Smt. Chandaben Mohanbhai Patel Institute of Computer Applications

CA621

Basics of Mobile Applications

BSc.IT-Semester-VI

Unit-3

Android Widgets and Adapters

Dr. Arpankumar Raval, Assistant Professor, CMPICA, CHARUSAT

Topics

1. Calendar: Manage Date and Time
2. Adapters: ArrayAdapter, BaseAdapter using ListView and ListActivity, GridView, Gallery
3. Menus: Option menu, Context menu, Sub menu
4. Spinner
5. Tabs and TabActivity
6. Alert Dialogs, Android Toolbar

1. Calendar: Manage Date and Time

- The Calendar View allows user to select the date from the calendar. It has different attributes to fetch date and time together as well as individual time and date with day of the month, month, year, hours, minutes and seconds from the view.
- The Date EditText and Time EditText to allow user to enter manually.
- The DatePicker and TimePicker views are deprecated and not in the Palette.
- The Calendar View can be placed on the activity or can be used by creating its instance in the application on any event.

❖ Code to use the calendar view without drag and drop

```
lateinit var calendar:Calendar

calendar = Calendar.getInstance()
binding.button.setOnClickListener {
val datePicker = DatePickerDialog(
            this, { _, year, month, dayOfmonth ->
              val selectedDate = "$dayOfmonth/${month+1}/$year"
              Toast.makeText(this, selectedDate, Toast.LENGTH_SHORT).show()
},
            calendar.get(Calendar.YEAR),
            calendar.get(Calendar.MONTH),
            calendar.get(Calendar.DAY_OF_MONTH)
         )
         datePicker.show() }
```

Use current date and time

❖ To use current date and time the LocalDate and LocalTime functions can be used. And that variable allows to separate the elements.

```
val date = LocalDate.now()
        val day = date.dayOfMonth
        val mon = date.monthValue
        val year = date.year

        val time = LocalTime.now()
        val h = time.hour
        val m = time.minute
        val s = time.second
```

2. Adapters: ArrayAdapter, BaseAdapter

Adapters: To fill data in a list or a grid we need to implement Adapter. Adapters act **like a bridge** between UI components and data sources. Here data source is the source from where we get the data and UI components are list or grid items in which we want to display that data.

Ex: Adapter of your phone connects your mobile to the electricity board in home.

Types of Adapters: Base, Array, Custom are three types of the Adapters.

- **ArrayAdapter** — It is used whenever we have a list of single items which is backed by an array.
- **BaseAdapter** — It is the parent adapter for all other adapters.
- **Custom ArrayAdapter** — It is used whenever we need to display a custom list. Means we can have multiple arrays.

Array Adapter

- **ArrayAdapter** is the most commonly used adapter in android. When you have a list of single type items which are stored in an array you can use ArrayAdapter. Likewise, if you have a list of phone numbers, names, or cities. ArrayAdapter has a layout with a single TextView.
- We can use with AutoCompleteTextView, Spinner, Grid or any UI component which requires the static data.
- The example of AutoCompleteTextView and Spinner will explain the use and functions of the ArrayAdapter.

❖ AutoCompleteTextView Example

```
var country = arrayOf("India", "Indonesia","Irland","Iran",
"Bangladesh","Brazil")
var adapter =
ArrayAdapter(this,android.R.layout.simple_dropdown_item_1line,count
ry)
        actvc.threshold = 1
        actvc.setAdapter(adapter)
```

- Spinner Example:

- var country = arrayOf("Select","India","Austrailiya","New Zealand")

var myadapter = ArrayAdapter(this, android.R.layout.simple_spinner_dropdown_item,country)

spinner1.adapter = myadapter

Base Adapter

- It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner etc. For more customization of views we uses the base adapter.
- BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView, GridView, Spinner etc.
- Whenever you need a customized list in a ListView or customized grids in a GridView you create your own adapter and extend base adapter in that.
- Base Adapter can be extended to create a custom Adapter for displaying a custom list item.

- ArrayAdapter is also an implementation of BaseAdapter.
- The BaseAdapter will provide the data to the list items dynamically but with provided arrays.
- The base adapter has four main member functions to be implemented when created. getCount(), getItem(), getItemId() and getView().
- Example: To use the base adapter we will create one app, that will show the images with its names.
- For that we need the view that can show those images and names. We will use the ListView to show the items with following code.

Layout

```
<ListView
android:id="@+id/mylist"
android:layout_width="match_parent"
android:layout_height="match_parent"/> // Defined in Main Activity.XML
<TextView
        android:id="@+id/nm"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content“ />
<ImageView
        android:id="@+id/imageView"
        android:layout_width="250dp"
        android:layout_height="250dp“ /> Defined in Separate File ListItem.XML
```

