LESSON – 7- DAY 2

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
- Design support Library – Material Design
    - FloatingActionButton
    - SnackBar
    - Tabs
    - NavigationDrawer

INTRODUCTION

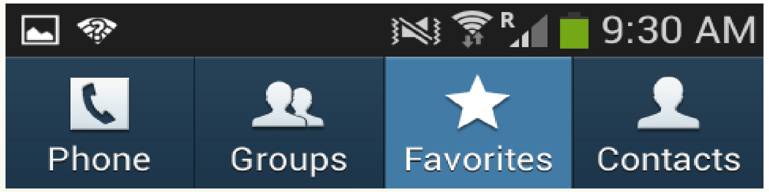
- Android 3.0 introduces a finer-grained application component called Fragment that lets you modularize the application and its user interface (into fragments).
- Think of fragments as “mini-activities”: reusable, independent but related mini portions of the application and screen that can be drawn independently 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

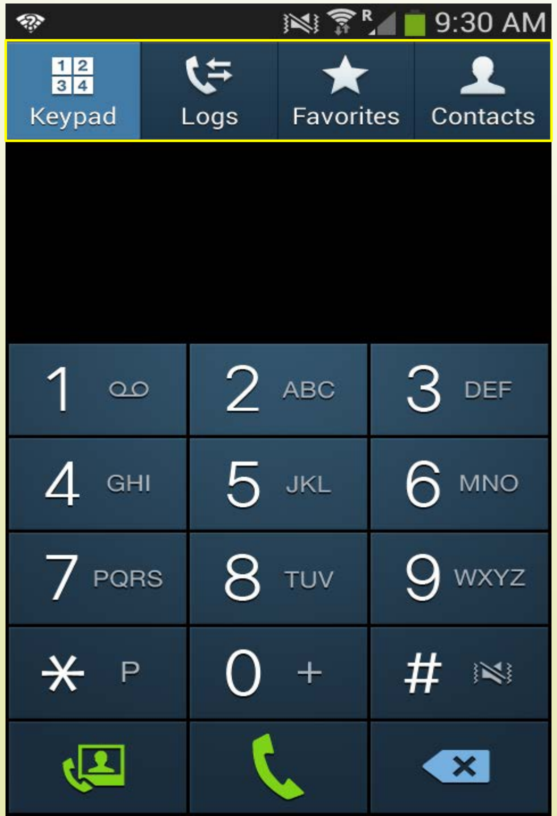
- You can create these modules with the Fragment class, which behaves somewhat like a nested activity that can define its own layout and manage its own lifecycle.
- When a fragment specifies its own layout, it can be configured in different combinations with other fragments inside an activity to modify 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

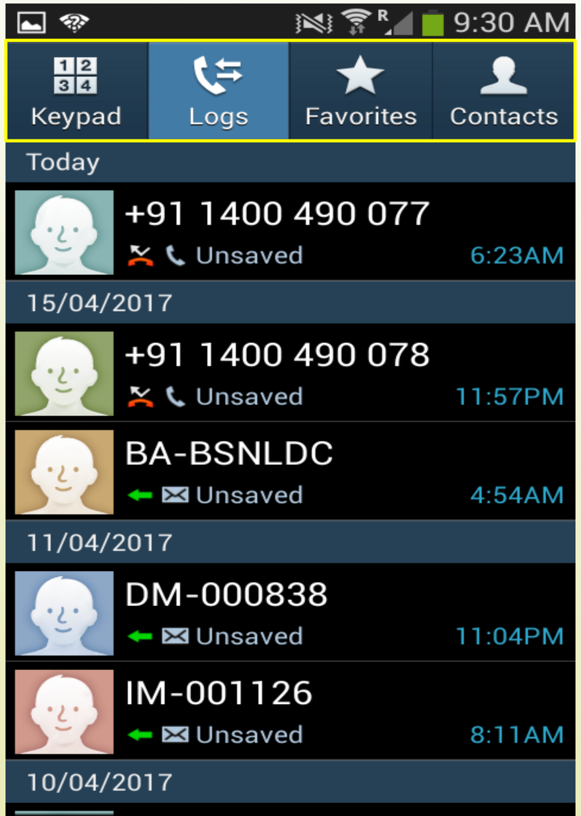
Why do we need Fragments

- Lets take a look on the screens shots.

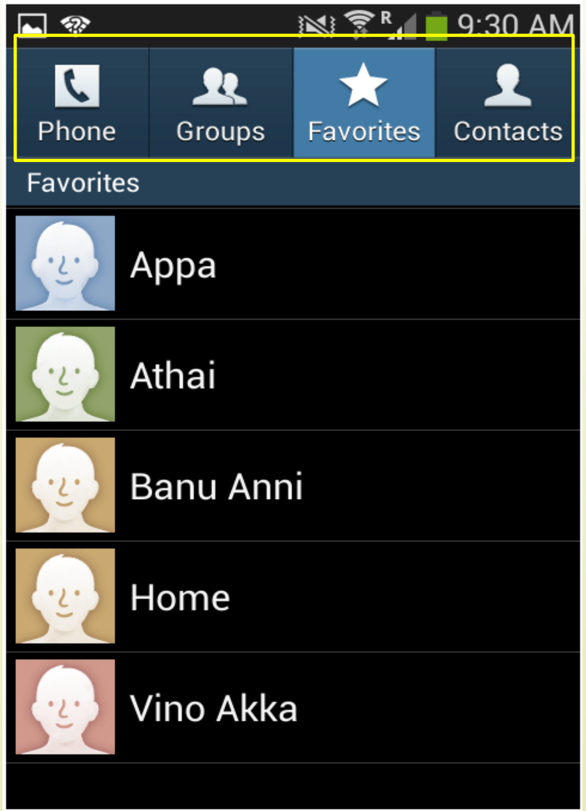


This top header is common to all the screens. Based on the user action only the body of the part changes. Here there is no need to create multiple activities. Instead of that we can use Fragments.

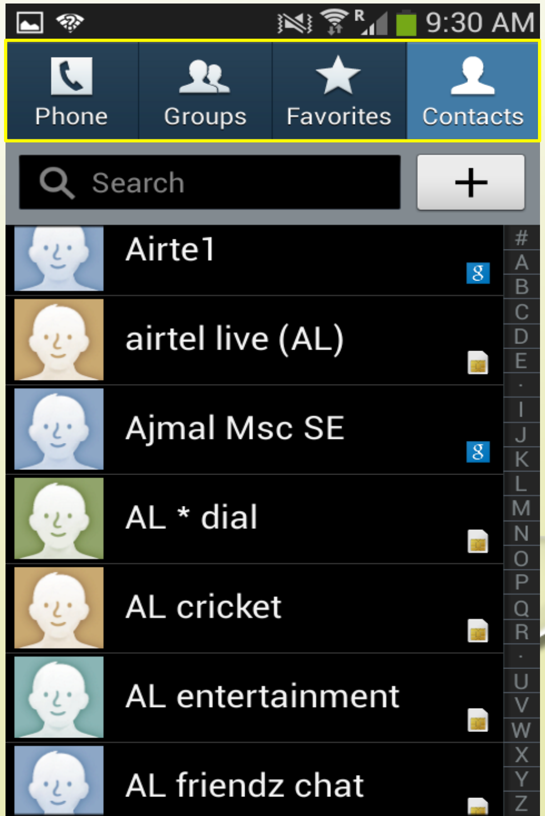




Today
+91 1400 490 077 Unsaved 6:23AM
15/04/2017
+91 1400 490 078 Unsaved 11:57PM
BA-BSNLDC Unsaved 4:54AM
11/04/2017
DM-000838 Unsaved 11:04PM
IM-001126 Unsaved 8:11AM
10/04/2017

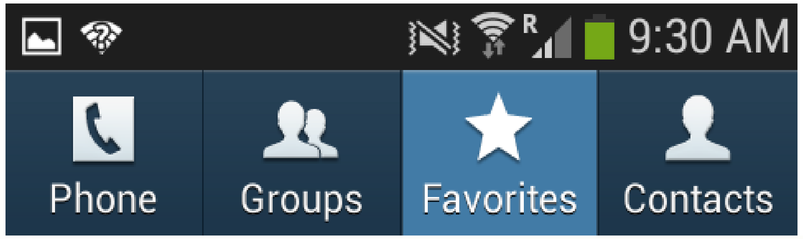


Favorites
Appa
Athai
Banu Anni
Home
Vino Akka



Airte1
airtel live (AL)
Ajmal Msc SE
AL * dial
AL cricket
AL entertainment
AL friendz chat

