

CS 473 - MDP

Mobile Device Programming

© 2021 Maharishi International University

All course materials are copyright protected by international copyright laws and remain the property of the Maharishi International University. The materials are accessible only for the personal use of students enrolled in this course and only for the duration of the course. Any copying and distributing are not allowed and subject to legal action.



Maharishi International
University

CS 473 - MDP

Mobile Device Programming

MS.CS Program
Department of Computer Science
Renuka Mohanraj , Ph.D.



Maharishi International
University

CS 473 – MDP

Mobile Device Programming

LESSON – 7- DAY 2

FRAGMENTS



Maharishi International
University

AGENDA

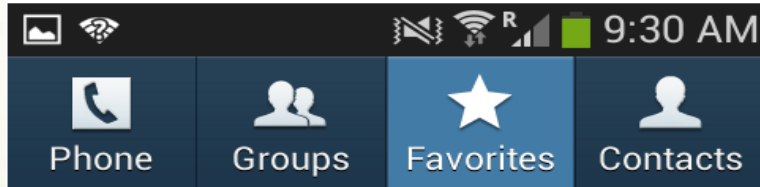
- Introduction to Fragments
- Why do we need Fragments
 - How to define Fragments in XML
 - How to Create a Fragment Class
 - How to add Fragments in Activity
 - About Weight property(How to design screen for multiple device size)
 - Tab Layout
 - BottomViewNavigation
 - Hands on Examples
 - Design support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

INTRODUCTION

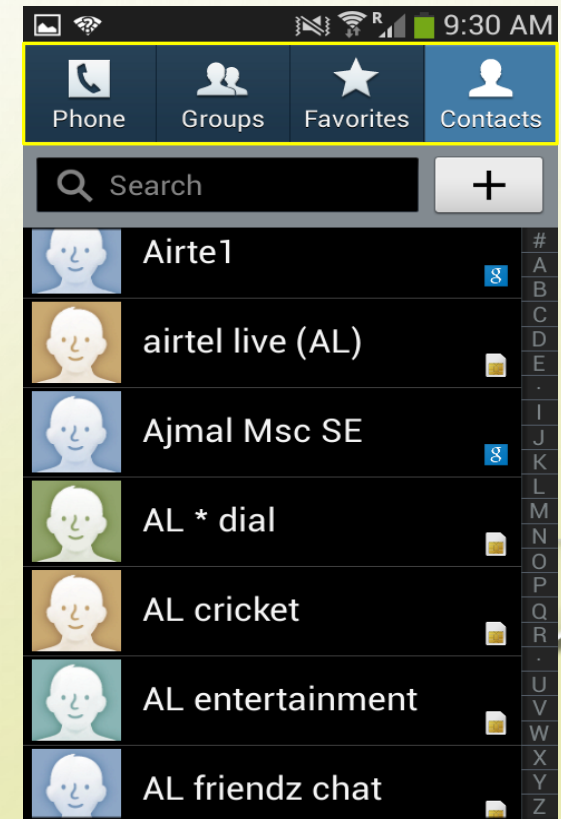
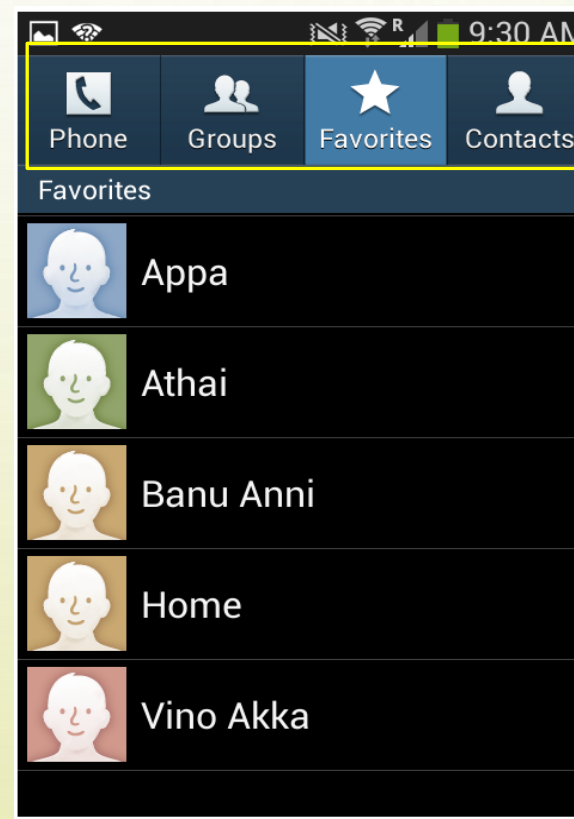
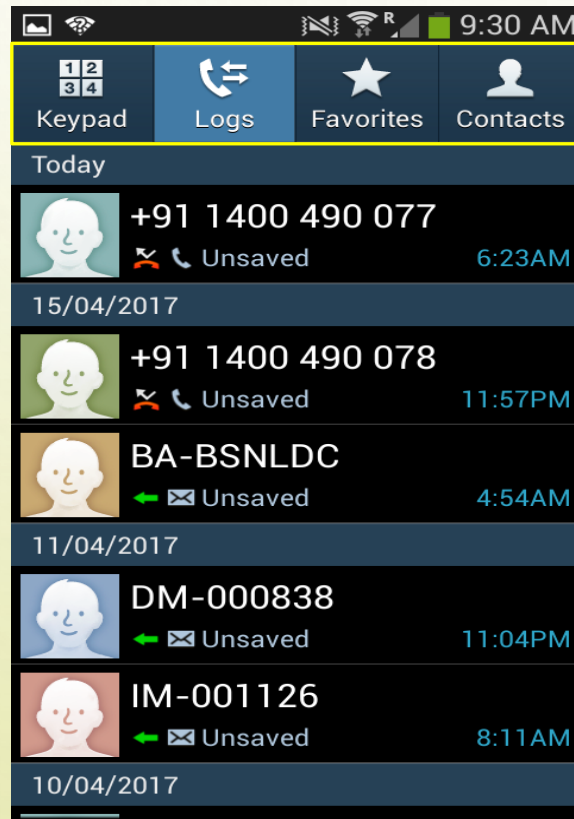
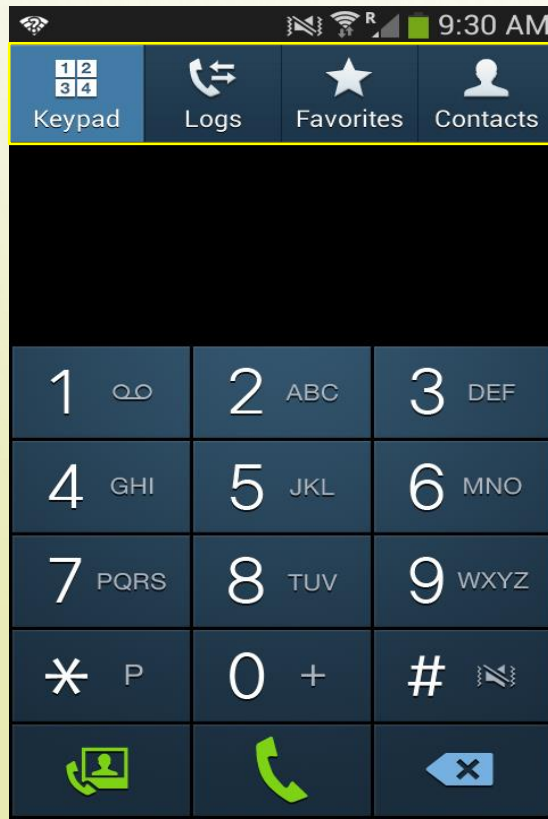
- Android 3.0 introduces a finer-grained application component called Fragment that lets you modularize the application and its user interface (into fragments).
- A Fragment represents a behavior or a portion of user interface in a FragmentActivity.
- You can combine multiple fragments in a single activity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

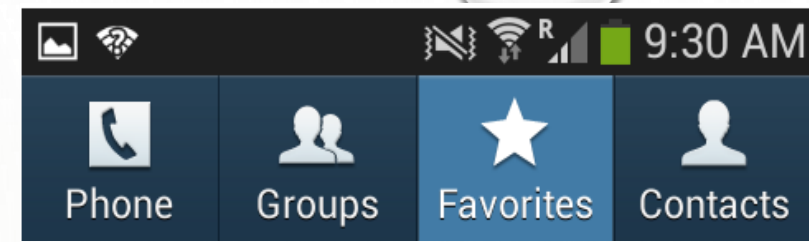
- Let's take a look on the screens shots.



