LESSON – 7- DAY 2

FRAGMENTS

AGENDA

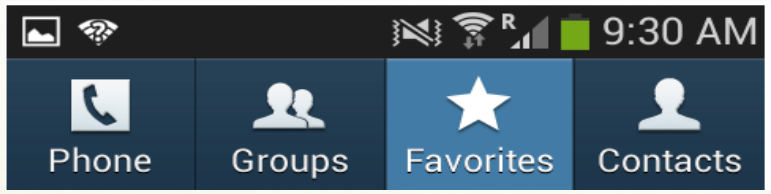
- Introduction to Fragments
- Why do we need Fragments
- How to define Fragments is XML
- How to Create a Fragment Class
- How to add Fragments in Activity
- About Weight property(How to design screen for multiple device size)
- Hands on Examples
- Design support Library – Material Design
  - FloatingActionButton
  - SnackBar
  - Tabs
  - NavigationDrawer

INTRODUCTION

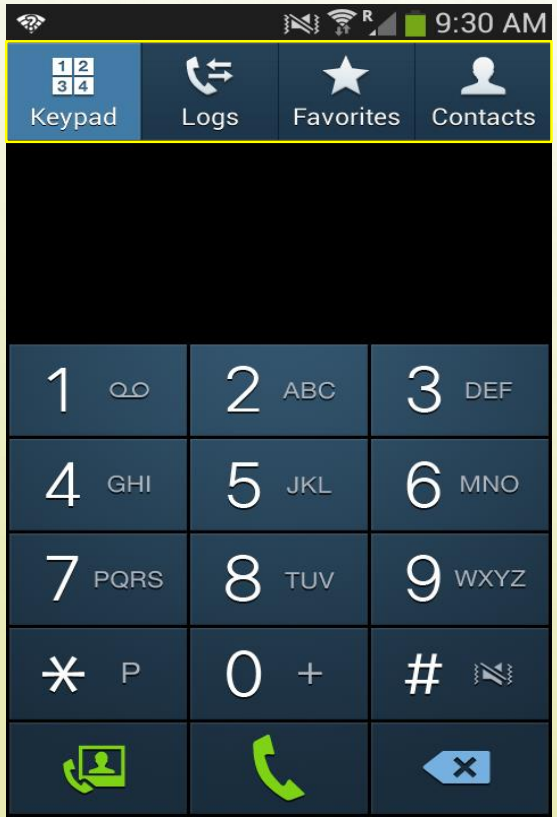
- Android 3.0 introduces a finer-grained application component called Fragment that lets you modularize the application and its user interface (into fragments).
- A Fragment represents a behavior or a portion of user interface in a FragmentActivity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities
- It 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).
- 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

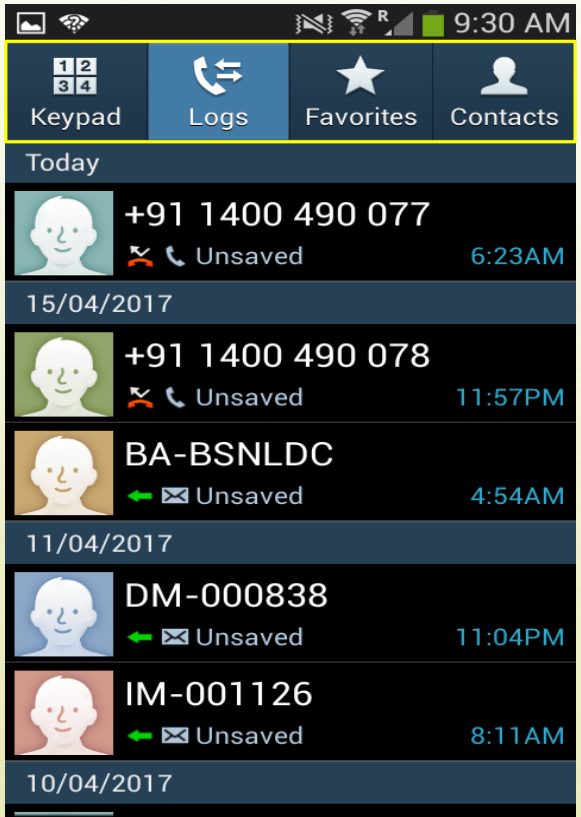
Why do we need Fragments

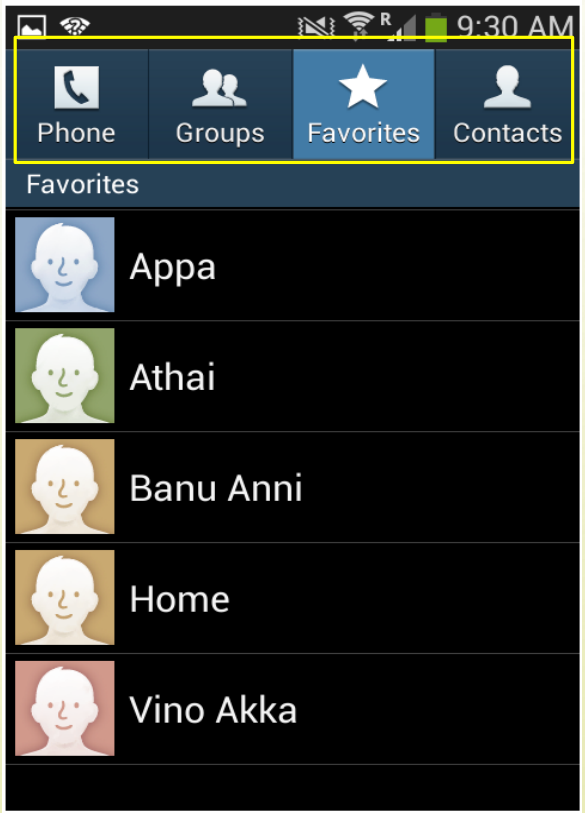
- **Let's** take a look on the screens shots.

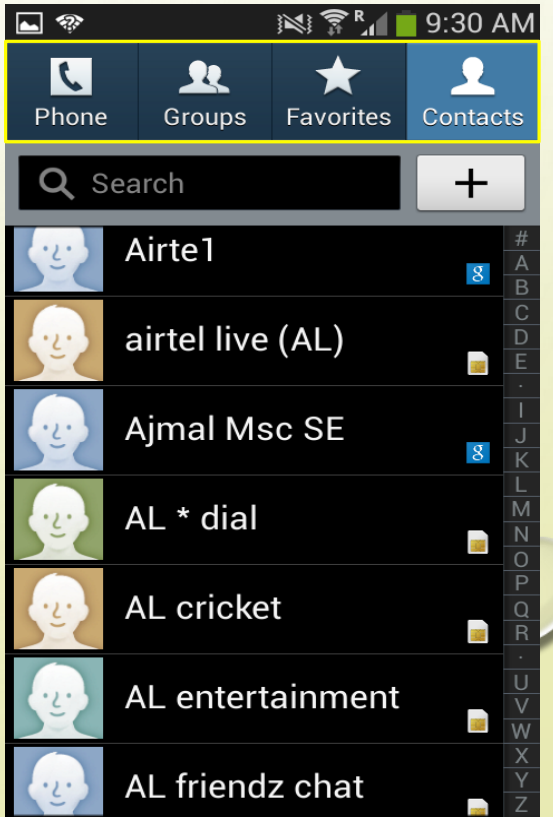


This top header is common to all the screens. Based on the user action only the body of the part changes. Here there is no need to create multiple activities. Instead of that we can use Fragments.