Drawbacks of having Individual Activity

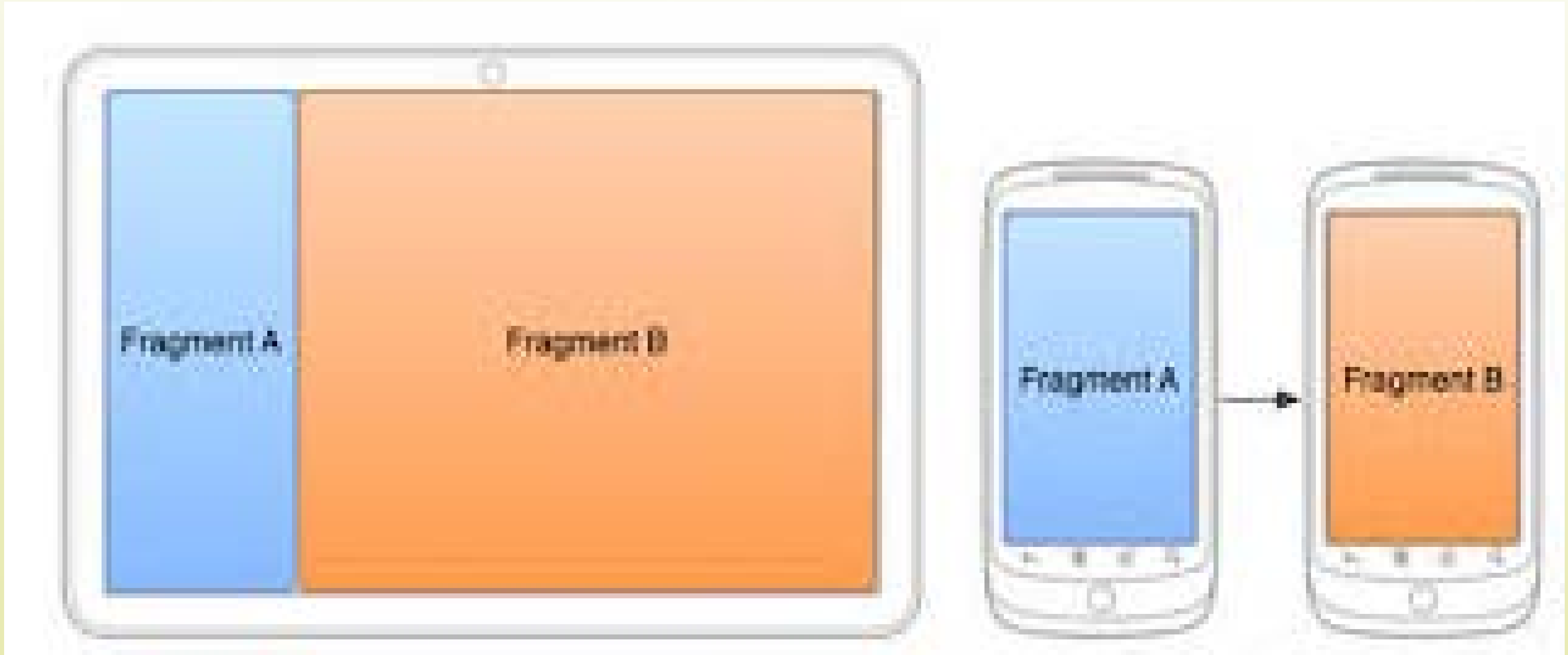


- In the previous example, if we go for the usage of 4 different individual activities, in each activity we have to write the logic for the common part. If we want to modify the common part, need to make changes in all the activities.
- So there are two drawbacks of having different individual activities.
  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

- → 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

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



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

FRAGMENT'S LIFECYCLE

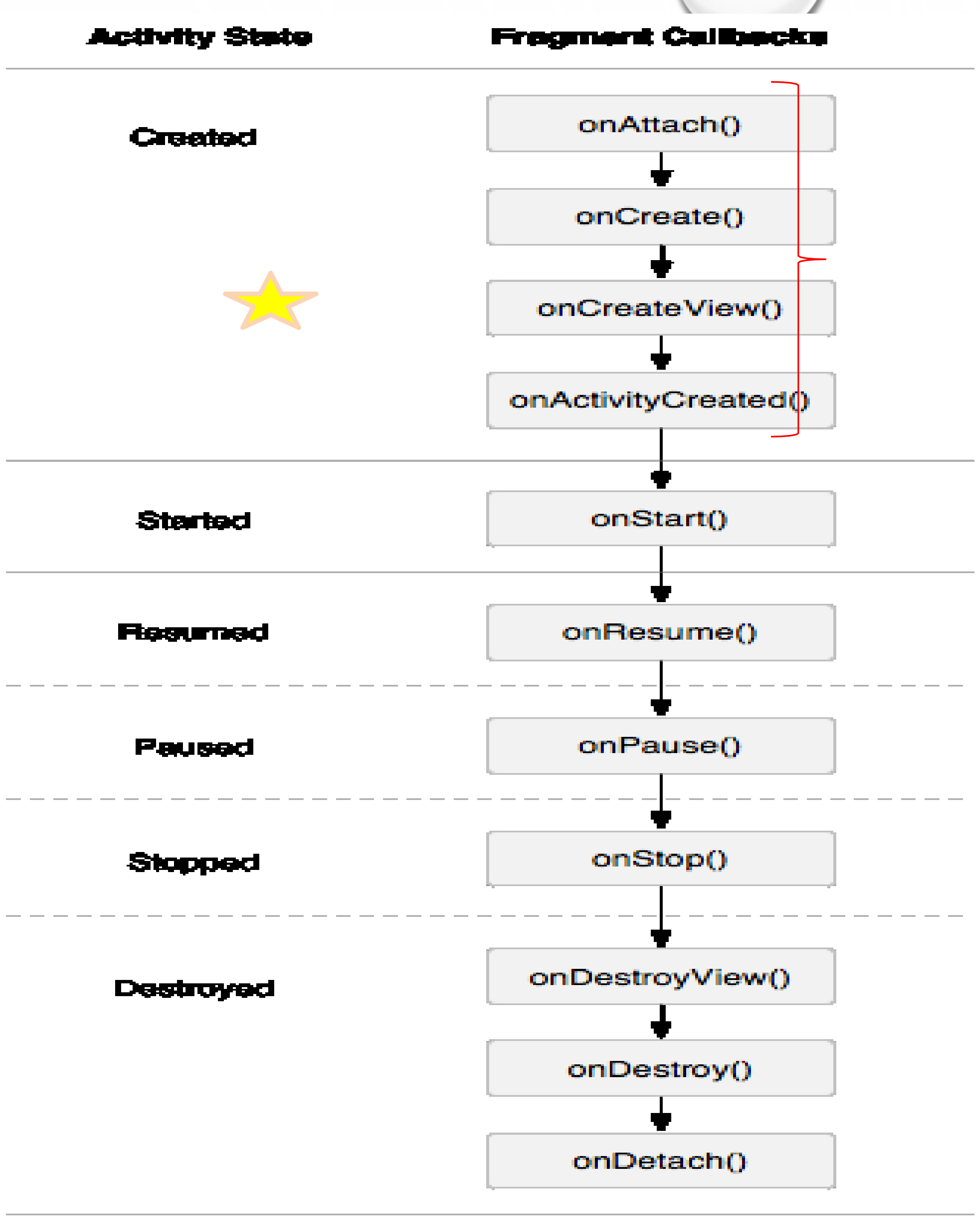
onAttach() Invoked when the fragment has been connected to the host activity.

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">
//    [ Name of your Fragment Class  with package name ]
    <fragment android:name="com.example.android.fragments.HeadlinesFragment
            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() with its Layout

- Class should inherit from Fragment and need to import androidx.fragment.app.Fragment

```
class GalleryFragment : Fragment() {
```

Instantiates a layout XML file into its corresponding View objects.

