ANDROID – FRAGMENTS

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 <u>has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running</u> (sort of like a "sub activity" that you can reuse in different activities).

Fragment

- Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately.
- With the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Fragment Uses

- **Modularity**: If a single activity is having too many functional components, its better to divide it into independent fragments, hence making the code more organized and easier to maintain.
- **Reusability**: If we define any particular feature in a fragment, then that feature more or less becomes a reusable component which can be easily integrated into any activity.
- **Adaptability**: If we break UI components of an app screen into fragments, then it becomes easier to change their orientation and placement, based on screen size etc.

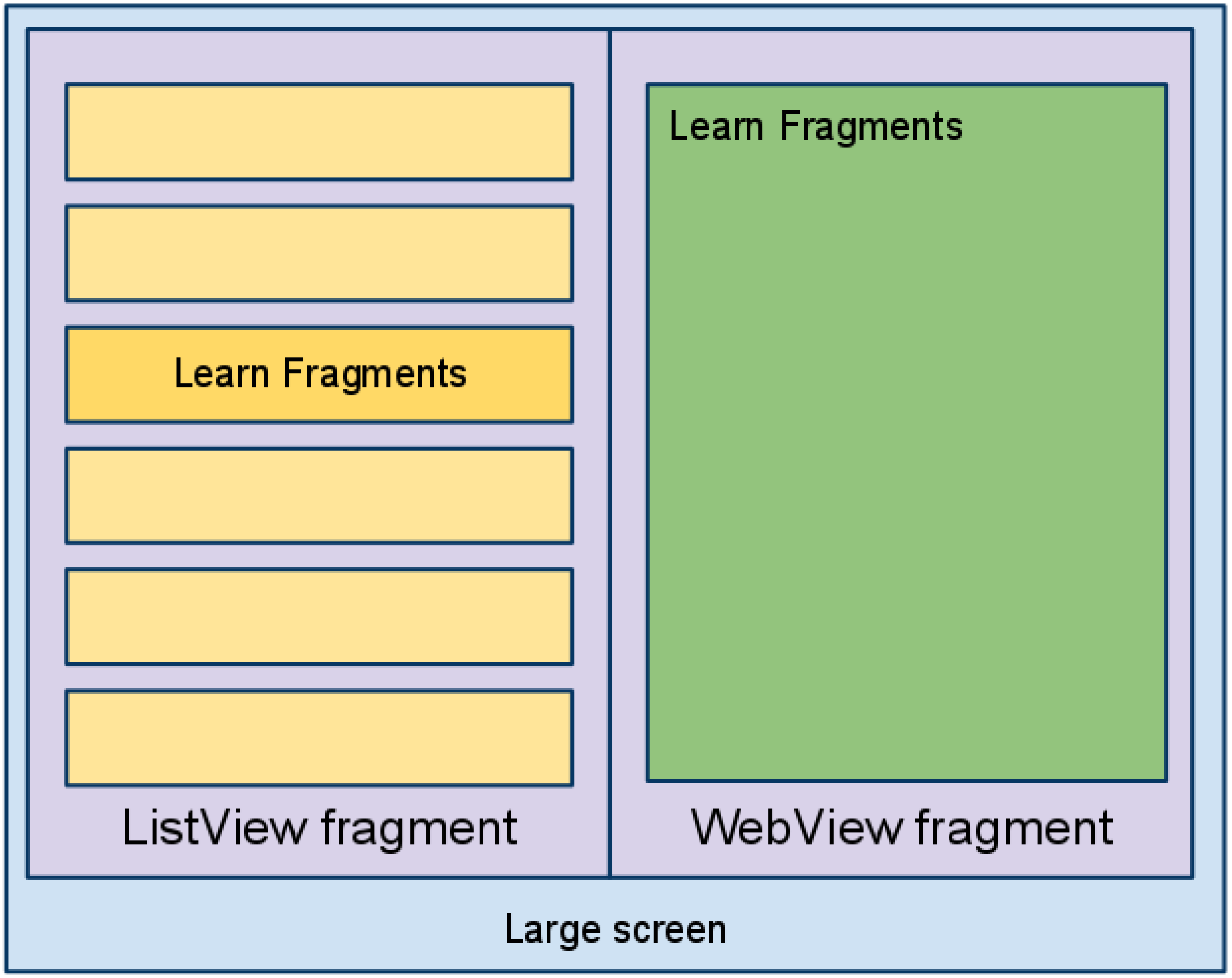
Fragment

- DESIGN PHILOSOPHY

- **Structure** an Activity as a collection of Fragments.