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

• → FRAGMENTS

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

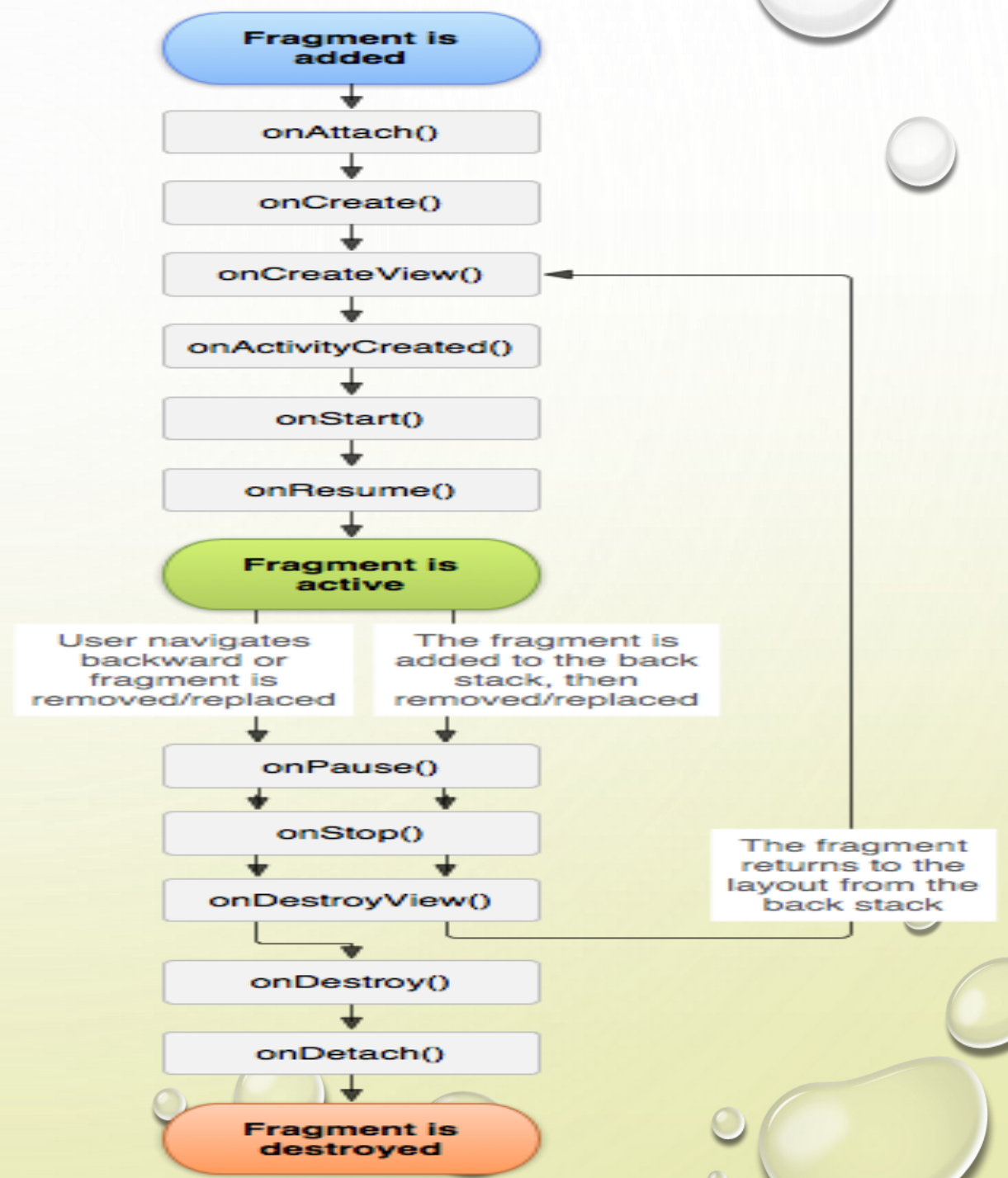
- Mini-activities, each with its own set of views
- One or more fragments can be embedded in an activity
- You can do this dynamically as a function of the device type (tablet or not) or orientation



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.
- 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. – Static Fragment

```
<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 – Dynamic Fragment

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" />
```

- Add a fragment to an activity using XML

```
<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() {
```

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

Method 1:

```
class GalleryFragment : Fragment() {  
    override fun onCreateView(  
        inflater: LayoutInflater, // Parameter 1  
        container: ViewGroup?, // Parameter 2  
        savedInstanceState: Bundle? // Parameter 3  
    ): View? {  
        // Convert an XML layout into the corresponding view  
        return inflater.inflate(R.layout.fragment_gallery, container, false) // return view  
    }  
}
```

Method 2: class GalleryFragment : Fragment(R.layout.fragment_gallery)

- 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 = fmanager.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