Activity parent's ViewGroup

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

                  savedInstanceState: Bundle?) : View?{
```

Returns a view object

Bundle that provides data about the instance of the fragment

```
     // inflate the layout for this fragment – Convert XML into View
    return inflater.inflate(R.layout.fragment_gallery, container, false)// First parameter is layout, Second
//parameter is ViewGroup object, third   parameter is boolean type  always false
  }
}
```

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

Example

```
class GalleryFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,  // Parameter 1
        container: ViewGroup?, // Parameter 2
        savedInstanceState: Bundle? // Parameter 3
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_gallery, container, false) // return value
    }
}
```

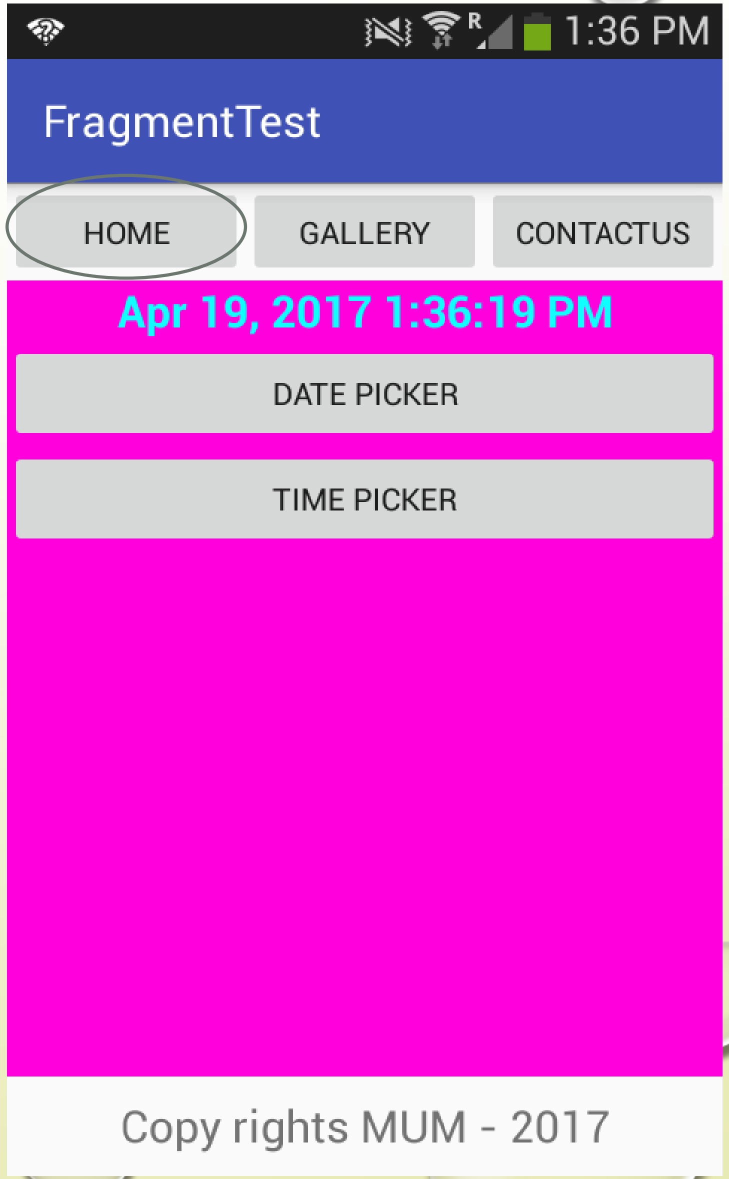
- Get the Fragment into your MainActivity.java by using the following code segments

```
//get FramgemtManager associated with this Activity

val fmanager = supportFragmentManager
// Begin a fragment transaction by calling beginTransaction() returns FragmentTransaction
val fragmentTransaction = fragmentManager.beginTransaction();

//Create instance of your Fragment

ExampleFragment fragment = 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 1

- 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 : FragmentDemo folder



Apr 19, 2017 1:36:19 PM

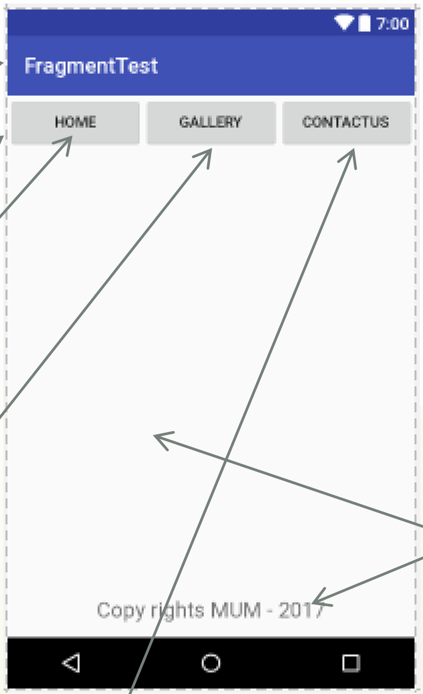
Copy rights MUM - 2017

Linear Layout Weight and Weight Sum

- In android, Linearlayout is a common layout that arranges “component” in vertical or horizontal order, via orientation attribute. In additional, the highest “weight” component will fill up the remaining space in Linearlayout.
- Weight can only be used in Linearlayout. If the orientation of Linearlayout is vertical, then use android:layout_height="0dp" and if the orientation is horizontal, then use android:layout_width = "0dp“
- you can use android:weightSum attribute to the linear layout in addition to android:layout_weight attribute. Ensure that the sum of the android:layout_weight attributes of the children are less than or equal to sum. Then there will be empty space matching for the rest of the room not allocated to the children.

Main Screen Design Code – activity_main.xml

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

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

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

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

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

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

<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>
```



Top Layout Code

Layout Code for all Buttons

Home Button Code

Gallery Button Code

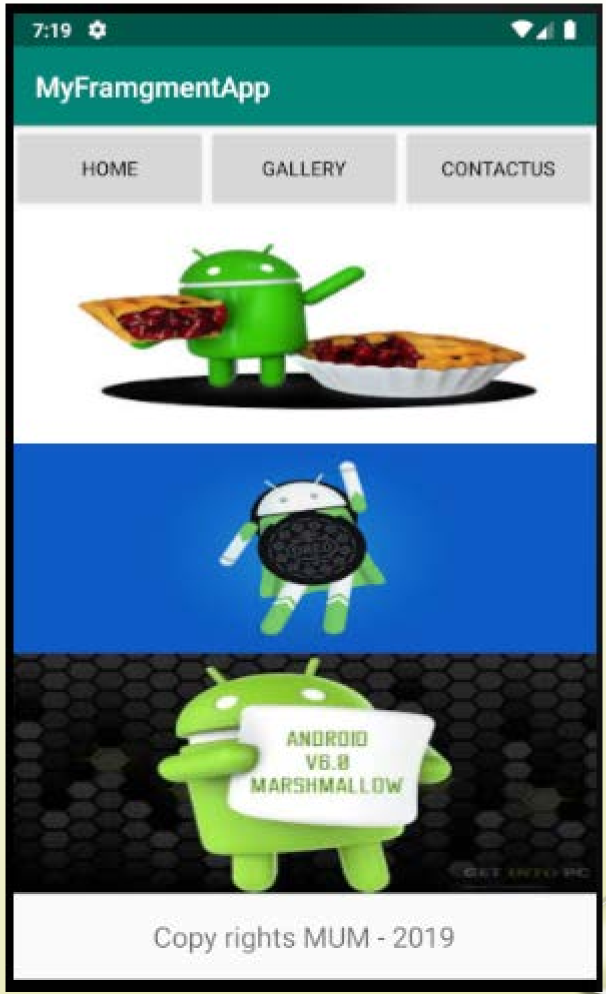
ContactUs button Code

Fragment Code

CODING PART

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

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res
/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="3">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scaleType="fitXY"
        android:src="@drawable/pie">
    </ImageView>
<ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scaleType="fitXY"
        android:src="@drawable/oreo">
    </ImageView>

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scaleType="fitXY"
        android:src="@drawable/mars">
    </ImageView>
</LinearLayout>
```



CODING PART

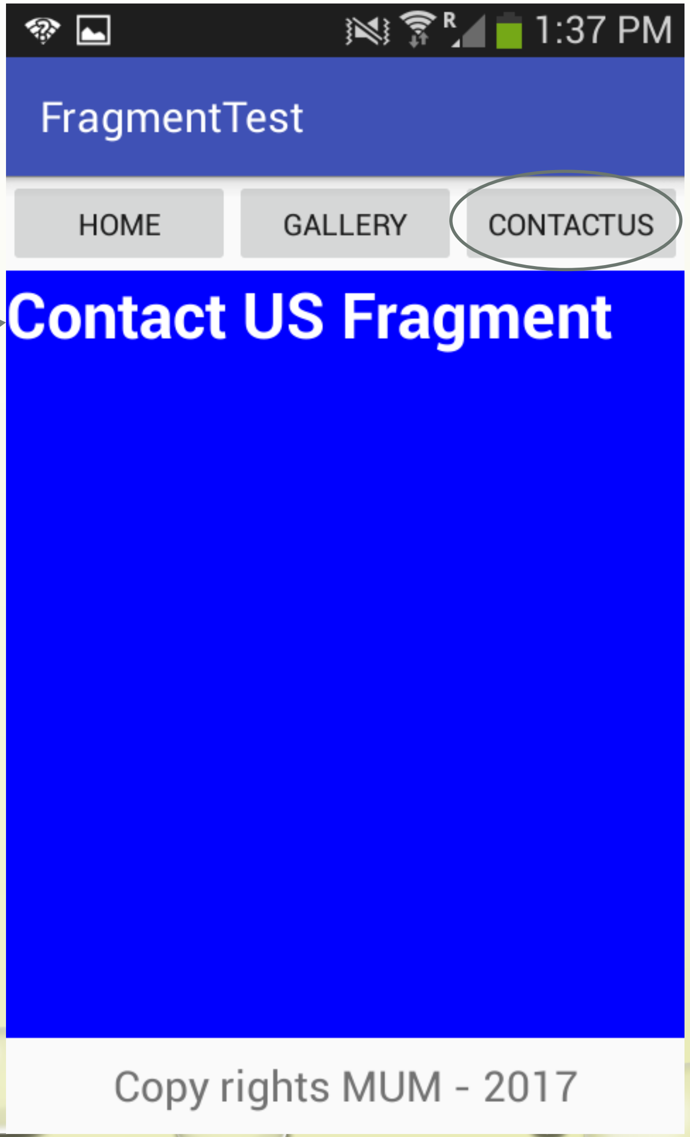
GalleryFragment.kt

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class GalleryFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_gallery, container, false)
    }
    }
```

CODING PART

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

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="3">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scaleType="fitXY"
        android:src="@drawable/pie">
    </ImageView>
    <ImageView
```



