Lesson 9: Android Navigation Components

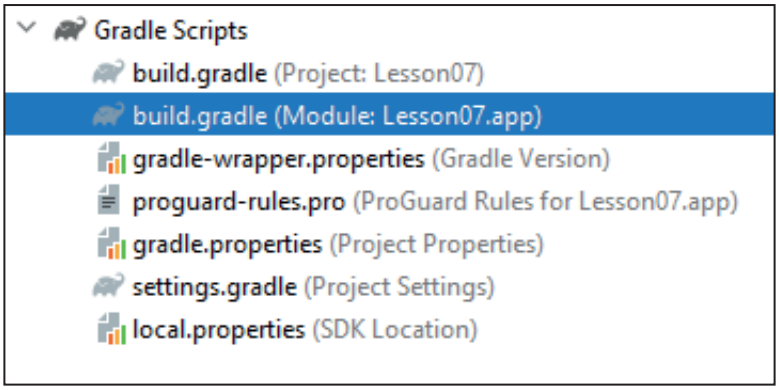
Menus

Menus are common user interface components used to provide user actions and other options in your activities. Menus are provided as a part of an action bar of the activity. Each activity in an application can get its own options' menu – the main collection menu items for an activity.

The user can access the options menu from the action bar by pressing on the three dots icon as illustrated in the figure below:

Creating an Options Menu

1- Open Android Studio, and then click **File → New → New Project**
2- Select **Empty Activity**, and click **Next**
3- Type: **Lesson09_Menus** for the application name, then click **Finish**.
4- Before start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson09_Menus.app)** file content as illustrated in the figure below:



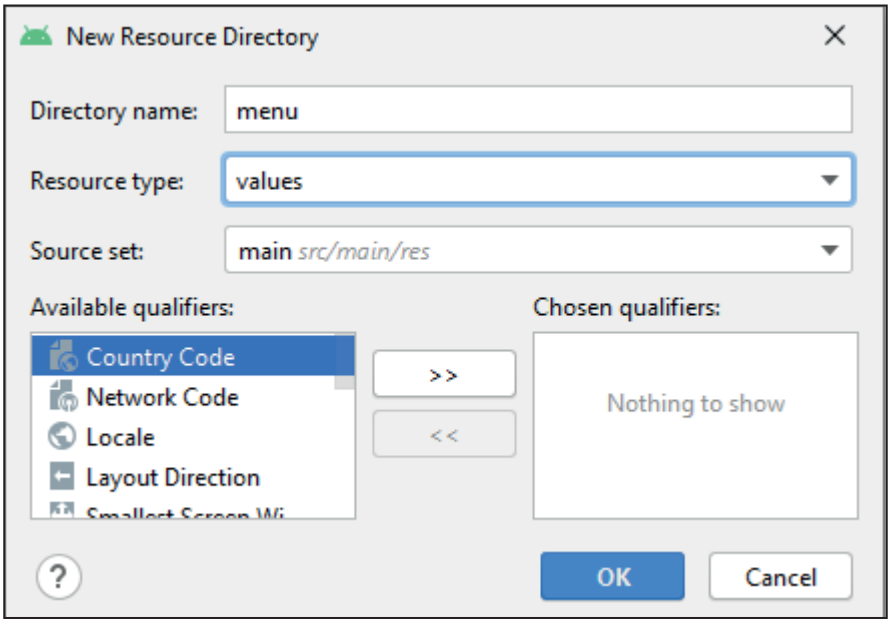
Be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**
And click the **Sync Now**. The configuration should be as follows:

```
1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }
```

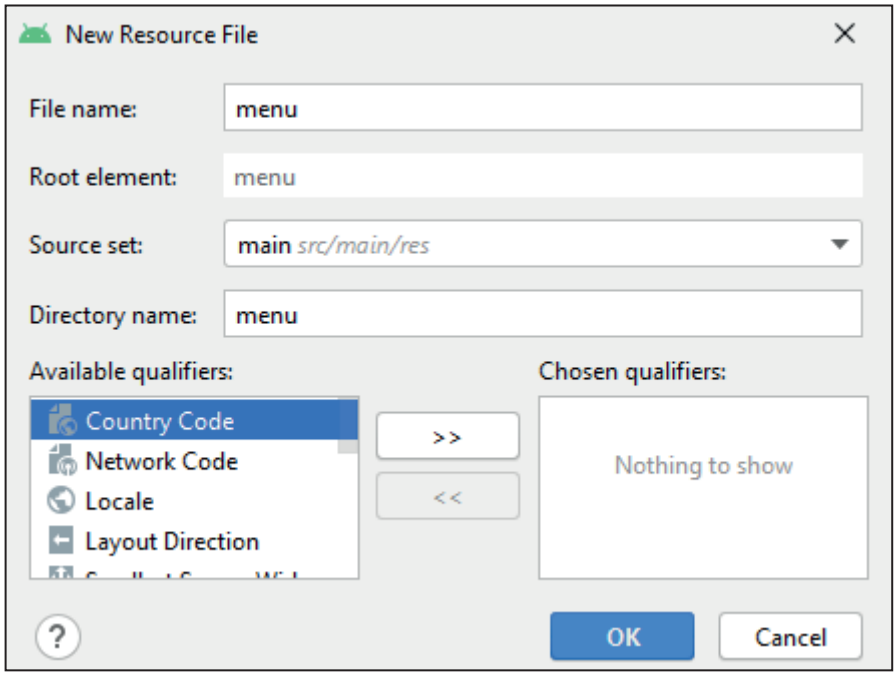
Now, you are going to create a separate file called **menu.xml** which will include the menu items (Menu content). Then, to use the menu in your activity, you need to inflate the menu resource (convert the **menu.xml** file into a programmable object) using `MenuInflater.inflate()` as you will do in the following steps:

5- Create a new director called **menu** under the **res** directory. To do so, right click the **res** directory**,** and then select **New → New Resource Directory.**

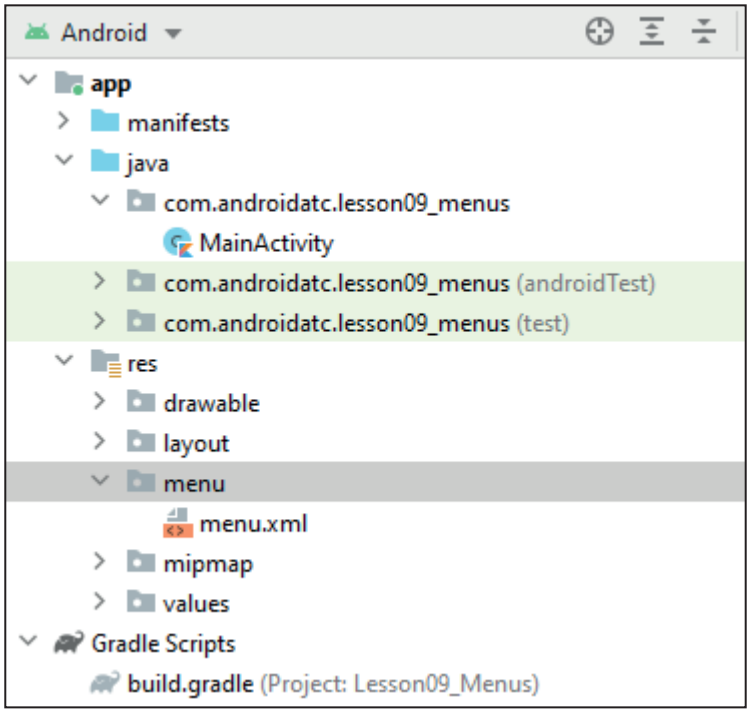
As illustrated in the figure below, in the **Directory Name** text box, enter the name of the menu: **menu**, then click **OK**.



6- Right click the **menu** directory then → **New** → **Menu Resource File**.
As illustrated in the dialog box below, enter the **File name: menu**, then click **OK**

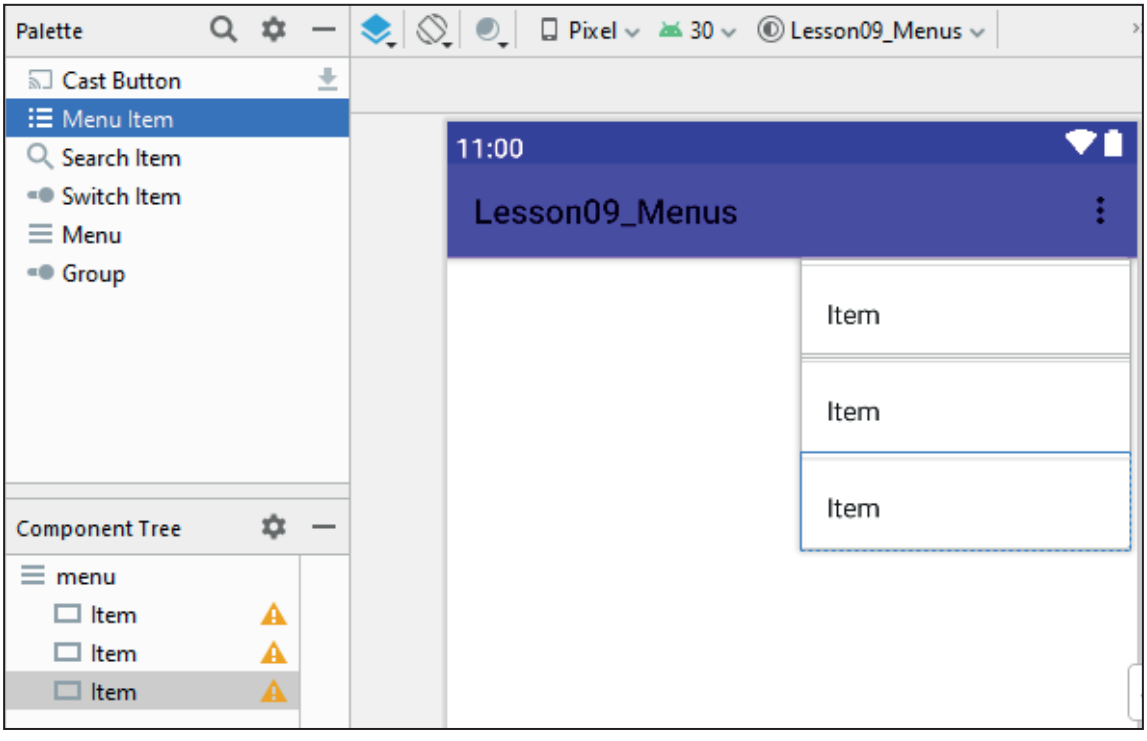


Now, you should have the following project files structure:



7- Open the **menu.xml** file in **Design** mode. From the **Palette** panel and using drag and drop technique, add **Menu Item** to your activity. The number of the menu items which you should add depending on the number of options which you want to have in your menu.

