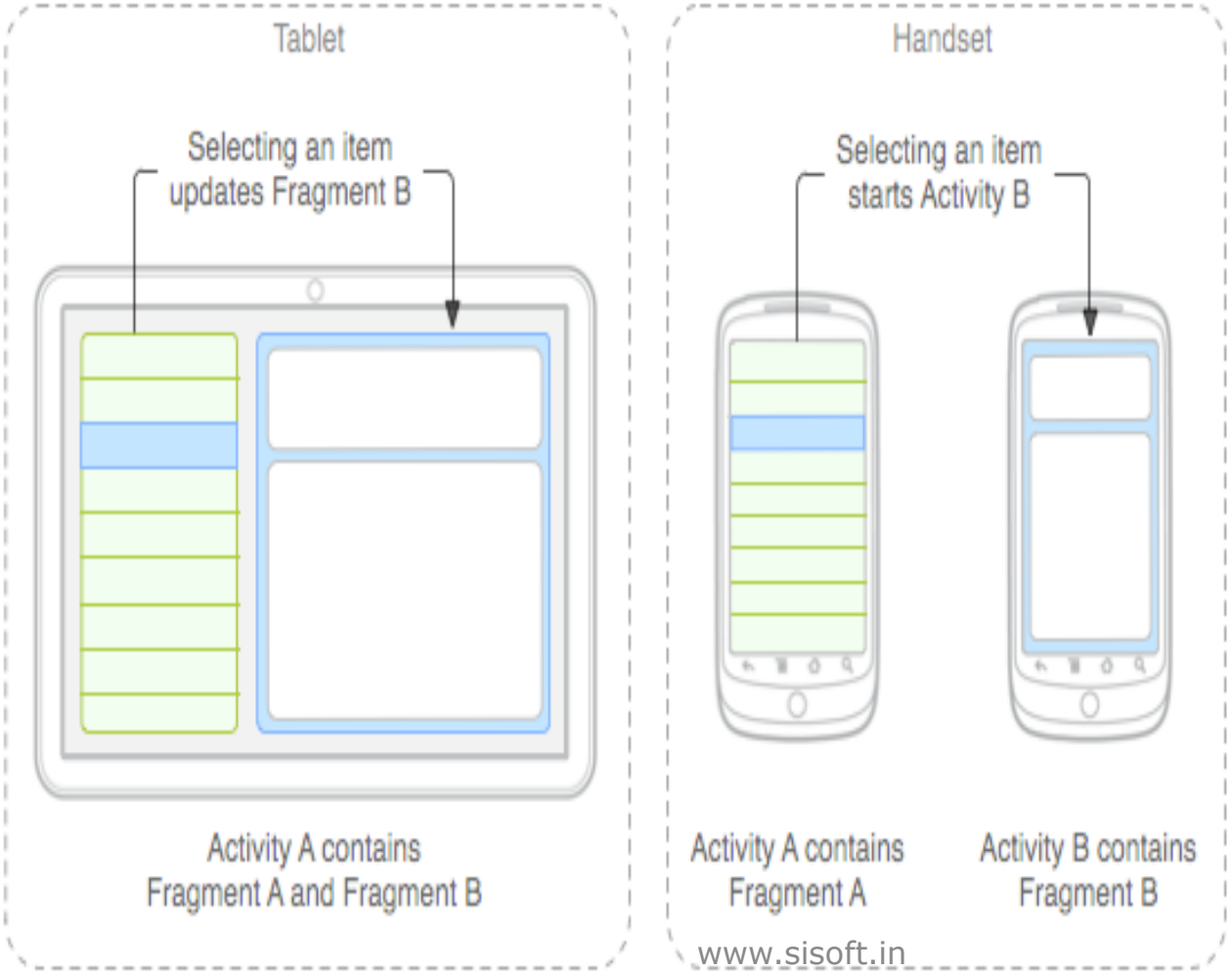
Contents

- Fragments
  - Overviews
  - Lifecycle of Fragments
  - Creating Fragments
  - Fragment Manager and Transactions
  - Adding Fragment to Activity
  - Fragment-to-Fragment Communication
  - Fragment SubClasses

Fragment

- A Fragment represents a behavior or a portion of user interface in an Activity.
- Multiple fragments can be combined in a single activity to build a multi-pane UI and a fragment can be reused in multiple activities.
- A fragment is a modular section of an activity, which has its own lifecycle ,receives its own input events and which can be added or removed while the activity is running (sort of like a "sub activity" that you can reuse in different activities).
- The “Minimum Required Sdk” should be the API 11 or larger, because Android introduced fragments in Android 3.0 – Honeycomb (API level 11).