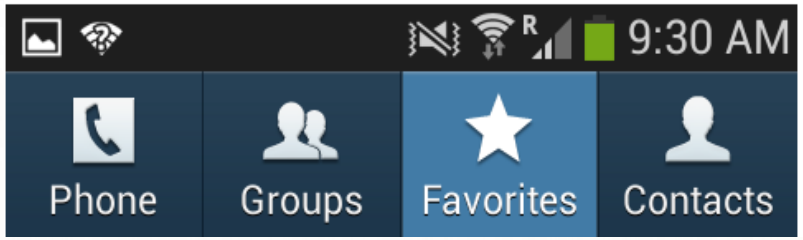








Drawbacks of having Individual Activity



- In the previous example, if we go for the usage of 4 different individual activities, in each activity we must 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

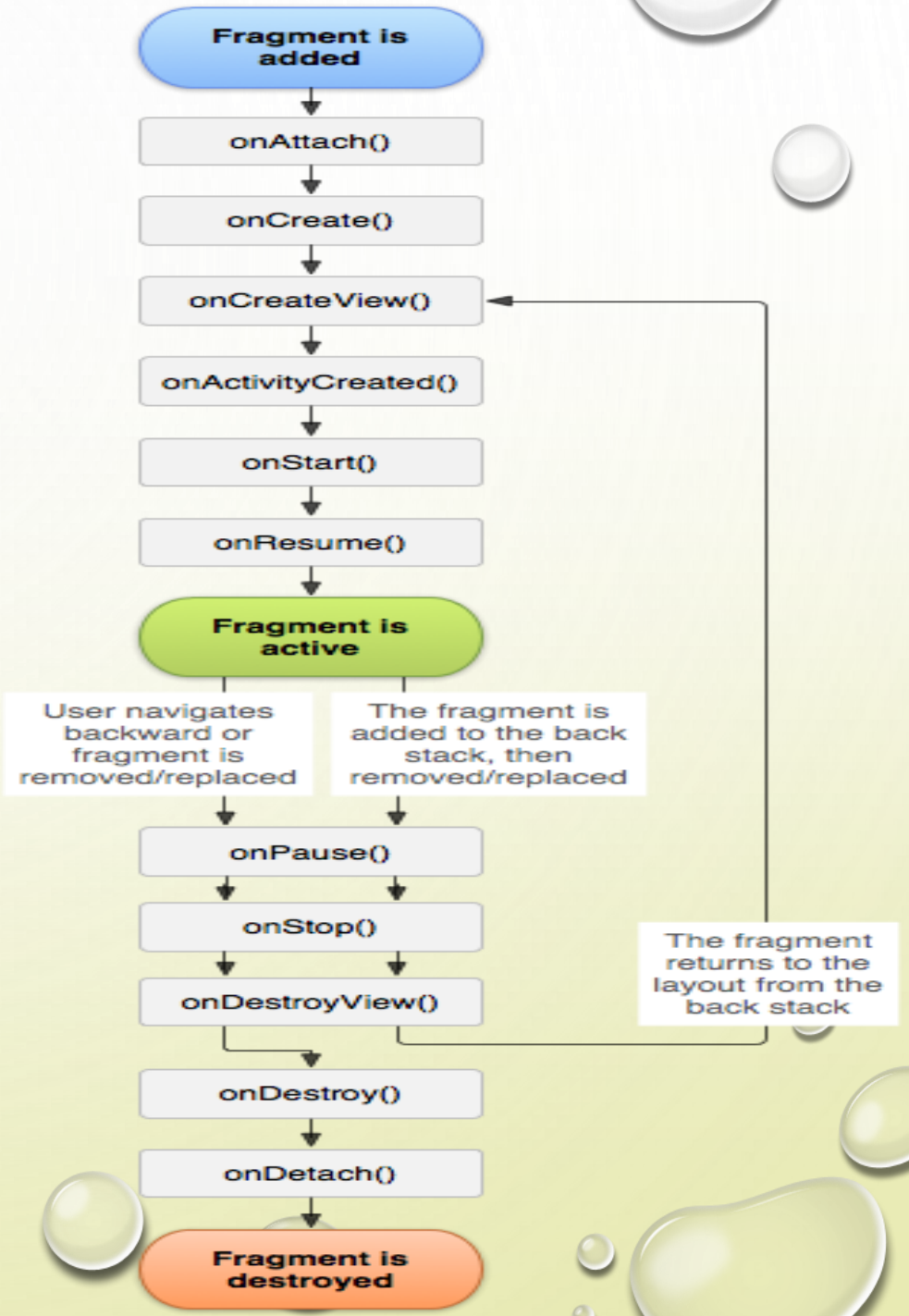
primarily to support more dynamic and **flexible UI designs on large screens, such as tablets**.

- → 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'S LIFECYCLE

- To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).
- The Fragment class has code that looks a lot like an Activity.
- It contains callback methods similar to an activity, such as onCreate(), onStart(), onPause(), and onStop().



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, extends the Fragment class, then override key lifecycle methods to insert your app logic, like 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() {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?) : View?{
    // 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
    }
}
```

Instantiates a layout XML file into its corresponding View objects.

Activity parent's ViewGroup

Returns a view object

Bundle that provides data about the instance of the fragment

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
    }
}
```

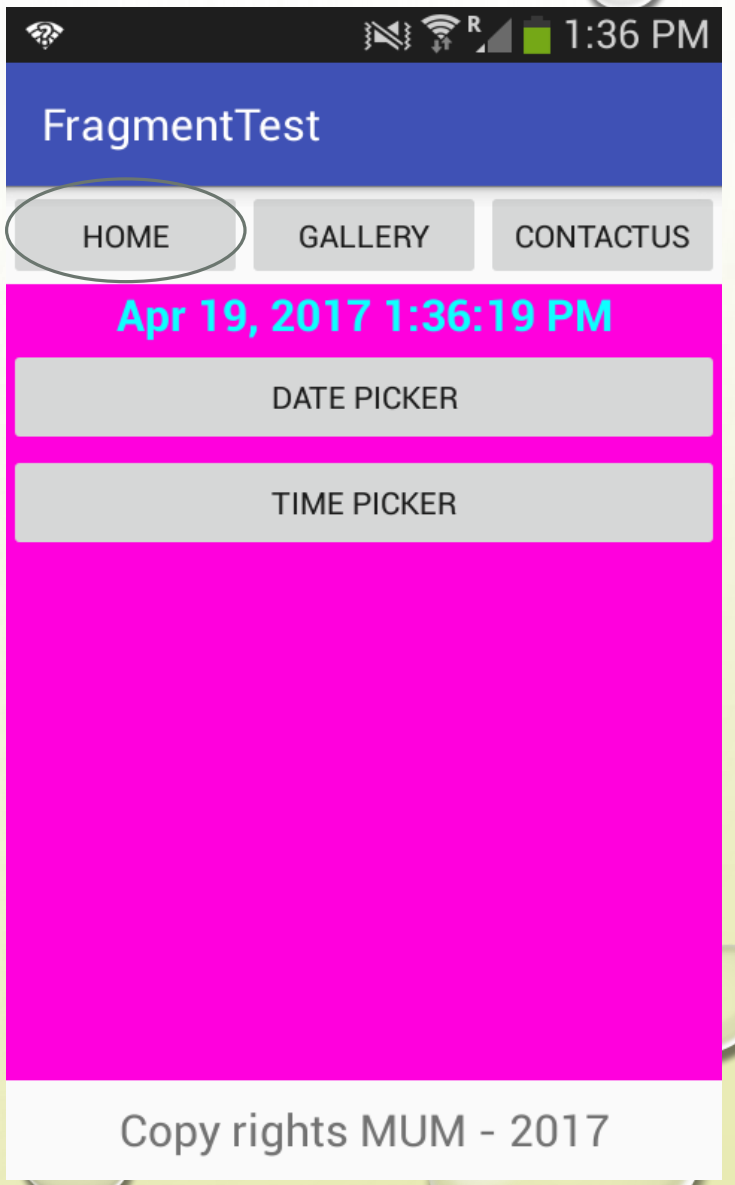
- Get the Fragment into your MainActivity.java by using the following code segments

```
//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



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

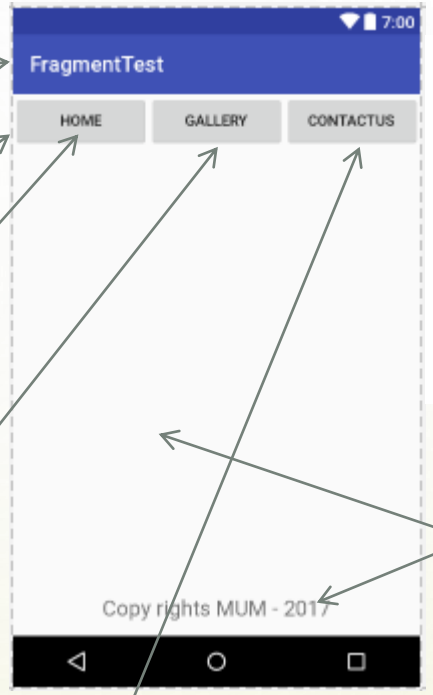
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="0.1"
    android:orientation="horizontal"
    >
```

