LESSON – 6

FRAGMENTS

AGENDA

- Introduction to Fragments
- Why do we need Fragments
- How to define Fragments is XML
- How to Create a Fragment Class
- How to add Fragments in Activity
- About Weight property
- Hands on Examples
- Procedure to create Dynamic Fragments
- Design support Library

INTRODUCTION

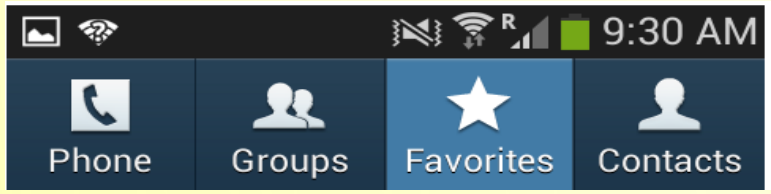
- Fortunately, Android 3.0 introduces a finer-grained application component called Fragment that lets you modularize the application and its user interface (into fragments).
- Think of fragments as “mini-activities”: reusable, independent but related mini portions of the application and screen that can be drawn independently of each other, each receiving its own events and having its own state, application lifecycle, and back stack.
- Fragment is one of the UI Component or Fragment is a subtype of an Activity.
- In Android each and every screen is an Activity. An activity is a container for views.
- To create a dynamic and multi-pane user interface on Android, you need to encapsulate UI components and activity behaviors into modules that you can swap into and out of your activities.

INTRODUCTION

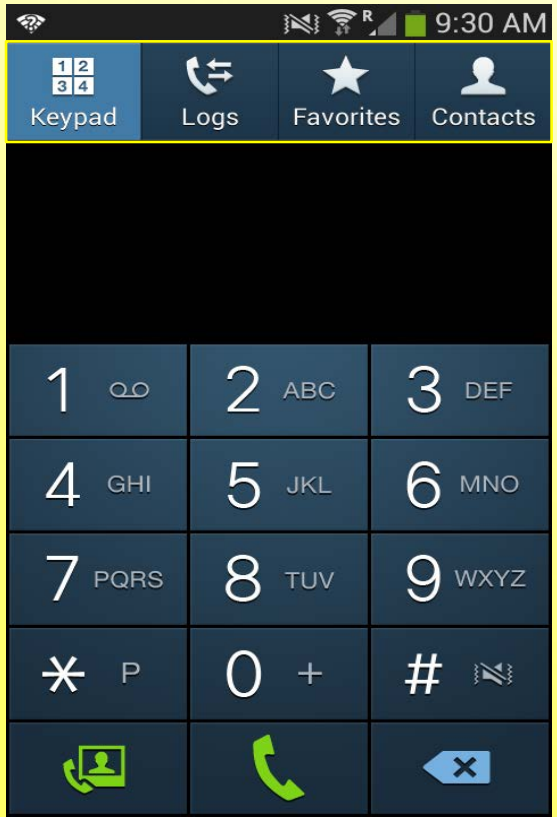
- You can create these modules with the Fragment class, which behaves somewhat like a nested activity that can define its own layout and manage its own lifecycle.
- When a fragment specifies its own layout, it can be configured in different combinations with other fragments inside an activity to modify your layout configuration for different screen sizes (a small screen might show one fragment at a time, but a large screen can show two or more).
- This class shows you how to create a dynamic user experience with fragments and optimize your app's user experience for devices with different screen sizes.
- Refer : https://developer.android.com/guide/components/fragments.html

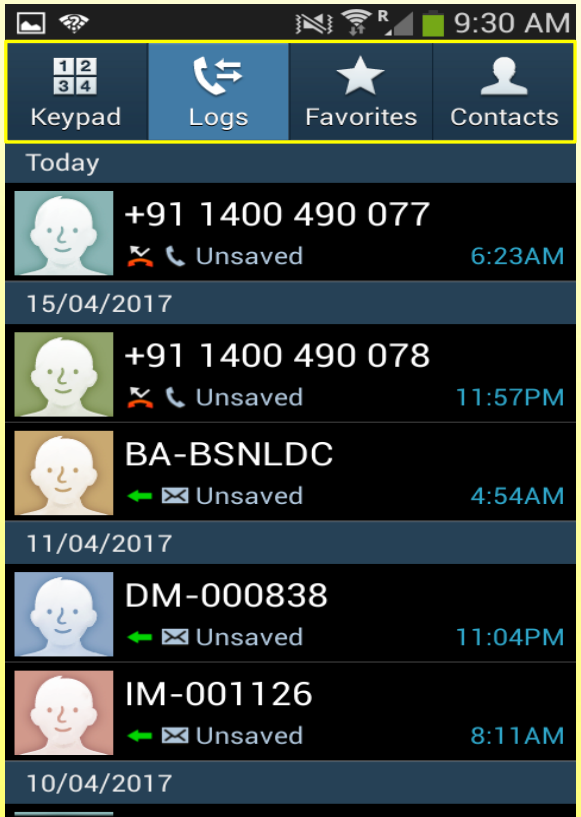
Why do we need Fragments

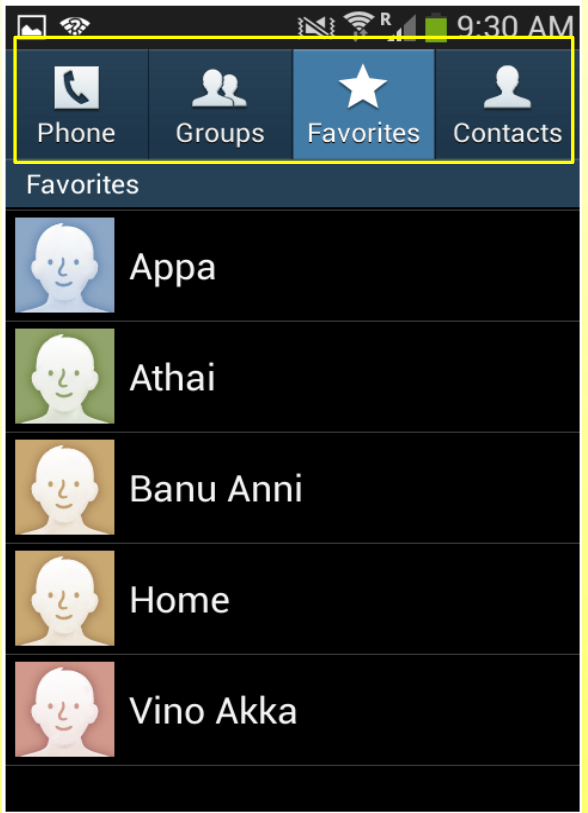
- Lets take a look on the screens shots.