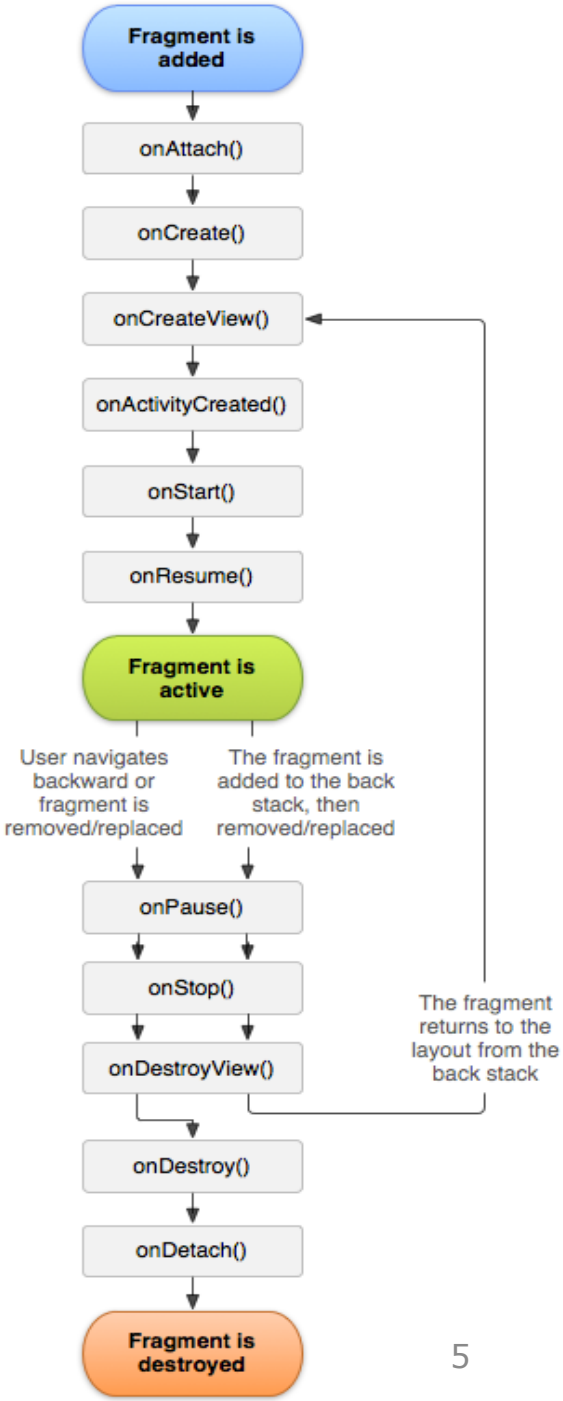
Fragment Idea



You might decide to run a tablet in portrait mode with the handset model of only one fragment in an Activity

Fragment Lifecycle

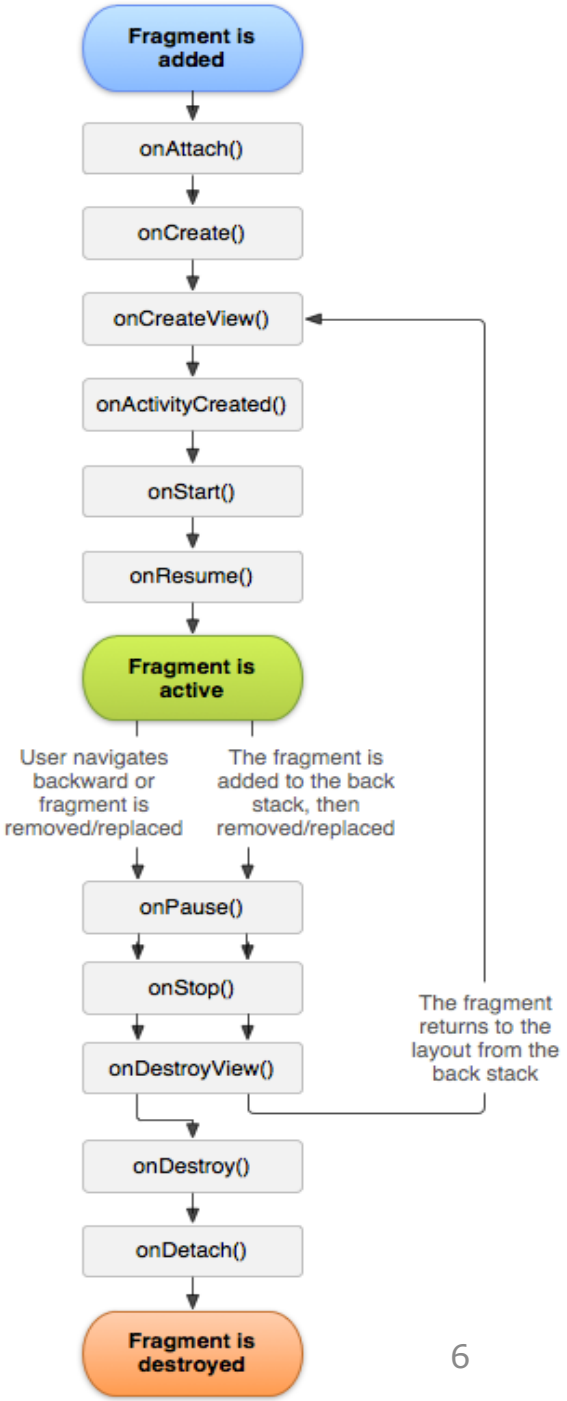
- onAttach(Activity) called once the fragment is associated with its activity.
- onCreate(Bundle) called to do initial creation of the fragment.
- onCreateView(LayoutInflater, ViewGroup, Bundle) creates and returns the view hierarchy associated with the fragment.
- onActivityCreated(Bundle) tells the fragment that its activity has completed its own Activity.onCreate.
- 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).

