Mobile Application Development

Building User Interfaces

(Professional Android 4 Application Development chapter 4)

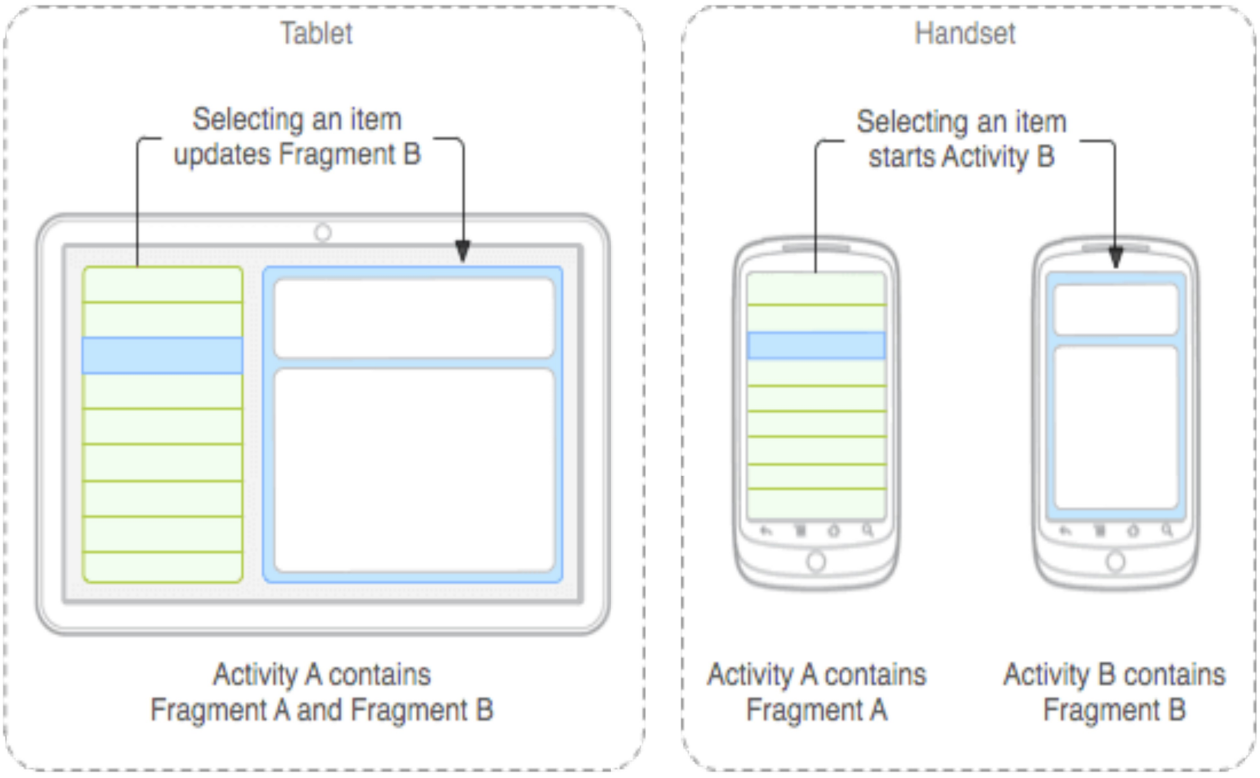
Outline

- Fundamental Android UI Design, Assigning UI To Activities, Layout Classes, Defining Layouts
- Linear Layout, Relative Layout, Grid Layout, Optimizing Layouts
- Fragments, Creating New Fragment, Fragment LifeCycle, Fragment Manager, Fragment Specific LifeCycle Events
- Fragment States, Adding Fragments to Activities, Using Fragment Transaction
- Interfacing Between Fragments and Activities, Fragment Without User Interfaces, Android Fragment Classes.