Contact US Fragment

Copy rights MUM - 2017

CODING PART

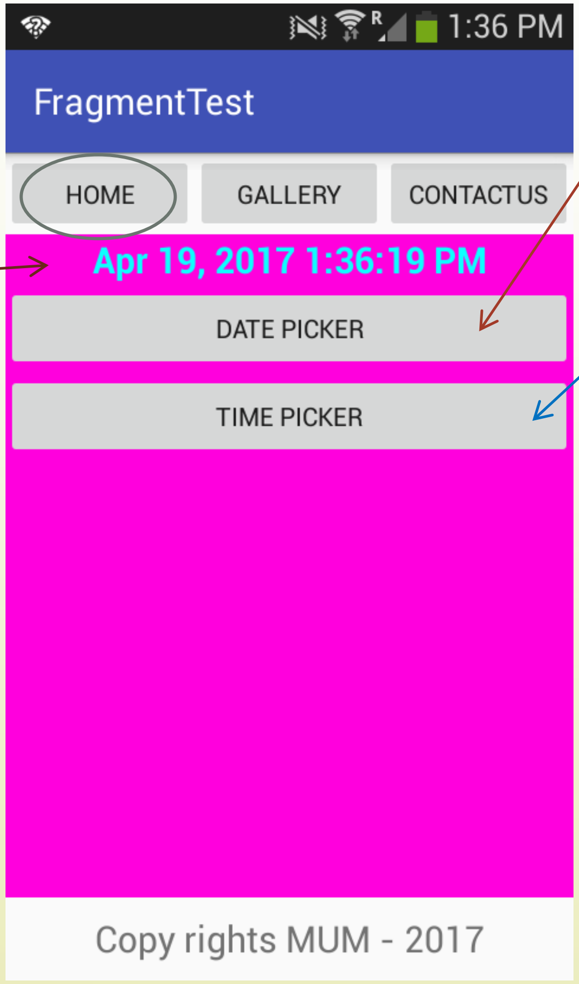
ContactusFragment.kt

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class ContactusFragment : Fragment() {
 override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_contactus, container, false)
    }
}
```

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"/>
```

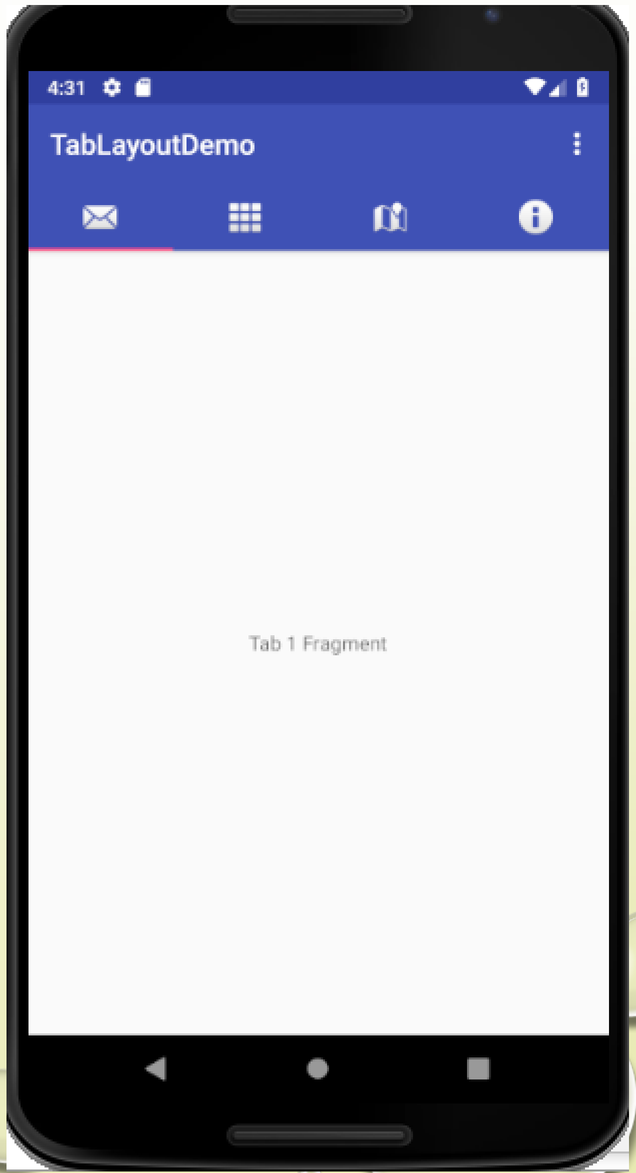


```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bt1"
    android:text="DATE PICKER"/>
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/bt2"
        android:text="TIME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

HomeFragment.kt code refer from Demo Code.

Tabs and Swipes – Hands on Example 2

- Swipe views provide lateral navigation between sibling screens such as tabs with a horizontal finger gesture (a pattern sometimes known as horizontal paging).
- Learn how to create a tab layout with swipe views for switching between tabs.
- You can create swipe views in your app using the ViewPager widget, available in the Support Library.
- TheViewPager is a layout widget in which each child view is a separate page (a separate tab) in the layout.
- To set up your layout with ViewPager, add a <ViewPager> element to your XML layout.
- Refer : Chapter 43 from the Kotlin / Android Studio 3.0 Development Essentials Android 8



Steps for implementing tabs

1. Add the Toolbar and inflate into your MainActivity.java
2. Define the tab layout using tablayout
3. Implement a fragment and its layout for each tab
4. Implement a pageradapter from fragmentpageradapter
5. Create an instance of the tab layout
6. Manage screen views in fragments
7. Set a listener to determine which tab is tapped

Tabs and Swipes

- CoordinatorLayout is a super-powered FrameLayout.
- Toolbar was introduced in Android Lollipop, API 21 release and is the spiritual successor of the ActionBar. Toolbar supports a more focused feature set than ActionBar.
- 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().
- ViewPager is most often used in conjunction with Fragment, which is a convenient way to supply and manage the lifecycle of each page. There are standard adapters implemented for using fragments with the ViewPager, which cover the most common use cases.

Add tab layout below toolbar in activity_main.xml

```
<android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>

<android.support.design.widget.TabLayout
    android:id="@+id/tab_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/toolbar"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

Add view pager below tablayout

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:layout_below="@id/tab_layout" />
```

Create a tab layout in oncreate()

```
var tabLayout = (TabLayout) findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
```

Add the view pager in onCreate()

```
/* Creating a Tabbed Interface using the TabLayout Component for the ViewPager and the
    TabLayout component added to the page change listener*/
      val adapter = PagerAdapter(supportFragmentManager, tab_layout.tabCount)
      pager.adapter = adapter  // pager is the ViewPager Component

pager.addOnPageChangeListener(TabLayout.TabLayoutOnPageChangeListener(tab_layout))
```

Add the listener in oncreate()

```
tab_layout.addOnTabSelectedListener(object : TabLayout.
        OnTabSelectedListener {
            override fun onTabSelected(tab: TabLayout.Tab) {
                pager.currentItem = tab.position
            }

            override fun onTabUnselected(tab: TabLayout.Tab) {
            }

            override fun onTabReselected(tab: TabLayout.Tab) {
            }
        })
```

Create a layout and class for each fragment

1. Choose File → New → Fragment → Fragment (Blank).
2. Name the fragment TabFragment1
3. Create a Fragment Layout XML file and named as tab_fragment1

```
class Tab1Fragment:Fragment() {
public override fun onCreateView(inflater:LayoutInflater?, container:ViewGroup?,
savedInstanceState:Bundle?):View? {
 // Inflate the layout for this fragment
        return inflater!!.inflate(R.layout.fragment_tab1, container, false)
}
}
```

tab_fragment1.xml

```
<?xml version="1.0"    encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="These   are    the top    stories:"
    android:textAppearance="?android:attr/textAppearanceLarge"/>
</RelativeLayout>
```

Add a PagerAdapter

The adapter-layout manager pattern lets you provide different screens of content within an activity—use an adapter to fill the content screen to show in the activity, and a layout manager that changes the content screens depending on which tab is selected.

PagerAdapter Class