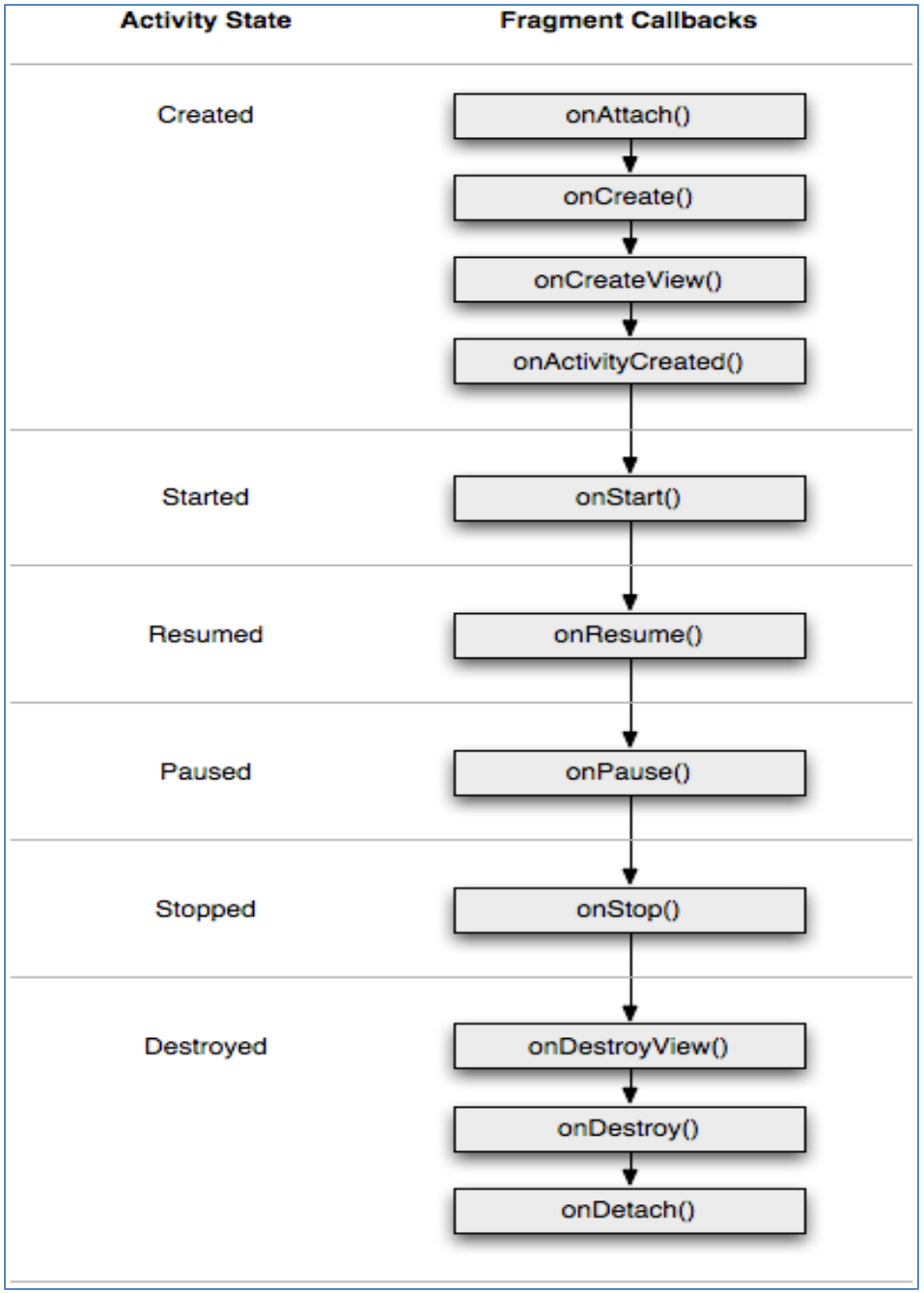


Fragment Lifecycle

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.





- Activity Lifecycle influences
  - –Activity paused: all its fragments paused
  - –Activity destroyed : all its fragments destroyed
  - –Activity running: manipulate each fragment independently.

Fragment: Classes

- 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

Defining a Fragment

- Extend android.app.Fragment
- 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().

..Defining a Fragment

- Fragment lives in a ViewGroup inside the activity's view hierarchy
- Most fragments will have a UI and will have its own layout
- There may also be a fragment without its own UI as an invisible worker for the activity(called headless fragment)
- The onCreateView() method is called by Android once the fragment should create its user interface. Here you can inflate a layout via the inflate() method call of the Inflator object passed as a parameter to this method.
- There is no need to implement this method for headless fragments

Defining Fragment - Example

Parent's Activity
ViewGroup

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

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

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


    }
```

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

Adding Fragment to an Activity

- Via XML(Statically)
  - Insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element

- Via CODE(Dynamically)
  - At any time while activity is running, fragment may be added to activity using Fragment Manager and FragmentTransaction

Adding Fragment to an Activity: Via XML

- 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 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.

Adding fragment to an Activity thru XML layout

```
<?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>
```

2 fragment classes

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

Adding Fragment to an Activity: Via Code

- At any time while activity is running, fragments can be added to activity layout
- The activity layout must include a container View in which you can insert the fragment.