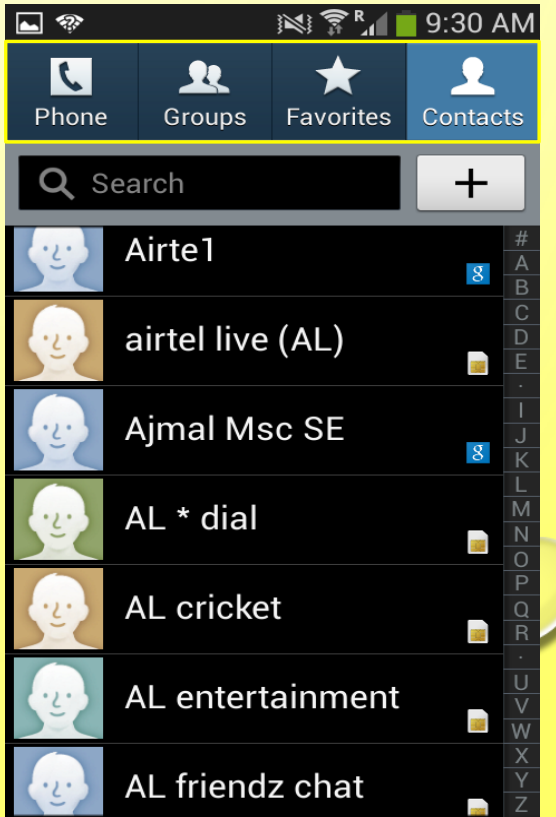


This top header is common to all the screens. Based on the user action only the body of the part changes. Here there is no need to create multiple activities. Instead of that we can use Fragments.

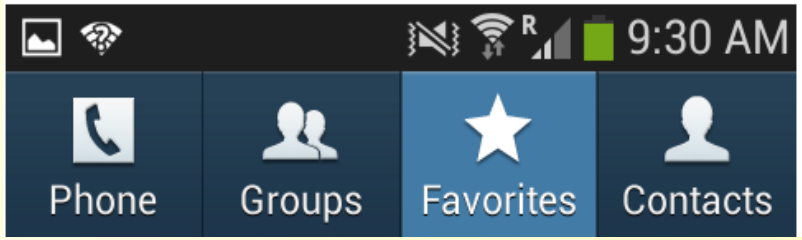








Drawbacks of having Individual Activity



- In the previous example, if we go for the usage of 4 different individual activities, in each activity we have to write the logic for the common part. If we want to modify the common part, need to make changes in all the activities.
- So there are two drawbacks of having different individual activities.
  1. Code duplication (in terms of, if you want to define header and footer part in an activity)
  2. Code Modification (in terms of activity, need to change the header and footer part in every activity)
- To overcome this drawback android apps prefers to use single activity with more fragments to achieve the benefit of Code reusability.

FRAGMENT IDEA

primarily to support more dynamic and **flexible UI designs on large screens, such as tablets**.

- → 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

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

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).

FRAGMENT'S LIFECYCLE

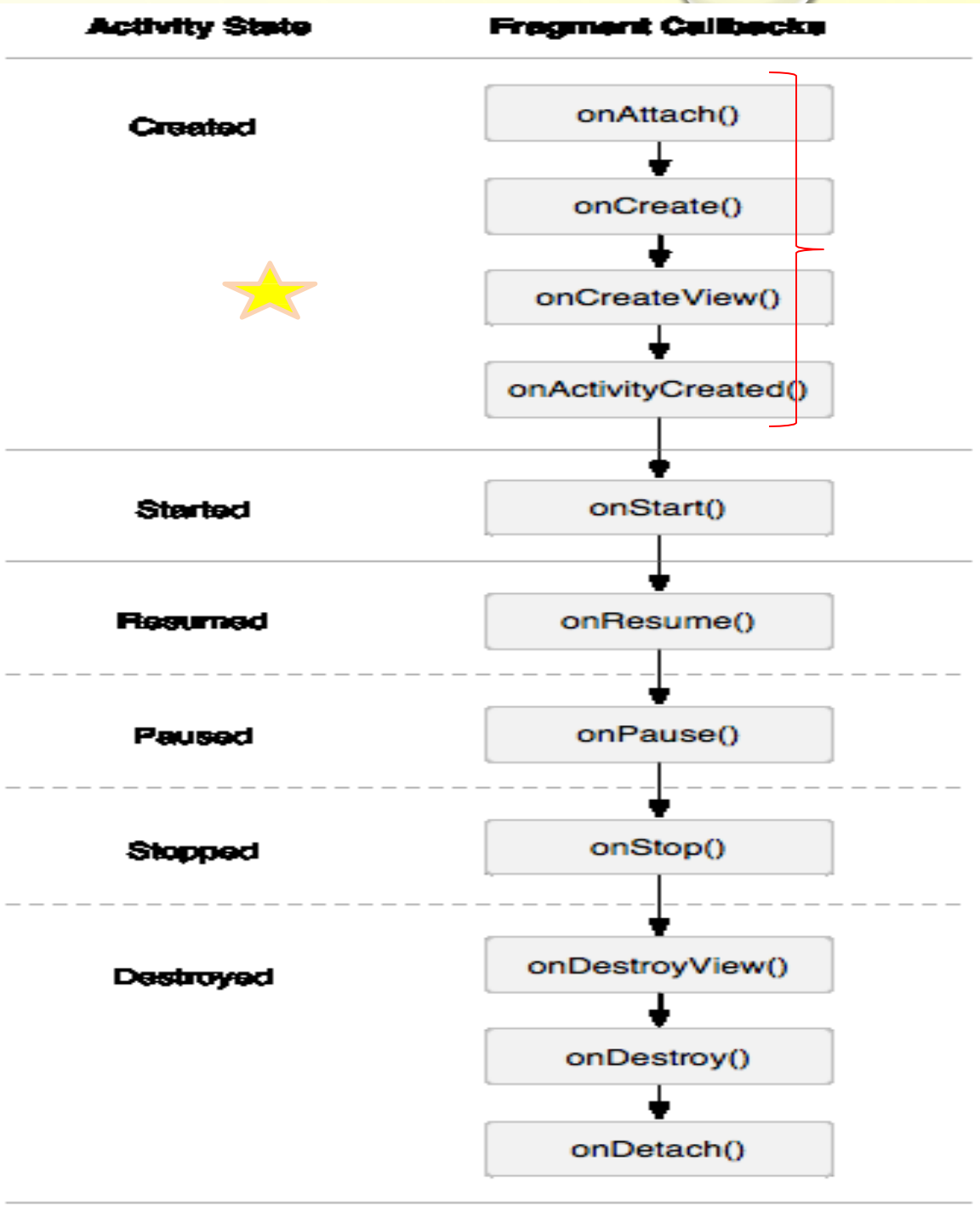
onAttach() Invoked when the fragment has been connected to the host activity.