```
class TabPagerAdapter(fm: FragmentManager, private var tabCount: Int) :
FragmentPagerAdapter(fm) {
    override fun getItem(position: Int): Fragment? {
        when (position) {
            0 -> return Tab1Fragment()
            1 -> return Tab2Fragment()
            2 -> return Tab3Fragment()
            3 -> return Tab4Fragment()
            else -> return null
        }
    }
    override fun getCount(): Int {
        return tabCount
    }
}
```

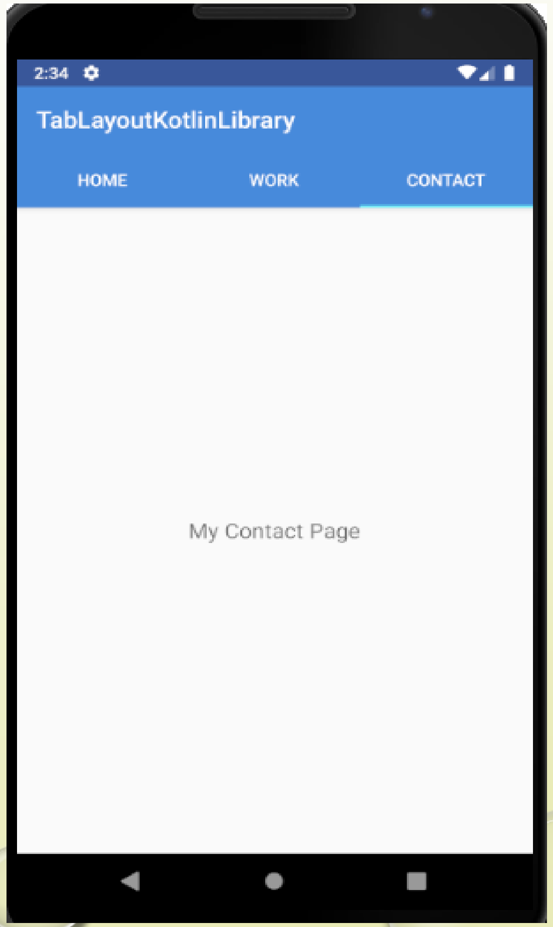
Refer : TabLayoutDemo

Example 3 -Tab Layout with new Kotlin support library

Refer the Step by Implementation : Lesoon-7-Day-2-Tab Layout Step by Step Implementation using new Kotlin support Library.pdf

Dependency : implementation **'com.google.android.material:material:1.0.0'**

Refer : TabLayoutKotlinLibrary



My Contact Page

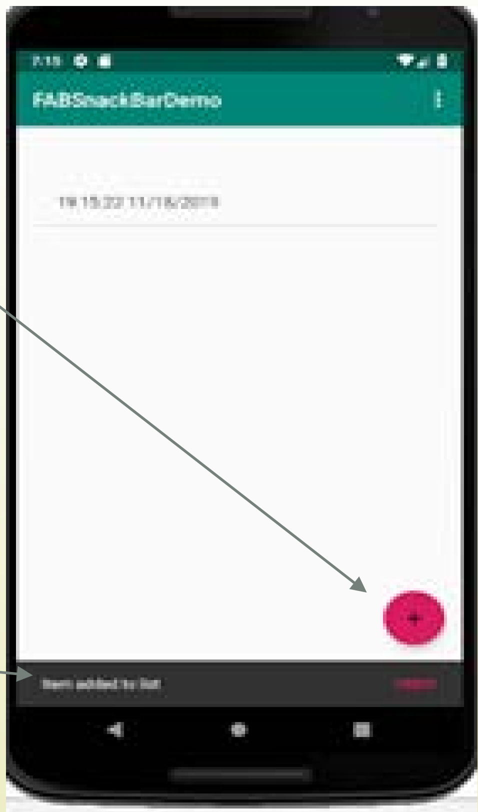
Design Support Library - Material Design

- Material Design is a visual language that synthesizes the classic principles of good design with the innovation of technology and science.
- Add the below dependency to get the Design support library

  implementation 'com.google.android.material:material:1.0.0'
- Learn more about Material Design in this source *https://www.google.com/design/spec/material-design/introduction.html*

Basic Activity
The Floating Action Button (FAB) with SnackBar

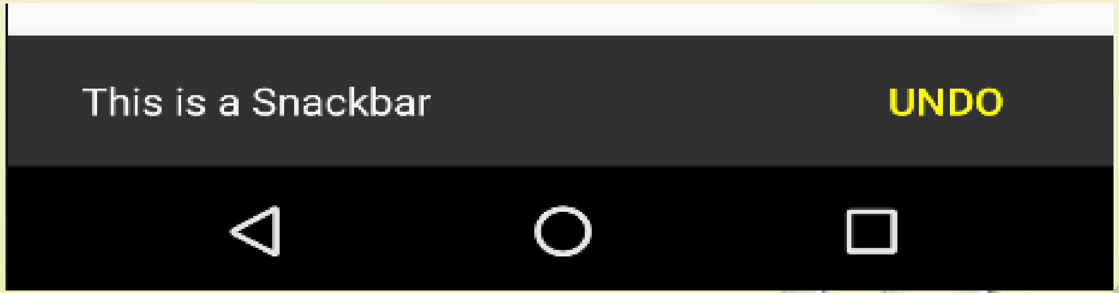
- Material design also dictates the layout and behavior of many standard user interface elements.
- The Floating Action Button(FAB) is a button which appears to float above the surface of the user interface of an app and is generally used to promote the most common action within a user interface screen.
- A floating action button might, for example, be placed on a screen to allow the user to add an entry to a list of contacts or to send an email from within the app.
- The Snackbar component provides a way to present the user with information in the form of a panel that appears at the bottom of the screen.
- Snackbar instances contain a brief text message and an optional action button which will perform a task when tapped by the user. Once displayed, a Snackbar will either timeout automatically or can be removed manually by the user via a swiping action.
- During the appearance of the Snackbar the app will continue to function and respond to user interactions in the normal manner.



SnackBar vs Toast

- Toasts are generally used when we want to display an information to the user regarding some action that has successfully (or not) happened and this action does not require the user to take any other action.