- Instantiate an instance of your fragment.
- Get a reference to the FragmentManager using Activity.getFragmentManager()
- Invoke FragmentManager.beginTransaction() to get an instance of FragmentTransaction.
- Use the FragmentTransaction.add() to add the fragment to a ViewGroup in the activity, specified by its ID. Optionally, a string tag can be provided to identify the fragment.
- Commit the transaction using FragmentTransaction.commit().

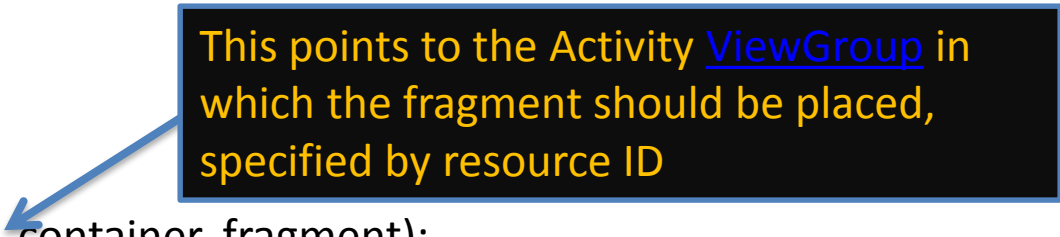
Adding fragment to an Activity via CODE

```
/*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();
```



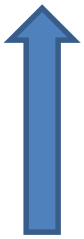
Fragment Transactions – adding, removing and replacing dynamically

```
// 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.

Adding Fragment that has NO UI

- A fragment is also used to provide a background behavior for the activity without presenting additional UI
- 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.
- Use fragement to activity using FragmentTransaction.add(Fragment, String) (supplying a unique string "tag" for the fragment, rather than a view ID)
- If you want to get the fragment from the activity later, you need to use findFragmentByTag().

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().

FragmentTransaction class Methods

- 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.

Fragment-to-Fragment Communication

- All Fragment-to-Fragment communication is done through the associated Activity. Two Fragments should never communicate directly
- To allow a Fragment to communicate up to its Activity, you can define an interface in the Fragment class and implement it within the Activity
- The Fragment captures the interface implementation during its onAttach() lifecycle method and can then call the Interface methods in order to communicate with the Activity
- The host activity can deliver messages to a fragment by capturing the Fragment instance with findFragmentById(), then directly call the fragment's public methods

Fragment 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
- WebViewFragment: Display a web view