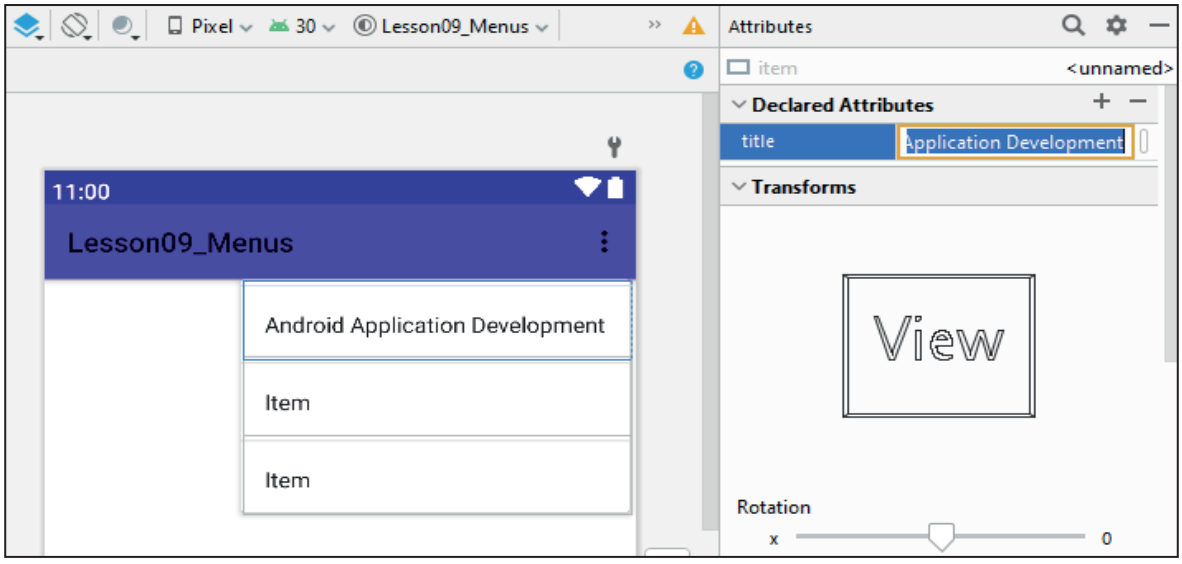
As you see in the following figure, we added three Menu Items to the menu activity.



8- Click the **Attributes** tab, and enter the **title** and **id** attributes values for each menu item as follows:

Id: developmentId **title**: Android Application Development	**Id**: securityId **title**: Android Security Essentials	**Id**: uiId **title**: Android UI/UX Design

As illustrated in the following figure:



Some developers prefer to open the **menu.xml** file in the **Code** mode, and use the **item** tag to add and configure these menu items. The **menu.xml** file in the **Code** mode as follows:

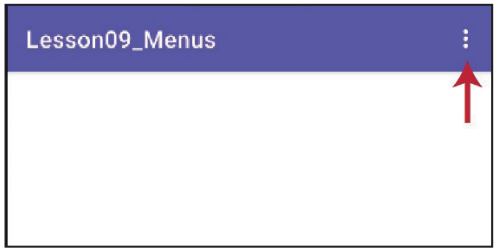
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/developmentId"
        android:title="Android Application Development"/>
    <item
        android:id="@+id/securityId"
        android:title="Android Security Essentials" />
    <item
        android:id="@+id/uiId"
        android:title="Android UI/UX Design" />
</menu>
```

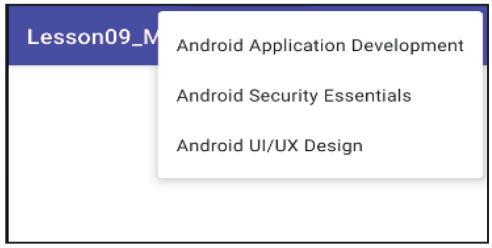
9- Now, to use or add this menu in the **MainActivity** file, you should open this file, then use the **onCreateOptionsMenu** () callback method. This method can inflate your menu resource **menu.xml** into the **Menu** provided in the callback, as illustrated in the following code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main) }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.menu, menu)
        return true

        }
}
```

Double click **Menu,** click the pop-up red lamp, and select **Import**.

10- **Run** your app. You should have a 3 dots icon. If you tap this 3 dots icon, you should get your menu items as illustrated in the following figures:





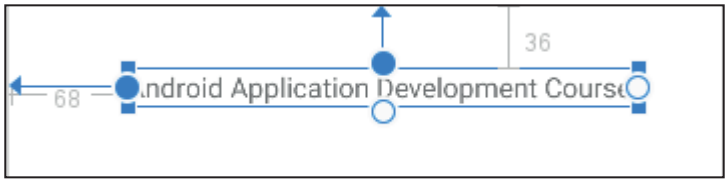
11- Now, you should configure the action which will be taken if your app users tap any of these menu items. To do this, you should use **OptionsItemSelecte()** method. In this example, you will use the **OptionsItemSelecte()** method with the first menu item (**id : developmentId**) to open a new activity called **AndroidDevelopment** using **Intent** class.

To do this, create this new activity first by right clicking **app → New → New Activity → Empty Activity**

Enter: **AndroidDevelopment** for the activity name, and click **Finish**

12- Open the **activity_android_development.xml** in the **Design** mode, add a **TextView** widget, set its constraints, set its **text** attribute value to: **Android Application Development Course** and set its **textSize** attribute value to: **16sp**

This activity should have the following design:



13- Open the MainActivity file, and add the following code to configure the **`OptionsItemSelecte()`** and **`Intent`** class to add the navigation feature for your menu items:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.menu, menu)
        return true

    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {

      when (item.itemId) {
        R.id.developmentId -> {
          startActivity(Intent(this, AndroidDevelopment::class.java))
        return super.onOptionsItemSelected(item)}
        else -> {
                return super.onOptionsItemSelected(item)
      }
    }

  }
}
```

Double click **`MenuItem`**, click the red pop-up lamp, and select **Import**.
Double click **`Intent`**, click the red pop-up lamp, and select **Import**.

14- **Stop**, then **Run** your app. If you tap your 3 dots menu, then tap the first menu item: **Android Application Development Course**., you will open the Android application development course activity.

15- If you create a new activity for the Android Security Essentials Course (**id**: securityId), the **`Intent`** and the **`OptionsItemSelecte()`** method configurations for this second menu item will be as follows:

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {

 when (item.itemId

      R.id.developmentId -> {
          startActivity(Intent(this, AndroidDevelopment::class.java))
          return super.onOptionsItemSelected(item)}

      R.id.securityId -> {
          startActivity(Intent(this, AndroidSecurity::class.java))
          return super.onOptionsItemSelected(item)}

      else -> {
          return super.onOptionsItemSelected(item)
      }
   }

}
```

Bottom AppBar

Bottom AppBar is a material widget that is displayed at the bottom of an app for selecting among a small number of views, typically between three and five. This bar consists of multiple items in the form of text labels, icons, or both, laid out on top of a piece of material. It provides quick navigation between the top-level views of an app.

Example:
In this example you will add a BottomAppBar widget to your activity and know how to add different navigation items to it.

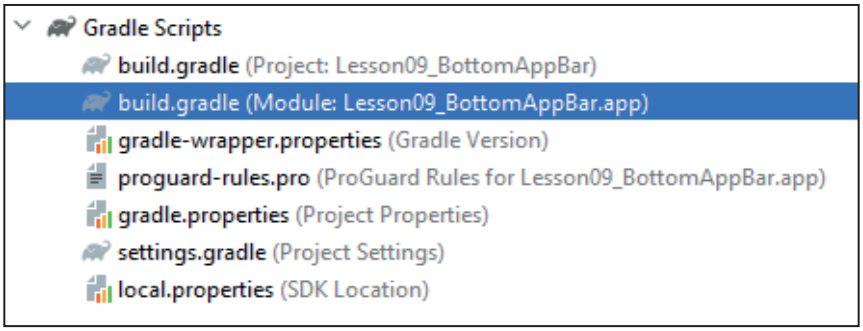
The steps are as follows:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson09_BottomAppBar** for the application name, then click **Finish**.

4- Before you start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson09_BottomAppBar.app)** file content as illustrated in the figure below:



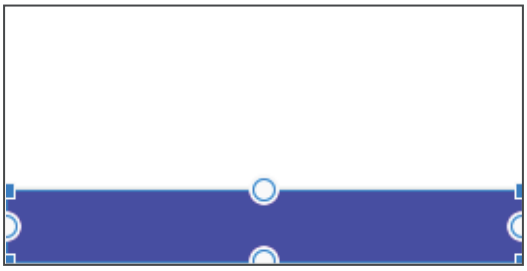
Be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** file in the **Design** mode, and then **delete** the "Hello World!" text.

6- From the **Palette** panel (Containers), add the **BottomAppBar** widget to the bottom of your activity using drag and drop technique. The following figure displays the bottom of the **activity_main.xml** file.



7- The **BottomAppBar** must be wrapped in a **coordinator** layout activity; therefore, open the **activity_main.xml** file in the Code mode, and in the second line of the XML code, replace the:

```
androidx.constraintlayout.widget.ConstraintLayout
```

with:

```
androidx.constraintlayout.widget.ConstraintLayout
```

The Full code should be as follows:

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

<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.bottomappbar.BottomAppBar
        android:id="@+id/bottomAppBar"
        style="@style/Widget.MaterialComponents.BottomAppBar.Colored"
```

```
        android:layout_width="409dp"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        tools:layout_editor_absoluteX="1dp"
        tools:layout_editor_absoluteY="674dp"

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Note: In the code above, the **`android:layout_gravity="bottom"`** attribute, if you change the gravity value to top, this bar will be moved to the top of your activity.

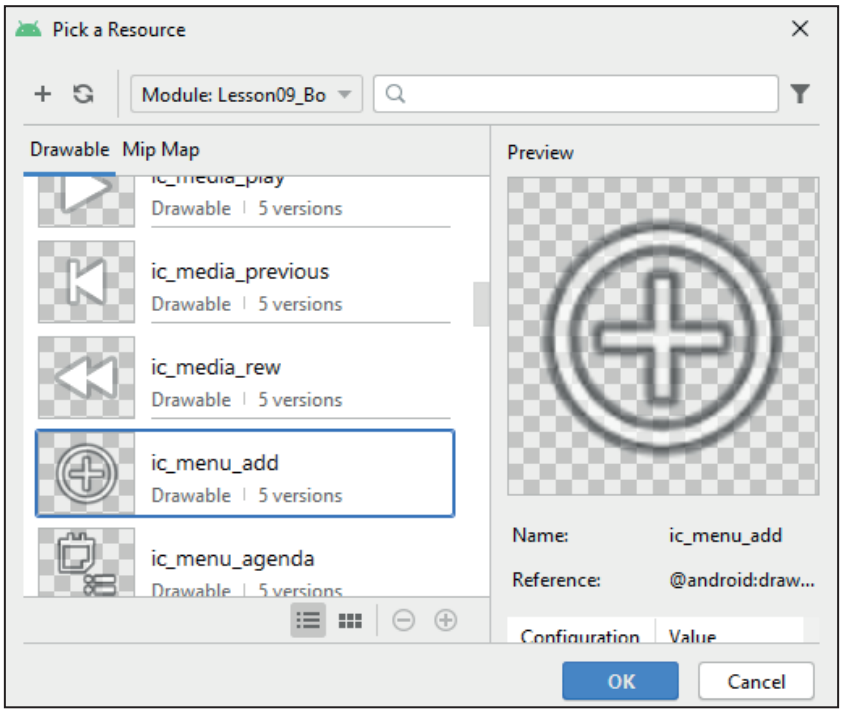
8- If you want to change the background color of this **BottomAppBar**, add the color which you want to from the **colors.xml** file (**app → res → values → colors.xml**), then add the following attribute to your **BottomAppBar**:

```
app:backgroundTint="@color/purple_500"
```