onCreate() Used for initializing components needed by the fragment.

onCreateView() *Most of the work is done here*. Called to create the view hierarchy representing the fragment. Usually inflates a layout, defines listeners, and populates the widgets in the inflated layout.

onPause() The session is about to finish. Here you should commit state data changes that are needed in case the fragment is re-executed.

onDetach() Called when the inactive fragment is disconnected from the activity.



Fragment Tag

Add a fragment to an activity using XML

- Create a Fragment in XML by using the following ways.

```
<fragment android:name="com.example.android.fragments.ArticleFragment"
        android:id="@+id/frag1"
        ...../>
```

(or)

// FrameLayout is the Parent of Fragment, use this to create a Fragments

AFrameLayout is a special type of view in Android that is used to block an area of the screen, in order to display a single element

```
<FrameLayout
        android:id="@+id/frag1"
        ...../>
```

Add a fragment to an activity using XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <fragment android:name="com.example.android.fragments.HeadlinesFragment“   [ Name of your Fragment Class ]
            android:id="@+id/headlines_fragment"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.android.fragments.ArticleFragment"
            android:id="@+id/article_fragment"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Creation of Fragments

- A Single activity can have many fragments. Each fragment is having its own life cycle and its own UI. You follow three main steps when implementing a fragment:
  1. Create the fragment subclass.
  2. Define the fragment layout.
  3. Include the fragment within the Activity.

Create a fragment class

- To create a fragment, extend the fragment class, then override key lifecycle methods to insert your app logic, similar to the way you would with an activity class.
- One difference when creating a fragment is that you must use the oncreateview() callback to define the layout. In fact, this is the only callback you need in order to get a fragment running.
- Automatically create Fragment by Click File→ New → Fragment(Blank)

Fragments and their UI – onCreateView() using XML

- Can implement onCreateView() using XML

```
public static class ExampleFragment extends andriod.app.fragment {

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedinstancestate) {
    // inflate the layout for this fragment – Convert XML into View
    return inflater.inflate(RLayout.Example_fragment, container, false); // First parameter is layout, Second
//parameter is ViewGroup object, third parameter is boolean type  always false
    }
}
```

Instantiates a layout XML file into its corresponding View objects.

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.

Example

```
public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

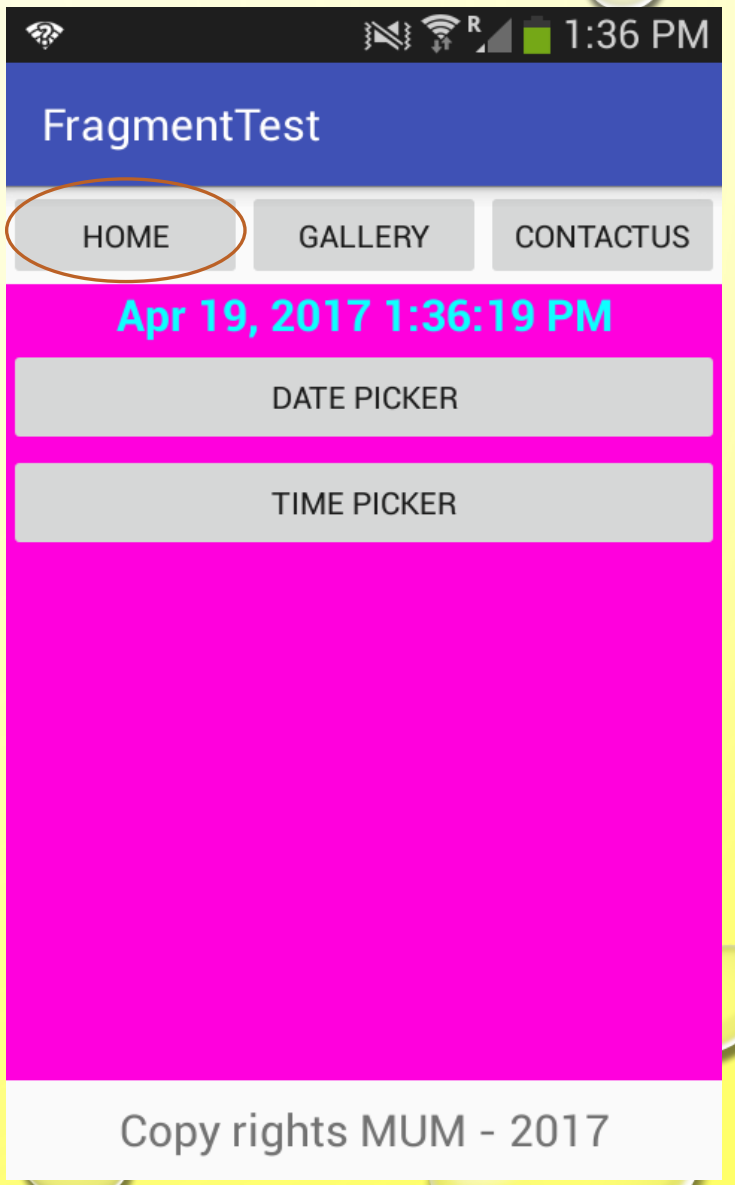
- Get the Fragment into your MainActivity.java by using the following code segments

```
//get FragmentTransaction associated with this Activity

FragmentManager fragmentManager = getFragmentManager();

// Begin a fragment transaction
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); // you can also call remove()/replace()