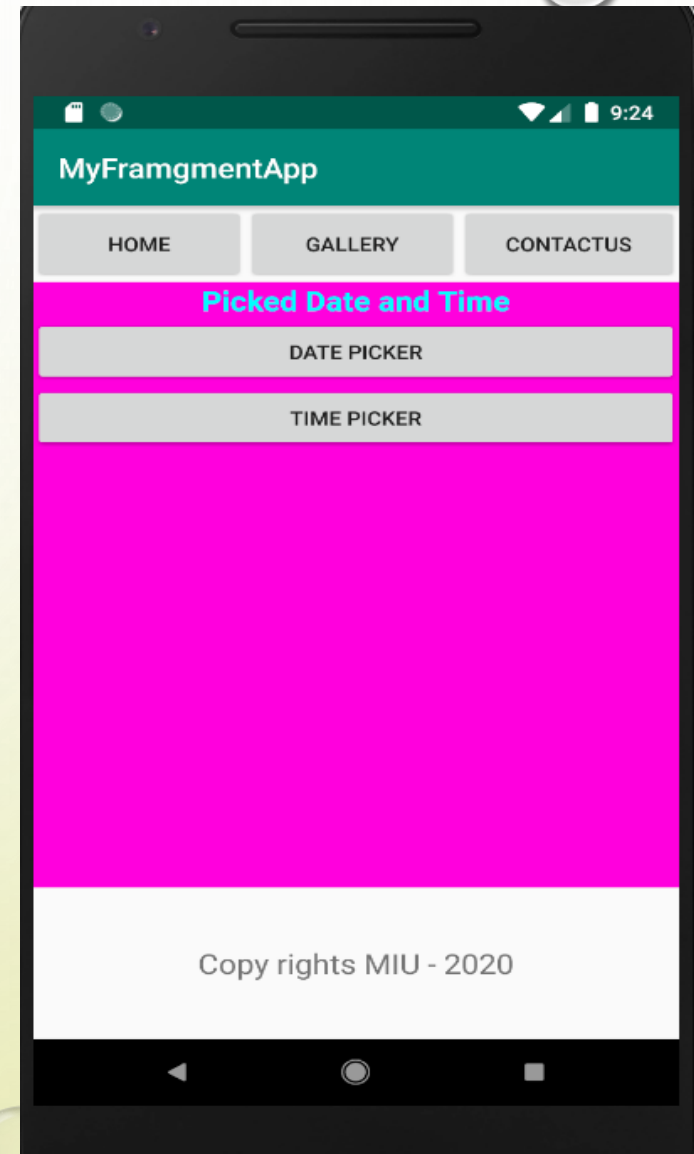
Main Point 1

- Fragments can be used to make your app more flexible and adaptable to different content and devices. You will learn how fragments have their own life cycle, can respond to events independently, and can be interchanged to create a rich user experience. *Life is found in layers – the lower the layers the more abstract, and thereby more powerful it is. The lowest level of all creation is the transcendental field.*

.

Hands on Example 1

- Create a 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 Fragement as a default.
- See : FragmentDemo folder



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="10"
    android:orientation="horizontal"
    >
```

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="33"
    android:text="Home"
    android:onClick="home"
    />
```

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="33"
    android:text="Gallery"
    android:onClick="gallery"
    />
```

Top Layout Code

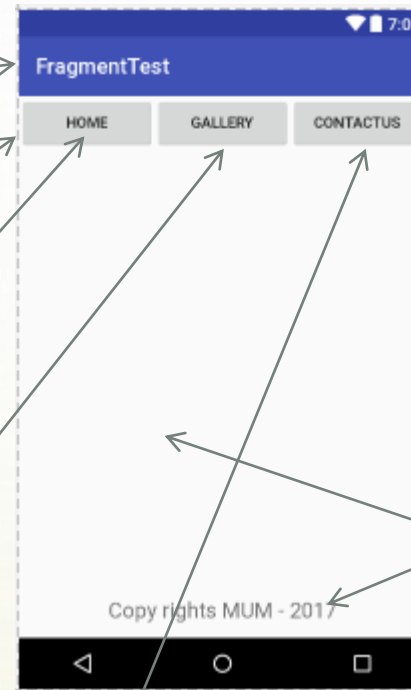
Layout Code for all Buttons

Home Button Code

Gallery Button Code

ContactUs button Code

```
<Button
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="34"
    android:text="ContactUs"
    android:onClick="contactus" />
</LinearLayout>
```



```
<TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="10"
    android:text="Copy rights MUM - 2020"
    android:textSize="20sp"
    android:gravity="center"/>
</LinearLayout>
```

Fragment Code

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="80"
    android:id="@+id/frame1">
</FrameLayout>
```

GalleryFragment.kt

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)
    }
}
```

fragment_gallery.xml

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
```

```
            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>
```



ContactFragment.kt

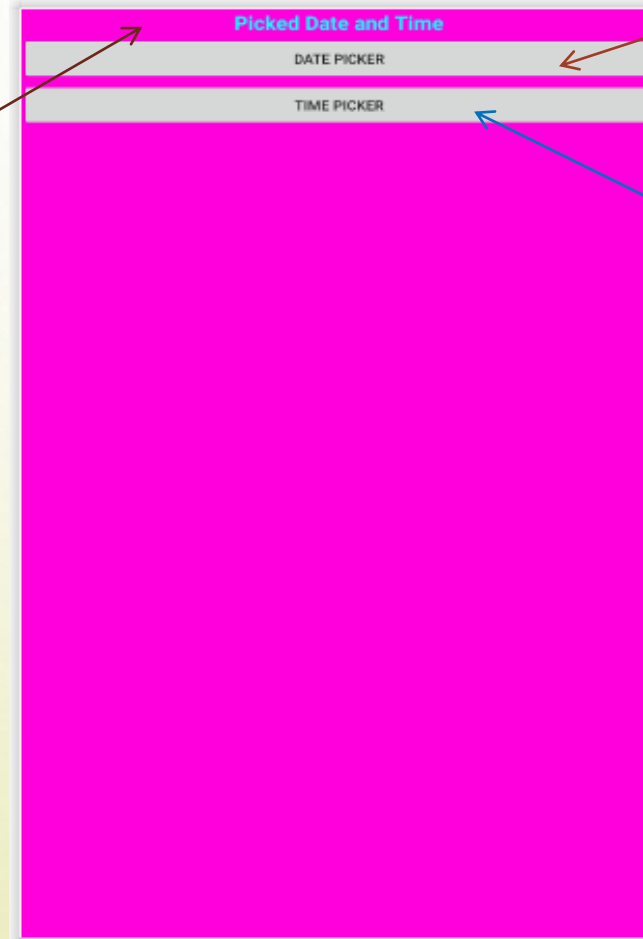
```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class ContactusFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: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


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="#0FFFFFFF"  
    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"/>
```