INTRODUCING FRAGMENTS

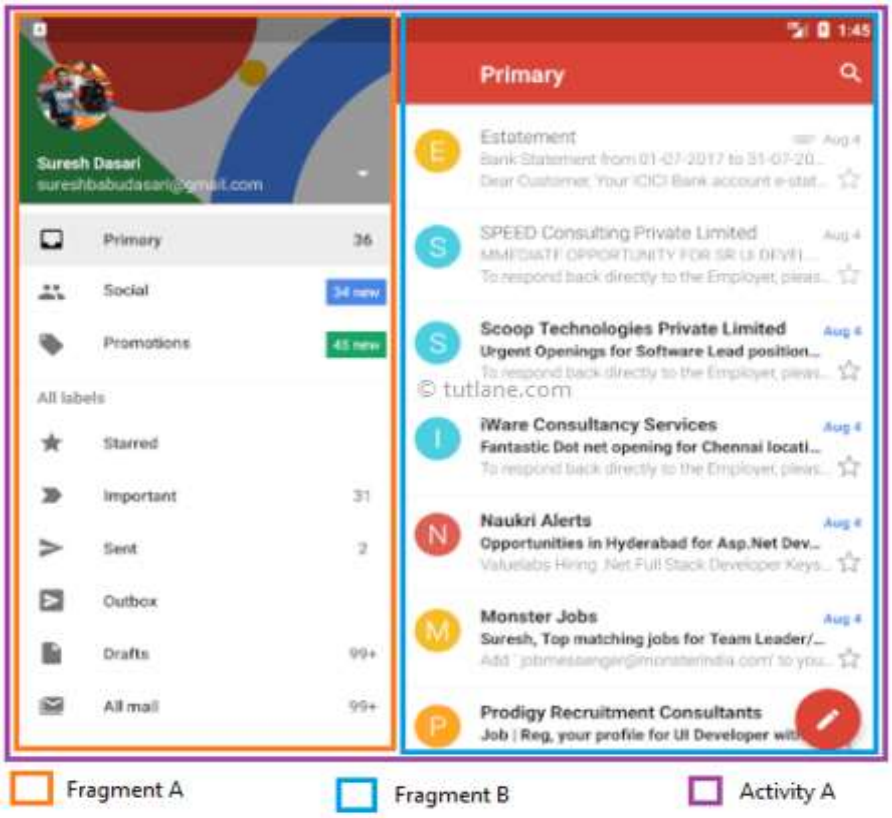
- Fragments enable you to divide your Activities into fully encapsulated reusable components, each with its own lifecycle and UI.
- The primary advantage of Fragments is the ease with which you can create dynamic and flexible UI designs that can be adapted to suite a range of screen sizes — from small-screen smartphones to tablets.
- Fragments provide a way to present a consistent UI optimized for a wide variety of Android device types, screen sizes, and device densities.
- Although it is not necessary to divide your Activities into Fragments, doing so will drastically improve the flexibility of your UI and make it easier for you to adapt your user experience for new device confi gurations.

Multiple Fragment Example



GMAIL app is designed with multiple fragments, so the design of GMAIL app will be varied based on the size of device such as tablet or mobile device.

Table View



Creating New Fragments

- Extend the Fragment class to create a new Fragment, (optionally) defining the UI and implementing the functionality it encapsulates.
- If your Fragment does require a UI, override the onCreateView handler to inflate and return the required View hierarchy
- Unlike Activities, Fragments don't need to be registered in your manifest.
- This is because Fragments can exist only when embedded into an Activity, with their lifecycles dependent on that of the Activity to which they've been added.

Creating New Fragments

```
package com.paad.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class MySkeletonFragment extends Fragment {
  @Override
  public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
    // Create, or inflate the Fragment's UI, and return it.
    // If this Fragment has no UI then return null.
    return inflater.inflate(R.layout.my_fragment, container, false);
  }
}
```

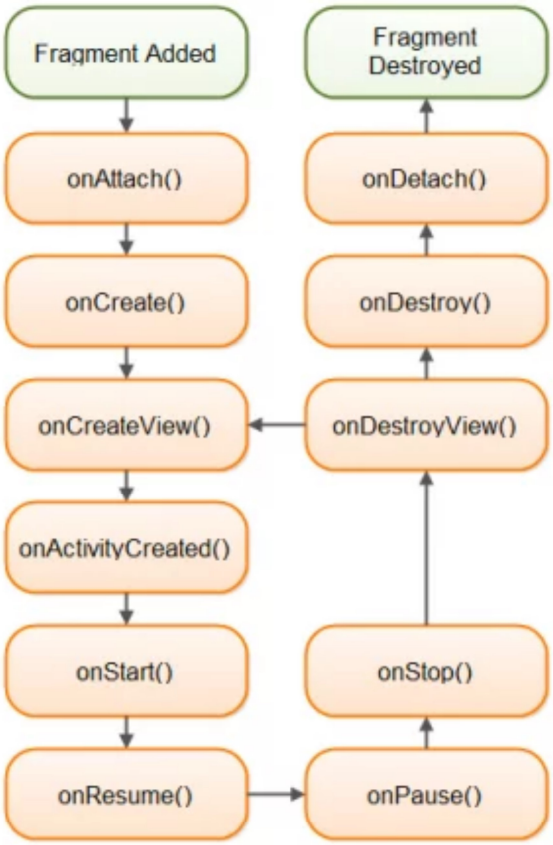

Creating New Fragments

- Fragment.OnCreateView(LayoutInflater, ViewGroup, Bundle)
- Parameters
- **inflater LayoutInflater**
- The LayoutInflater object that can be used to inflate any views in the fragment
- **Container ViewGroup**
- If non-null, this is the parent view that the fragment's UI should be attached to. The fragment should not add the view itself, but this can be used to generate the LayoutParams of the view.
- **savedInstanceState Bundle**
- If non-null, this fragment is being re-constructed from a previous saved state as given here.