Main Activity Code

```
var
image=arrayOf(R.drawable.computer,R.drawable.laptop,R.drawable.iphone,R.drawable.router)
var name= arrayOf("Computer","Laptop","iPhone 13","Router")
mylist.adapter = CustomAdapter(this,image,name)
```

listview

Custom Adapter Code –Create Class

```
class CustomAdapter(
    var mainActivity: MainActivity, var image:Array<Int>, var name:
Array<String>,    ) : BaseAdapter() {
    override fun getCount(): Int {  return image.size }
    override fun getItem(p0: Int): Any {  return p0  }
    override fun getItemId(p0: Int): Long { return p0.toLong()  }
```

class custom adapter extends base adapterr

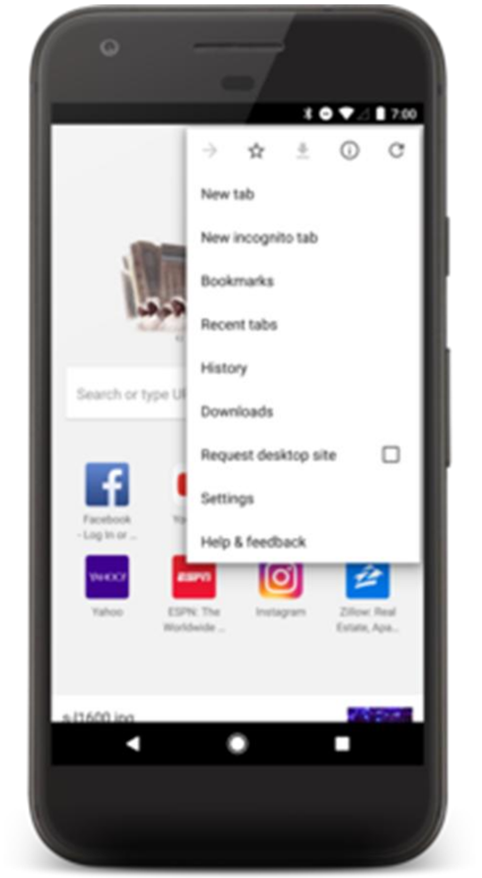
```
override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
    var inview = LayoutInflater.from(mainActivity).inflate(R.layout.listitem,p2,false)
    var image1 = inview.findViewById<ImageView>(R.id.imageView)
    var name1 = inview.findViewById<TextView>(R.id.nm)

    image1.setImageResource(image[p0])
    name1.text = name[p0]

    return inview
}
```

3. Menu

- In android, Menu is an important part of UI component which is used to provide some common functionality around the application. With the help of menu user can experience smooth and consistent experience throughout the application.
- Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience.
- In order to use menu, we should define it in separate XML file and use that file in our application based on our requirements. Also, we can use menu APIs to represent user actions and other options in our android application activities.



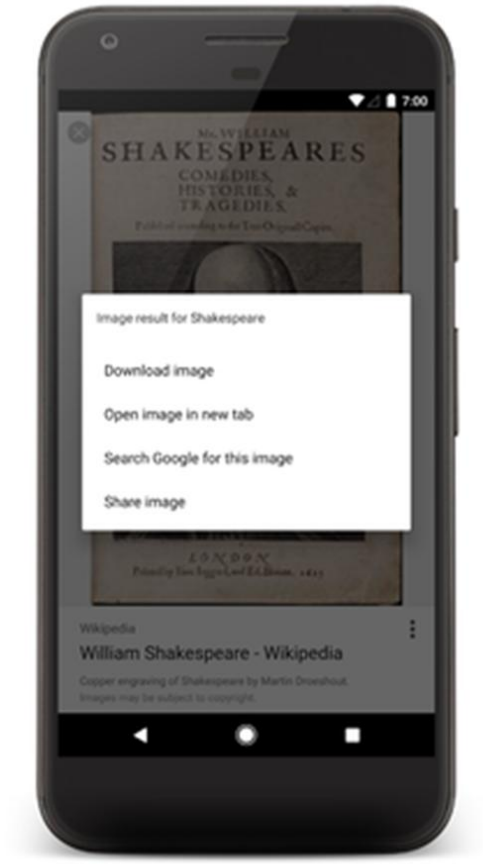
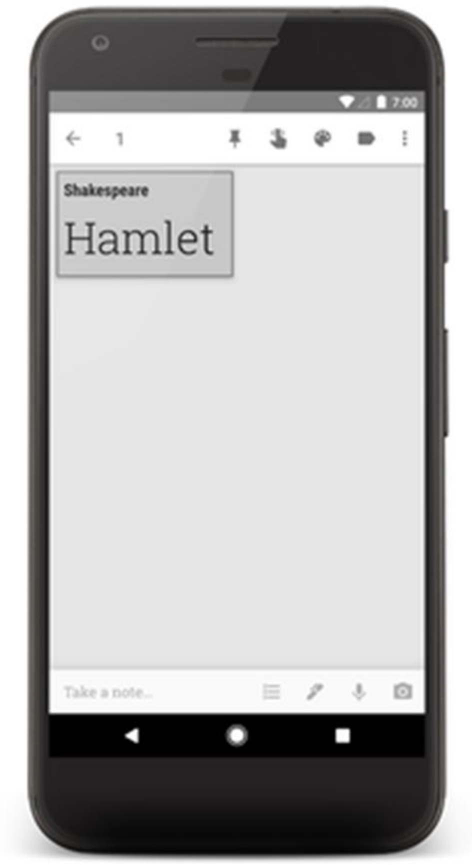
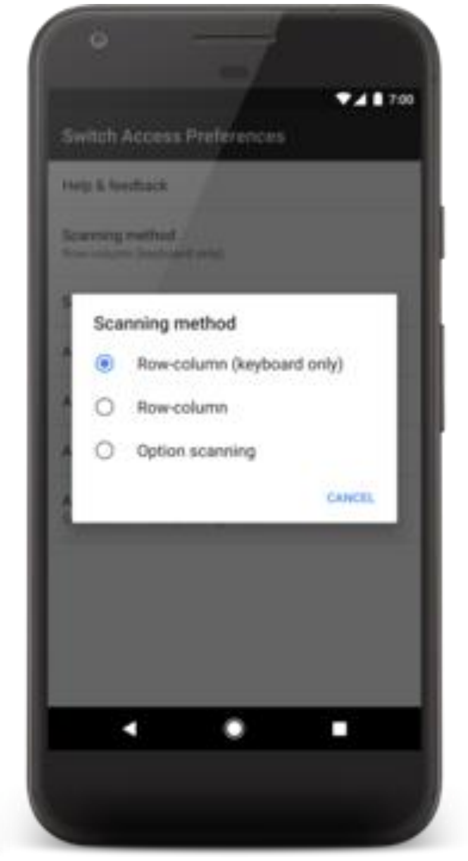
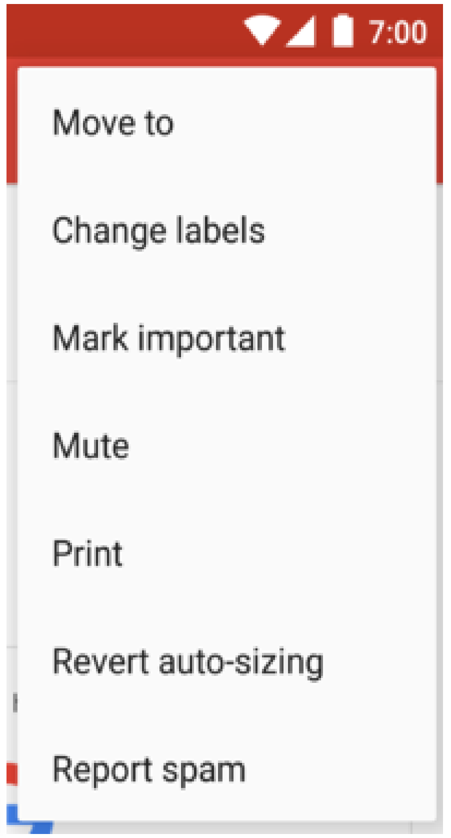