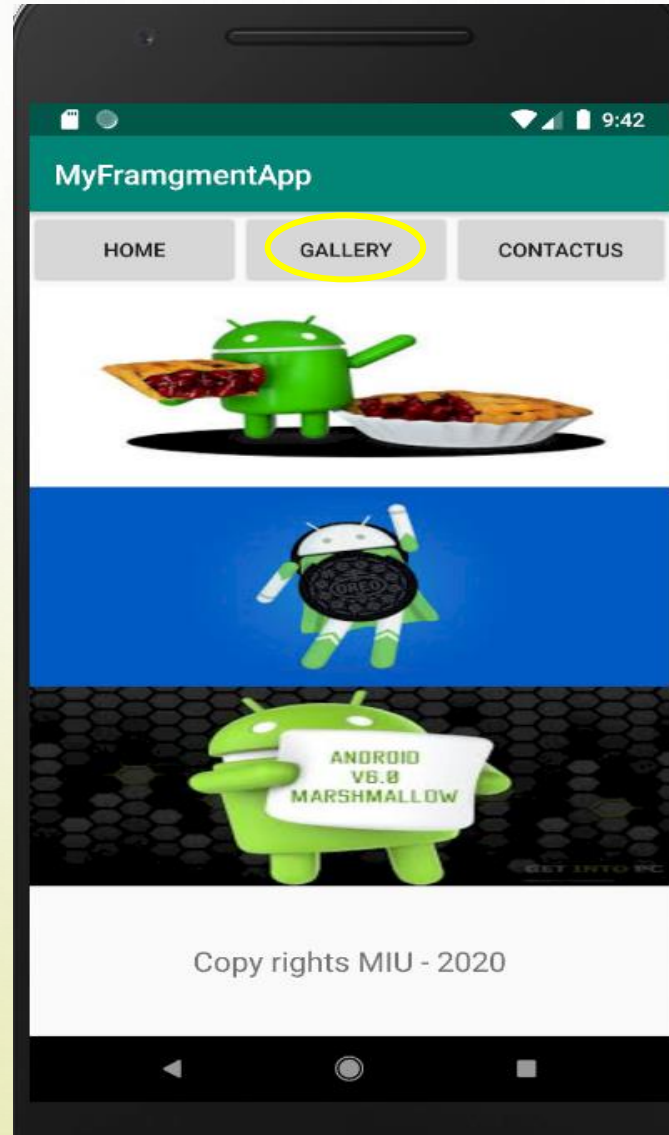
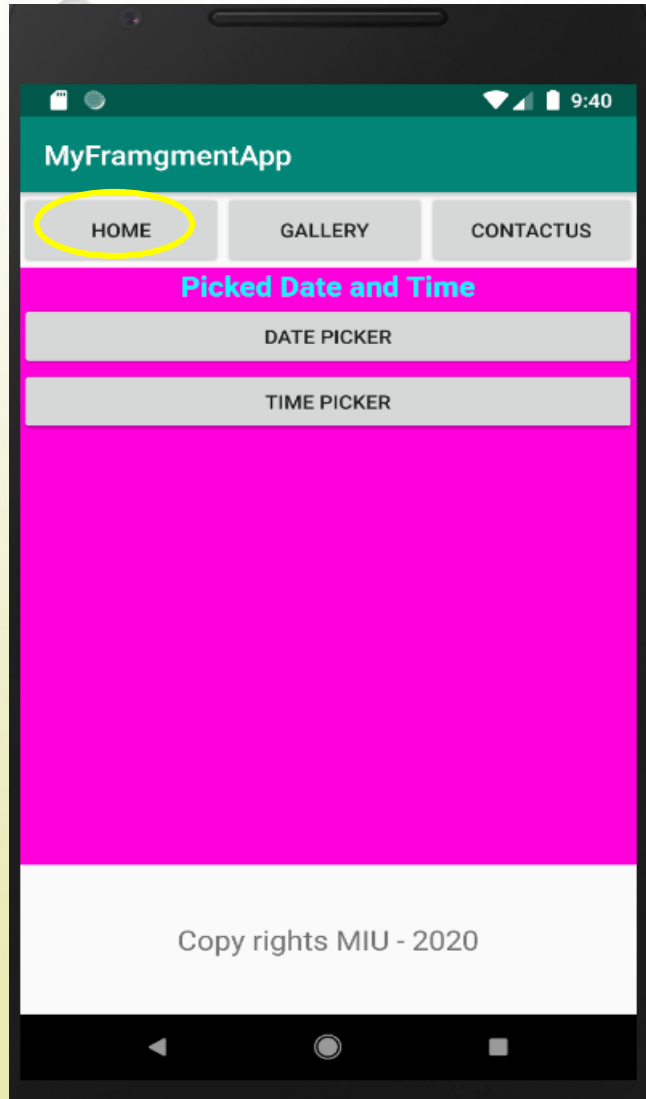
```
<Button
```

```
    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.

Outcome Screen Shots

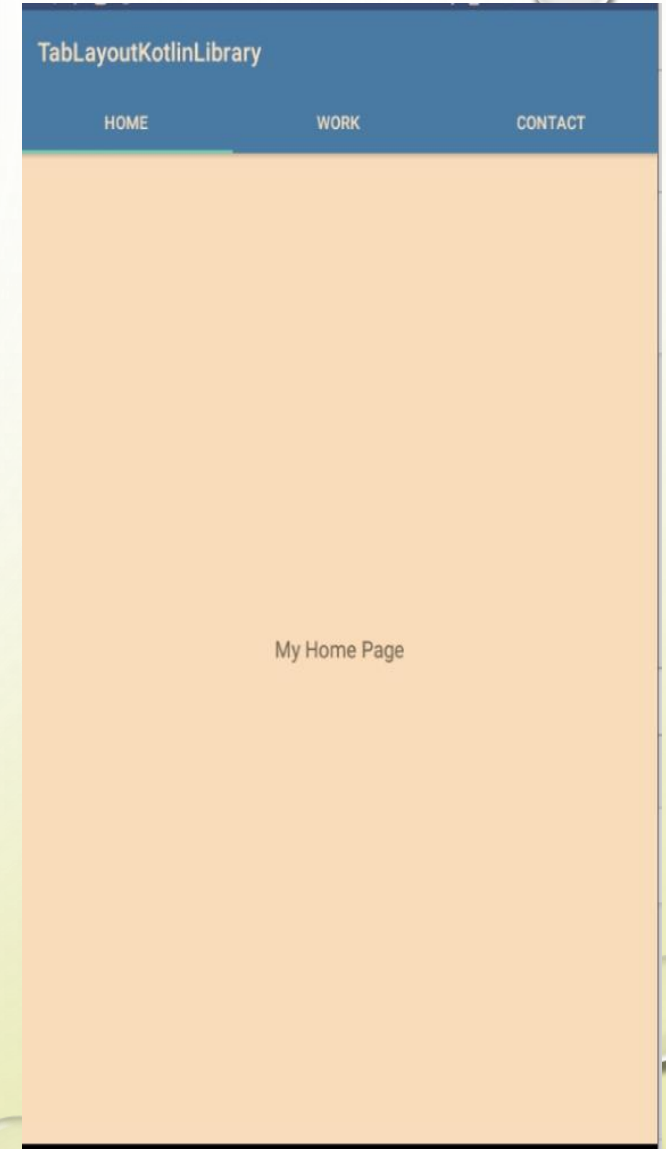


Main Point 2

- Fragments are independent entities that are contained within an android app. Fragments can be interchanged with one another to allow users to choose the content they desire. *Science of consciousness: In cosmic consciousness one can established in inner fullness, one acts to fulfill the needs of own what they desire, others and society with greatest effectiveness.*

Tabs and Swipes using ViewPager2 – 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 ViewPager2 widget, available in the Support Library.
- The ViewPager2 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 ViewPager2, add `<ViewPager2>` element to your XML layout.
- Refer : <https://material.io/develop/android/components/tabs>



Steps for implementing tabs

1. Add dependency on your build.gradle module

// Material Design Library - also for ViewPager2

implementation 'com.google.android.material:material:1.5.0-alpha02'

2. Add the below UI components in your main activity layout file

```
<com.google.android.material.tabs.TabLayout  
    android:id="@+id/tlayout"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
</com.google.android.material.tabs.TabLayout>
```

```
<androidx.viewpager2.widget.ViewPager2  
    android:id="@+id/viewpager"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
/androidx.viewpager2.widget.ViewPager2>
```

Steps for implementing tabs

3. Add a Fragments and its layouts for each Tab.

4. Add your own Adpater class to set up the Fragments to the ViewPager2

```
class MyViewAdapter(fm:FragmentManager,lc:Lifecycle) : FragmentStateAdapter(fm,lc)
```

- ViewPager2 objects have built-in swipe gestures to transition through pages
- ViewPager2 uses FragmentStateAdapter objects as a supply for new pages to display, so the FragmentStateAdapter will use the fragment class that you created earlier.