- Snackbars are also used to display an information. But we can give the user an opportunity to take an action. For example, let's say the user deleted a picture by mistake and he wants to get it back.



SnackBar

val mySnackbar = Snackbar.make(view, string, duration)

Parameters

view

- The view to attach the Snackbar to. The method actually searches up the view hierarchy from the passed view until it reaches either a CoordinatorLayout, or the window decor's content view. Ordinarily, it's simplest to just pass the CoordinatorLayout enclosing your content.

string

- The resource ID of the message you want to display. This can be formatted or unformatted text.

duration

- The length of time to show the message. This can be either LENGTH_SHORT or LENGTH_LONG.

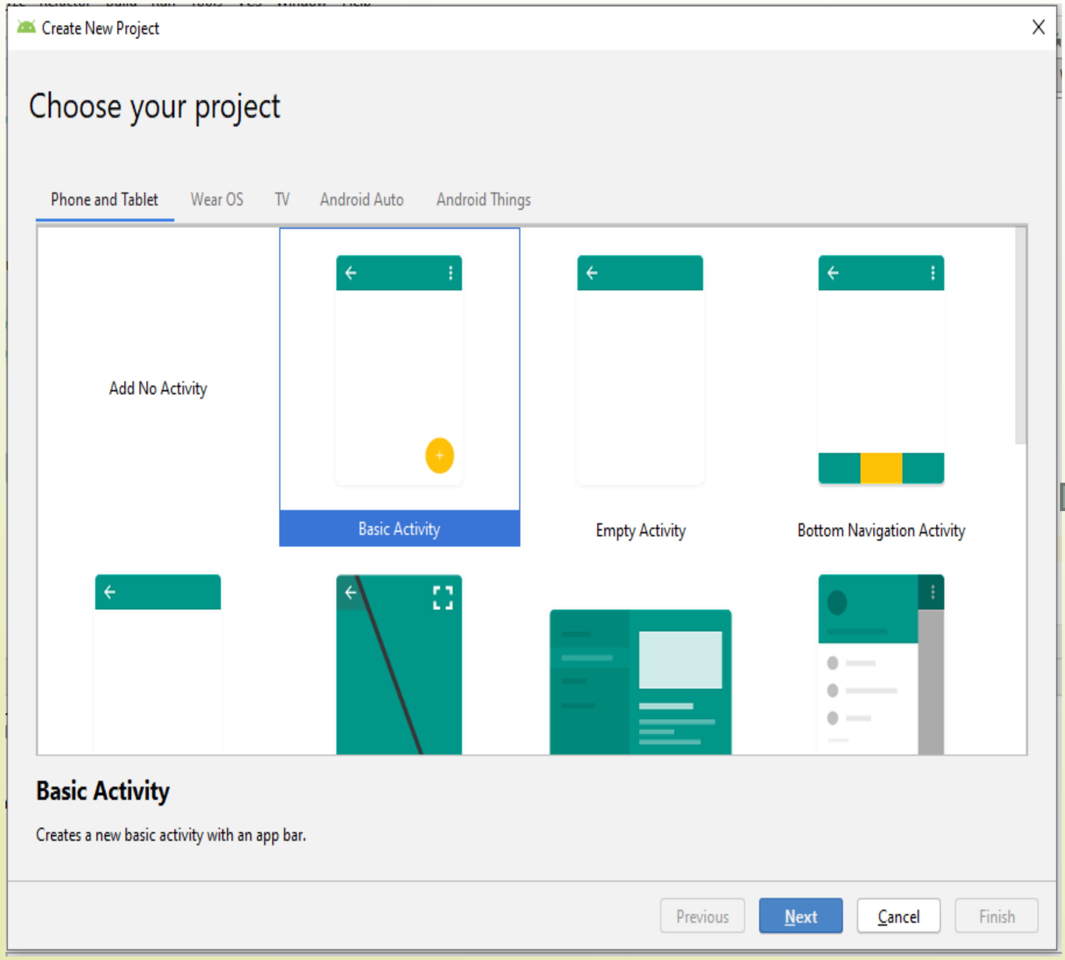
Once you have created the Snackbar, call its show() method to display the Snackbar to the user

mySnackbar.show()

- To add an action to a Snackbar message, you need to define a listener object that implements the View.OnClickListener interface. The system calls your listener's onClick() method if the user clicks on the message action.

To Create FAB with SnackBar

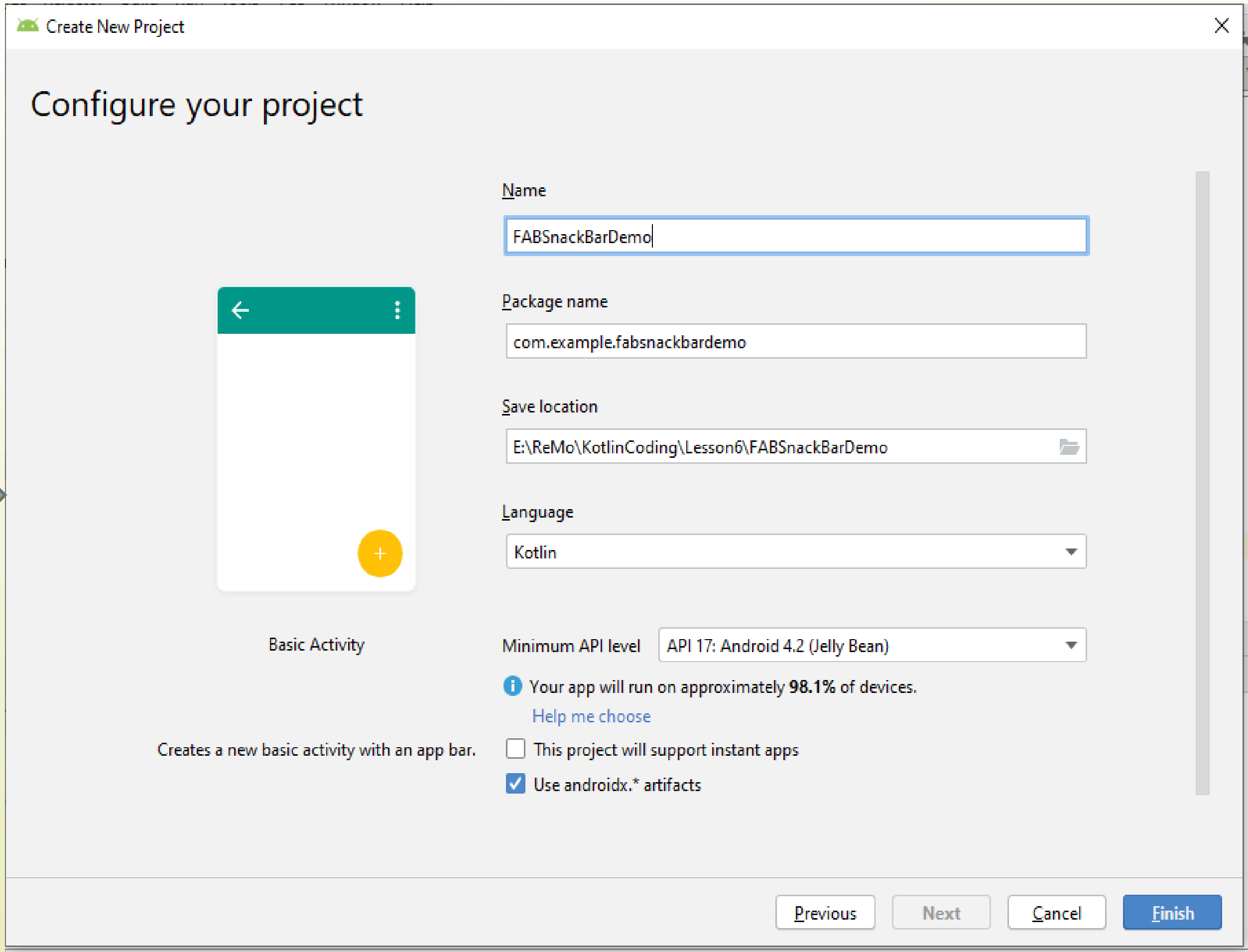
- Create a New Project and Choose Basic Activity, you will get the UI with Start up Code


Create New Project

Choose your project

Basic Activity

Creates a new basic activity with an app bar.


Create New Project

Configure your project

Name: FABSnackBarDemo

Package name: com.example.fabsnackbardemo

Save location: E:\ReMo\KotlinCoding\Lesson6\FABSnackBarDemo

Language: Kotlin

Minimum API level: API 17: Android 4.2 (Jelly Bean)

Your app will run on approximately **98.1%** of devices.

Help me choose

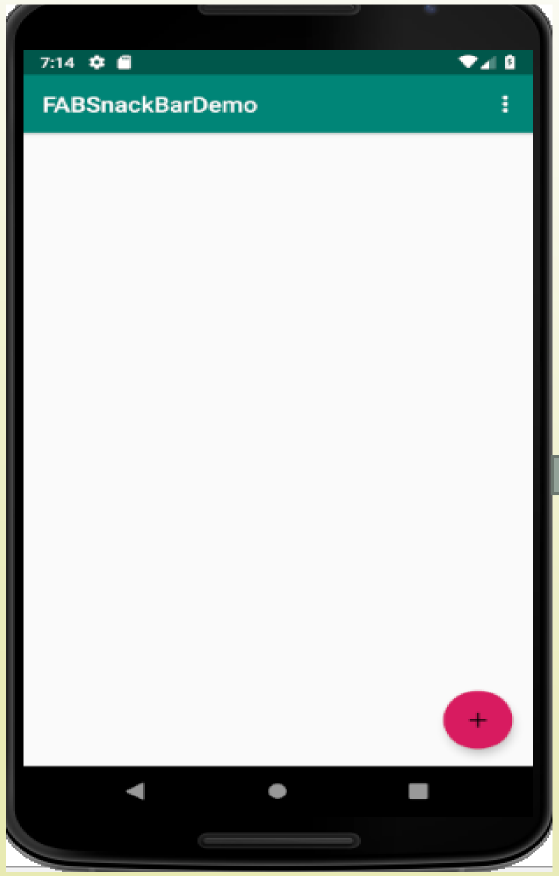
Basic Activity

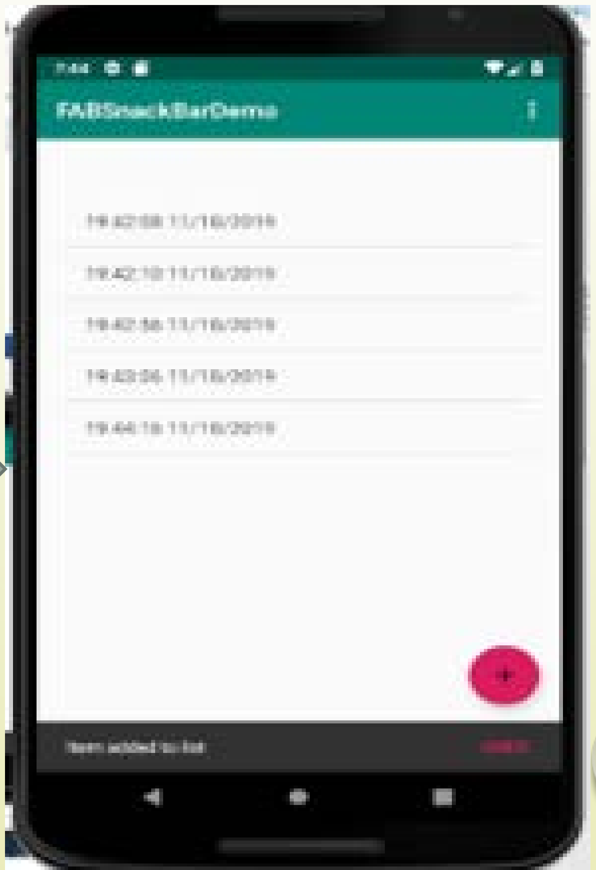
Creates a new basic activity with an app bar.

- [ ] This project will support instant apps
- [x] Use androidx.* artifacts

Example - 4

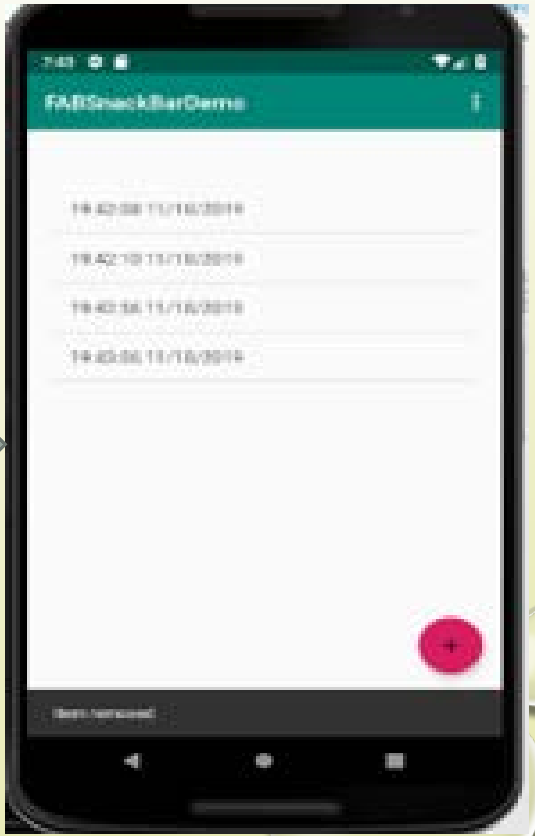
- Initial screen appeared with Floating Action Button. Once you clicked the FAB will add current date and time in the listview at the same time you will get SnackBar message at bottom, click the UNDO action from the SnackBar to Undo the last item in the list.
- Refer Example : FABSnackBarDemo


7:14
FABSnackBarDemo


FABSnackBarDemo

19:42:08 11/16/2019
19:42:10 11/16/2019
19:42:56 11/16/2019
19:43:06 11/16/2019
19:44:16 11/16/2019

Item added to list


FABSnackBarDemo

19:42:08 11/16/2019
19:42:10 11/16/2019
19:42:56 11/16/2019
19:43:06 11/16/2019

Item removed

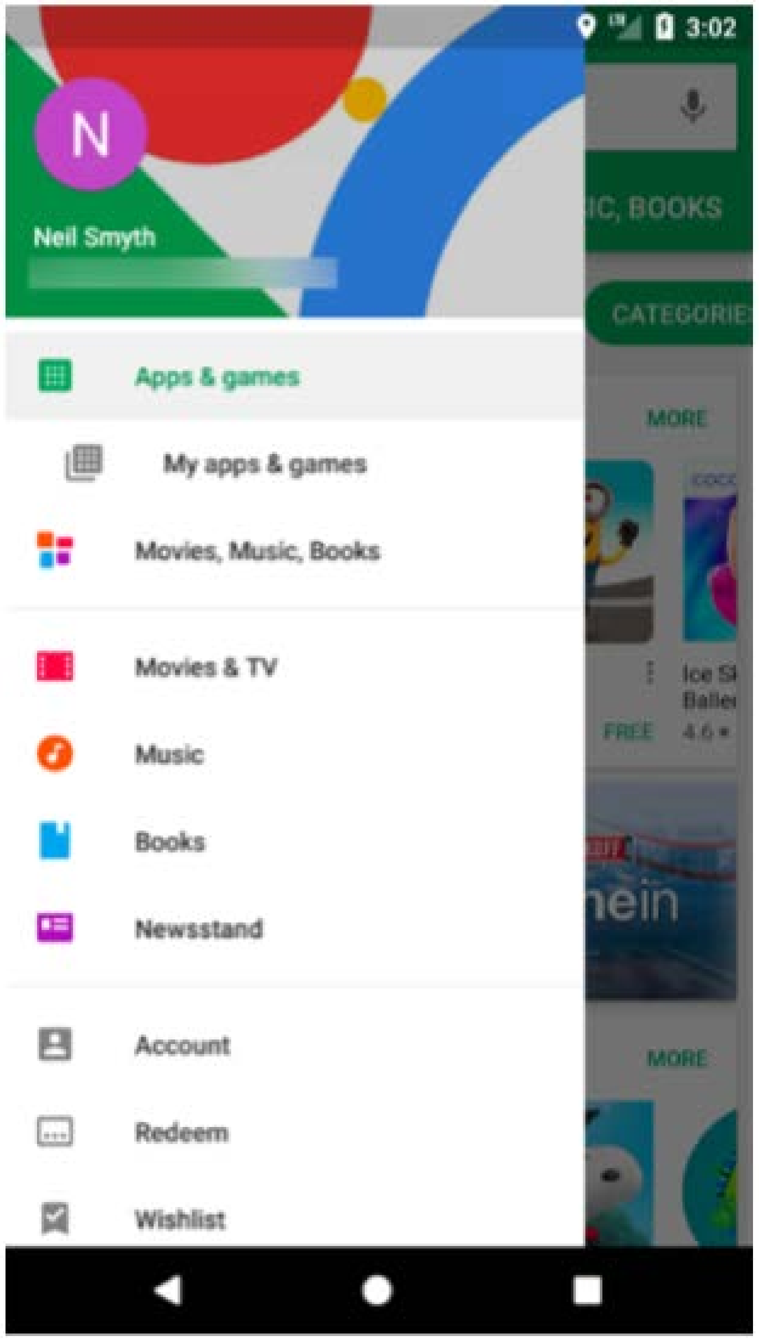
Navigation Drawer