Creating New Fragments

- Fragment.OnCreateView(LayoutInflater, ViewGroup, Bundle)
- return inflater.inflate(R.layout.fragment_first, container, false)
- The inflate() method has three arguments first one is the resource layout which we want to inflate, second is the ViewGroup to be the parent of the inflated layout.
- third parameter is a boolean value indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.

The Fragment Lifecycle



The Fragment Lifecycle

- OnAttach(): The very first method to be called when the fragment has been associated with the activity. This method executes only once during the lifetime of a fragment.
- OnCreate(): This method initializes the fragment by adding all the required attributes and components.
- OnCreateView(): System calls this method to create the user interface of the fragment. The root of the fragment's layout is returned as the View component by this method to draw the UI.
- OnActivityCreated(): It indicates that the activity has been created in which the fragment exists.

The Fragment Lifecycle

- OnStart(): The system invokes this method to make the fragment visible on the user's device.
- OnResume(): This method is called to make the visible fragment interactive.
- OnPause(): It indicates that the user is leaving the fragment. System call this method to commit the changes made to the fragment.
- OnStop(): Method to terminate the functioning and visibility of fragment from the user's screen.
- OnDestroyView(): System calls this method to clean up all kinds of resources as well as view hierarchy associated with the fragment.

The Fragment Lifecycle

- OnDestroy(): It is called to perform the final clean up of fragment's state and its lifecycle.
- OnDetach(): The system executes this method to disassociate the fragment from its host activity.

Introducing the Fragment Manager

- Each Activity includes a Fragment Manager to manage the Fragments it contains.
- You can access the Fragment Manager using the getFragmentManager method:
- FragmentManager fragmentManager = getFragmentManager();
- The Fragment Manager provides the methods used to access the Fragments currently added to the Activity, and to perform Fragment Transaction to add, remove, and replace Fragments.

Adding Fragments to Activities

- The simplest way to add a Fragment to an Activity is by including it within the Activity's layout using the fragment tag.

```
<?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.paad.weatherstation.MyListFragment"
    android:id="@+id/my_list_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
  />
```


Adding Fragments to Activities

- This technique works well when you use Fragments to define a set of static layouts based on various screen sizes.
- If you plan to dynamically modify your layouts by adding, removing, and replacing Fragments at run time, a better approach is to create layouts that use container Views into which Fragments can be placed at runtime, based on the current application state.

Adding Fragments to Activities

```
<?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">
  <FrameLayout
    android:id="@+id/ui_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
  />
  <FrameLayout
    android:id="@+id/details_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="3"
  />
</LinearLayout>
```


Using Fragment Transactions

- Fragment Transactions can be used to add, remove, and replace Fragments within an Activity at run time. Using Fragment Transactions, you can make your layouts dynamic — that is, they will adapt and change based on user interactions and application state.
- Each Fragment Transaction can include any combination of supported actions, including adding, removing, or replacing Fragments.
- A new Fragment Transaction is created using the beginTransaction method from the Activity's Fragment Manager. Modify the layout using the add, remove, and replace methods, as required.
- When you are ready to execute the change, call commit to add the transaction to the UI queue.

Using Fragment Transactions

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

// Add, remove, and/or replace Fragments.
// Specify animations.
// Add to back stack if required.

fragmentTransaction.commit();
```


Adding and Removing Fragments

- When adding a new UI Fragment, specify the Fragment instance to add, along with the container View into which the Fragment will be placed. Optionally, you can specify a tag that can later be used to fi nd the Fragment by using the findFragmentByTag method:

```
// Begin the transaction
FragmentManager fm = getFragmentManager();
FragmentTransaction ft = fm.beginTransaction();

// Create the Fragment and add
BookListFragment2 listFragment = new BookListFragment2();
ft.add(R.id.layoutRoot, listFragment, "bookList");

// Commit the changes
ft.commit();
```

Adding and Removing Fragments

- To remove a Fragment, you first need to find a reference to it, usually using either the Fragment Manager's findFragmentById or findFragmentByTag methods. Then pass the found Fragment instance as a parameter to the remove method of a Fragment Transaction:

```
FragmentManager fm = getFragmentManager();
Fragment listFragment = fm.findFragmentByTag("bookList");
BookDescFragment bookDescFragment = new BookDescFragment();
FragmentTransaction ft = fm.beginTransaction();
ft.remove(listFragment);
ft.add(R.id.layoutRoot, bookDescFragment, "bookDescription");
ft.commit();
```


Interfacing Between Fragments and Activities

- Use the getActivity method within any Fragment to return a reference to the Activity within which it's embedded.
- This is particularly useful for fi nding the current Context, accessing other Fragments using the Fragment Manager, and fi nding Views within the Activity's View hierarchy.

```
TextView textView = (TextView)getActivity().findViewById(R.id.textview);
```


THANKS