```
class MyPageAdapter(fragmentActivity:FragmentActivity) : FragmentStateAdapter(fragmentActivity)
```

- Override the below two methods inside your class

```
override fun getItemCount()
```

```
override fun createFragment(position: Int): Fragment
```


ViewPage2 Adapter Class

```
class MyPageAdapter(fragmentActivity: FragmentActivity) :
```

```
FragmentStateAdapter(fragmentActivity) {
```

```
    override fun getItemCount() = 4 // We have 4 fragments
```

```
    // Provide a new Fragment associated with the specified position.
```

```
    override fun createFragment(position: Int): Fragment {
```

```
        return when (position) {
```

```
            0 -> HomeFragment()
```

```
            1 -> WorkFragment()
```

```
            2 -> ContactFragment()
```

```
            3 -> HelpFragment()
```

```
            else -> Fragment()
```

```
        }
```

```
    }
```

```
}
```

MainActivity.kt

5. A mediator helps to link a TabLayout with a ViewPager2. It will synchronize the ViewPager2's position with the selected tab.

Inside your MainActivity add the below codes

```
val myPagerAdapter = MyPagerAdapter(this)
```

```
// Set the Adapter to your Viewpager UI
```

```
vpager.adapter = myPagerAdapter
```

```
// Will align the space according to the Screen size to equally spread
```

```
tlayout.tabGravity = TabLayout.GRAVITY_FILL
```

MainActivity.kt

/* Setting up Tab Layout with the ViewPager2 is handled by the TabLayoutMediator
* by passing your tablayout id and viewpager id*/

```
TabLayoutMediator(tabLayout, viewPager) { tab, position ->
    when(position) {
        0 -> {
            tab.text = "Home"
            tab.setIcon(R.drawable.home)
        }
        1 -> {
            tab.text = "Work"
            tab.setIcon(R.drawable.work)
        }
        2 -> {
            tab.text = "Contact"
            tab.setIcon(R.drawable.contact)
        }
        3 -> {
            tab.text = "Help"
            tab.setIcon(R.drawable.help)
        }
    }
}.attach()
```

Refer Demo: TabLayoutOctober2022

Step by Step implementation file from Lectures: Lesson-7-Day-2-Tab Layout Step by Step Implementation-Ver2

Bottom Navigation View Implementation

1. Create a menu resource to show the items on the Bottom Navigation Bar.
2. Add Fragment UI and BottomNavigation UI on the main_activity.xml. Set the menu resource with BottomNavigationView as highlighted below.

```
<com.google.android.material.bottomnavigation.BottomNavigationView
```

```
    android:id="@+id/bottomNavigationView"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="75dp"
```

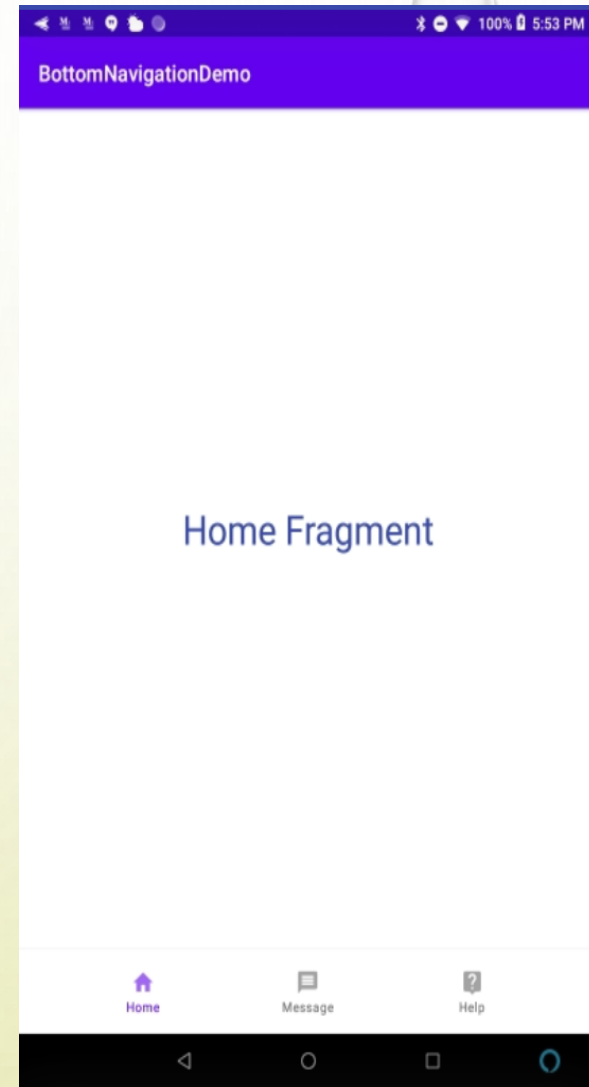
```
    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent"
```

```
    app:layout_constraintHorizontal_bias="0.5"
```

```
    app:layout_constraintStart_toStartOf="parent"
```

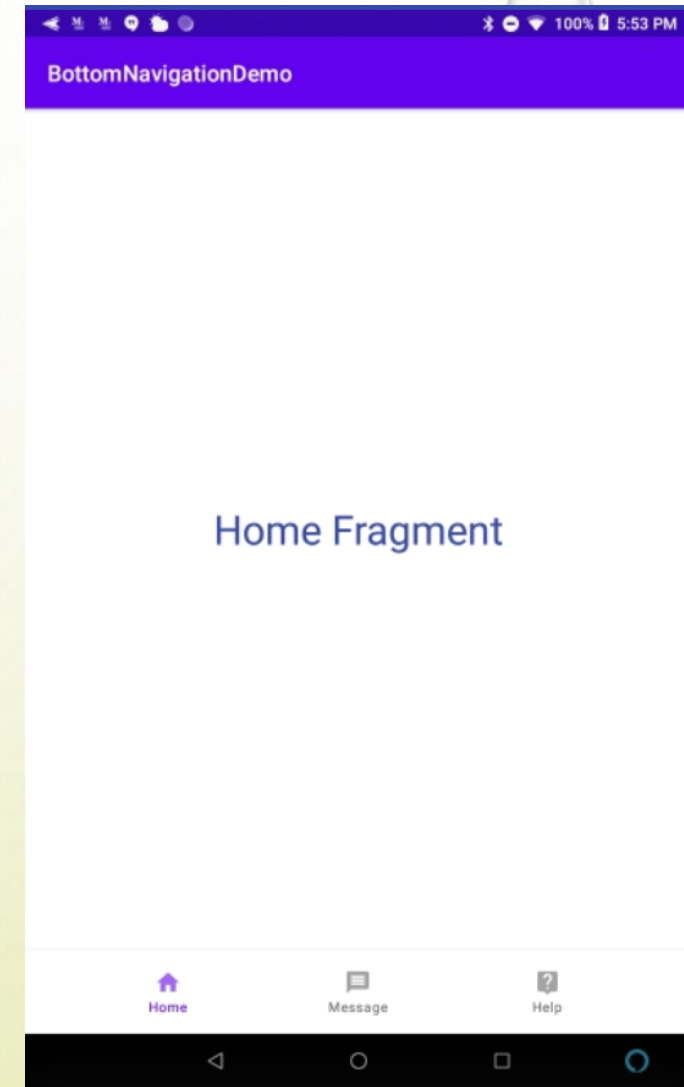
```
    app:menu="@menu/mymenu">
```



Bottom Navigation View Implementation

3. Create three Fragments with its layout for Home, Message and Help.
4. Do the Implementation in your MainActivity to show the Fragments of each click from the BottomNavigationView as mentioned below.

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val home = HomeFragment();  
        val msg = MessageFragment();  
        val help = HelpFragment();  
        setCurrentFragment(home)  
    }  
}
```



Bottom Navigation View Implementation

```
bottomNavigationView.setOnNavigationItemSelectedListener {
```

```
    when(it.itemId){
```

```
        R.id.menuhome->setCurrentFragment(home)
```

```
        R.id.menumsg->setCurrentFragment(msg)
```

```
        R.id.menuhelp->setCurrentFragment(help)
```

```
    }
```

```
    true // return true
```

```
}}
```

```
private fun setCurrentFragment(fragment: Fragment){
```