Image result for Shakespeare
Download image
Open image in new tab
Search Google for this image
Share image
William Shakespeare - Wikipedia





Scanning method
[✓ Row-column (keyboard only)]
Row-column
Option scanning



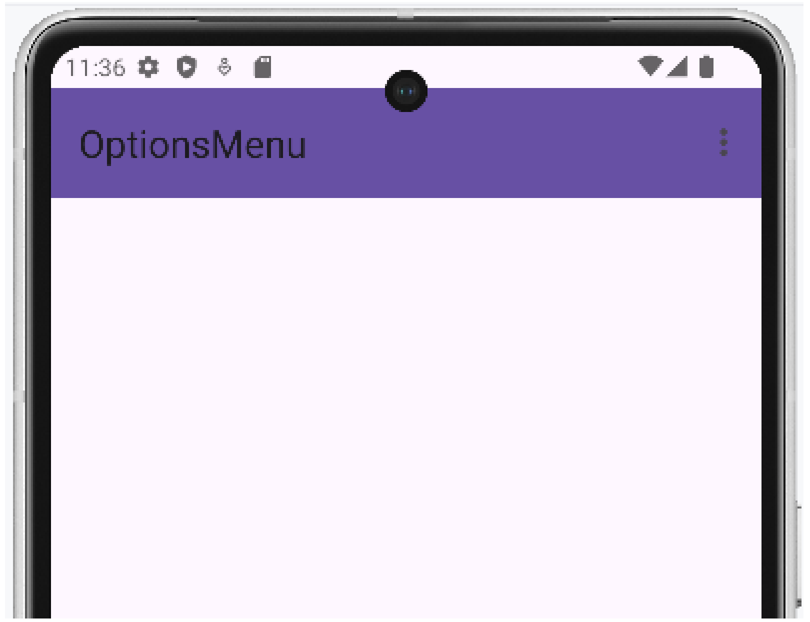
Types of Menus

- **Android Options Menu, Android Context Menu, Android Popup Menu**
- **Android Options Menu** – Android Options Menu is a primary collection of menu items in an android application and useful for actions that have a global impact on the searching application.
- **Android Context Menu** – Android Context Menu is a floating menu only appears when user click for a long time on an element and useful for elements that effect the selected content or context frame.
- **Android Popup Menu** – Android Popup Menu displays a list of items in a vertical list which presents to the view that invoked the menu and useful to provide an overflow of actions that related to specific content.

Options Menu

- ❖ In Android, options menu can be attached in the Appbar. It will be shown as three vertical dots. To open the Menu, you have to single click on the Options Menu.
- ❖ In early versions of the Android OS, the options menu was directly created and shown on the App.
- ❖ But, from API Level 21 (Android 5.0 Lollipop), if you're using an Activity without an ActionBar. (e.g., a custom theme or NoActionBar theme), you must manually create an AppBar (Toolbar) to display the options menu.

- There are two ways to define the Menu Items in Android. Either you can create separate resource file under menu package and inflate it or you can add the Menu items in its method.
- **Through Resource File:** To create menu item file, create package(folder) under **res** (Resource) directory Right Click **New**>, **Android Resource Directory** > Name provide name as **Menu**, select resource type **menu >** Press Ok.
- Now under Menu package Right click and Create > New > Menu Resource File > provide name menuitems and press ok.
- Now provide items to that menuitem file.