// Commit a fragment transaction

fragmentTransaction.commit();
```

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

Hands on Example

- Create an Main Activity with three buttons Home, Gallery and ContactUs along with one Fragement and One textView to display the Copy Rights Contents. If the user clicks the Home button the Home Fragement will work. Similarly for Gallery and Contact us. This page shows Home Fragment as a default.
- See : Lesson6\MyApplication folder



Main Screen Design Code – activity_main.xml

Top Layout Code

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

Layout Code for all Buttons

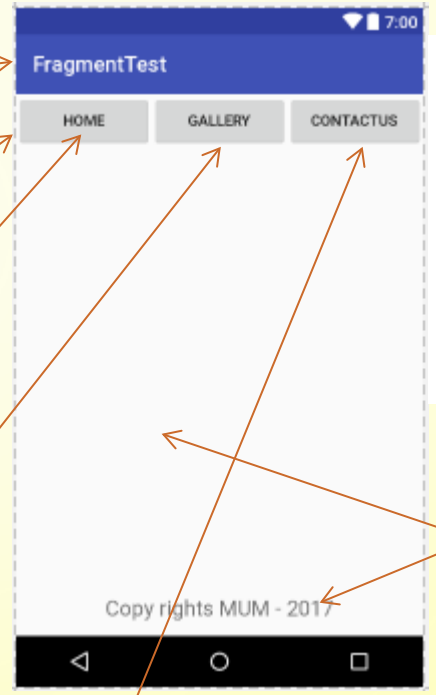
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="0.1"
    android:orientation="horizontal"
    >
```

Home Button Code

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="0.33"
    android:text="Home"
    android:onClick="home"
    />
```

Gallery Button Code

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="0.33"
    android:text="Gallery"
    android:onClick="gallery"
    />
```



ContactUs button Code

```
<Button
  android:layout_width="0dp"
  ndroid:layout_height="match_parent"
  android:layout_weight="0.33"
  android:text="ContactUs"
  android:onClick="contactus"/>
</LinearLayout>
```

Fragment Code

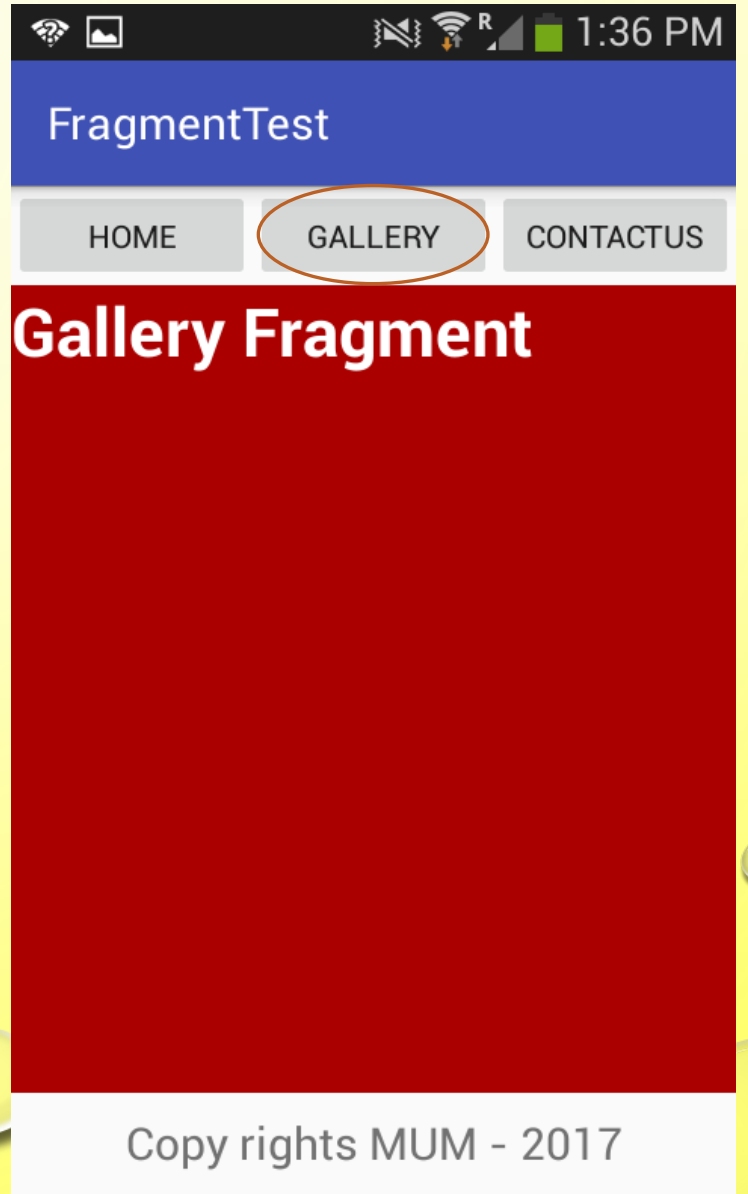
```
<TextView
android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="0.1"
android:text="Copy rights MUM -
2017"
android:textSize="20sp"
android:gravity="center"/>
</LinearLayout>
```

```
<FrameLayout
android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="0.8"
    android:id="@+id/frame1">
</FrameLayout>
```

CODING PART

Each Fragment we need to create .java file and .xml file. This is the code example for gallery_fragment.xml.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#AA0000">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:text="Gallery Fragment"
        android:id="@+id/textView"
           android:textColor="@android:color/white"
        android:textSize="30sp"
        android:textStyle="bold"
        android:gravity="center"/>
</RelativeLayout>
```



CODING PART

GalleryFragment.java

```
public class GalleryFragment extends Fragment {

public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.gallery_fragment,container,false);

        return view;

    }

}
```

CODING PART

Each Fragment we need to create .java file and .xml file. This is the code example for contactus_fragment.xml.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0000FF">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:text="Contact US Fragment"
        android:id="@+id/textView"
           android:textColor="@android:color/white"
        android:textSize="30sp"
        android:textStyle="bold"
        android:gravity="center"/>
</RelativeLayout>
```