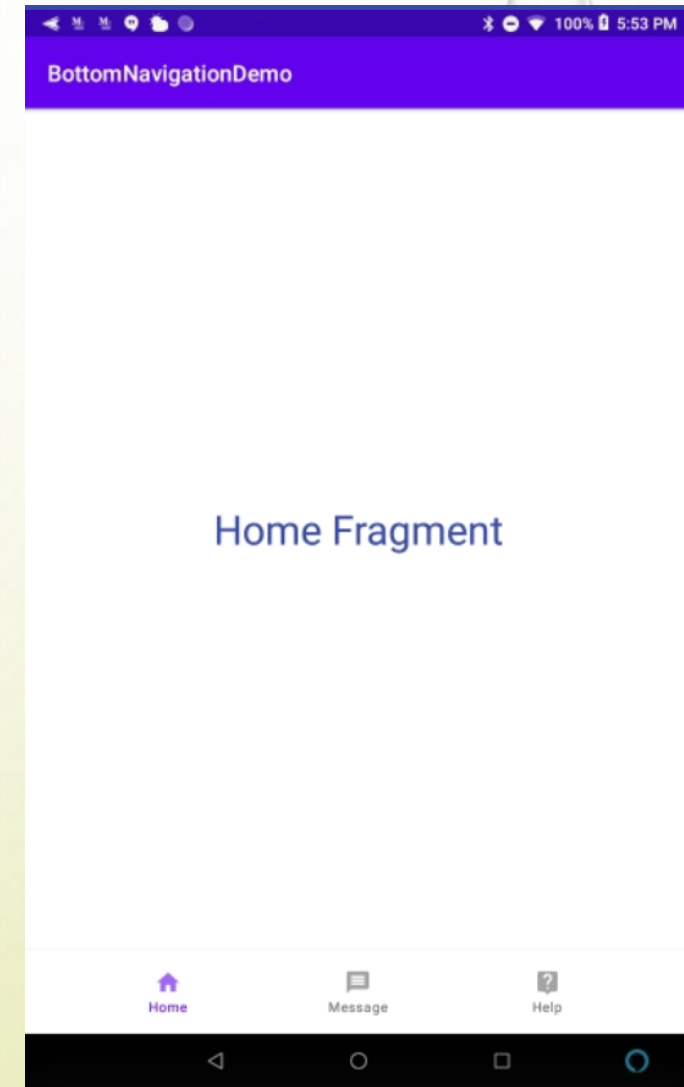
```
    supportFragmentManager.beginTransaction().apply {
```

```
        replace(R.id.fragment,fragment)
```

```
        commit()
```

```
    } }
```

```
}
```



Menu Resource

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
  <item android:title="Home"
```

```
    android:id="@+id/menuhome"
```

```
    android:icon="@drawable/ic_home"/>
```

```
  <item android:title="Message"
```

```
    android:id="@+id/menumsg"
```

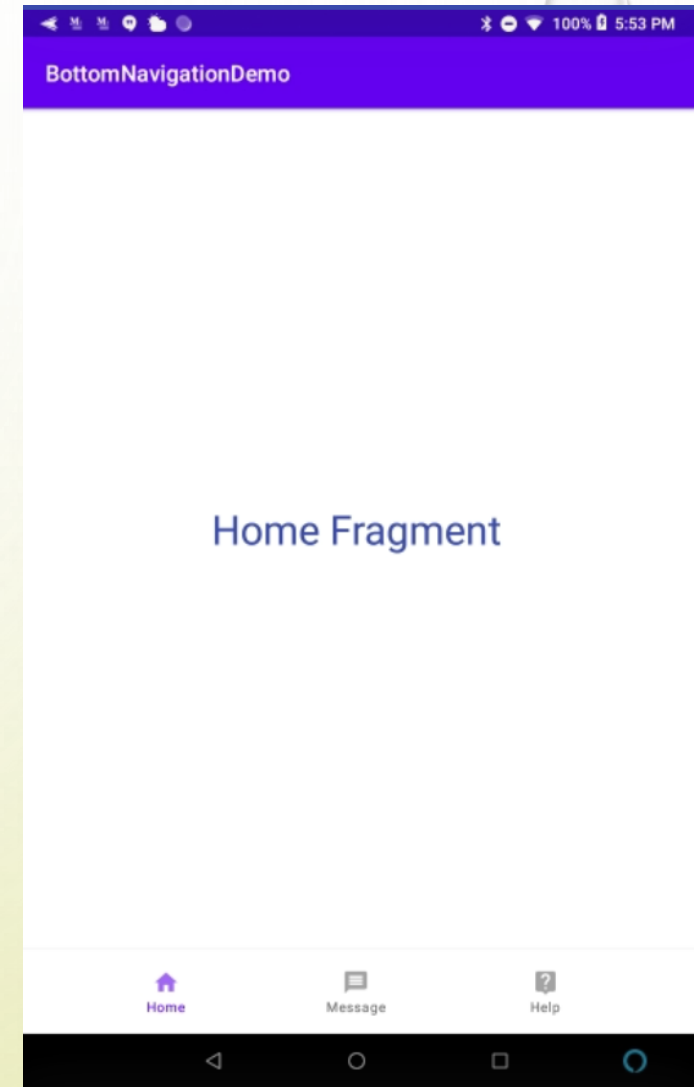
```
    android:icon="@drawable/ic_message"/>
```

```
  <item android:title="Help"
```

```
    android:id="@+id/menuhelp"
```

```
    android:icon="@drawable/ic_help"/>
```

```
</menu>
```



Refer : BottomNavigationDemo

View Binding

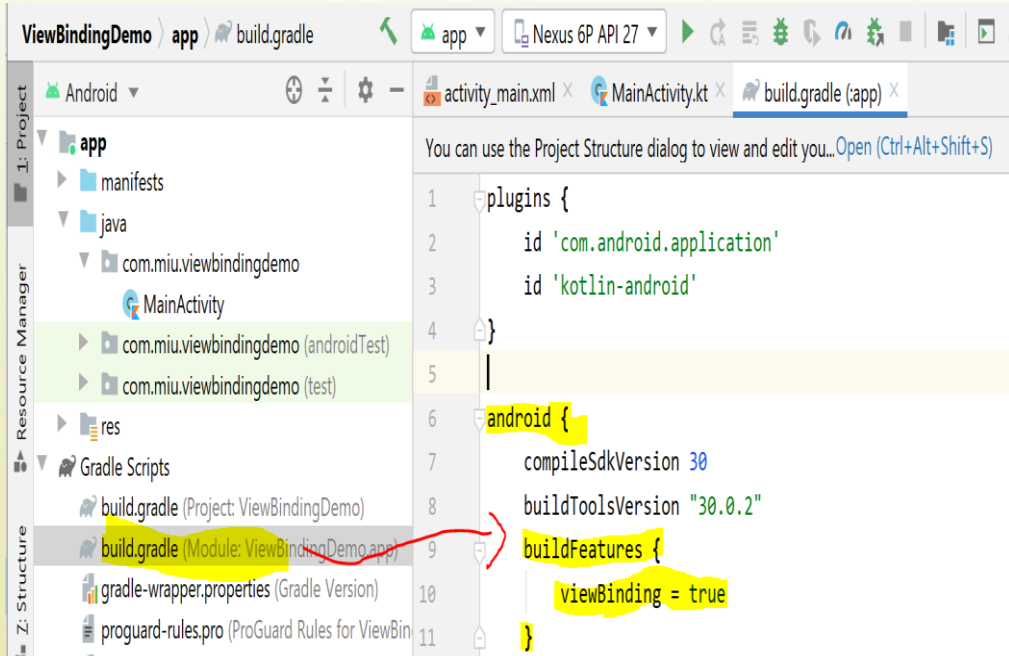
- View binding is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled, it generates a binding class for each XML layout file.
- An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.
- The name of the binding class is generated by converting the name of the XML file and adding the word "Binding" to the end.
- **Advantages**
 - Faster
 - Replaces findViewById
 - Safe & Concise
 - Less Boiler Plate

View Binding

The below plugins are added build.gradle module to get the XML ids into Kotlin code. Can replace it with View Binding

id 'kotlin-android-extensions'

You need to enable the ViewBinding is true in build.gradle module as per the screenshot below. Android creates a Binding Class for your Layout. Bind the View in your Activity.