Layout code for MenuItem file

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

    <item android:id="@+id/yes"
        android:title="YES"
        app:showAsAction="never"/>
    <item android:id="@+id/no"
        android:title="NO"
        app:showAsAction="never"/>

</menu>
```

Layout code for Appbar and Toolbar in MainActivity.XML

```
<com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:ignore="MissingConstraints" >
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:layout_scrollFlags="enterAlways|scroll"/>
    </com.google.android.material.appbar.AppBarLayout>
```

Kotlin Code

- ❖ As we are using API level above 21, we have to create the Toolbar and attach that in Appbar.

```
var toolbar = findViewById<Toolbar>(R.id.toolbar)
        setSupportActionBar(toolbar)
```

Now There are two member functions to handle Options Menu in Android. One to create Options Menu and one to handle the menu selection event.

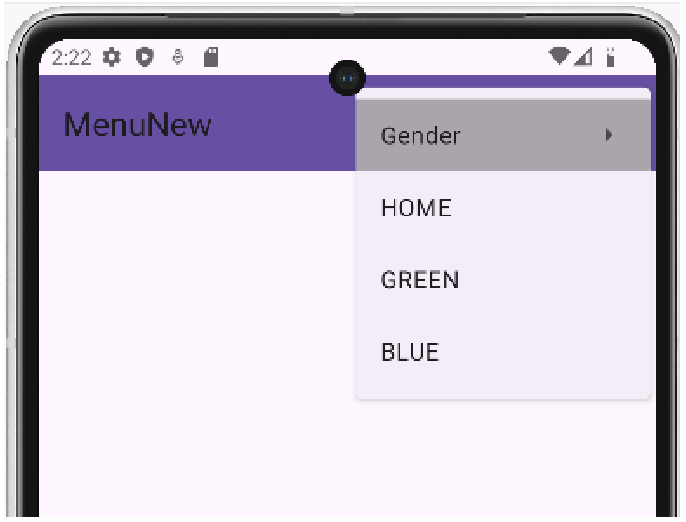
1. override fun onCreateOptionsMenu
2. override fun onOptionsItemSelected

Options Menu code using Inflate Method

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menuitems,menu)
        return super.onCreateOptionsMenu(menu)    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when(item.itemId){
  R.id.yes->    Toast.makeText(this,"YES",Toast.LENGTH_LONG).show()
  R.id.no-> Toast.makeText(this,"NO",Toast.LENGTH_LONG).show()
        }
        return super.onOptionsItemSelected(item)
    }
```

Options Menu/Sub menu code using Manual Method

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menu?.add(1,1001,1,"HOME") // Add Menuitems Manually
        menu?.add(1,1002,2,"GREEN")
        menu?.add(1,1003,3,"BLUE")
                var smenu = menu?.addSubMenu("Gender")  // To add SubMenu
                smenu?.add(2,11,4,"Male“)
                smenu?.add(2,12,5,"Female")
        return super.onCreateOptionsMenu(menu)      }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when(item.itemId)
        {  1001-> Toast.makeText(this,"HOME",Toast.LENGTH_SHORT).show()
            1002-> Toast.makeText(this,"GREEN",Toast.LENGTH_SHORT).show()
            11-> Toast.makeText(this,“MALE",Toast.LENGTH_SHORT).show()}
        return super.onOptionsItemSelected(item)    }
```



Context Menu

- In Android, the context menu is like a floating menu and arises when the user has long-pressed or clicked on an item and is beneficial for implementing functions that define the specific content.
- It can be assign on any context of the Activity like, TextView, EditText, Image etc.
- There are four steps to assign the context menu.
- Create the context, set the menu on it, create and use on selection.
- To create write code in Layout file with ID.

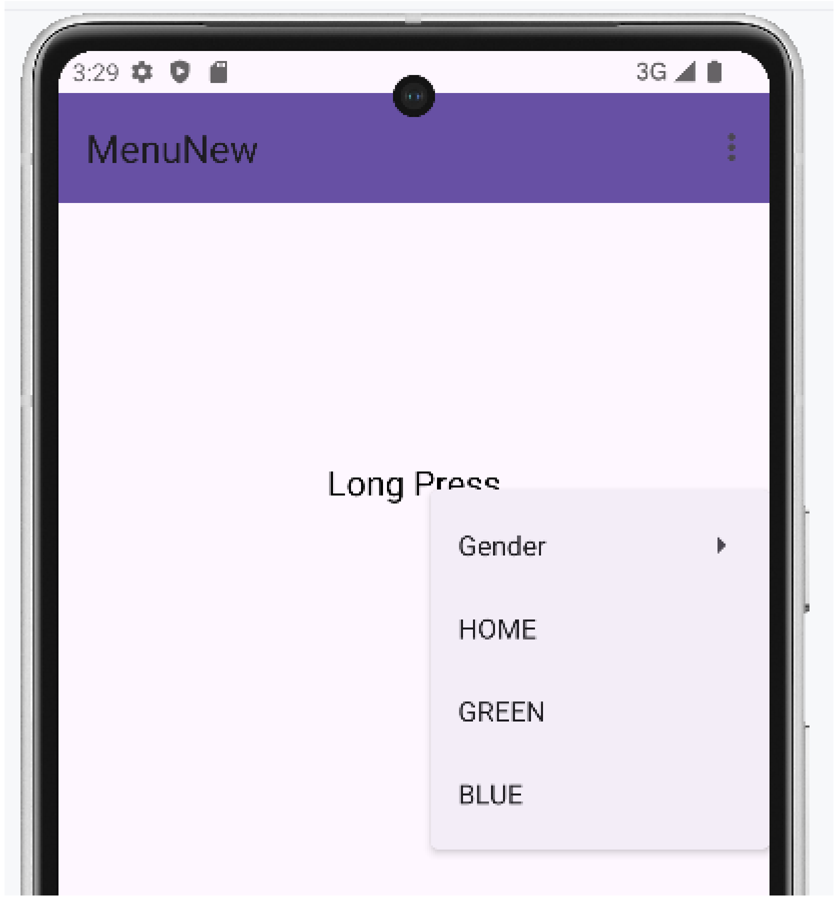
- To assign: registerForContextMenu(ViewId) in Oncreate Function
- Create Context Menu: use

override fun onCreateContextMenu

- To Use Item selection:

override fun onContextItemSelected

Here also either you can inflate the Menu Items or you can assign manually. Where ever the view is, it will show the Menu options.



Code