- The Navigation drawer is a panel that slides out from the left of the screen and contains a range of options available for selection by the user, typically intended to facilitate navigation to some other part of the application.

A navigation drawer is made up of the following components:

- An instance of the DrawerLayout component.
- An instance of the NavigationView component embedded as a child of the DrawerLayout.

Refer : NavigationDrawerDemo



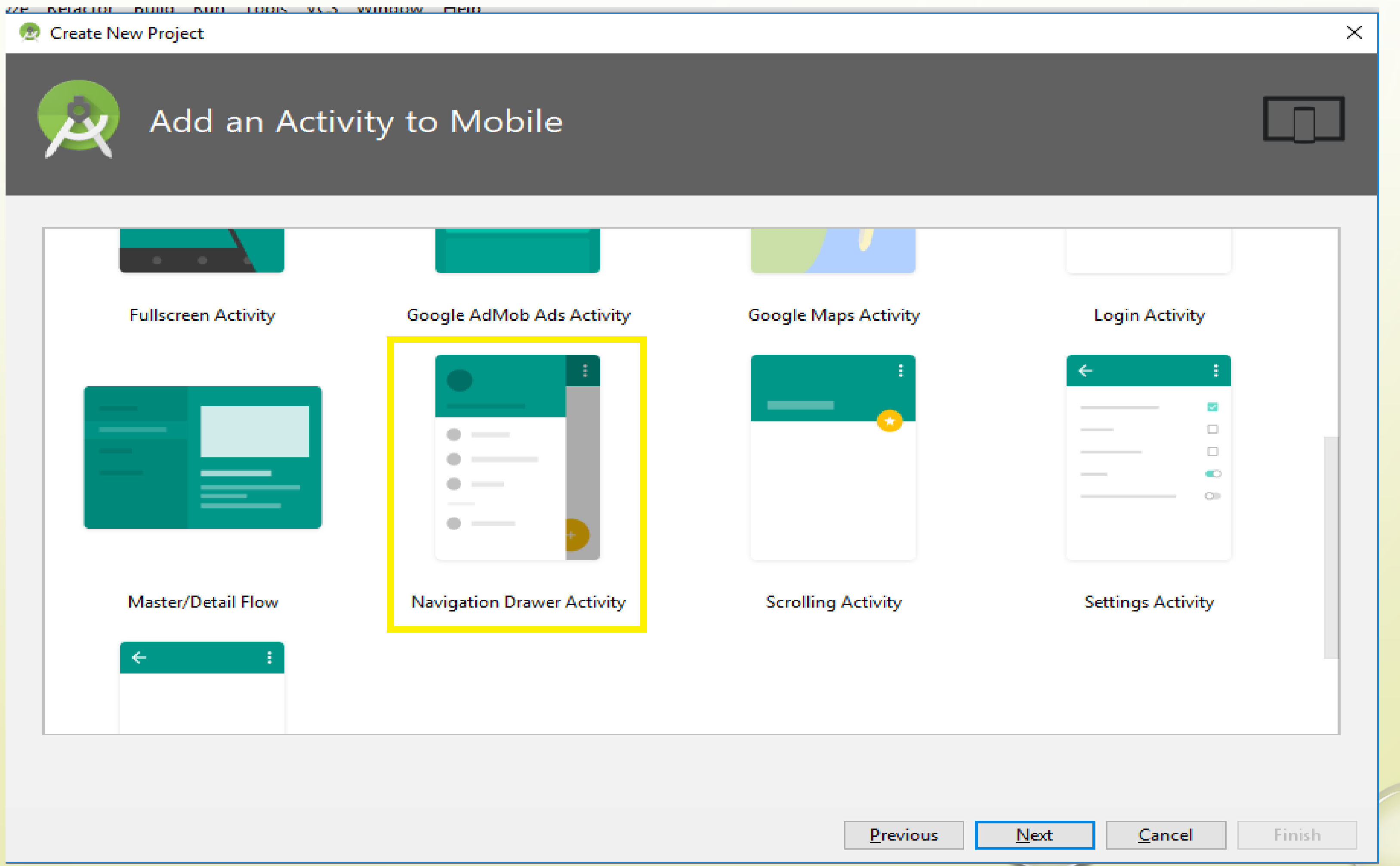
Implementing Navigation Drawer

- A menu resource file containing the options to be displayed within the navigation drawer.
- An optional layout resource file containing the content to appear in the header section of the navigation drawer.
- A listener assigned to the NavigationView to detect when an item has been selected by the user.
- An ActionBarDrawerToggle instance to connect and synchronize the navigation drawer to the app bar.
- The ActionBarDrawerToggle also displays the drawer indicator in the app bar which presents the drawer when tapped.

Navigation Drawer

- **Adding the Navigation Drawer**

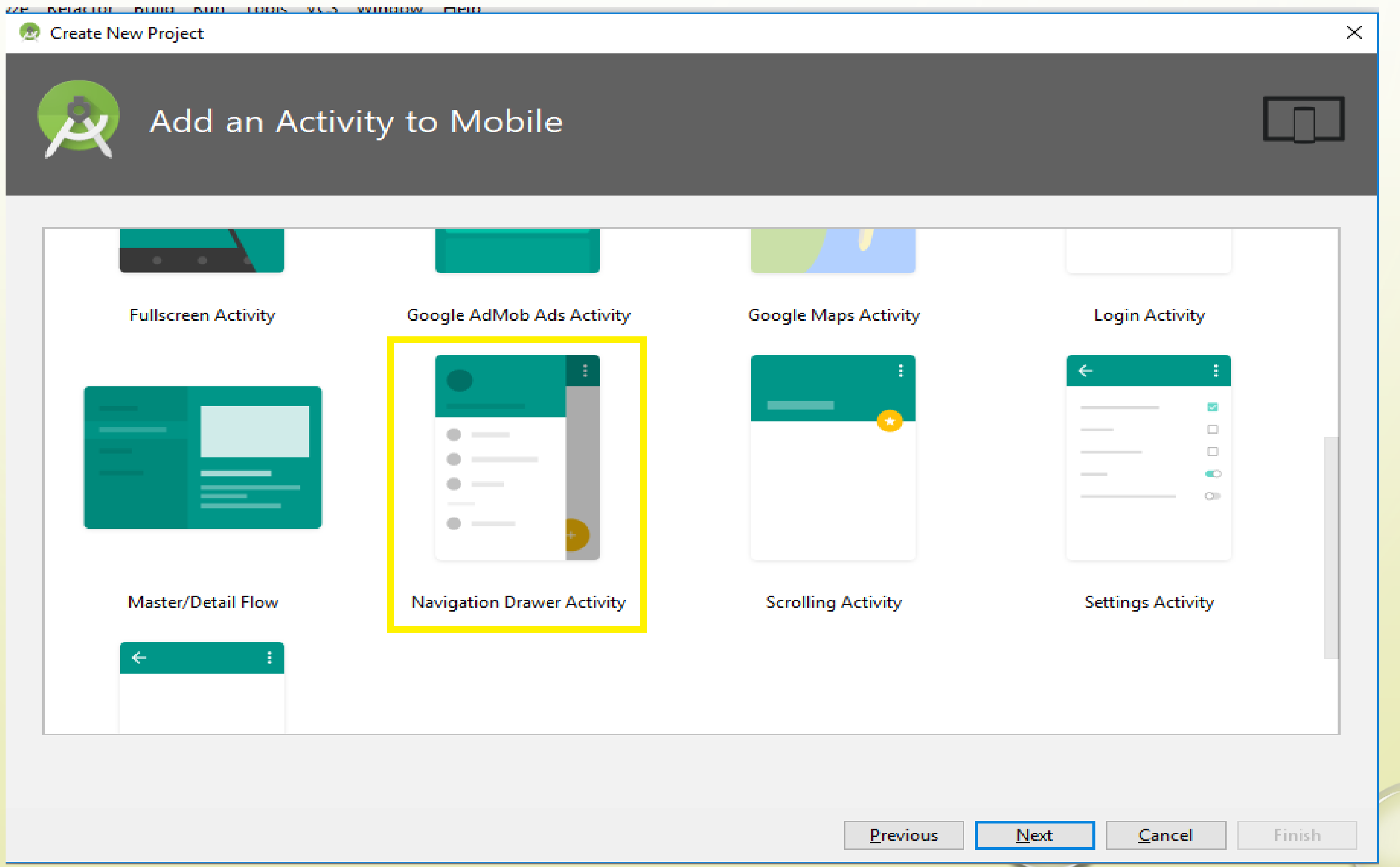
- Create a new Project, choose Navigation Drawer Activity instead of Empty activity.



Navigation Drawer

- **Adding the Navigation Drawer**

- Create a new Project, choose Navigation Drawer Activity instead of Empty activity.



The Template Layout Resource Files

- **activity_main.xml** – This is the top level layout resource file. It contains the DrawerLayout container and the NavigationView child. The NavigationView declaration in this file indicates that the layout for the drawer header is contained within the nav_header_main.xml file and that the menu options for the drawer activity_mainare located in the activity_main_drawer.xml file. In addition, it includes a reference to the app_ bar_main.xml file.
- **app_bar_main.xml** – This layout resource file is included by the activity_main.xml file and is the standard app bar layout file built within a CoordinatorLayout container as covered in the preceding chapters. As with previous examples this file also contains a directive to include the content file which, in this case, is named content_main.xml.
- **content_main.xml** – The standard layout for the content area of the activity layout. This layout consists of a ConstraintLayout container and a "Hello World!" TextView.
- **nav_header_main.xml** – Referenced by the NavigationView element in the activity_main.xml

onBackPressed()

This method is added to handle situations whereby the activity has a “back” button to return to a previous activity screen.