CODING PART

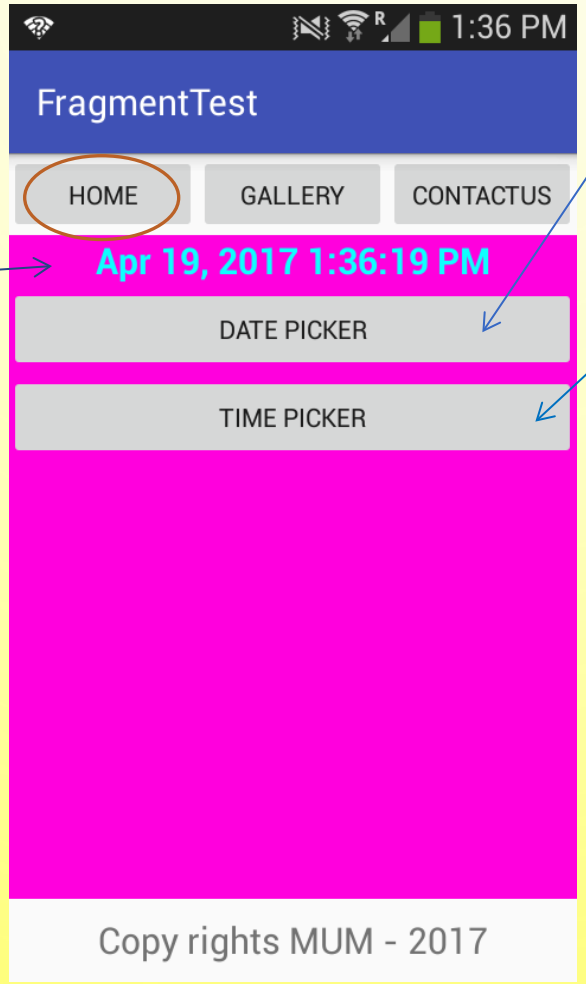
ContactusFragment.java.

```
public class ContactusFragment extends Fragment {

    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
container, Bundle savedInstanceState) {

        View view = nflater.inflate(R.layout.contactus_fragment,container,false);

        return view;

    }

}
```

CODING PART

Each Fragment we need to create .java file and .xml file. This is the code example for home_fragment.xml.

```
 <LinearLayout
xmlns:android="http://schemas.android.com/apk/res
/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#FF00DD">
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tv1"
        android:text="Picked Date and Time"
        android:textColor="#0FFFFF"
        android:textStyle="bold"
        android:textSize="20dp"
        android:gravity="center"/>
```




```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bt1"
    android:text="DATE PICKER"/>
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/bt2"
        android:text="TIME PICKER"/>
</LinearLayout>
```

HomeFragment.java

```
public class HomeFragment extends Fragment {

        // get the Current Date and Time
        Calendar datetime = Calendar.getInstance();

        // use the Date Time format of Month,DD,YYYY HH:mm:SS

        DateFormat df = DateFormat.getDateTimeInstance();

       TextView tv1;

         Button bt1, bt2;

          View view;

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle
        savedInstanceState) {

    view = inflater.inflate(R.layout.home_fragment, container, false);

    // To get the components from fragment, use view object of fragement.findViewById();

    tv1 = (TextView) view.findViewById(R.id.tv1);

    bt1 = (Button) view.findViewById(R.id.bt1);

    bt2 = (Button) view.findViewById(R.id.bt2);
```

HomeFragment.java

```
// Implementing Button Listener for Date Picker
        bt1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            updateDate(); // Call to set the Date from DataPicker
            }
        });
// Implementing Button Listener for Time Picker
        bt2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            updateTime(); // Call to set the Time from TimePicker
            }
        });
       updateLabel(); // Calling this method to set the Data and Time in TextView
        return view;
    }
```

HomeFragment.java

```
// Method helps to set the Date and Time in TextView
 private void updateLabel() {
     tv1.setText(df.format(datetime.getTime()));
 }
 /* Creating object for DatePickerDialog by passing current activity, object of DatePickerDialog, YYYY,Month,Day.
 * Then call the show method to display the DatePickerDialog*/
 private void updateDate(){
     new DatePickerDialog(getActivity(),d,datetime.get(Calendar.YEAR),
        datetime.get(Calendar.MONTH),
        datetime.get(Calendar.DAY_OF_MONTH)).show();
 }
 /* Creating object for TimePickerDialog by passing current activity, object of TimePickerDialog, HH,mm,boolean.
    Here boolean is24HourView as true or false.  Then call the show method to display the DatePickerDialog*/
 private void updateTime(){
     new TimePickerDialog(getActivity(),t,datetime.get(Calendar.HOUR_OF_DAY),
        datetime.get(Calendar.MINUTE),true).show();
 }
```

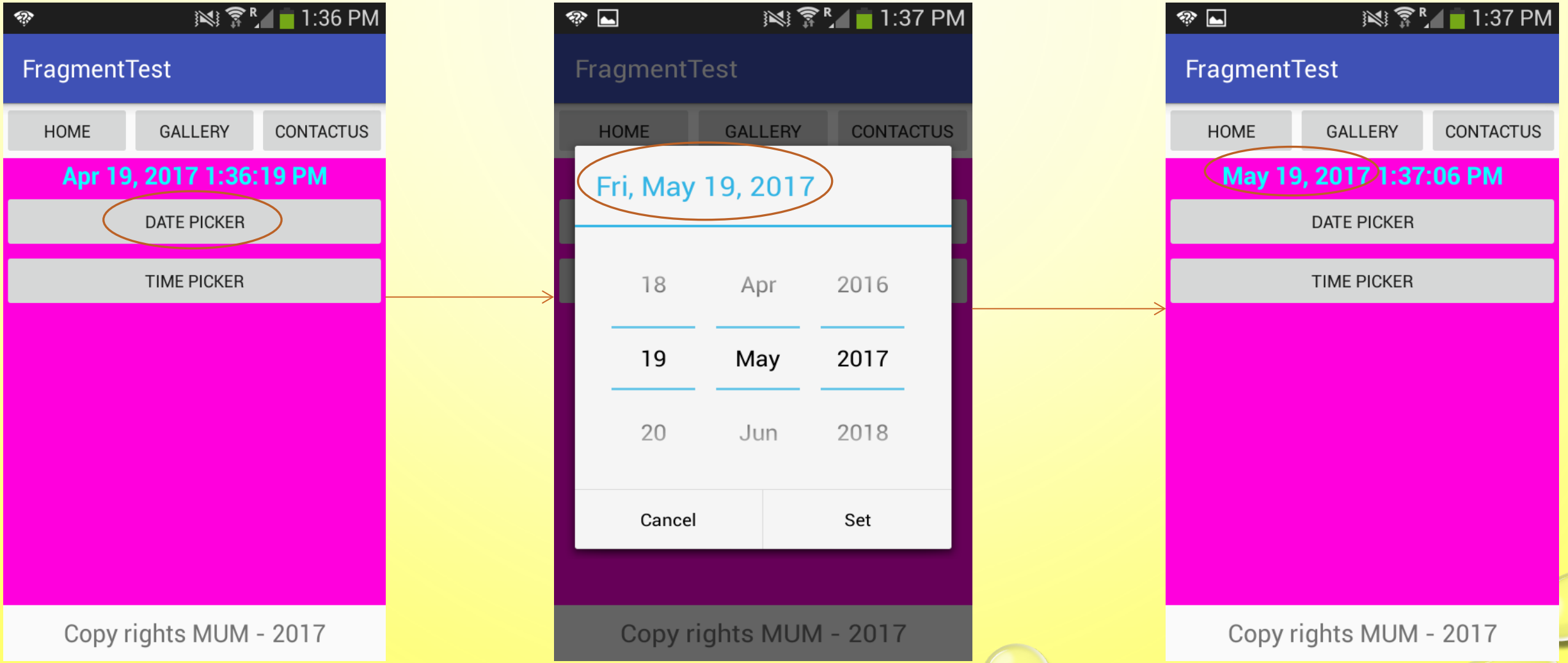
HomeFragment.java

```
// TimePickerDialog Listener Implementation to set the Picked Time

TimePickerDialog.OnTimeSetListener t = new TimePickerDialog.OnTimeSetListener() {

    @Override

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {

    datetime.set(Calendar.HOUR_OF_DAY,hourOfDay);

     datetime.set(Calendar.MINUTE,minute);

      updateLabel();

    }

};
```