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"
    />
```




```
<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>
```

Fragment Code

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

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>
```

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
```

CODING PART

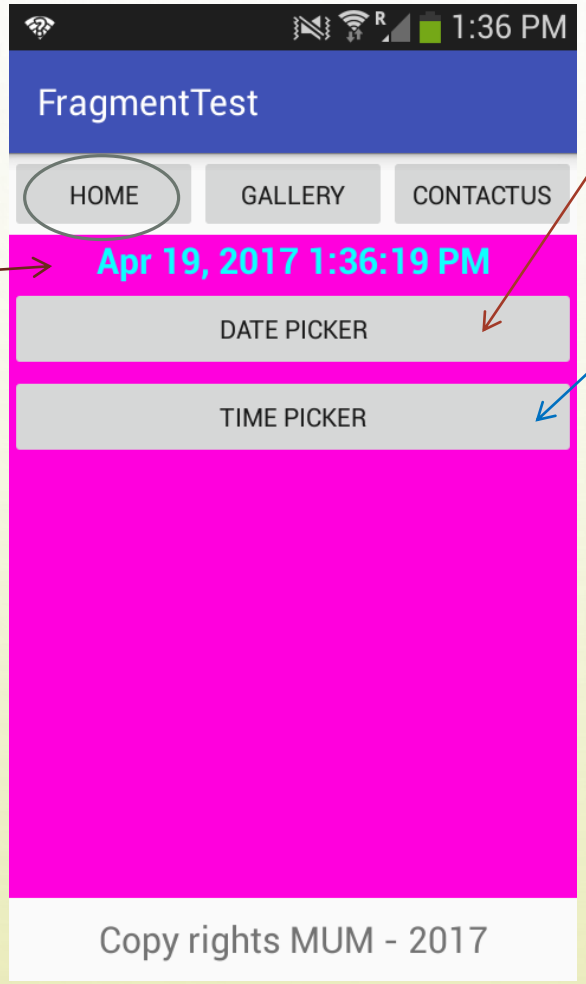
ContactusFragment.kt

```
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.