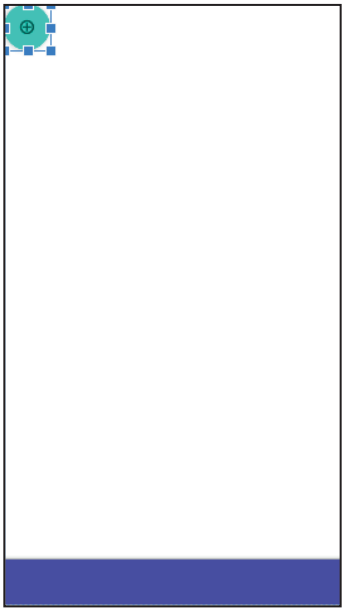
The XML code for the BottomAppBar tag should be as follows:

```
<com.google.android.material.bottomappbar.BottomAppBar
    android:id="@+id/bottomAppBar"
    style="@style/Widget.MaterialComponents.BottomAppBar.Colored"
    android:layout_width="409dp"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    tools:layout_editor_absoluteX="1dp"
    tools:layout_editor_absoluteY="674dp"
    app:backgroundTint="@color/purple_500" />
```

9- Open the **activity_main.xml** file in the **Design** mode, from the **Palette** panel (Buttons), add a **FloatingActionButton** to your activity using drag and drop technique, select the icon shape of this button from the **Pick a Resource** dialog box which you got (**ic_menu_add** or any icon you want) as illustrated in the following figure and click **OK**.



Until now, your activity interface should have the following design:



10- If you open the **activity_main.xml** file in the **Code** mode, you will find the id attribute value of your BottomAppBar is: **bottomAppBar**

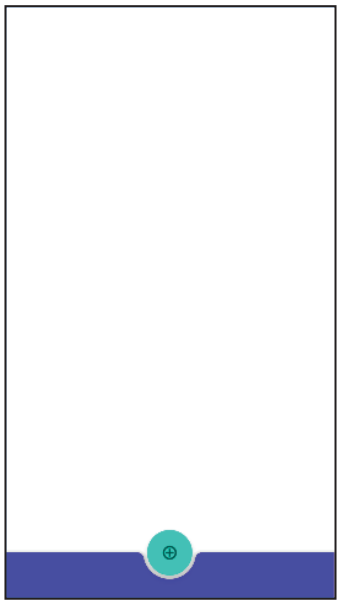
I will use this **BottomAppBar id** attribute value in changing the location of the floating action button. Open the **activity_main.xml** file in the **Code** mode, and add the following attribute to your floating button tag:

```
app:layout_anchor="@id/bottomAppBar"
```

The full XML code is as follows:

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

    <com.google.android.material.bottomappbar.BottomAppBar
        android:id="@+id/bottomAppBar"
        style="@style/Widget.MaterialComponents.BottomAppBar.Colored"
        android:layout_width="409dp"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        tools:layout_editor_absoluteX="1dp"
        tools:layout_editor_absoluteY="674dp"
        app:backgroundTint="@color/purple_500" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/floatingActionButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:clickable="true"
        app:srcCompat="@android:drawable/ic_menu_add"
        app:layout_anchor="@id/bottomAppBar"
        />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Your activity interface should have the following design:



11- Because users of accessibility services such as screen readers, rely on content labels to understand the meaning of elements in an interface, when using an `ImageView`, `ImageButton`, `CheckBox`, or other View widgets that conveys information graphically, use an **`android:contentDescription`** attribute to provide a content label for that view. Therefore, it is recommended to add a description about your floating action button in your activity and you may add the following attribute to your floating action button tag:

```
android:contentDescription="Type here the role of this button"
```

However, to do that in a professional way, open the **strings.xml** file (**app → resource → values**), then add the gray highlighted tag in the following XML code:

```
<resources>
<string name="app_name">Lesson09_BottomAppBar</string>
<string name="floatingBtnInfo"> This button to place a book order
</string>
</resources>
```

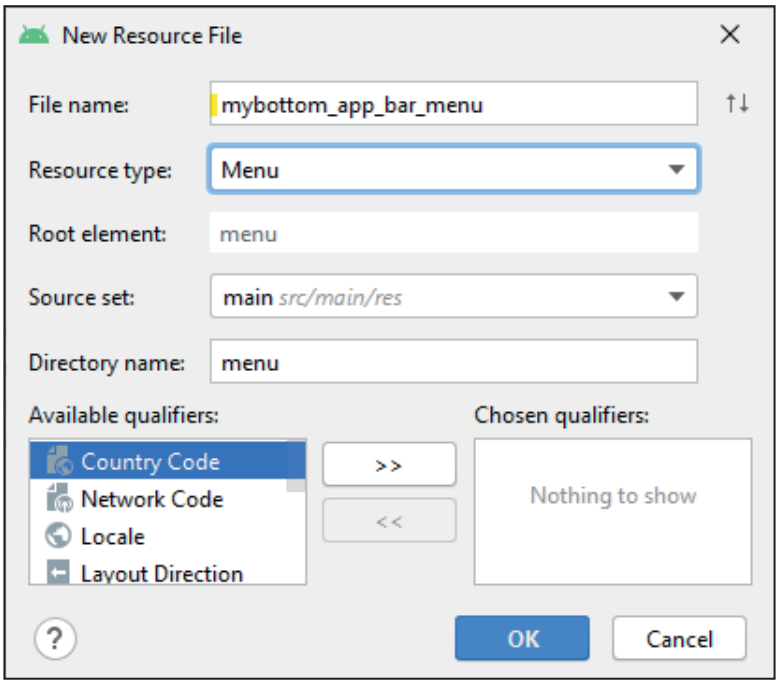
Then, open the **activity_main.xml** file in the **Code** mode and add the following attribute to your floating action button tag:

```
android:contentDescription="@string/floatingBtnInfo"
```

12- Now, to add Android icons to your bottom app bar, you may create a menu including all these icons, then add this menu to your **BottomAppBar** tag in the **activity_main.xml** file.

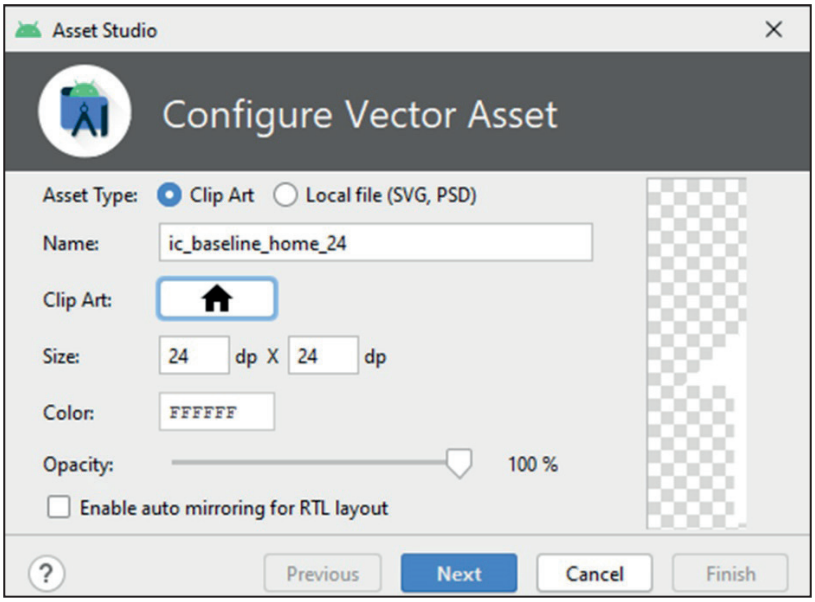
So, let us create this menu by right clicking the **res → New → Android Resource File**

As illustrated in the figure below, enter the **File name**: **`mybottom_app_bar_menu`**, and select the **Resource type**: Menu, then click **OK**

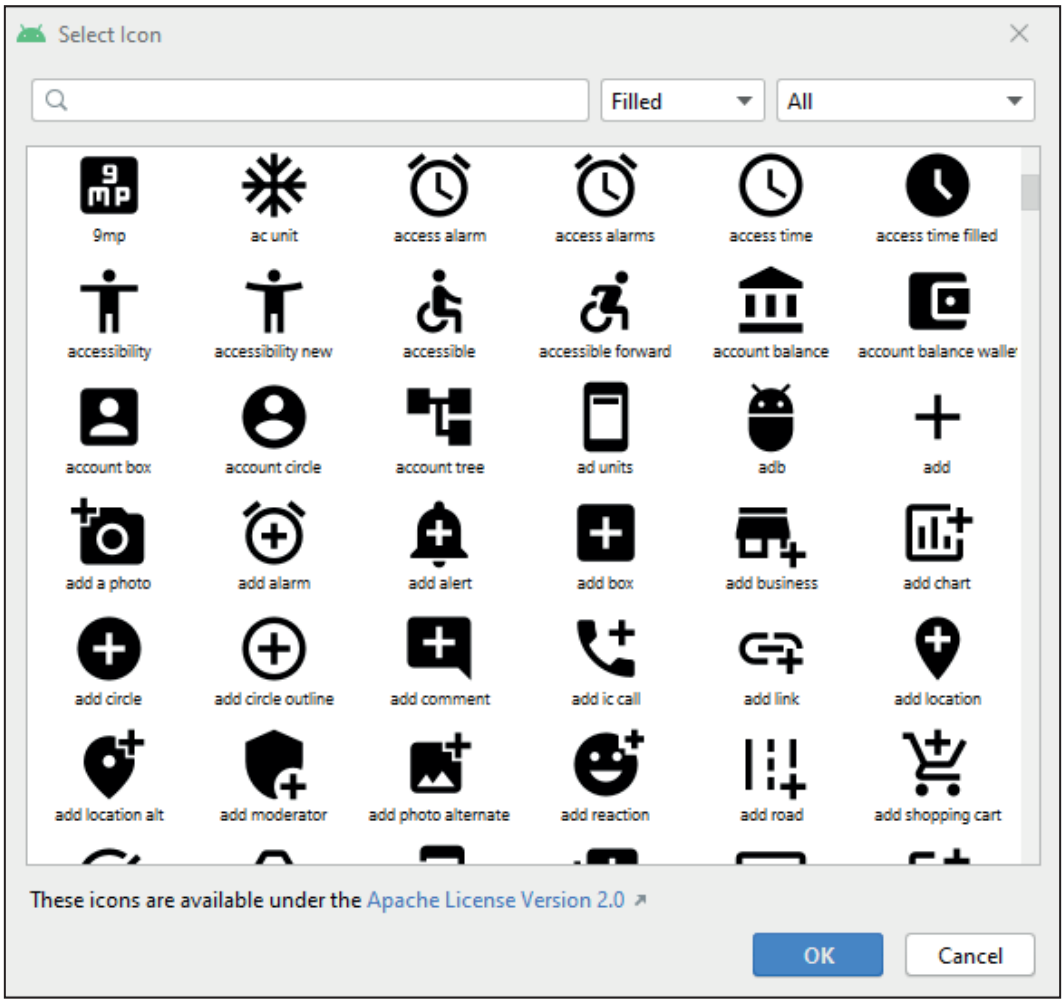


You will get **mybottom_app_bar_menu.xml** file (app → res → menu). This file will include all the items which will display in your bottom app bar as icons.