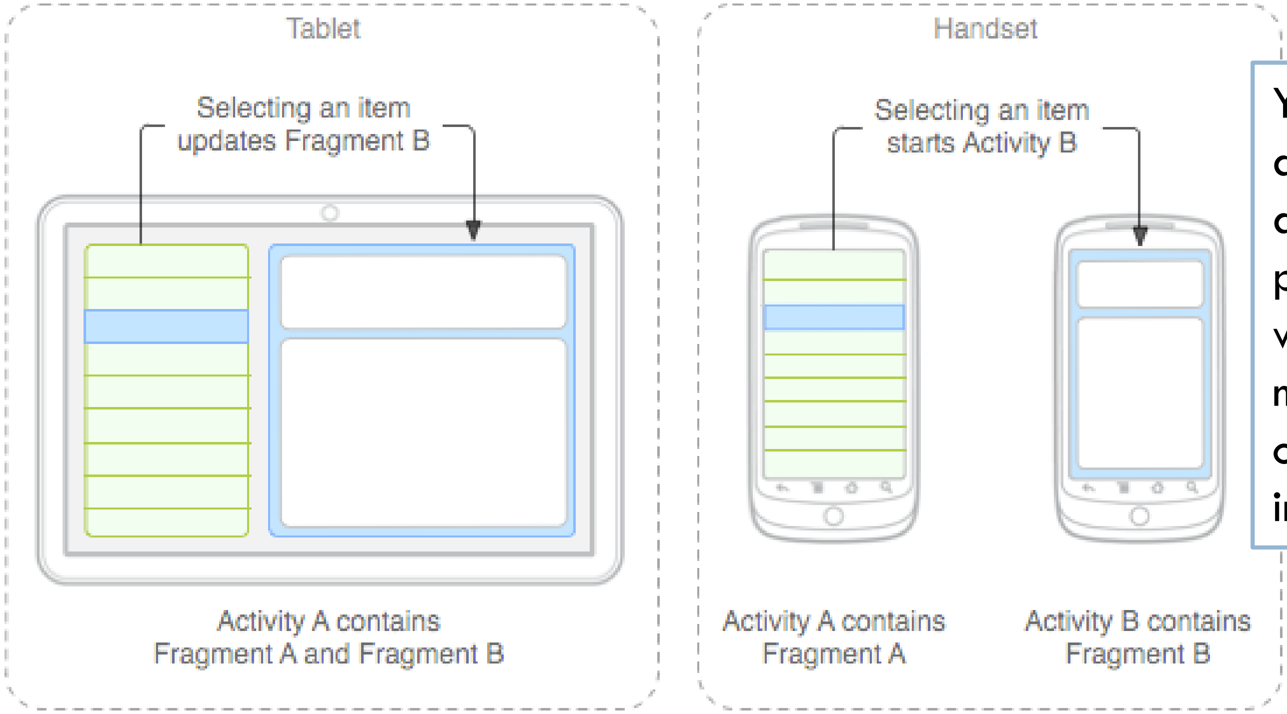
- **Reuse** a Fragment on different Activities …

Fragment


Learn Fragments
Learn Fragments
ListView fragment
WebView fragment
Large screen

Fragment Idea

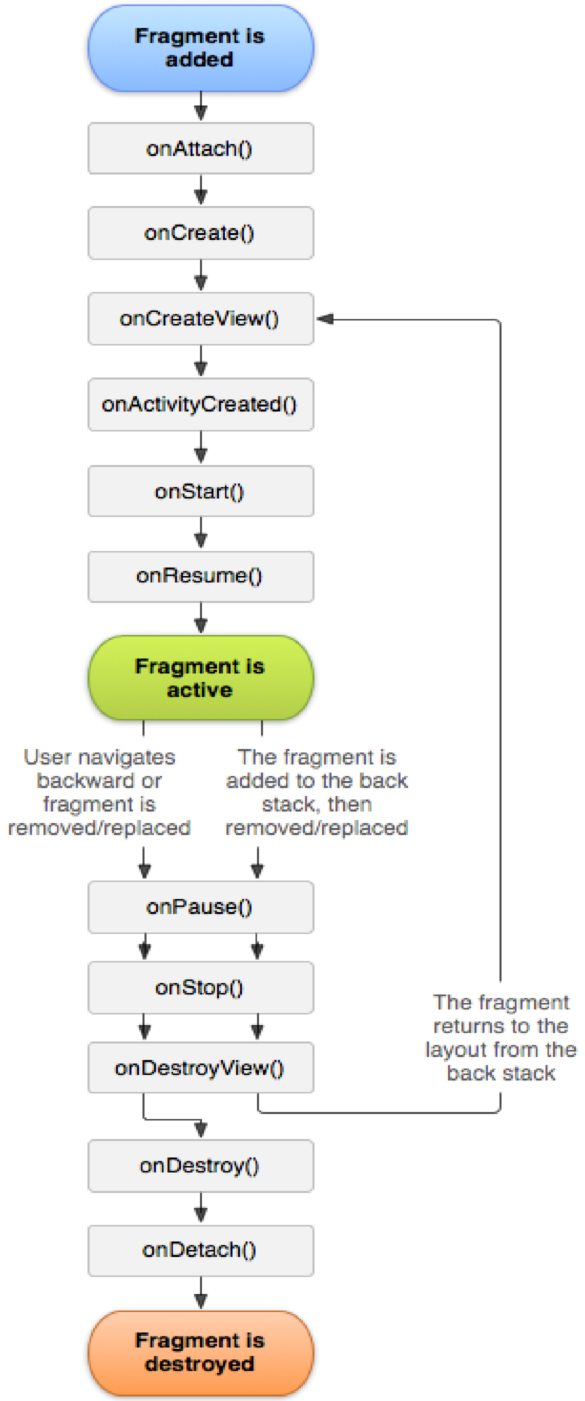
- → Fragments
  - Mini-activities, each with its own set of views
  - One or more fragments can be embedded in an Activity
  - You can do this dynamically as a function of the device type (tablet or not) or orientation


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

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

Fragment Lifecycle

- Fragment in an Activity---Activity Lifecyle influences
  - Activity paused → all its fragments paused
  - Activity destroyed → all its fragments are destroyed
  - Activity running → manipulate each fragment independently.
- Fragment transaction →add, remove, etc.
  - adds it to a back stack that's managed by the activity : each back stack entry in the activity is a record of the fragment transaction that occurred.
  - The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.



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.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 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.

How to use Fragments ?

- First of all decide how many fragments you want to use in an activity. For example we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has all the callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

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.

OPTION 1 – 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>
```

2 fragment classes

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

OPTION 2 – 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().

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.