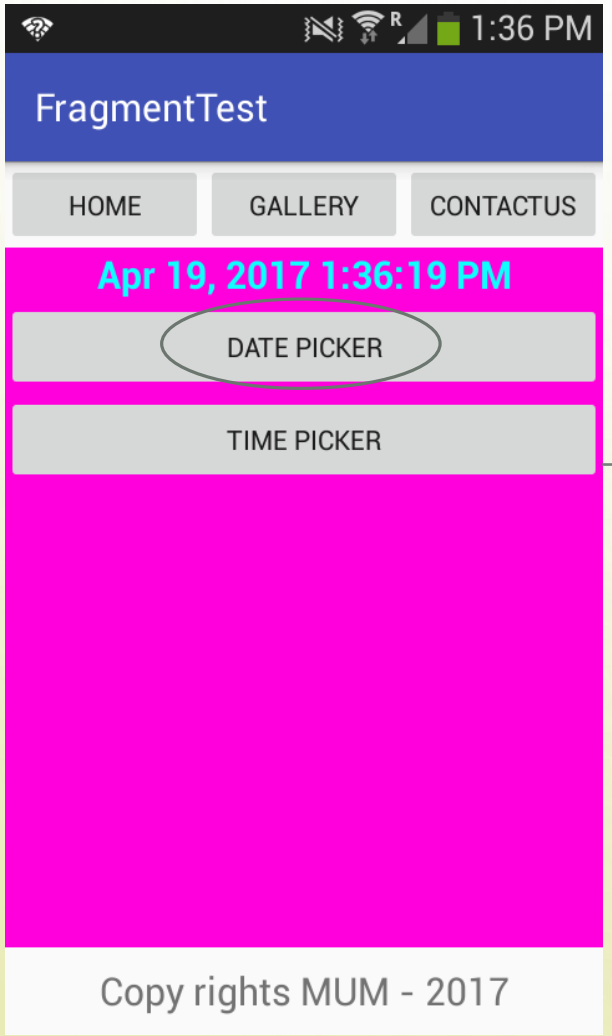
```
 <LinearLayout
xmlns:android="http://schemas.android.com/apk/res
/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#FF00DD">
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"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

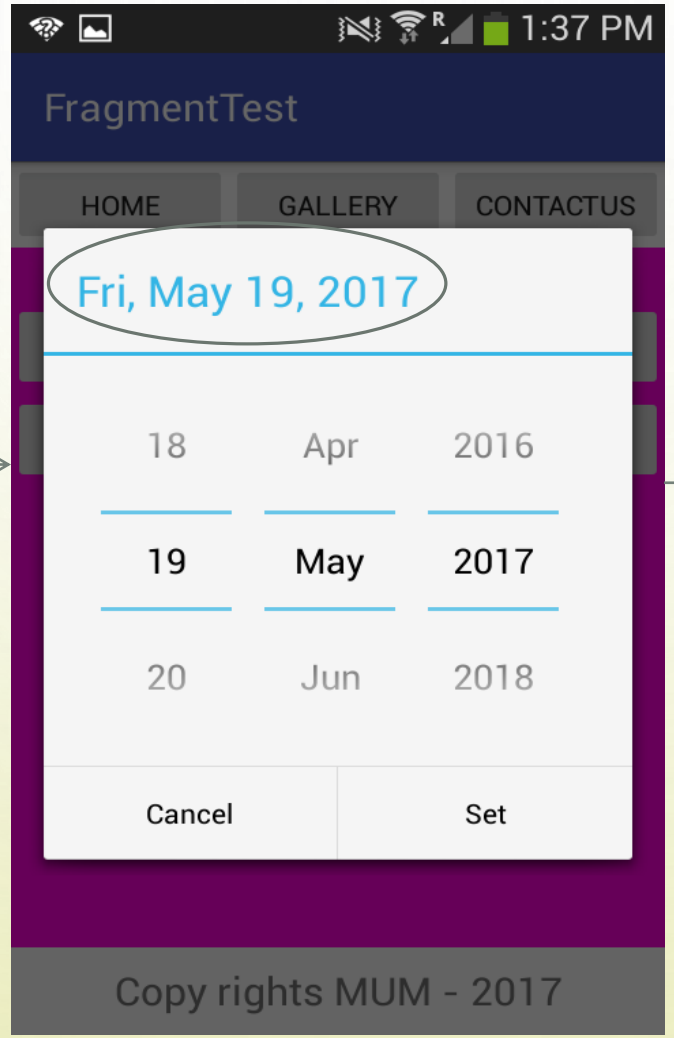


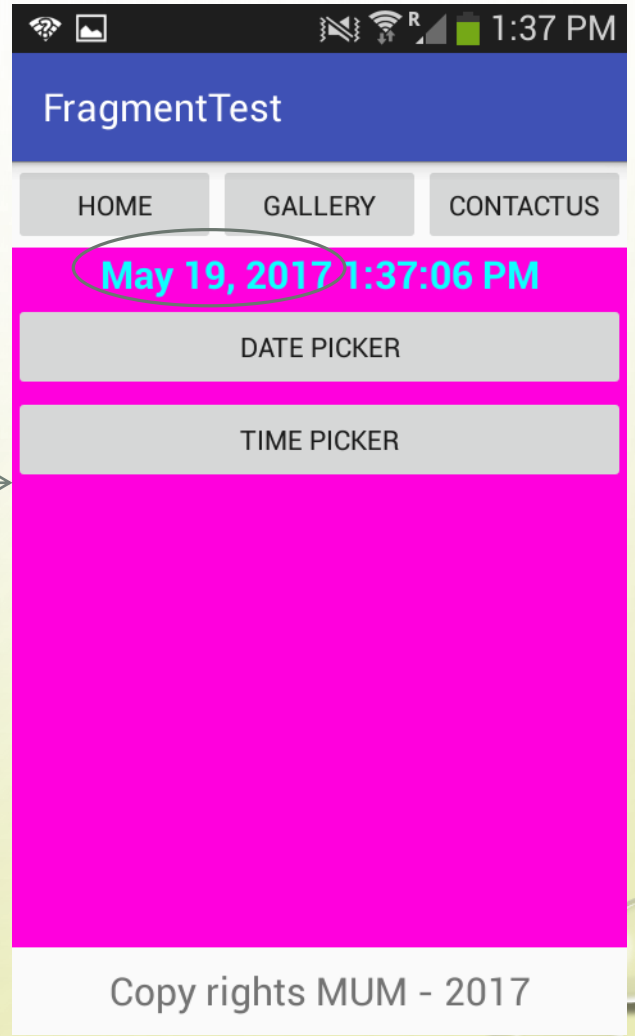

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bt1"
    android:text="DATE PICKER"/>
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/bt2"
        android:text="TIME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

HomeFragment.kt code refer from Demo Code.

Home Fragment Screen Shots

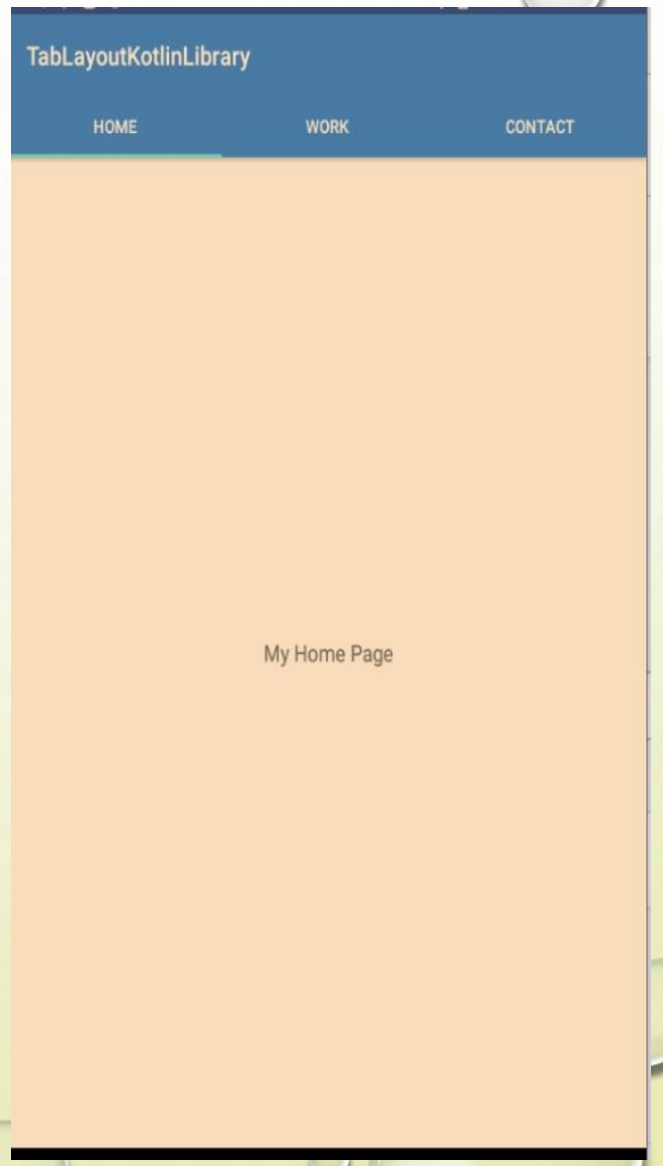






Tabs and Swipes – Hands on Example 2

- 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.



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 or fragmentstatepageradapter
5. Create an instance of the tab layout
6. Manage screen views in fragments
7. Set a listener to determine which tab is tapped

Tool Bar

- 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.
- To use Toolbar as an ActionBar, first ensure to include the below dependency library is added to your application build.gradle(Module:app) file:

```
dependencies {
 implementation 'com.google.android.material:material:1.0.0'
}
```

Refer – Step by step implementation Lesson-7-Day-2-Tab Layout Step by Step Implementation using new Kotlin support Library.pdf from DemoCode

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:layout_constraintBottom_toTopOf="@+id/viewPager"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

activity_main.xml

```
        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabBackground="@color/colorPrimary"
            app:tabGravity="fill"
            app:tabMode="fixed"
            app:tabTextColor="@android:color/white" />
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/appBarLayout">
    </androidx.viewpager.widget.ViewPager>
</androidx.constraintlayout.widget.ConstraintLayout>
```

styles.xml

- Modify the styles.xml to get the customized app bar and tool bar, use the below code

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Refer – Step by step implementation Lesson-7-Day-2-Tab Layout Step by Step Implementation using new Kotlin support Library.pdf from DemoCode

ViewPageAdapter Class

```
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentStatePagerAdapter
class ViewPagerAdapter (fm:FragmentManager) : FragmentStatePagerAdapter(fm){
    private val mFragmentList = ArrayList<Fragment>()
    private val mFragmentTitleList = ArrayList<String>()
    // return the right fragment tabbed
    override fun getItem(position: Int): Fragment {
        return mFragmentList[position]
    }
    // return the count of tabs
    override fun getCount(): Int {
        return mFragmentList.size
    }
    override fun getPageTitle(position: Int): CharSequence? {
        return mFragmentTitleList[position]
    }
    fun addFragment(fragment: Fragment, title: String) {
        mFragmentList.add(fragment)
        mFragmentTitleList.add(title)
    }
}
```

MainActivity.kt