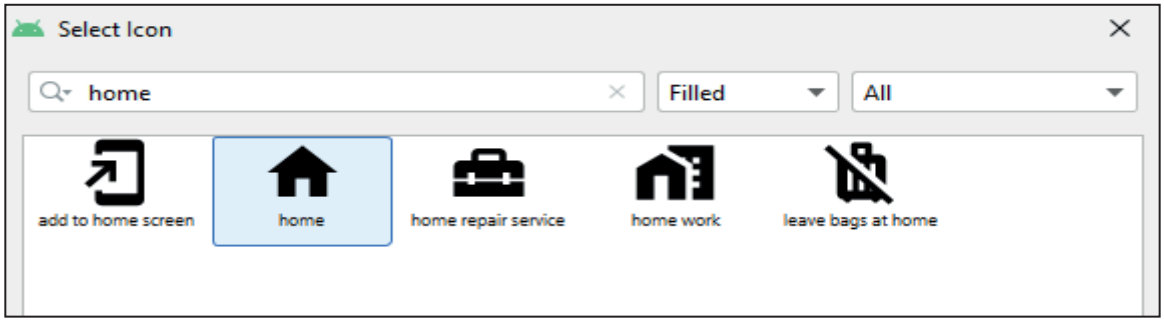
13- You should add the icons which you will use in your app. There are many methods to add icon files to your **drawable** container (app → res → drawable). In this example, we will use the clip art icons which already exist with your Android Studio. To add a home icon, right click the **drawable** directory, click: **New**, then select **Vector Asset**. You should get the following dialog box:



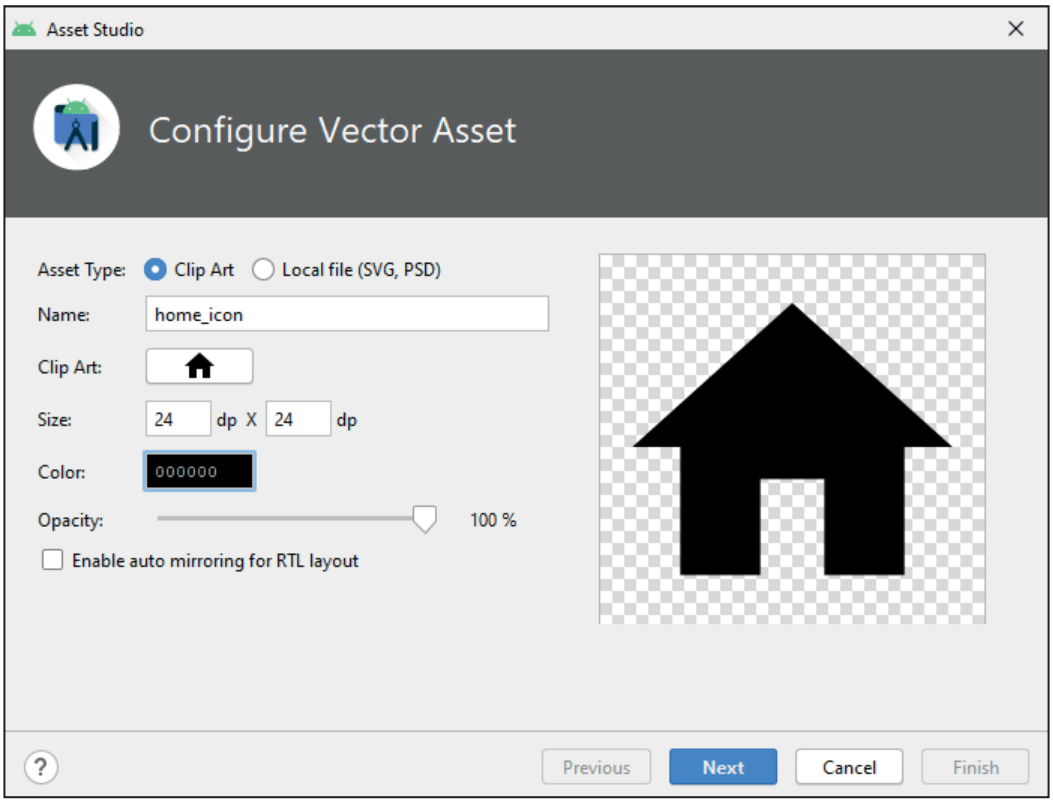
Change the **Name** to **home_icon**, and click the **Clip Art** image to get the following dialog box which includes all the available icons which you may use in your app:



Type **home** in the search area as illustrated in the following figure, then click **OK**.

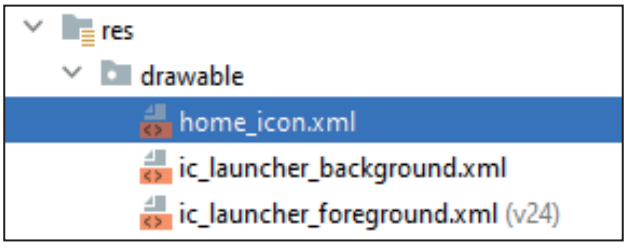


You should get the following:



In this dialog box, click the **Color**: 000000, and change the Hex value to: **FFFFFF** to get the white color (RGB colors), click **Next**, then click **Finish**

As you see in the following figure, you should get an XML file for this icon:

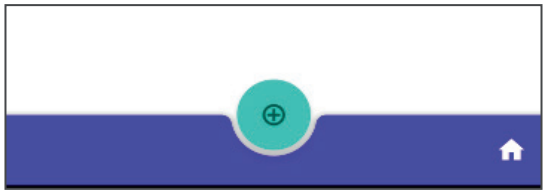


You may repeat the previous steps to add another icon to your app.

14- Now, you should add this new icon as an Item to your menu. To do that, open the **mybottom_app_bar_menu.xml** file in the **Code** mode, and add the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/home"
        android:icon="@drawable/home_icon"
        android:title="home icon"
        app:showAsAction="ifRoom" />
</menu>
```

15- Check your **activity_main.xml** file in the Design mode, you should have the following design:



If you want to change the color, size, or edit this home icon, you may edit the file: **home_icon.xml**

You may link the action of this button with a function by adding the following attribute to the **item** tag in the previous menu xml file:

```
android:onClick="method or function name"
```

In the lab of this lesson, you will do more practice about adding BottomAppBar, and use it in your app navigation.

Recycler View

The **`RecyclerView`** widget is a more advanced, efficient, powerful and flexible version of the ListView which was using before. This widget is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views. You use the **`RecyclerView`** widget when you have data collections whose elements change at runtime based on user action or network events.

The **`RecyclerView`** class simplifies the display and handling of large data sets by providing:

- Layout managers for positioning items.
- Default animations for common item operations, such as removal or addition of items.

The **`RecyclerView`** solves the problem of including a complex layout within a list if you were to use a **`ListView`**.

Example:
In this example, you will create a list of **CardView** widgets, then use it to build a **RecyclerView** list. The steps are as follows:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson09_RecyclerView** for the application name, then click **Finish**.

4- Before you start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson09_RecyclerView.app)** file content and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- To use **CardView** and **RecyclerView** in your project, add the following dependency to your app module: **build.gardle (Module: Lesson09_RecyclerView.app),** and click **Sync Now**

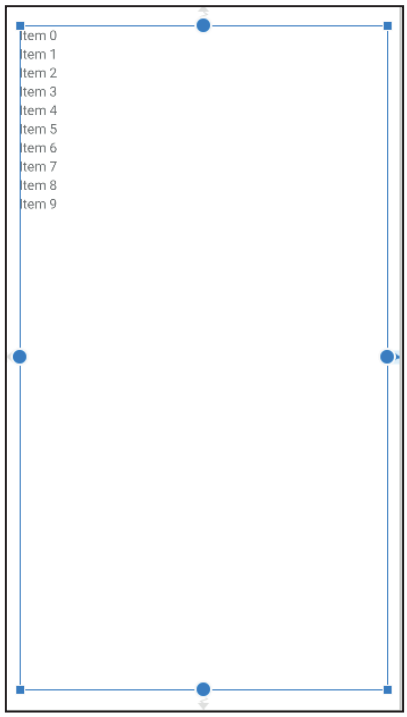
```
implementation("androidx.cardview:cardview:1.0.0")
implementation("androidx.recyclerview:recyclerview:1.2.1")
implementation("androidx.recyclerview:recyclerview-selection:1.1.0")
```

Your project dependency configuration should be similar to the following code (this depending on your SDK version):

```
dependencies {

implementation 'androidx.core:core-ktx:1.6.0'
implementation 'androidx.appcompat:appcompat:1.3.1'
implementation 'com.google.android.material:material:1.4.0'
implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
implementation("androidx.cardview:cardview:1.0.0")
implementation("androidx.recyclerview:recyclerview:1.2.1")
implementation("androidx.recyclerview:recyclerview-selection:1.1.0")
}
```

6- Open the **activity_main.xml** file in the **Design** mode, and then **delete** the “Hello World!” text.

7- From the **Palette** panel (Containers), add the **RecyclerView** widget to your main activity interface using drag and drop technique, change its size to make it look a little smaller and then set its constraints as illustrated in the following figure:



8- Set the RecyclerView **id** : **myRecyclerView**

9- Open the **activity_main.xml** file in the **Code** mode, and the configure your **RecyclerView** scroll bar to be vertical by adding the following XML attribute to your **RecyclerView** tag:

```
android:scrollbars="vertical"
```

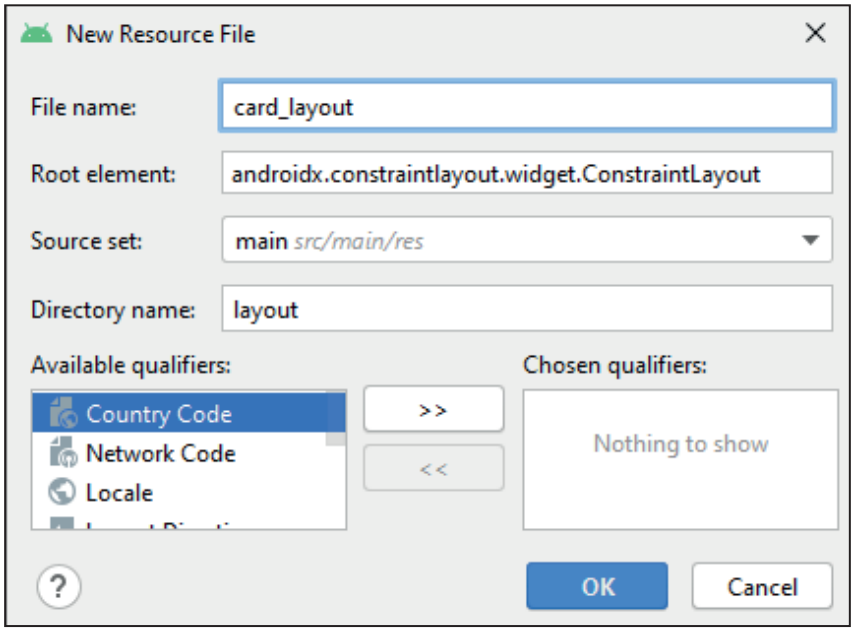
Apps often need to display data in similarly styled containers. These containers are often used in lists to hold each item's information.

The RecyclerView depends on its structure on the **CardView** widget or container. This container is often used in lists to hold each item's information. The system provides the CardView API as an easy way for you to show information inside cards that have a consistent look across the platform.

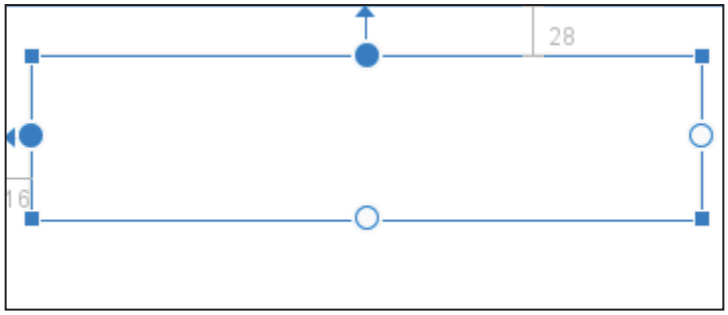
CardView is a Frame Layout with a rounded corner background and shadow. We are going to use this CardView to show information as a card having a round corner background. You will repeat (use again) this CardView to build the rows of the RecyclerView where each row in your **RecycleView** will be a CardView widget with different data.

In the next steps, you are going to configure the integration of these two views (RecyclerView and CardView). You are going to create a card view as a new XML file which will work as a template for the other **CardViews**. These Card Views will be used later to constitute your RecyclerView.

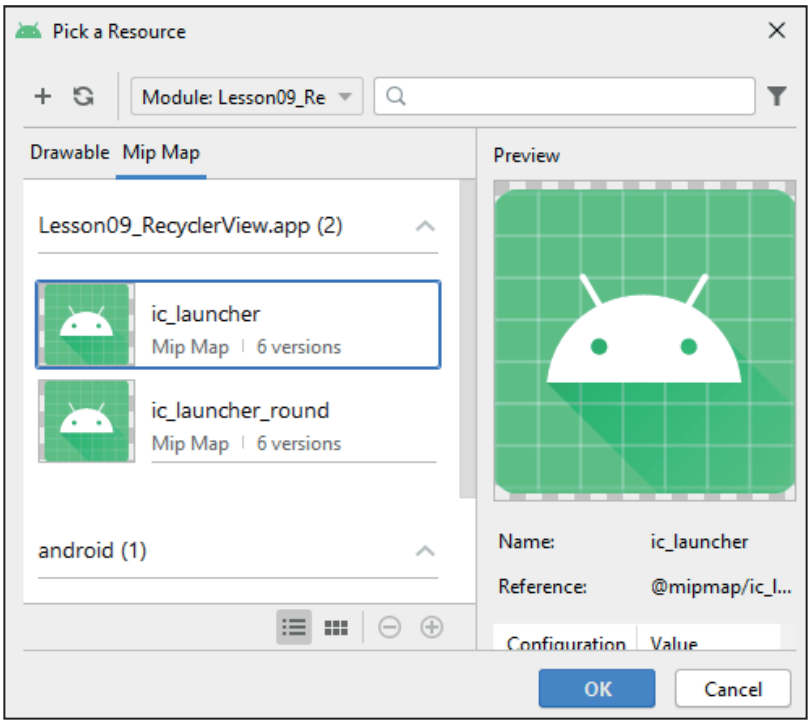
10- Right click **Layout → New → Layout Resource File**
As illustrated in the following figure, type : **card_layout** for the file name, and click **OK**.



11- Open the **card_layou.xml** file in the **Design** mode, then from the **Palette** panel (Containers) drag and drop a **`CardView`** widget to this activity. Set its location and constraints to have a similar design to the following figure:

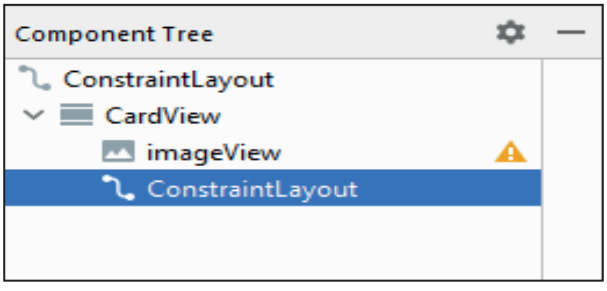


12- From the **Palette** panel (Widgets), drag and drop an **ImageView** widget to your **CardView**, click the **Mip Map** tab, and as illustrated in the figure below, select **ic_laucher** image, then click **OK**.

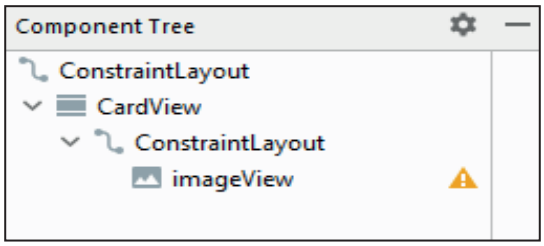


You should have the following design:

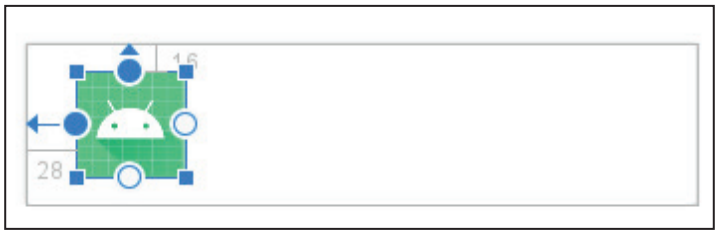
You may notice that it is difficult to move this image inside your CardView. Therefore, insert a **ConstraintLayout** to include this image, and then you can move this image as a child to this ConstraintLayout. To do so, from the Palette panel (Layouts) drag and drop a **ConstraintLayout** widget to your **Component Tree** panel as illustrated in the following figure:



Move your **imageView** inside the **ConstraintLayout**. You should have the following widgets structure:



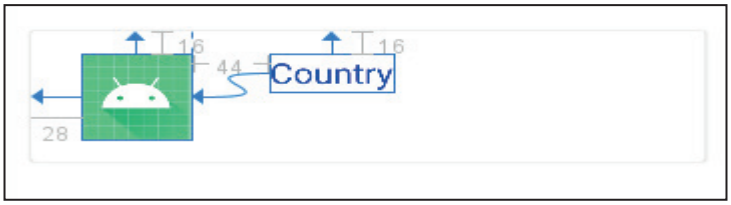
13- Now, you can set your image location and its constraints. Set your image constraints to have the following design:



14- Add a **TextView** widget using drag and drop technique to your CardView, set its constraints and set its attributes values as follows:

id: country_ID	**textSize**: 20sp
text : Country	**textColor**: #0000FF (Blue)

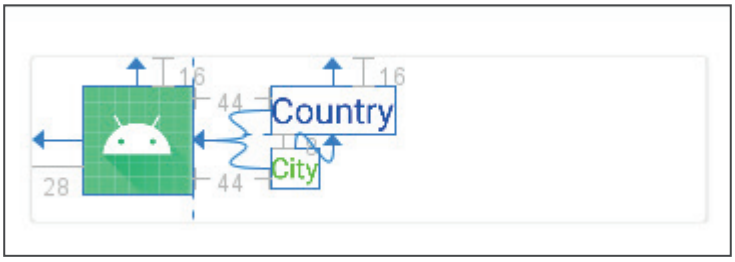
Your **CardView** should look like the following figure:



15- Add a **TextView** widget using drag and drop technique to your CardView, set its constraints and set its attributes values as follows:

id: city_ID	**textSize**: 16sp
text : City	**textColor**: #00FF00 (Green)

Your **CardView** should look like the following figure:

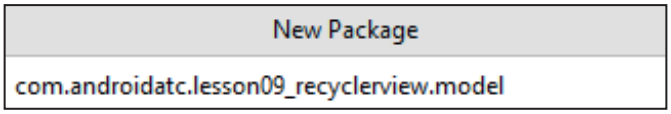


In the next steps, you are going to create a class called "**PlaceAdapter**" to connect your RecyclerView with its data. You are also going to create another class called "**Place**" which works as a data model for the data which will be loaded later to your RecyclerView.

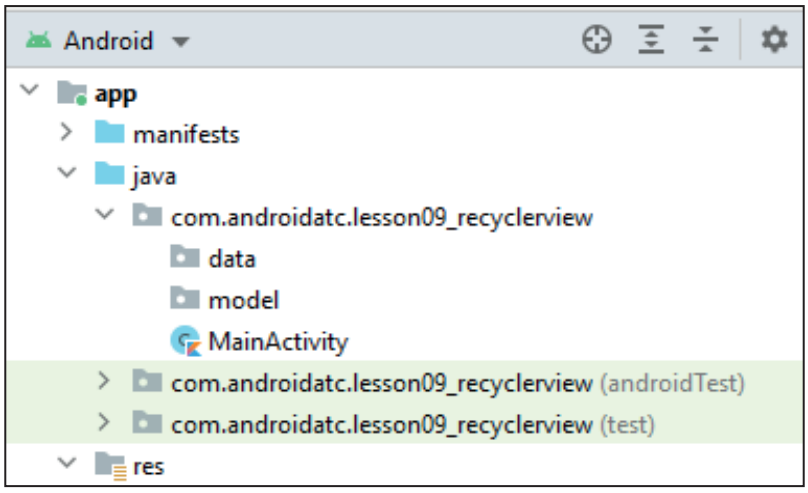
In any web site project, the web designer should create a folder for each data type such as images, database files, animation files, and web pages. Similarly, you should use the same technique in Android Studio to arrange your app files. Therefore, you should create a package for user interface, data, and models. This is what you are going to do in the next steps.

16- Right click → **com.androidatc.lesson09_recyclerview** → **New** → **Package**

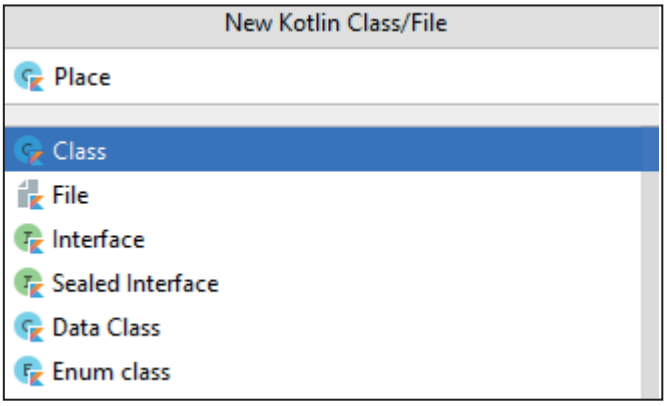
As illustrated in the following figure, just add **model**, then press **Enter**.



17- Repeat the same step as before to create a package for the **data**. You should have the following package structure:



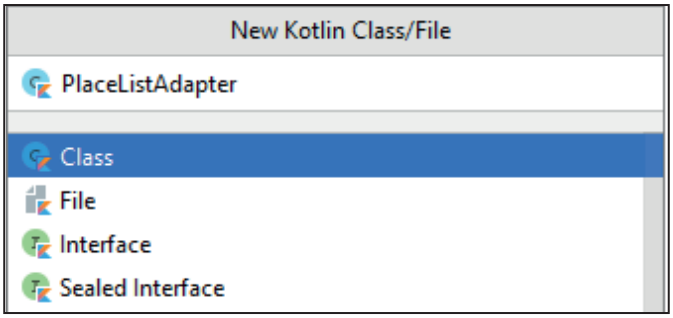
18- Now, create a Kotlin class called **Place** in the **model** package. To do that, right click the **model** → **New** → **Kotlin File/Class**. As illustrated in the dialog box below, type **Place** for the class name, and press **Enter**.



19- Open the **Place** class and add the following code. This code defines a class that models the state and behavior of the data:

```
package com.androidatc.lesson09_recyclerview.model

class Place {
    var CountryName: String?=null
    var CityName :String?=null
}
```

20- To create your **RecyclerView Adapter**, right click the **data → New → Kotlin File/Class**. Type the class name: **PlaceListAdapter** as illustrated in the figure below, and press **Enter**.



21- Open the **PlaceListAdapter** and write the following code:

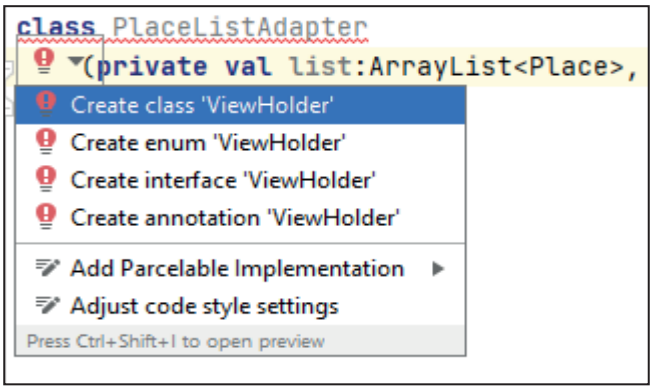
```
package com.androidatc.lesson09_recyclerview.data


class PlaceListAdapter(private val list: ArrayList<Place>, private
val context: Context) :
    RecyclerView.Adapter<PlaceListAdapter.ViewHolder>() {


    }
```

Double click **Place**, click the red pop-up lamp, and select **Import**.

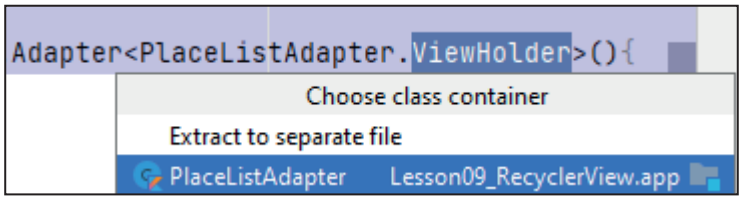
Double click **Context**, click the red pop-up lamp, and select **Import**.

Double click **RecyclerView**, click the red pop-up lamp, and select **Import**.

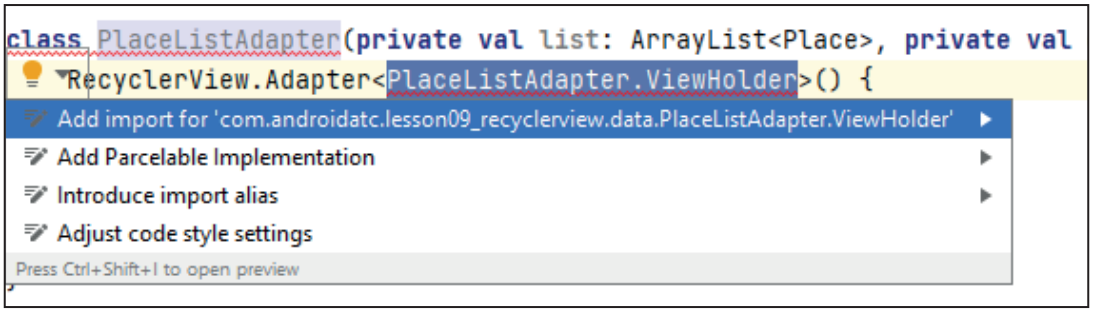
Double click **ViewHolder**, click the red pop-up lamp, and as illustrated in the figure below, select **Create class ViewHolder**.



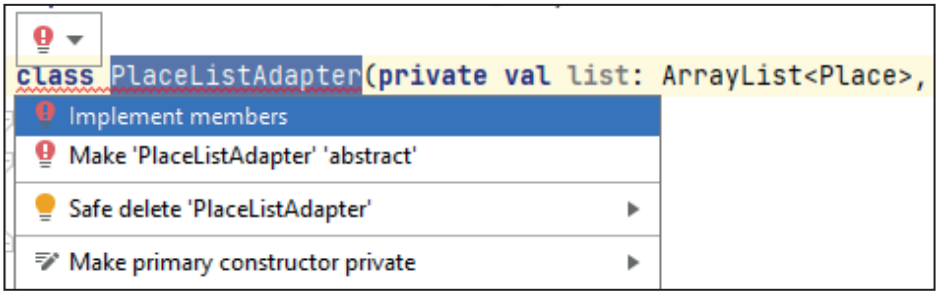
Then as illustrated in the following figure, select **PlaceListAdapter**, and press **Enter**



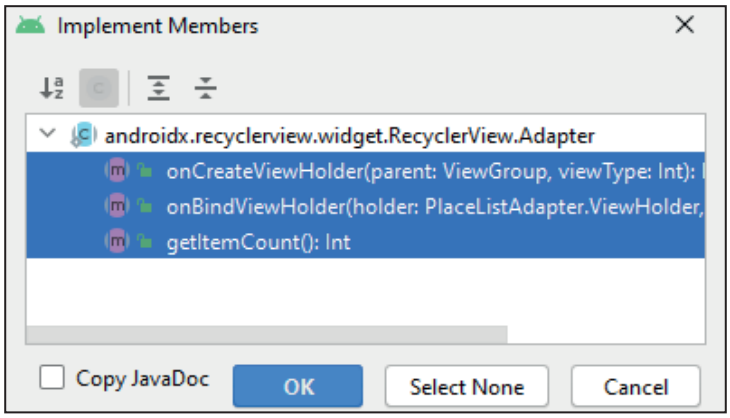
22- Double click: **`PlaceListAdapter.ViewHolder`**, click the yellow pop-up lamp, and as illustrated in the figure below, select: **`Add import for 'com.androidatc.lesson09_recycleview.data.PlaceListAdapter.ViewHolder'`**



23- Double click: **`PlaceListAdapter`**, click the red pop-up lamp, and select **Implement members**



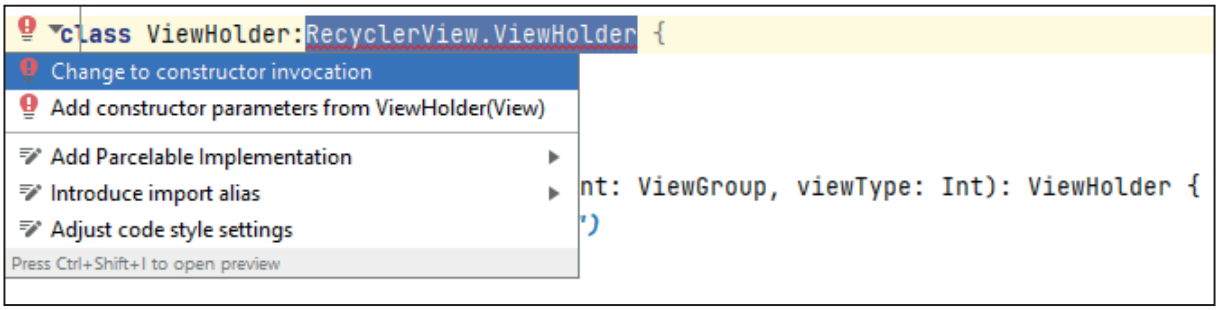
24- As illustrated I the figure below, press and hold **SHIFT** key, select all these classes, and click **OK**



25- Configure the **`ViewHolder`** class to inherit its properties from the **`RecyclerView`** class by adding the following configuration to the **`ViewHolder`** class:

```
class ViewHolder:RecyclerView.ViewHolder {
}
```

26- Double click: **`RecyclerView.ViewHolder`**, click the red pop-up lamp as illustrated in the figure below, and select: **Change to constructor invocation**



27- Configure the **`ViewHolder`** class as follows:

```
class ViewHolder (itemView:View): RecyclerView.
ViewHolder(itemView) {

}
```

Double click **`View`** in the code above, click the red pop-up lamp, and select **Import**.
The full code of the **PlaceListAdapter** class is as follows:

```
package com.androidatc.lesson09_recyclerview.data

import android.content.Context
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.androidatc.lesson09_recyclerview.data.PlaceListAdapter.
ViewHolder
import com.androidatc.lesson09_recyclerview.model.Place
```

```
class PlaceListAdapter(private val list: ArrayList<Place>, private val
context: Context) : RecyclerView.Adapter<ViewHolder>() {

 class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

}
```

In the next steps, you are going to configure the following three functions of the RecyclerView adapter in the PlaceListAdapter class:

- onBindViewHolder
- onCreateViewHolder
- getItemCount

28- Start with **`getItemCount`** function and add: **`return list.size`** to its configuration as follows:

```
override fun getItemCount(): Int {
    return list.size
}
```

29- Configure the function **`onCreateViewHolder`** which will create a view from the **card_layout.xml** file as follows:

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {

val view = LayoutInflater.from(context).inflate(R.layout.card_
layout, parent, false)

    return ViewHolder(view)
}
```

Double click **LayoutInflater** in the code above, click the red pop-up lamp, and select **Import**.

Double click **R** in the code above, click the red pop-up lamp, and select **Import**.

30- Before you configure the function **onBindViewHolder**, add the following function to **ViewHolder** class:

```
class ViewHolder(itemView: View) : RecyclerView.
ViewHolder(itemView) {

fun bindItem(place: Place) {
var country: TextView = itemView.findViewById(R.id.country_ID) as
TextView
var city: TextView = itemView.findViewById(R.id.city_ID) as
TextView

     country.text = place.CountryName
     city.text = place.CityName

    }
}
```

Double click **TextView** in the code above, click the red pop-up lamp, and select **Import**.