HomeFragment.java

```
    // DatePickerDialog Listener Implementation to set the Picked Date
    DatePickerDialog.OnDateSetListener d = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
            datetime.set(Calendar.YEAR,year);
            datetime.set(Calendar.MONTH,month);
            datetime.set(Calendar.DAY_OF_MONTH,dayOfMonth);
            updateLabel();
        }
    };
}
```

Home Fragment Screen Shots



More Example useful for Project Time

Concepts

- Through Workout Example
  - Dynamic Fragment
  - List Fragment
  - Nested Fragments
- Design Support Library(chapter12\BitsandPizzas
  - How to implement Material design
  - Coordinator Layout – Root Tag by adding dependencies
    - compile 'com.android.support:design:25.2.0‘ dependencies
  - Viewpager
  - Tab Layout
  - AppBar Layout

Creation of Dynamic Fragment Steps

- Possible want to add, remove or replace fragments during Activity's run time programmatically.
- Create an object of FragmentTransaction class using the FragmentManager object.
- To add fragment at runtime you must provide a container view for the fragment in your Activity's layout file in which you can insert the fragment. Here is the code.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Creation of Dynamic Fragment Steps

Add a fragment at run time.

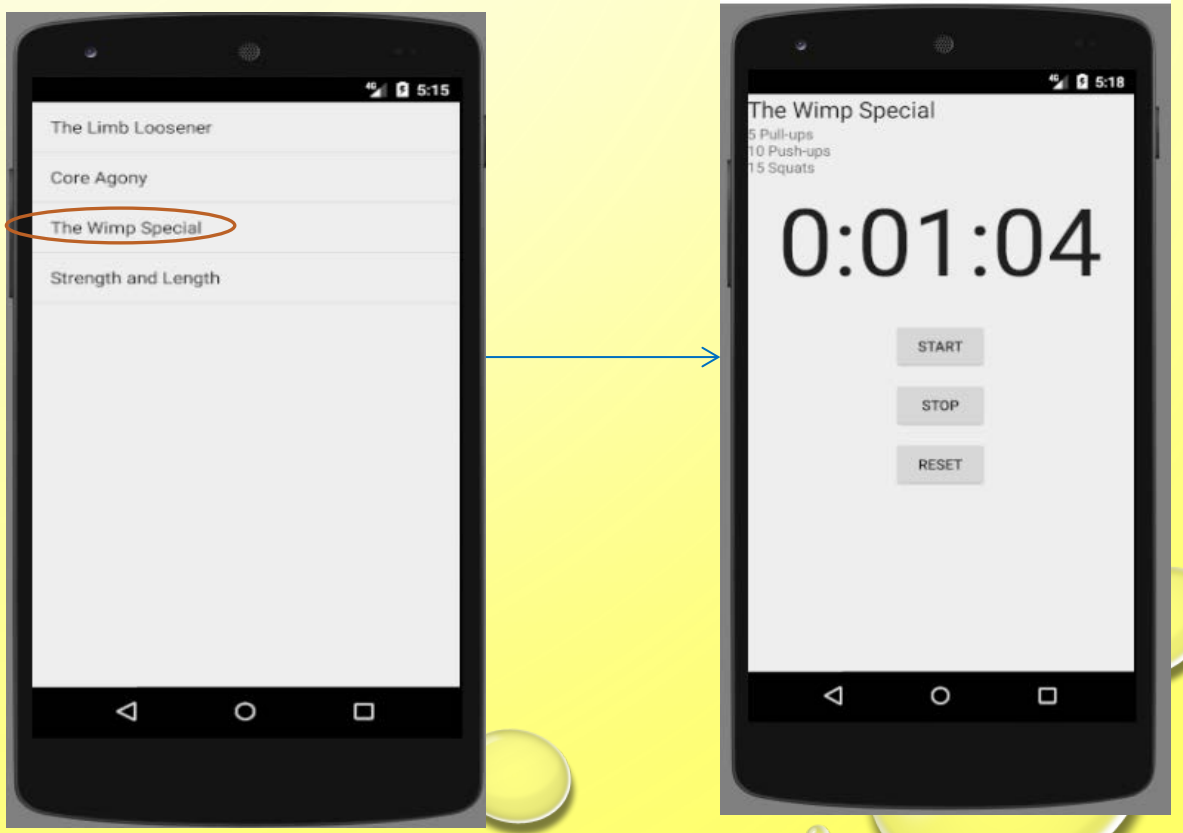
- You have to call the **getSupportFragmentManager()** to get the FragmentManager object.
- For getting the **FragmentTransaction** object you have to call the **beginTransaction()** method on the **FragmentManager** object.
- For add a new fragment call the **add()** method on FragmentTransaction object.
- You can perform multiple fragment transaction for the activity using the same **FragmentTransaction** Object.
- When you ready to make the changes, simply call the **commit()** method on the FragmentTransaction object.
- If you want replace a fragment, then you have to call the **replace()** method instead of add() method().
- To allow the user to navigate backward through the fragment transactions you must call the **addToBackStack()** before commit the **FragmentTransaction**.