```
// activity_main.xml  
<ConstraintLayout>  
    <TextView android:id="@+id/greeting">  
</ConstraintLayout>
```

```
// generates AndroidMainBinding  
class ActivityMainBinding {  
    val greeting: TextView  
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {  
    ...  
    val binding = ActivityMainBinding.inflate(layoutInflater)  
    setContentView(binding.root)
```

Hands on Example

Problem requirement: activity_main.xml designed with TextView and HomeFragment.

Applied ViewBinding to access the ids.

```
class MainActivity : AppCompatActivity() {  
    /* Get the instance of Binding Object,  
    ActivityMainBinding is autogenerated ViewBinding class for the activity_main.xml*/  
    private lateinit var binding: ActivityMainBinding  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // Initialize the binding object with the given below code  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        // Replace this line as mentioned below --> setContentView(R.layout.activity_main)  
        setContentView(binding.root) // root is your root layout specified in activity_main.xml  
        binding.tv.text = "Sample View Binding Demo"  
    }  
}
```



Hands on Example

```
class HomeFragment : Fragment() {  
    /* Get the instance of Binding Object,  
    FragmentHomeBinding is autogenerated ViewBinding class for the fragment_home.xml */  
    private lateinit var binding: FragmentHomeBinding  
  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View? {  
        // Inflate the layout for this fragment  
        var view = inflater.inflate(R.layout.fragment_home, container, false)  
  
        // Initialize the binding object with the given below code, layout already inflated, so call bind()  
        binding = FragmentHomeBinding.bind(view)  
  
        binding.fragtv.text = "Home Fragment"  
  