```kotlin
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Set the toolbar into your appbar in onCreate
        setSupportActionBar(toolbar)
        // Create an object of ViewPagerAdapter by passing supportFragmentManager
        val adapter = ViewPagerAdapter(supportFragmentManager)
        // Call the addFragment to add Fragment tabs and Tab Title
        adapter.addFragment(HomeFragment(), title: "HOME")
        adapter.addFragment(WorkFragment(), title: "WORK")
        adapter.addFragment(ContactFragment(), title: "CONTACT")
        // set your adpater to the ViewPager UI on the Layout
        viewPager.adapter = adapter
        // set the ViewPager to the respective tabs
        tabs.setupWithViewPager(viewPager)
    }
}
```

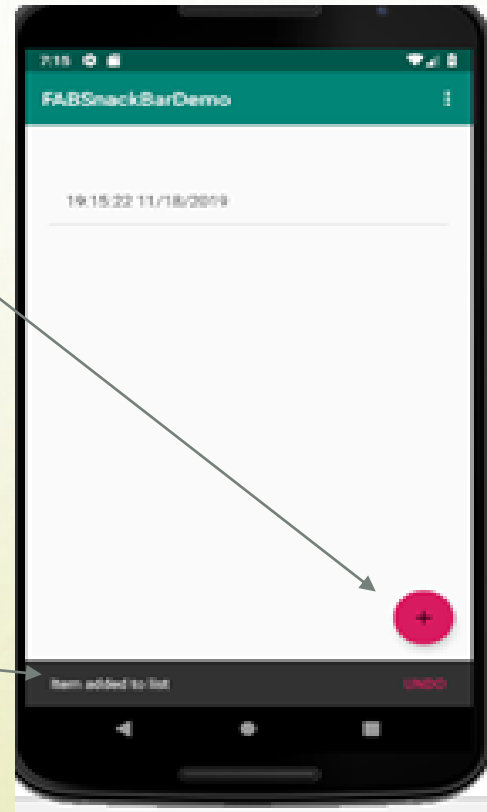
Design Support Library - Material Design

Learn more about Material Design in this source

https://www.google.com/design/spec/material-design/introduction.html

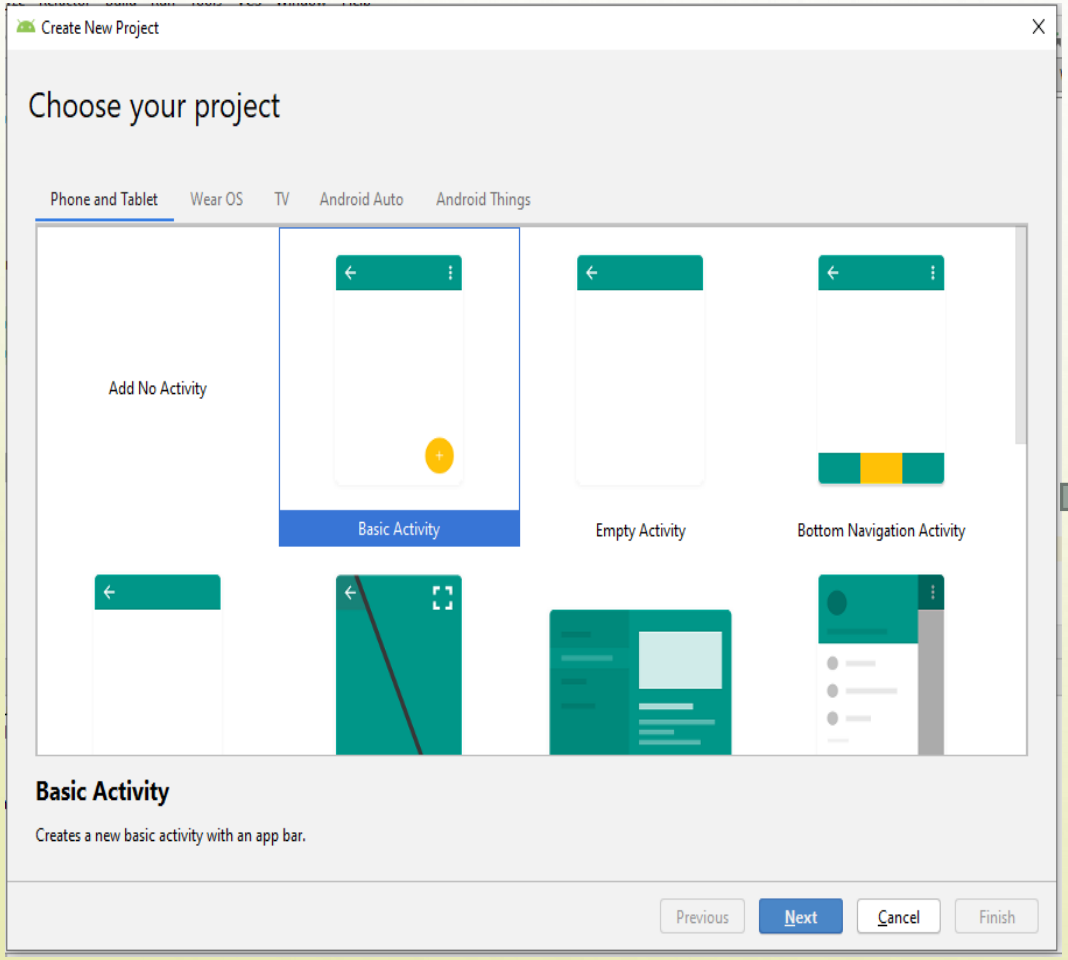
Basic Activity
The Floating Action Button (FAB) with SnackBar

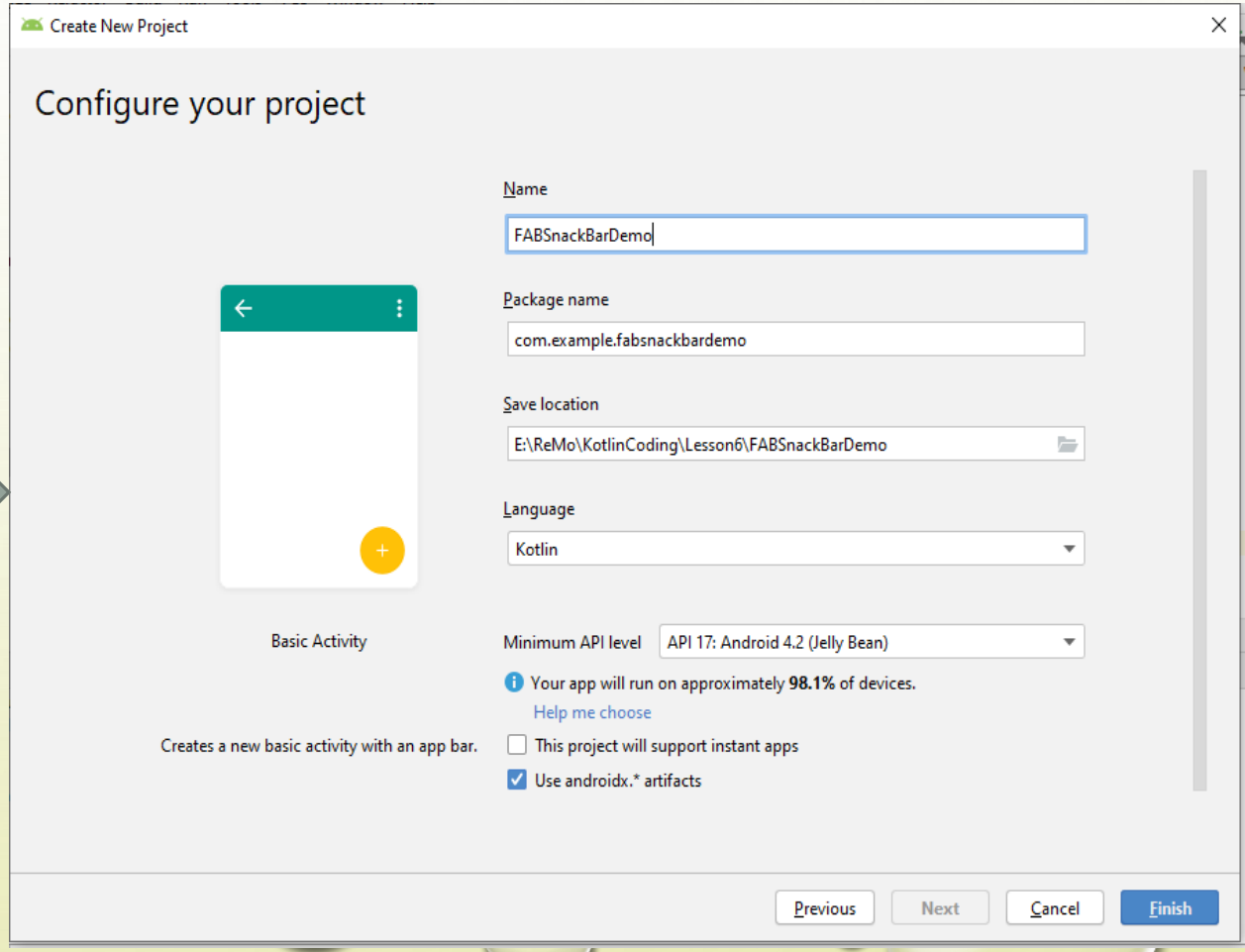