Creation of Dynamic Fragment

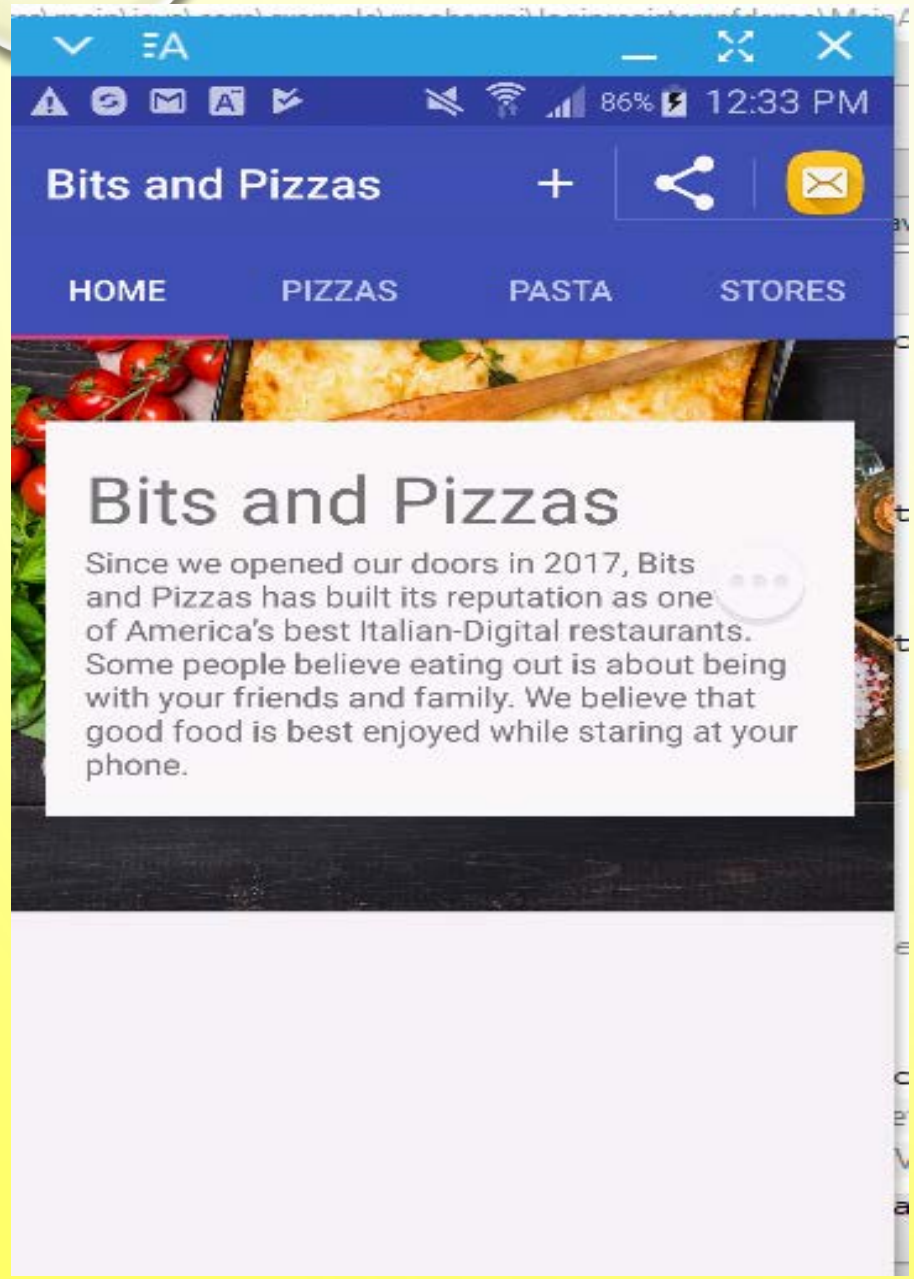
To know about Multiple Fragments and Nested Fragments refer Text book Chapter 11 from the Head First Android Development second edition. It is an good example to run the code in Phone as well as Tablet. In Phone you can choose the Item from ListFragment and detail information will be displayed Detail Fragment. But if you choose the Tablet both ListFragment and DetailFragment will displayed together. See : Lesson6\Workout folder

Tablet View

Phone View



Design support library [Example 1]