```
override fun onCreateContextMenu( menu: ContextMenu?, v: View?,
        menuInfo: ContextMenu.ContextMenuInfo?) {
        menu?.add(3,201,3,"Red")
        menu?.add(3,202,2,"Green")
        menu?.add(3,203,1,"Blue")
        menu?.setHeaderTitle("Select Color to change")
        super.onCreateContextMenu(menu, v, menuInfo)
    }
```

Code

```
override fun onContextItemSelected(item: MenuItem): Boolean {
        when(item.itemId) {
            201 -> (textView.setTextColor(Color.RED))
            202 -> (textView.setTextColor(Color.GREEN))
            203 -> (textView.setTextColor(Color.BLUE))
        }
            return super.onContextItemSelected(item)
    }
```

PopUp Menu

- Android Popup Menu is a list menu that appears vertically to the view. The popup menu appears below the view if there is room otherwise, appears above. Touching outside to popup menu makes it disappear.
- Popup Menu provides actions that are related to the specific content, and it does not affect the corresponding content.
- It only allows you to inflate the menu items, you can not create the menu items in the activity code file.
- You can assign on any event of the View, most of the time click or tap.

Layout Code for Menu Items of Popup menu

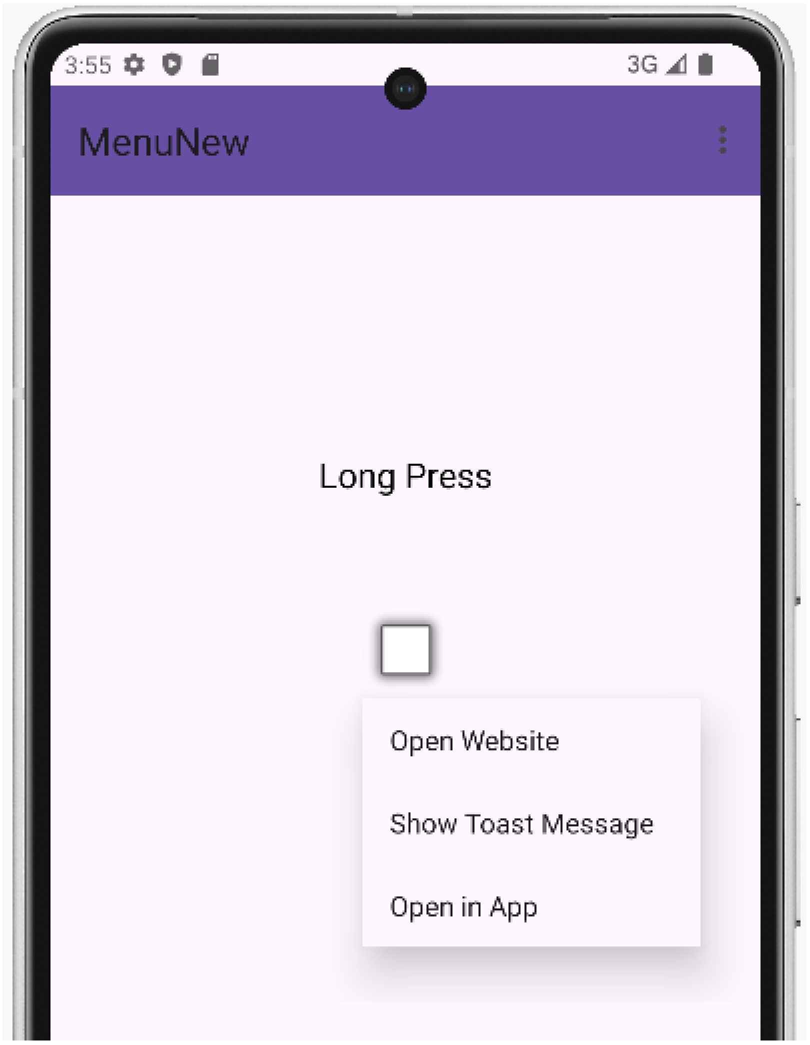
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_open_website"
    android:title="Open Website"
    android:icon="@drawable/ic_baseline_public_24">

</item>
    <item android:id="@+id/menu_show_toast"
        android:title="Show Toast Message"
        android:icon="@drawable/ic_baseline_message_24">

    </item>
    <item
        android:id="@+id/open_in_app"
        android:title="Open in App">

    </item>
</menu>
```