- Material design also dictates the layout and behavior of many standard user interface elements.
- The Floating Action Button(FAB) is a button which appears to float above the surface of the user interface of an app and is generally used to promote the most common action within a user interface screen.
- 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.



To Create FAB with SnackBar

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





Tab Layout using View Pager

- Open the module level *build.gradle* file (*gradle scripts -> build.gradle (module: app)*) and note that the android design support library has been added as a dependency:

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

- In android, we can create re-usable UI layouts by putting them into an extra XML and then by <include/> we can re-use that extra XML where we want. Example activity_main.xml uses the below line

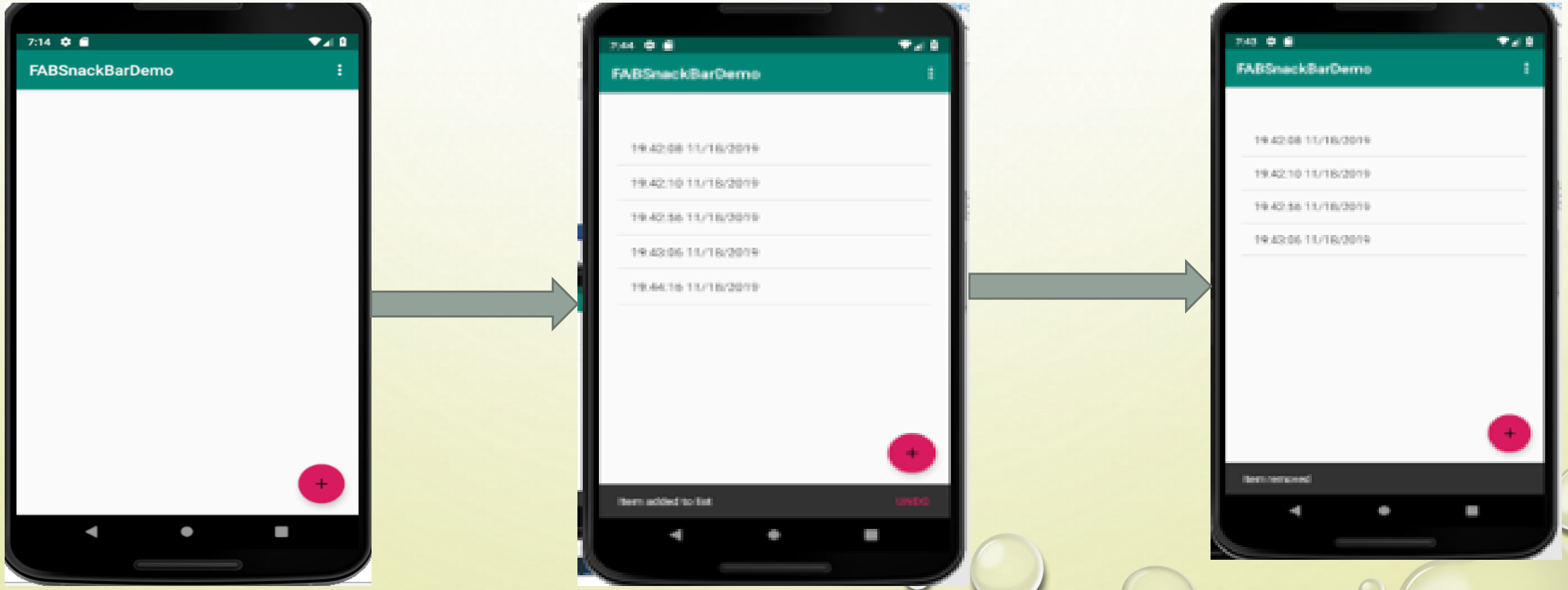
  <include layout="@layout/content_main" />

- The blank template has also configured the floating action button to display a Snackbar instance when tapped by the user.