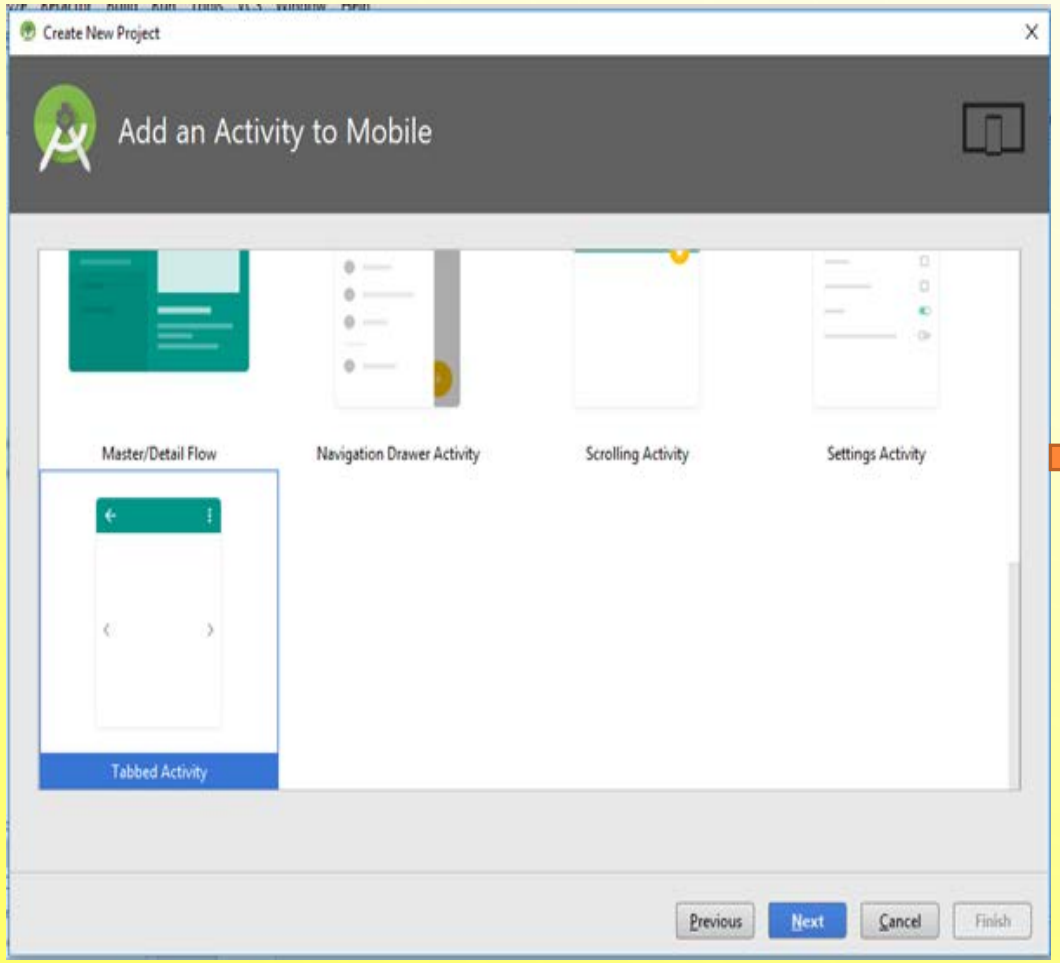


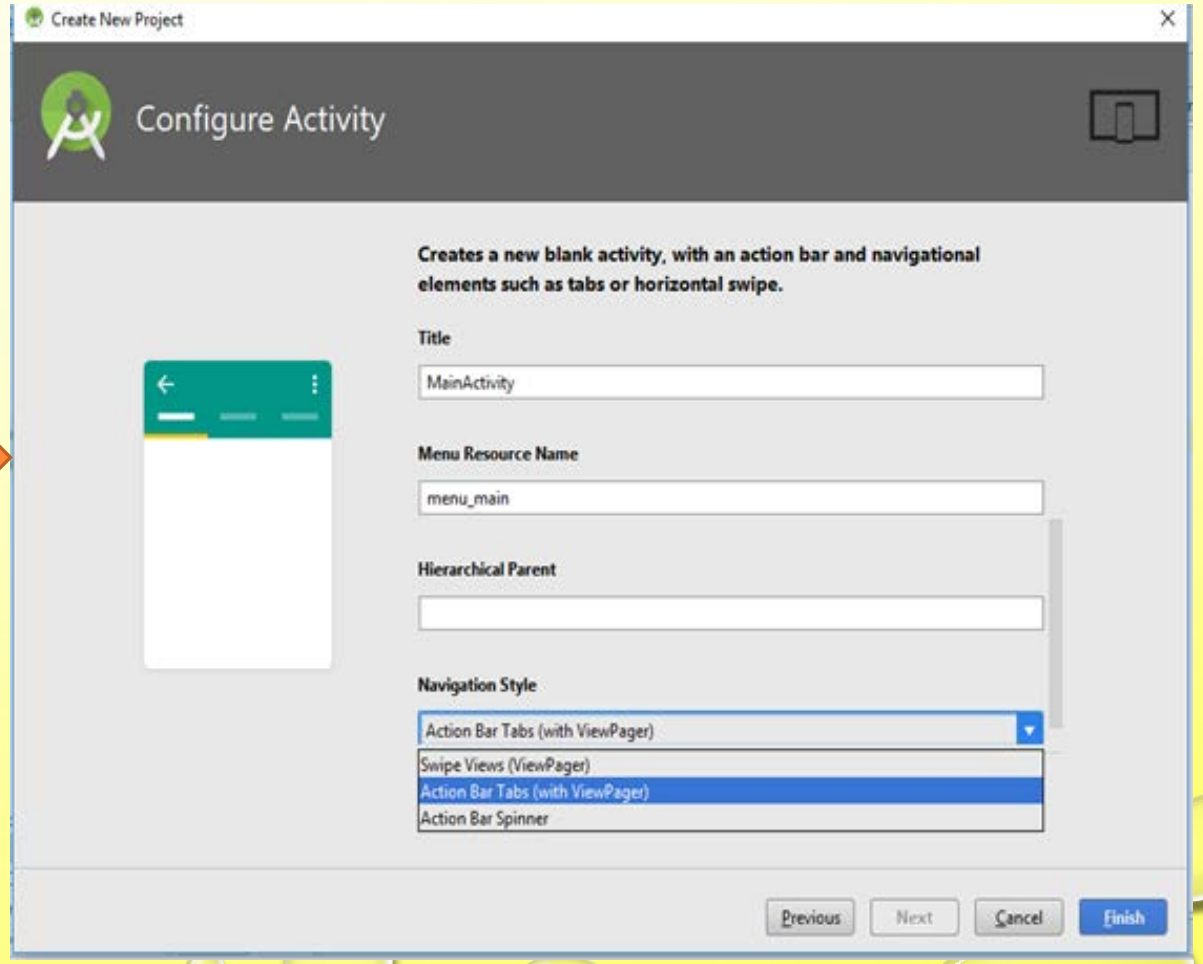
Refer : Chapter 12 Design Support Library from the Head First Android Development second edition.

From your Demo code : Lesson6\chapter12 package

Design support library for swipe views with Initial code

Create a new Project as usual, instead of Empty Activity follow the steps as per the screen shots.





Tab Layout using View Pager – Example 2

- Android Tab Layout is basically a view class required to be added into the layout of our app to create sliding tabs.
- However, you can create sliding as well as non-sliding tabs by using Android tablayout. If you want to add Android tabs without sliding, then replace the layout with the fragment on tab selected listener event. And, if you want to add sliding tabs, then use ViewPager.
- CoordinatorLayout is a super-powered FrameLayout.
- AppBarLayout is a vertical LinearLayout which implements many of the features of material designs app bar concept, namely scrolling gestures.
- A standard toolbar for use within application content.
- A Toolbar is a generalization of action bars for use within application layouts. While an action bar is traditionally part of an Activity's opaque window decor controlled by the framework, a Toolbar may be placed at any arbitrary level of nesting within a view hierarchy. An application may choose to designate a Toolbar as the action bar for an Activity using the setSupportActionBar() method.

Tab Layout using View Pager

- TabLayout provides a horizontal layout to display tabs.
- Population of the tabs to display is done through TabLayout.Tab instances. You create tabs via newTab(). From there you can change the tab's label or icon via setText(int) and setIcon(int) respectively. To display the tab, you need to add it to the layout via one of the addTab(Tab) methods.
- Inside build.gradle
- Add this line in dependencies

  compile 'com.android.support:design:25.2.0‘

  Refer : The Complete coding in Lesson6\TabFragment Layout