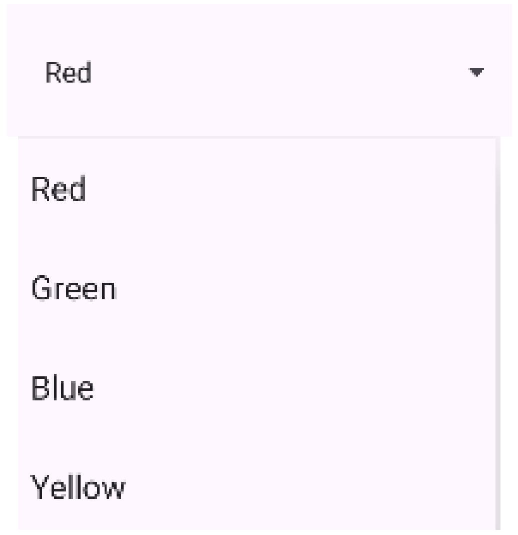


```
img.setOnClickListener {  // On the imageview we have provided the options
            val popupmenu = PopupMenu(this, it)
            popupmenu.inflate(R.menu.menu_main)
            popupmenu.show()
            popupmenu.setOnMenuItemClickListener { item ->
                when (item.itemId) {
                    R.id.menu_open_website -> {val intent =
                            Intent(Intent.ACTION_VIEW, Uri.parse("http://www.facebook.com"))
                            startActivity(intent)  true   }
                    R.id.menu_show_toast -> { Toast.makeText(this, "You clicked Facebook",
Toast.LENGTH_LONG).show()
                        true   }
                            R.id.open_in_app -> {
  Toast.makeText(this, "Open Facebook Applicaiton",
Toast.LENGTH_LONG).show()
true }  else -> false
                }  }
```

4. Spinner

- ❖ Spinners are like a drop-down menu that contains a list of items to select from.
- ❖ Once a value is selected the Spinner returns to its default state with that selected value.
- ❖ There are two requirements, provide data through Adapter and use method to handle the spinner.
- ❖ Use Array Adapter and provide, onItemSelectedListener method.
- ❖ It will implement 2 member functions, **onItemSelected** and **onNothingSelected**

- ❖ Here in this spinner we will assign the different colors to it and will assign the data to the variable and we assign this data to Toast message.
- ❖ The Spinner is available in Palette to drag and drop on the activity.
- ❖ Then Assign the data and use the ready made method for selection of the option.



Code

```
var spinner = findViewById<Spinner>(R.id.spinner)
var colors = arrayOf("Red","Green","Blue","Yellow")
var adapter = ArrayAdapter(this,android.R.layout.simple_spinner_dropdown_item,colors)
spinner.adapter = adapter

spinner.onItemSelectedListener =object:AdapterView.OnItemSelectedListener{
    override fun onItemSelected(
        parent: AdapterView<*>?,
        view: View?,
        position: Int,
        id: Long
    ) {
        Toast.makeText(this@MainActivity,"You have Selected "+colors[position],Toast.LENGTH_SHORT).show()
    }
    override fun onNothingSelected(parent: AdapterView<*>?) {
        TODO("Not yet implemented")
    }   }
```

5. Tab and Tabs Activity using Tab Layout

- In some Android apps, **Tabs** are used, which allows developers to combine multiple tasks (operations) on a single activity.
- On the other hand, it provides a different look to that app. It is also possible to provide a different feel like left and right swipes by using **ViewPager2**.
- And to implement this, a few widgets are required such as **ViewPager2**, **Fragments**, and **TabLayout**.

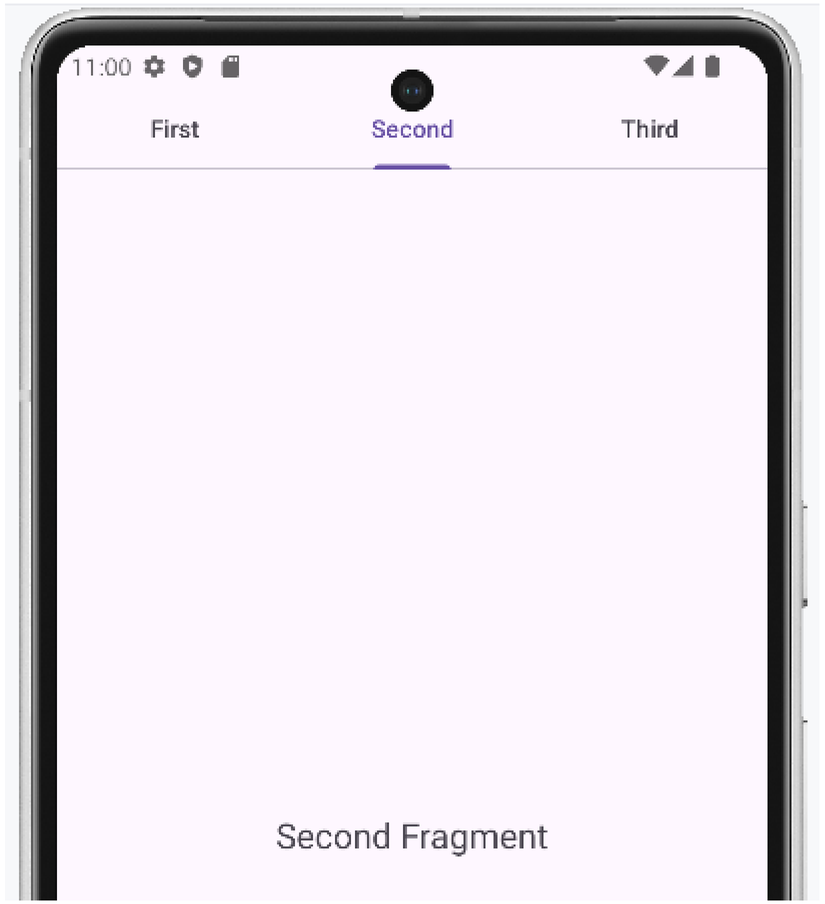
What are TabLayout, ViewPager, and Fragment?