```
fab.setOnClickListener { view ->
Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
.setAction("Action", null).show()
}
```

Example

- 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

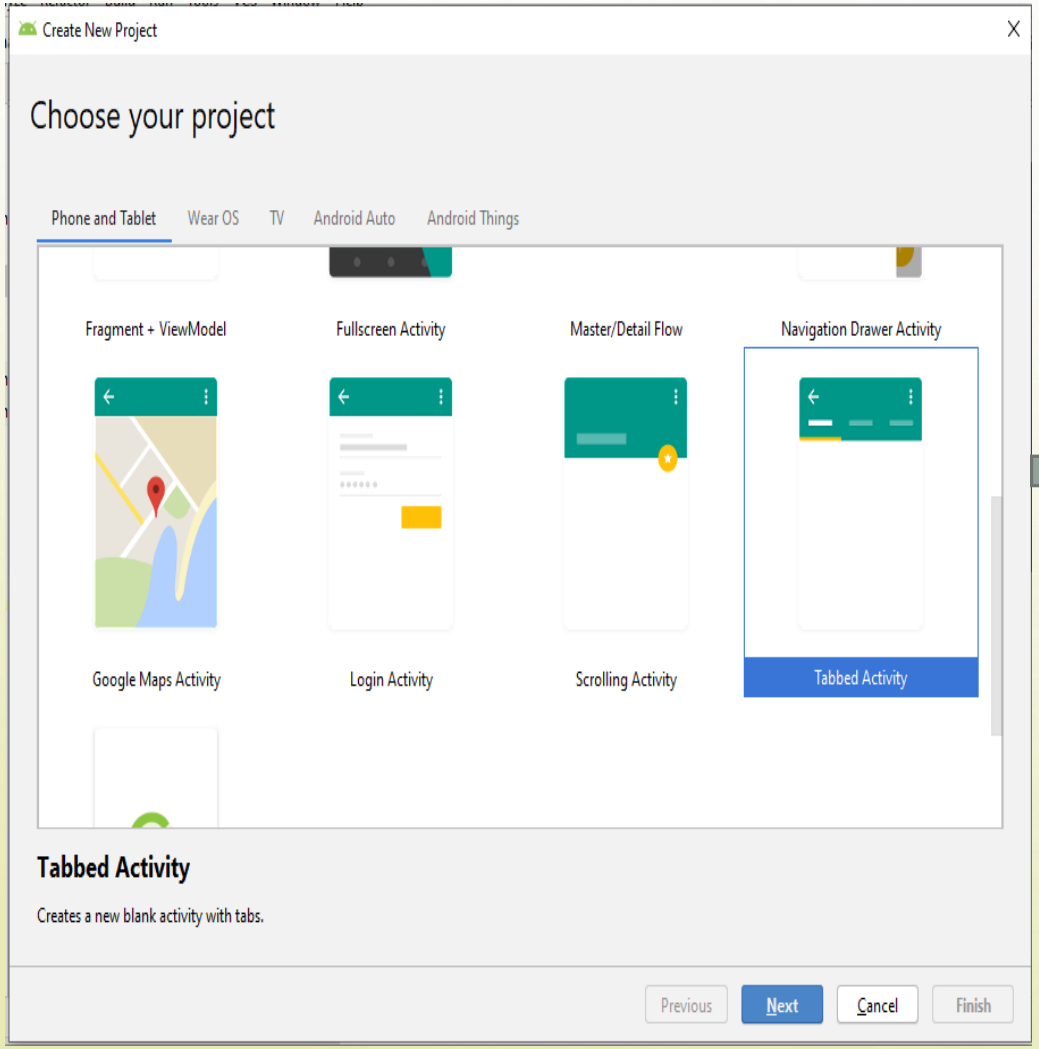


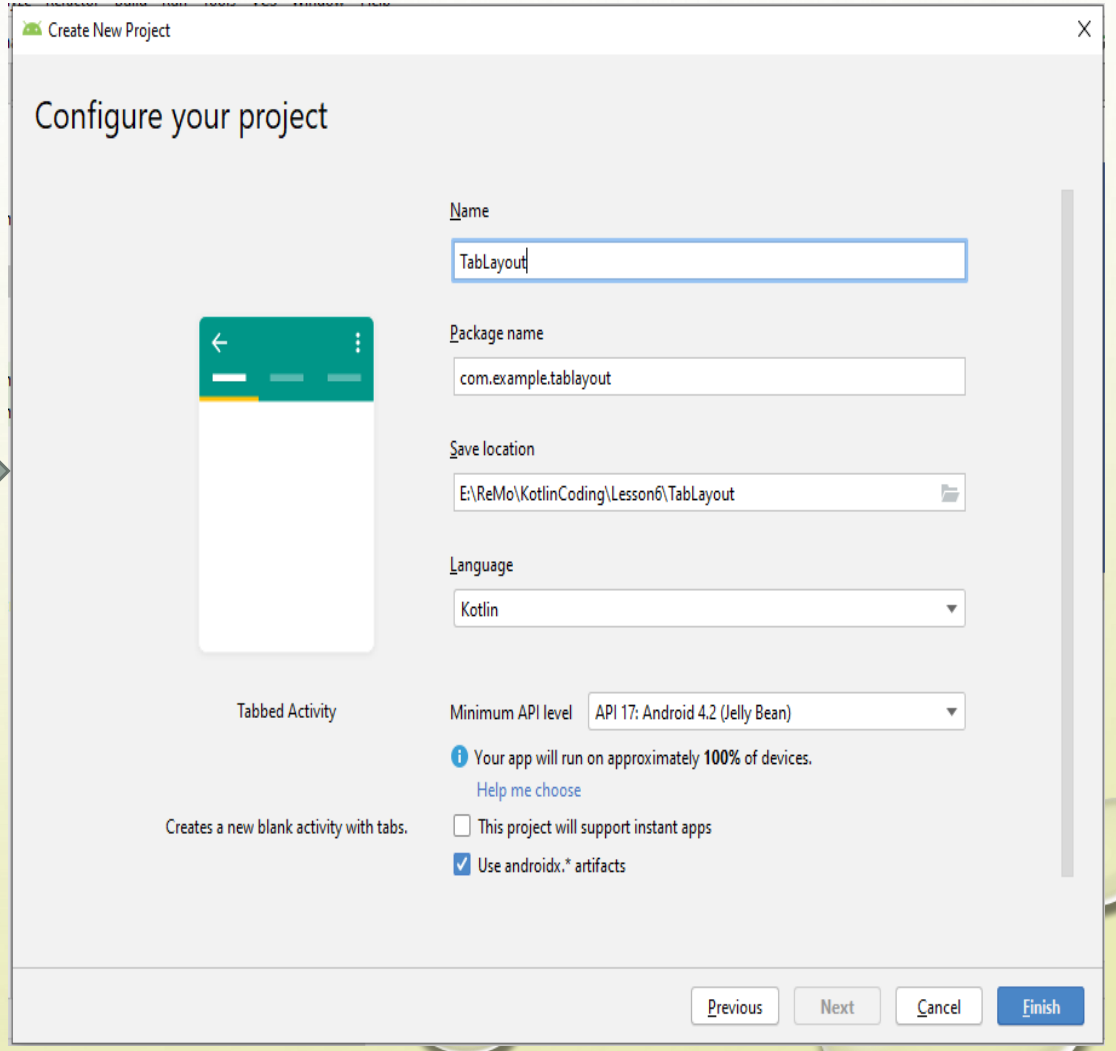
Tab Layout using View Pager

- 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.

Design support library for Tab Layout and swipe views with Initial code

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





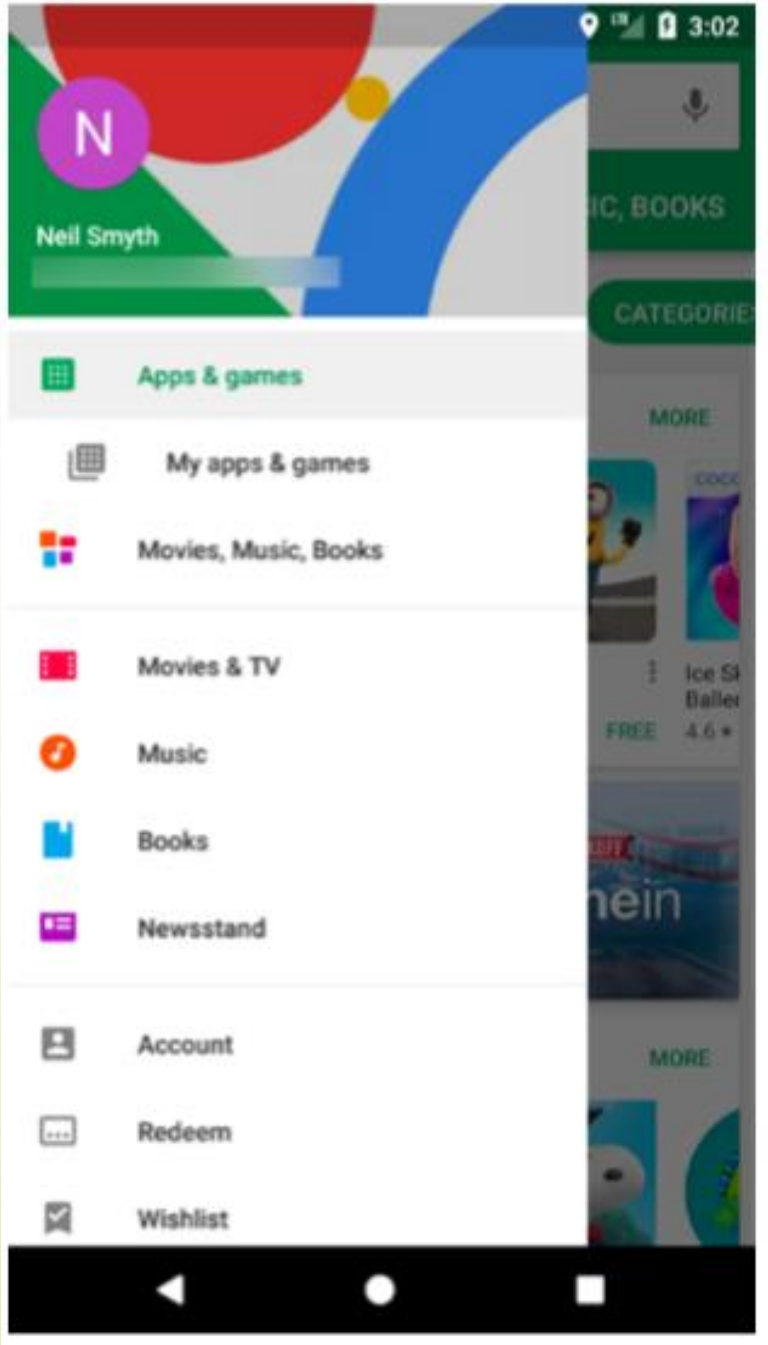
Navigation Drawer

- 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



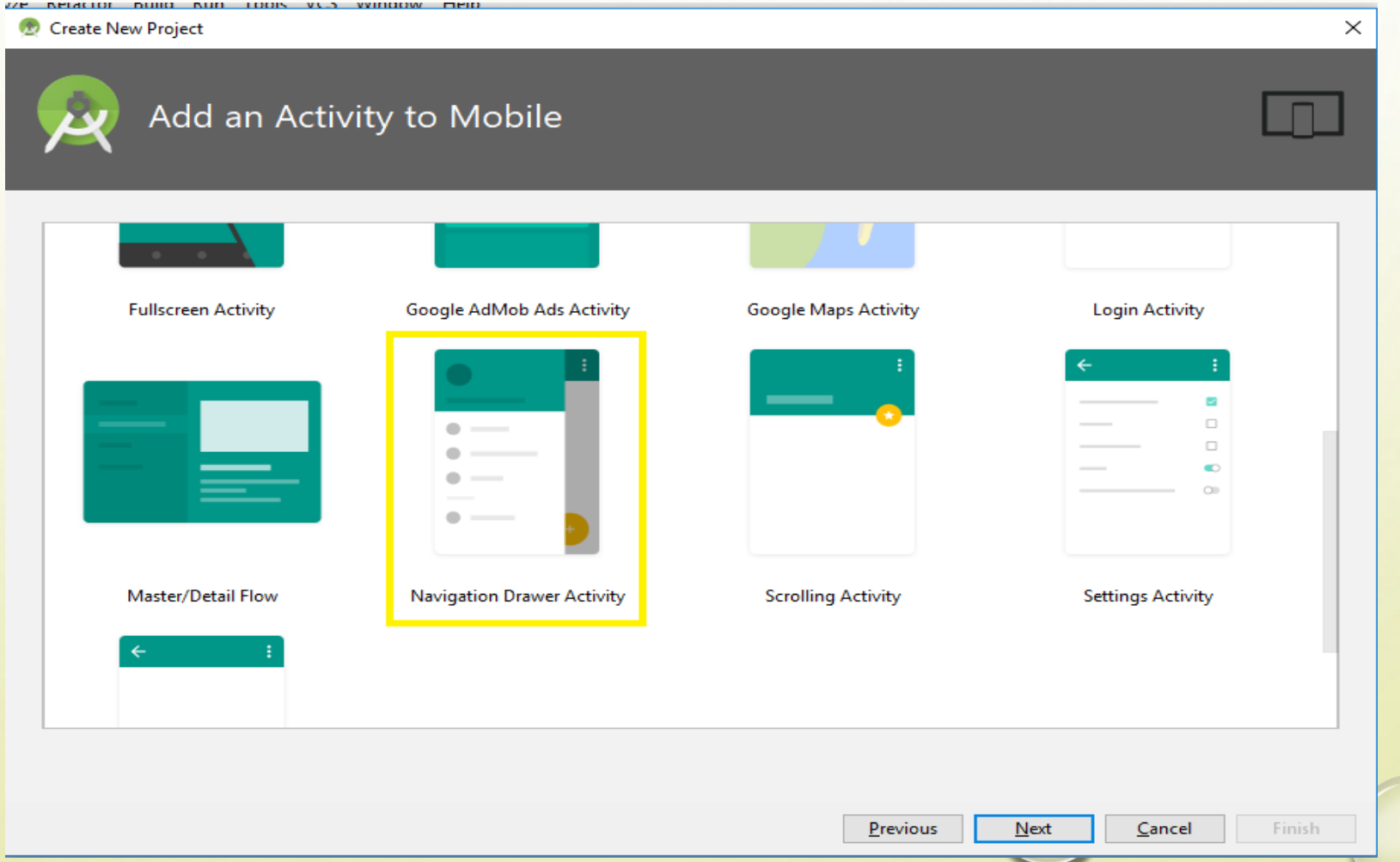
Implementing Navigation Drawer