```
override fun onBackPressed() {
        val drawer = findViewById<DrawerLayout>(R.id.drawer_layout)
        if (drawer.isDrawerOpen(GravityCompat.START)){
            drawer.closeDrawer(GravityCompat.START)
        }
        else {
            super.onBackPressed()
        }
    }
```

Responding to Drawer Item Selections

- Handling selections within a navigation drawer is a two-step process. The first step is to specify an object to act as the item selection listener. This is achieved by obtaining a reference to the NavigationView instance in the layout and making a call to its setNavigationItemSelectedListener() method, passing through a reference to the object that is to act as the listener.
- Implement NavigationView.OnNavigationItemSelectedListener

```
    class MainActivity : AppCompatActivity(), NavigationView.OnNavigationItemSelectedListener
```

- The second step is to implement the onNavigationItemSelected() method within the designated listener.

```
        val navView: NavigationView = findViewById(R.id.nav_view)
         navView.setNavigationItemSelectedListener(this)
```

- Override the onNavigationItemSelected() method

```
override fun onNavigationItemSelected(item: MenuItem): Boolean {// Handle navigation view item clicks here.
        return when (item.itemId) {
         R.id.nav_gallery -> { // Handle the camera action
            Toast.makeText(applicationContext, "Gallery Selected", Toast.LENGTH_LONG).show()
            return true
         }
```

Responding to Drawer Item Selections

```
R.id.nav_slideshow -> {
            Toast.makeText(applicationContext, "Gallery Selected", Toast.LENGTH_LONG).show()
            true }
 R.id.nav_send -> {
            Toast.makeText(applicationContext, "Gallery Selected", Toast.LENGTH_LONG).show()
            true  }
         else -> {
            val drawer = findViewById<DrawerLayout>(R.id.drawer_layout)
           drawer.closeDrawer(GravityCompat.START)
            true  }
      }
   }
```