- **TabLayout:** This view allows us to make use of multiple tabs in the Android app. This **Layout** holds different tabs. In which, tabs are used to navigate from one Fragment to another Fragment.
- **ViewPager2:** This view allows us to make use of the left and right swipe features to show another fragment.
- **Fragments:** This is a part of the **Activity**. This view is necessary, to do multiple tasks in a single activity. The **Fragment** also makes use of a layout file which contains view/s as needed.

XML Code to Insert TabLayout and ViewPager2

```
<com.google.android.material.tabs.TabLayout
    android:id="@+id/tablayout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    tools:ignore="SpeakableTextPresentCheck"
    app:layout_constraintTop_toTopOf="parent" />

<androidx.viewpager2.widget.ViewPager2
    android:id="@+id/viewpager2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteX="1dp"
    tools:layout_editor_absoluteY="49dp" />
```



Creating Fragments

- To handle the individual activities through Fragments, include 3 fragments in Package from Right click on Package and New> Fragment.
- The Fragment has two files exactly same as Activity.
- These fragments will be loaded on ViewPager2 widget on tab click or swipe. And load dynamically.

Code

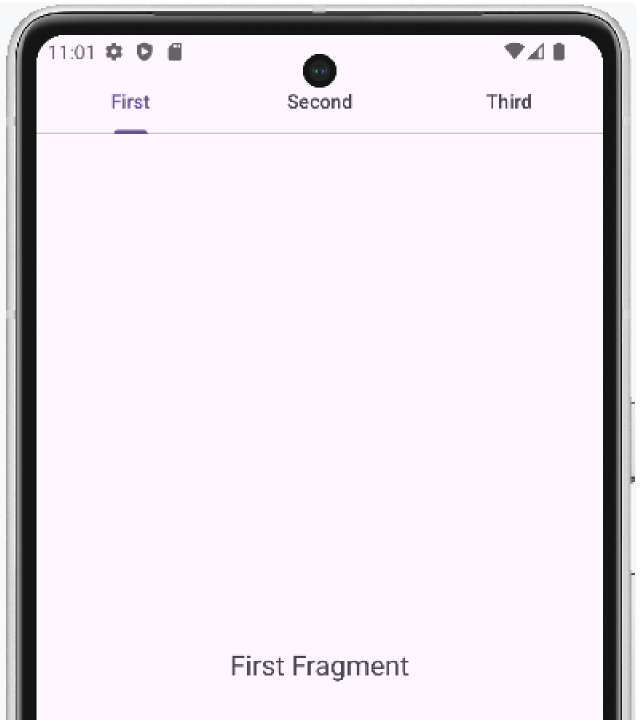
```
//Code to Create Adapter to show fragments and assign to viewoager2

val adapter = ViewPagerAdapter(supportFragmentManager,lifecycle)
binding.viewpager2.adapter = adapter

// Code to create dynamic tabs on the TabLayout
TabLayoutMediator(binding.tablayout,binding.viewpager2){tab, position->
when(position){
    0->{tab.text="First"}
    1->{tab.text="Second"}
    2->{tab.text="Third"}
     }
}.attach()
```

Code of Fragment State Adapter

```
class ViewPagerAdapter(supportFragmentManager: FragmentManager,
lifecycle: Lifecycle)
:FragmentStateAdapter(supportFragmentManager,lifecycle) {
    override fun getItemCount(): Int {
        return 3
    }
    override fun createFragment(position: Int): Fragment {
        return when(position){
        0->{Fragment1()}
            1->{Fragment2()}
            2->{Fragment3()}
            else->{Fragment()}
        }
    }
}
```



First Fragment

6. Alert Dialog and Toolbar

- **Alert Dialog:** Alert Dialog is a window that pops up on the screen. They generally show some information and ask for a user action. There are three core components that build an Alert Dialog.
- It is also the UI element that displays a warning or notification message and asks the user to respond with options such as **Yes** or **No**. Based on the user's response, appropriate actions are executed. Android Alert Dialog is built with the use of three fields: **Title, Message area, and Action Button.**
- **Alert Dialog code has three methods and Button Options :**

setTitle(): method for displaying the Alert Dialog box Title

setMessage(): method for displaying the message

setIcon(): method is used to set the icon on the Alert dialog box.