- 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**

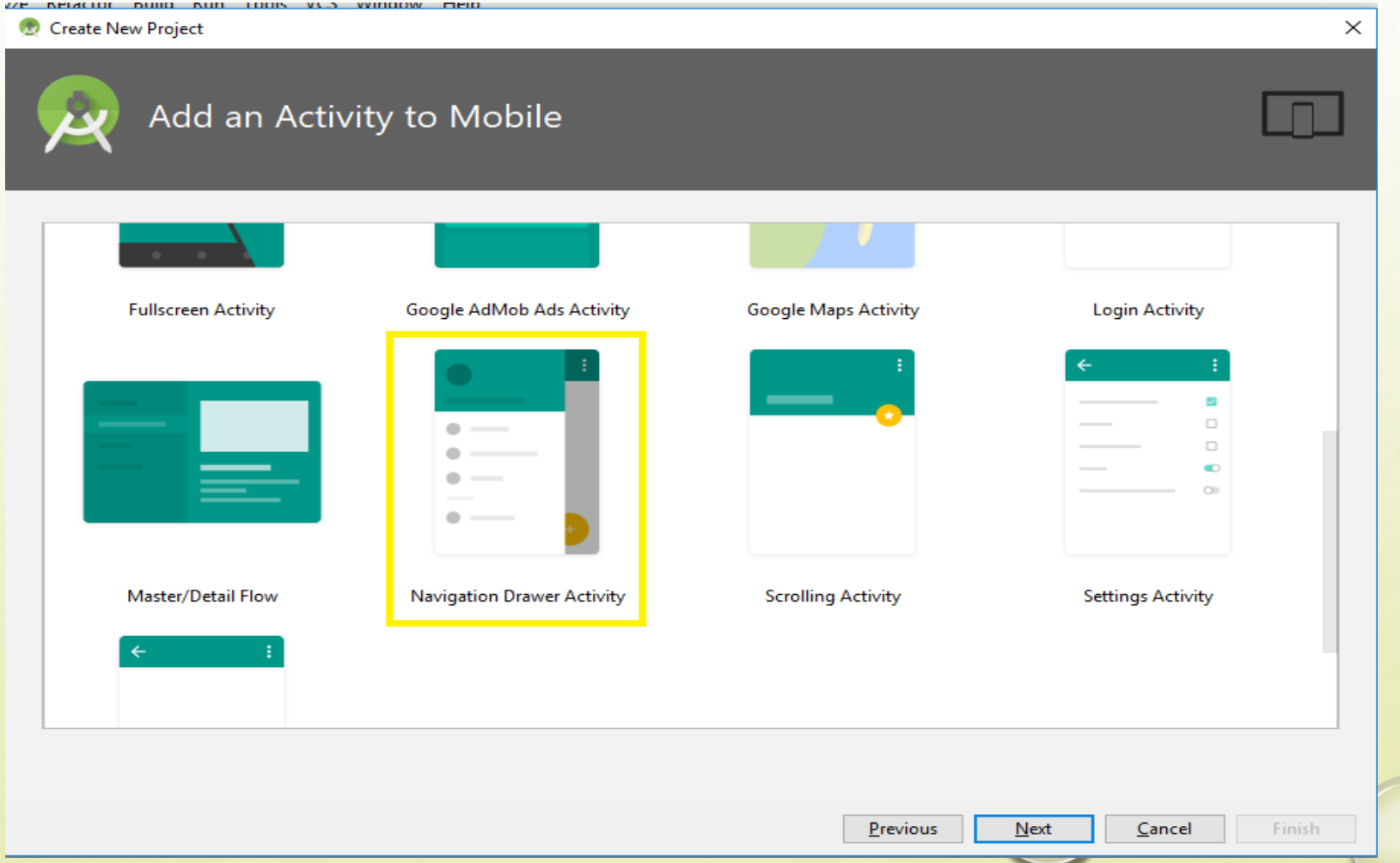
- Create a new Project, choose Navigation Drawer Activity instead of Empty activity.



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

Updates on Android Studio 3.6

- The Android studio 3.6, NavigationDrawer activity includes, Navigation Components, Navigation Graphs from Android Jetpack Library.
- I will introduce Navigation Components, Navigation Graphs in Lesson 10 with RoomDB.
- If you are not using Navigation Component, follow the next two slides to perform action for the menu selected from the NavigationDrawer.

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  }
      }
   }
```