        return view  
    }  
}
```

Refer : ViewBindingDemo





Design Support Library - Material Design

Learn more about Material Design in this source

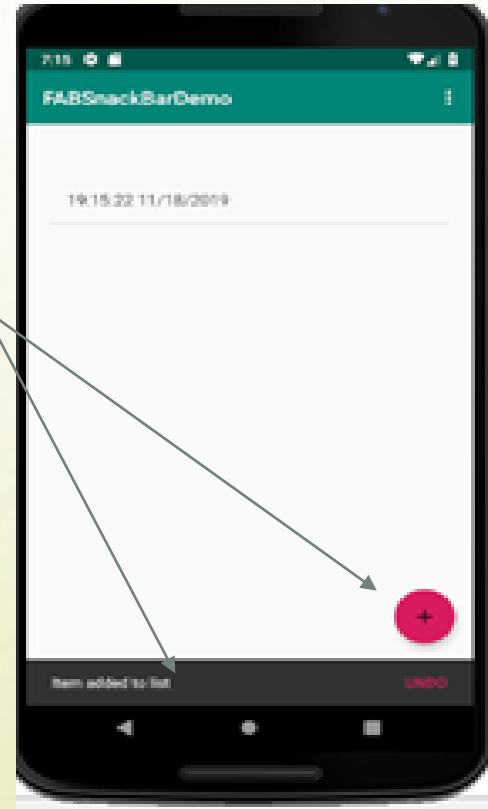
<https://material.io/develop/android>



Basic Activity

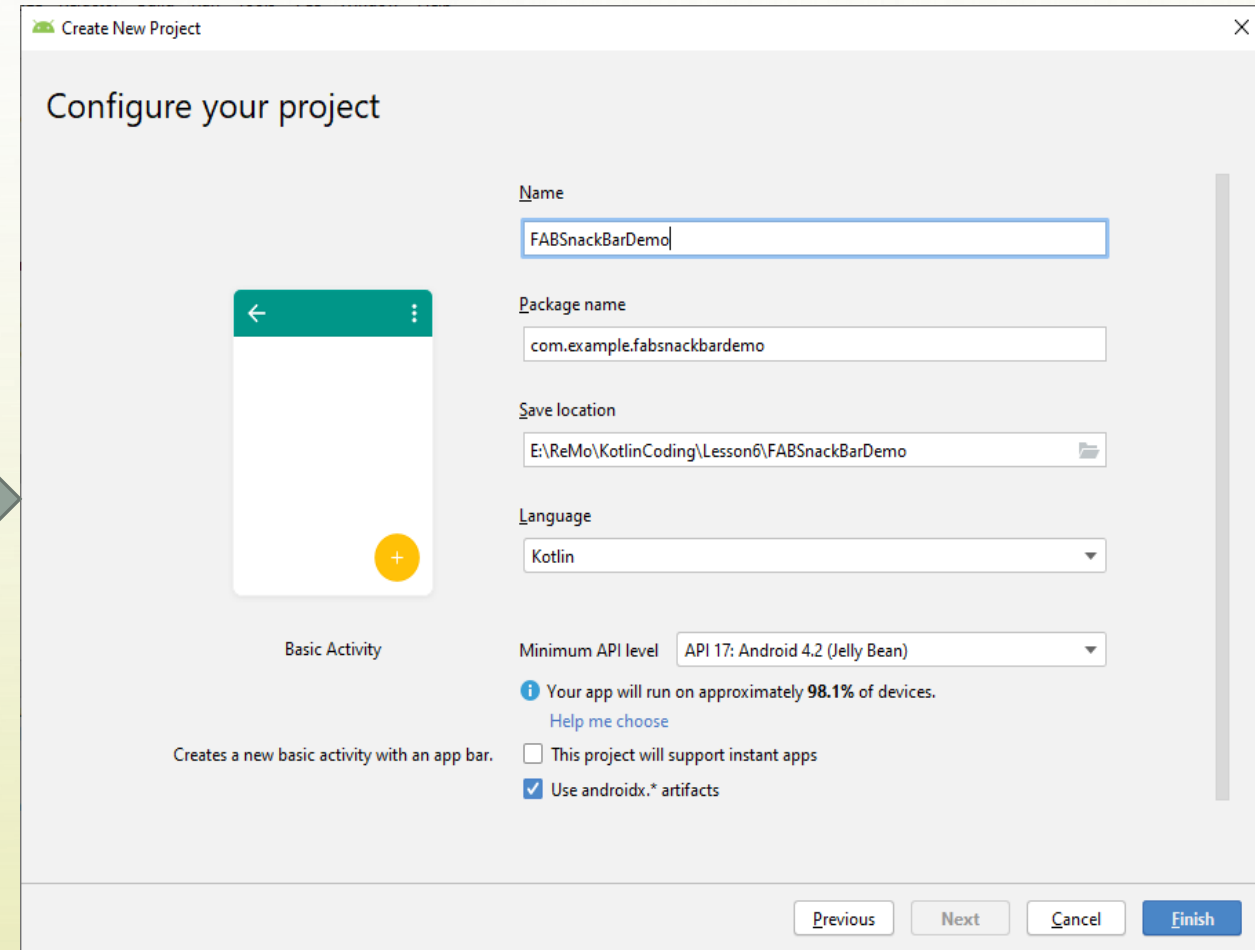
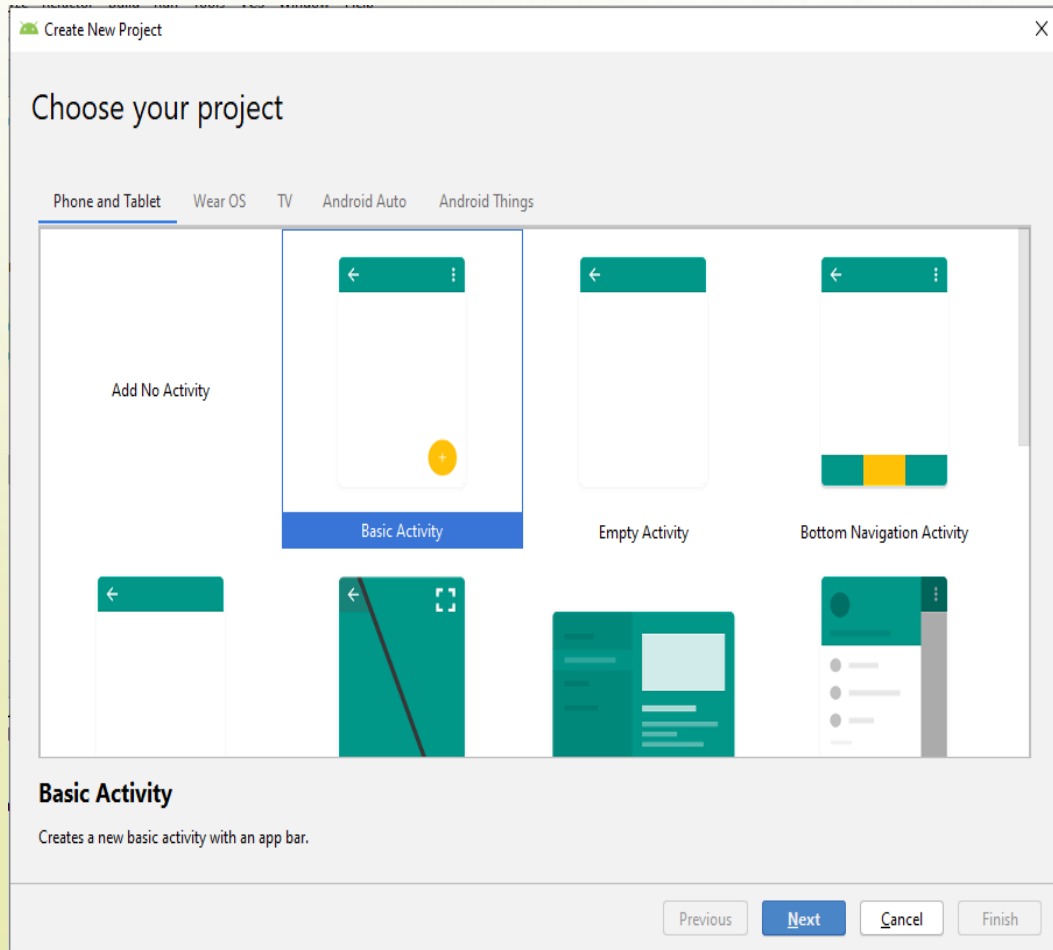
The Floating Action Button (FAB) with Snackbar

- 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.



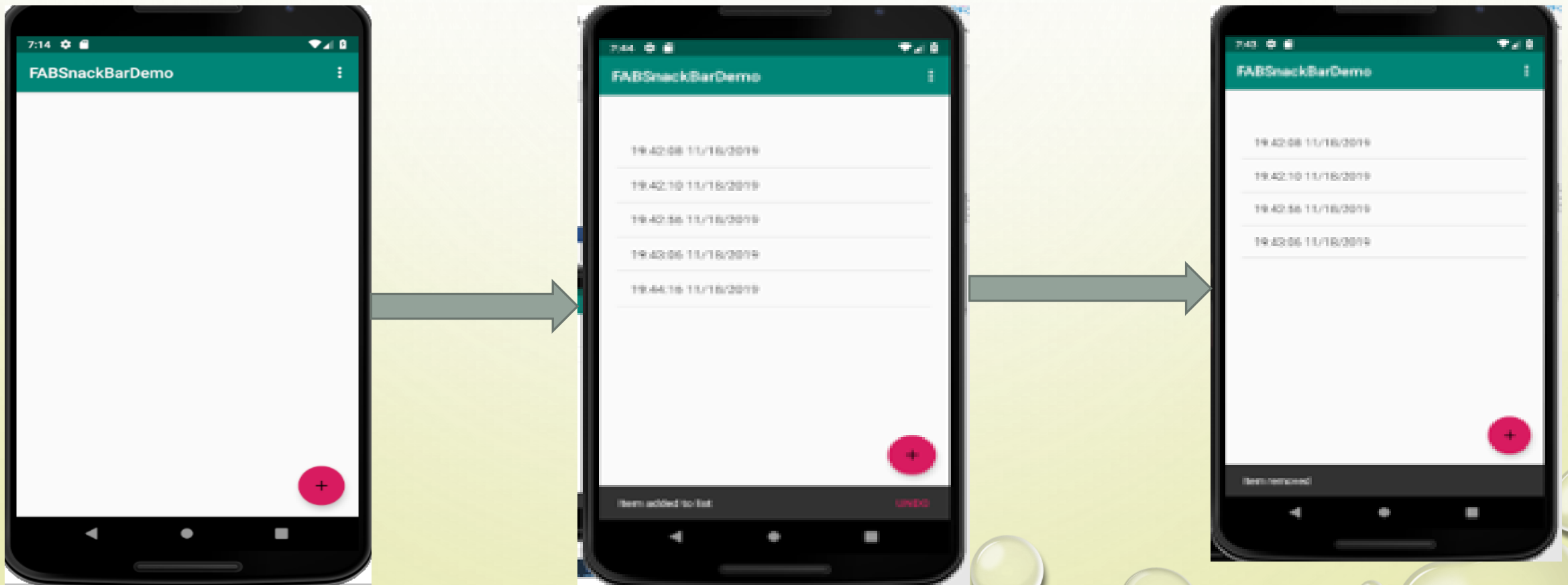
Create FAB with Basic Activity

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



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



Main Point 3

- Swipe views provide the navigation between tabs using finger gesture. Material design library such as float action button, snack bar, tab layout and navigation drawer provides components and tools that supports the best practices of user interface design. *Science of consciousness: As a field of ALL POSSIBILITIES, anything can be created and navigated within the self, using transcendental consciousness.*

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Inner class is an integral part of the unlimited potential

1. A Fragment represents a behavior or a portion of user interface in an Activity.
2. Fragments add stable and dynamic experiences taken from stable or dynamic resources just as Creative Intelligence is Stable and Adaptable: Anchored to the depths of life while playing in the waves of change.

-
3. **Transcendental Consciousness:** TC is the unbounded context for individual awareness.
 4. ***Impulses within the Transcendental field:*** *The impulses within the transcendental field are the simplest (and most powerful) ones to achieve the desired result.*
 5. ***Wholeness moving within Itself:*** *When individual awareness is permanently and fully established in its transcendental "context" – pure consciousness – every impulse of creation is seen to be an impulse of one's own awareness.*