Buttons - There are three types of buttons: Positive, Negative, and Neutral

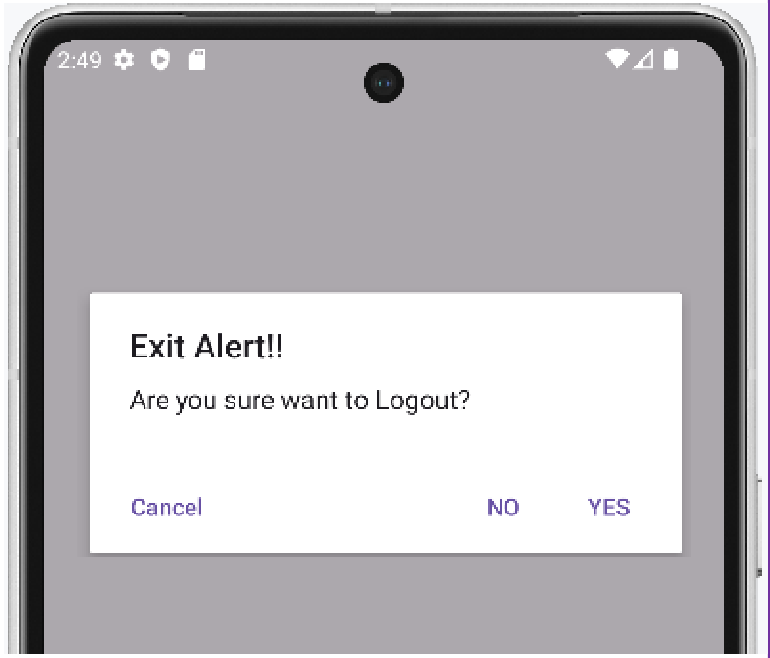
- The three buttons are,
- alertbuilder.setPositiveButton : When Agree what process
- alertbuilder.setNegativeButton: When disagree what process
- alertbuilder.setNeutralButton: When neutral no action or process

Example:

Here we will create one Alert dialog to select YES, NO or CANCEL.

You have to implement the

{ dialogInterface:DialogInterface, i:Int -> } Method and provide necessary action after lambda expression.



Code

```
var alertbuilder =AlertDialog.Builder(this)
alertbuilder.setTitle("Exit Alert!!")
alertbuilder.setMessage("Are you sure want to Logout?")

alertbuilder.setPositiveButton("YES",{dialogInterface :DialogInterface,i:Int->
    Toast.makeText(this,"Go to Login Page",Toast.LENGTH_SHORT).show()
})
alertbuilder.setNegativeButton("NO",{dialogInterface:DialogInterface,i:Int->
    Toast.makeText(this,"Go to Stay on Same page",Toast.LENGTH_SHORT).show()
})
alertbuilder.setNeutralButton("Cancel",{dialogInterface:DialogInterface,i:Int->
    Toast.makeText(this,"Do Nothing",Toast.LENGTH_SHORT).show()
})
alertbuilder.show()
```

Toolbar

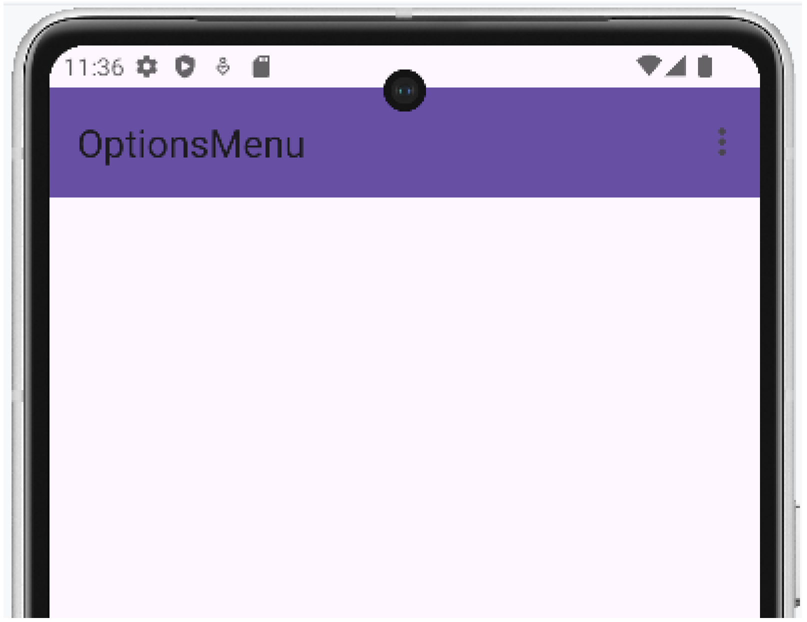
- In Android applications, **Toolbar** is a kind of **ViewGroup** that can be placed in the **XML layouts** of an activity. It was introduced by the Google Android team during the release of **Android Lollipop(API 21)**. The Toolbar is basically the advanced successor of the **ActionBar**. It is much more flexible and customizable in terms of appearance and functionality.
- Unlike ActionBar, its position is not hardcoded i.e., not at the top of an activity. Developers can place it anywhere in the activity according to the need just like any other **View** in android.

- One can use the Toolbar in the following two ways:
- **Use as an ActionBar:** In an app, the toolbar can be used as an ActionBar in order to provide more customization and a better appearance. All the features of ActionBar such as menu inflation, ActionBarDrawerToogle, etc. are also supported in Toolbar.
- **Use a standalone Toolbar:** It can be used to implement a certain kind of design to an application that can not be fulfilled by an ActionBar. For example, showing a Toolbar at a position other than the top of the activity or showing multiple Toolbars in an activity.

Example

❖ To show the Options Menu, we have already used this toolbar shown in Appbar.

Code:
```
<com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:ignore="MissingConstraints" >
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:layout_scrollFlags="enterAlways|scroll"/>
    </com.google.android.material.appbar.AppBarLayout>
```



THANK
YOU