31- Now, configure the third function **onBindViewHolder** by adding the following code:

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bindItem(list[position])
}
```

The full code of the **PlaceListAdapter** file is as follows:

```
package com.androidatc.lesson09_recyclerview.data

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.androidatc.lesson09_recyclerview.R
import com.androidatc.lesson09_recyclerview.data.PlaceListAdapter.
ViewHolder
import com.androidatc.lesson09_recyclerview.model.Place

class PlaceListAdapter(private val list: ArrayList<Place>, private
val context: Context) : RecyclerView.Adapter<ViewHolder>() {

class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
```

```
fun bindItem(place: Place) {
var country: TextView = itemView.findViewById(R.id.country_ID) as
TextView
var city: TextView = itemView.findViewById(R.id.city_ID) as TextView

    country.text = place.CountryName
    city.text = place.CityName

        }
    }


    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): ViewHolder {
        val view = LayoutInflater.from(context).inflate(R.layout.
card_layout, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position:
Int) {
        holder.bindItem(list[position])
    }


    override fun getItemCount(): Int {
        return list.size
    }
}
```

Now, your **PlaceListAdapter** class is ready. In the next step, you are going to use this class inside the **MainActivity** file.

32- Open the **MainActivity** file and add the following variables, as illustrated in the gray highlighted color:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var adapter:PlaceListAdapter?=null
        var countryList:ArrayList<Place>?=null
        var layoutManager:RecyclerView.LayoutManager?=null

    }
}
```

Double click **PlaceListAdapter** in the code above, click the red pop-up lamp, and select **Import**.

Double click **Place** in the code above, click the red pop-up lamp, and select **Import**.

Double click **RecyclerView** in the code above, click the red pop-up lamp, and select **Import**.

33- In this step, you are going to configure these variables, as illustrated in the gray highlighted color:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

         var adapter: PlaceListAdapter?=null
         var countryList:ArrayList<Place>?=null
         var layoutManager: RecyclerView.LayoutManager?=null

        countryList=ArrayList<Place>()
        layoutManager= LinearLayoutManager(this)
        adapter=PlaceListAdapter(countryList,this)

    }
}
```

Double click **LinearLayoutManager** in the code above, click the red pop-up lamp, and select **Import**.

34- Now, in the **MainActivity** file, setup the **RecyclerView** (initialize RecyclerView) by adding the following code, as illustrated in the gray highlighted color below:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

         var adapter: PlaceListAdapter?=null
         var countryList:ArrayList<Place>?=null
         var layoutManager: RecyclerView.LayoutManager?=null

        countryList=ArrayList<Place>()
        layoutManager= LinearLayoutManager(this)
        adapter=PlaceListAdapter(countryList,this)

        myRecyclerView.layoutManager=layoutManager
        myRecyclerView.adapter=adapter
    }
}
```

Double click **`myRecyclerView`** in the code above, click the red pop-up lamp and select **Import**.

Adding Data to your RecyclerView:

35- Now you can load your data to the **RecyclerView** which has two arrays; the first one includes 15 country names and the second one includes the capital city of each country in the first array. You are going to use "**For loop**" to place this data 15 times. Add the following code to **MainActivity** file:

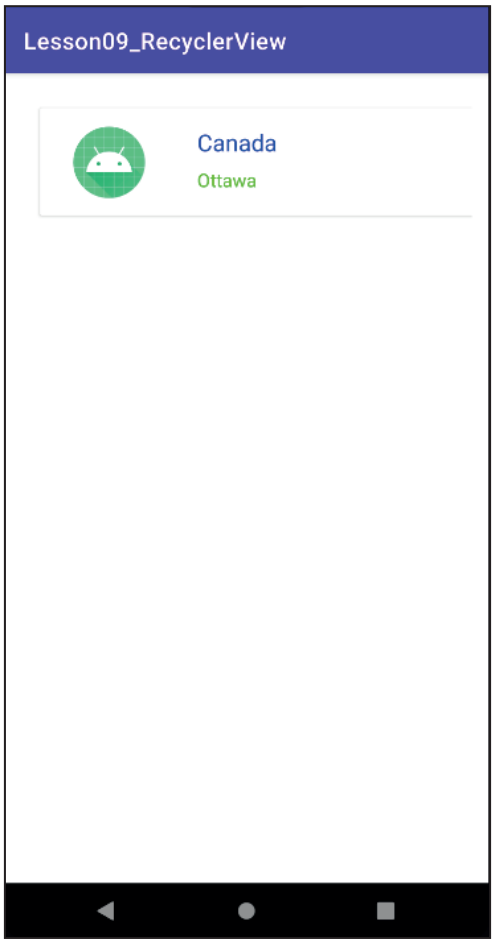
```
val countryNameList:Array<String> =
arrayOf("Canada","USA","Mexico","Columbia","Malaysia","Singapore",
"Turkey","Nicaragua","India","Italy","Tunisia","Chile","Argentina",
"Spain","Peru")

val citiesNameList:Array<String> =
arrayOf("Ottawa","Washington,D.C." ,"Mexico City","Bogotá","Kuala
Lumpur","Singapore" ,"Ankara","Managua","New
Delhi","Rome","Tunis","Santiago","Buenos Aires","Madrid","Lima")

for (i in 0..14) {
    val place=Place()
    place.CountryName = countryNameList[i]
    place.CityName = citiesNameList[i]
    countryList.add(place)
}

adapter.notifyDataSetChanged()
```

The full code of the **MainActivity** file is as follows:

```
package com.androidatc.lesson09_recyclerview

import android.annotation.SuppressLint
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.androidatc.lesson09_recyclerview.data.PlaceListAdapter
import com.androidatc.lesson09_recyclerview.model.Place
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    @SuppressLint("NotifyDataSetChanged")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
        var adapter: PlaceListAdapter?=null
        var countryList:ArrayList<Place>?=null
        var layoutManager: RecyclerView.LayoutManager?=null

        countryList=ArrayList<Place>()
        layoutManager= LinearLayoutManager(this)
        adapter=PlaceListAdapter(countryList,this)

        myRecyclerView.layoutManager=layoutManager
        myRecyclerView.adapter=adapter

       val countryNameList:Array<String> =
arrayOf("Canada","USA","Mexico","Columbia","Malaysia","Singapore",
"Turkey","Nicaragua","India","Italy","Tunisia","Chile","Argentina",
"Spain","Peru")

        val citiesNameList:Array<String> =
arrayOf("Ottawa","Washington,D.C." ,"Mexico City","Bogota","Kuala
Lumpur","Singapore" ,"Ankara","Managua","New
Delhi","Rome","Tunis","Santiago","Buenos Aires","Madrid","Lima")

        for (i in 0..14) {
            val place=Place()
            place.CountryName = countryNameList[i]
            place.CityName = citiesNameList[i]
            countryList.add(place)
        }

        adapter.notifyDataSetChanged()

    }
}
```

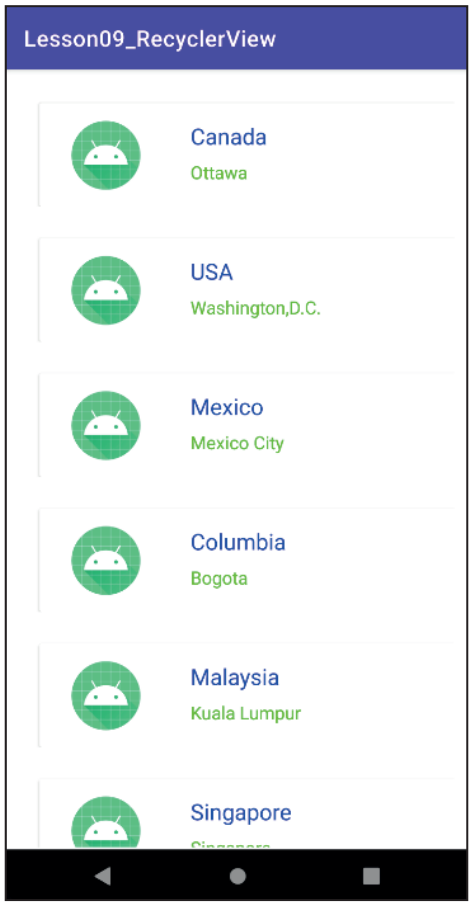
36- Stop your app, then Run it again. You should get the run result below. Scroll down to see the remaining list items.



37- To solve this gap between the **CardView**s, open the **card_layout.xml** file in the **Code** mode, and then replace **android:layout_height="match_parent"** with **android:layout_height="wrap_content"** as illustrated in the following code:

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
```

38- **Stop** your app, and then **Run** it again. You should get the following run result:



39- To set your CardViews with a corner radius shape, open the **card_layout.xml** file in the **Code** mode, and add the following attribute to your **CardView** tag:

```
app:cardCornerRadius="10dp"
```

If you want to add a shadow to your cards, add the following attribute to your **CardView** tag:

```
app:cardElevation="10dp"
```

To avoid any overlap between your **CardView** cards in your **RecyclerView**, add the following attribute to your **CardView** tag:

```
app:cardPreventCornerOverlap="true"
```

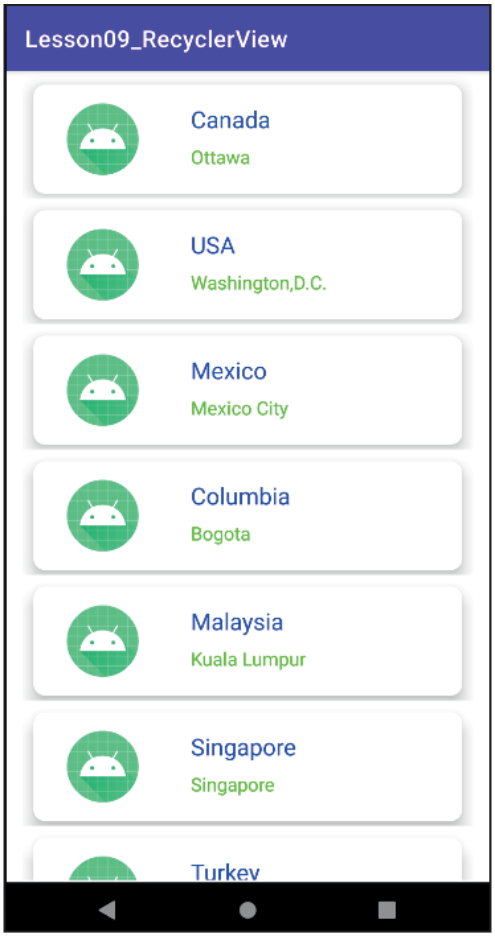
To make a space between your **RecyclerView** cards, add the following attribute to your **CardView** tag:

```
android:layout_marginBottom="4dp"
```

The following is the code of the **CardView** tag in the **card_layout.xml** file:

```
<androidx.cardview.widget.CardView
    android:layout_marginTop="10dp"
    android:layout_width="365dp"
    android:layout_height="93dp"
    app:cardCornerRadius="10dp"
    app:cardElevation="10dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:cardPreventCornerOverlap="true"
    android:layout_marginBottom="4dp">
```

If you added the changes above to your **card_layout.xml** file, stop your app, then run it again. You should get the following interface:

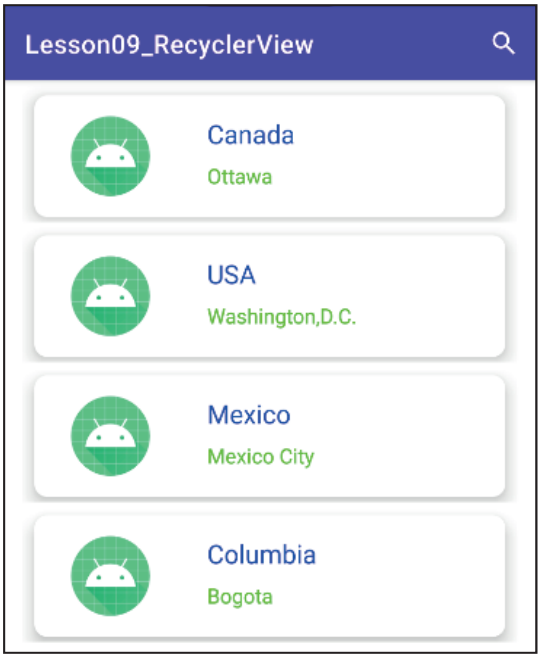


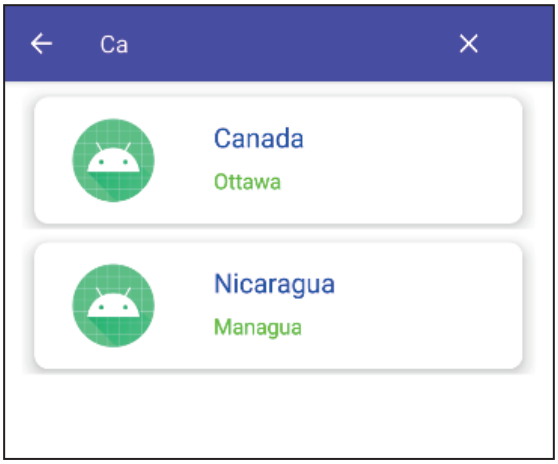
You may also use an array of images to add them to each CardView.

Because this app has a lot of steps and code to type, you may find this project code in the Lab folder. If you get any error in your code, just open this app in your Android Studio and check the code.

SearchView

SearchView is a widget that provides a user interface for the user to enter a search query and submit a request to a search provider. It shows a list of query suggestions or results, if available, and allows the user to pick a suggestion or result to launch into. Also, you may use this class to filter the content of your RecyclerView content.



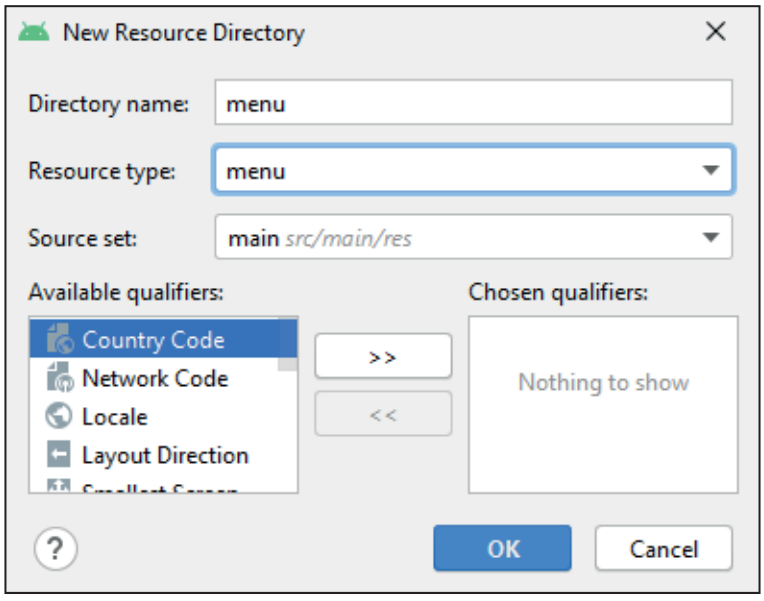


Example:

In this example you will continue using the same **RecyclerView** Android project code and add a SearchView widget (Class) to apply a filter on its contents. The steps as are follows:

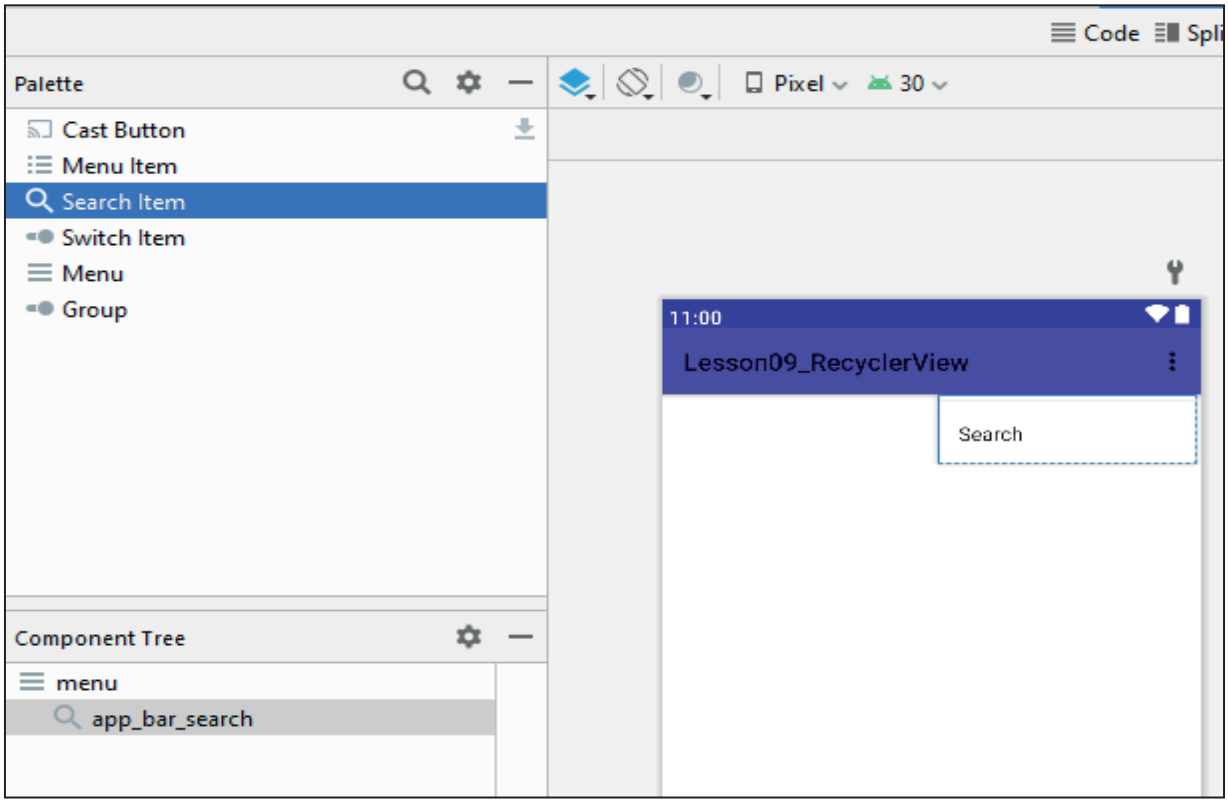
1- First, create a resource directory is called menu. This directory will include your **SearchView** widget (class). To do that, right click the **res** directory → **New** → **Android Resource Directory**

In the **New Resource Directory** dialog box and as illustrated in the dialog box below, type **menu** for the **Directory name**, select **menu** form the **Resource type**, and click **OK**



2- You should have a resource XML file (Menu Resource File), then configure your **SearchView** widget as an item in this resource file. To create this resource file, right click the **menu** directory → **New** → **Menu Resource File**. Type **menu** for the **File name**, and click **OK**.

3- Open the **menu.xml** file in the **Design** mode. From the **Palette** panel and as illustrated in the figure below, add a **Search Item** widget using drag and drop technique.



4- Open the **menu.xml** file in the **Code** mode, change the **id** attribute value for the **item** tag to: **search**.

Also, add the following attribute to your **item** tag:

```
app:showAsAction="always|collapseActionView"
```

5- Replace the attribute:

```
app:actionViewClass="android.widget.SearchView"
```

With:

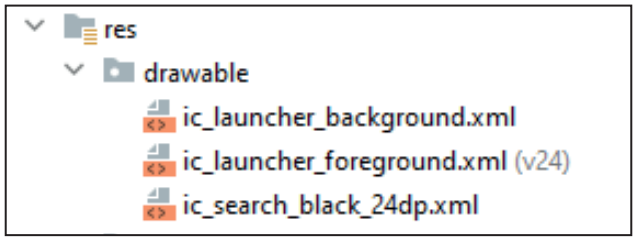
```
app:actionViewClass="androidx.appcompat.widget.SearchView"
```

This step is very important.

The **menu.xml** file should have the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/search"
        android:icon="@drawable/ic_search_black_24dp"
        android:title="Search"
        app:actionViewClass="androidx.appcompat.widget.SearchView"
        app:showAsAction="always|collapseActionView"/>
</menu>
```

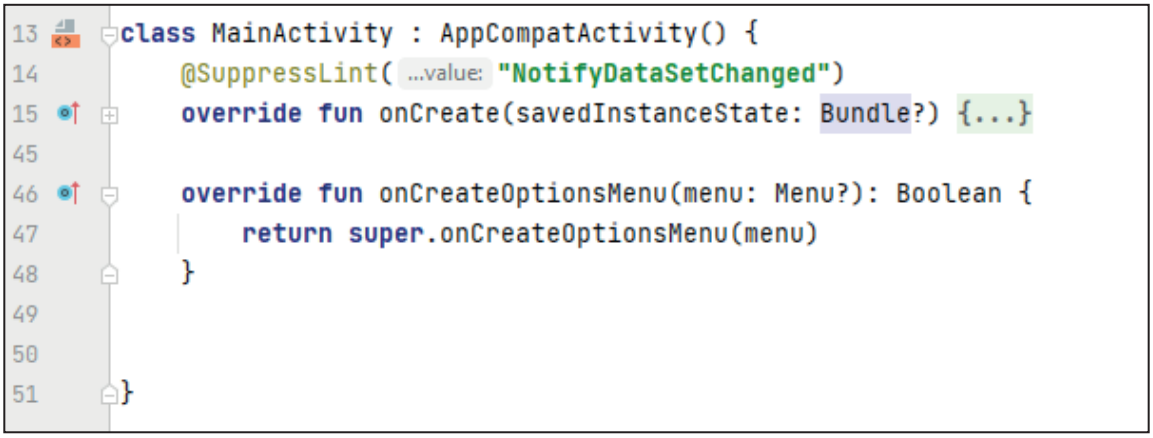
Note that an icon image has been added to your Android app files automatically. Check your **drawable** directory, and as illustrated in the figure below, the **ic_search_black_24dp.xml** icon configuration XML file has been added here:



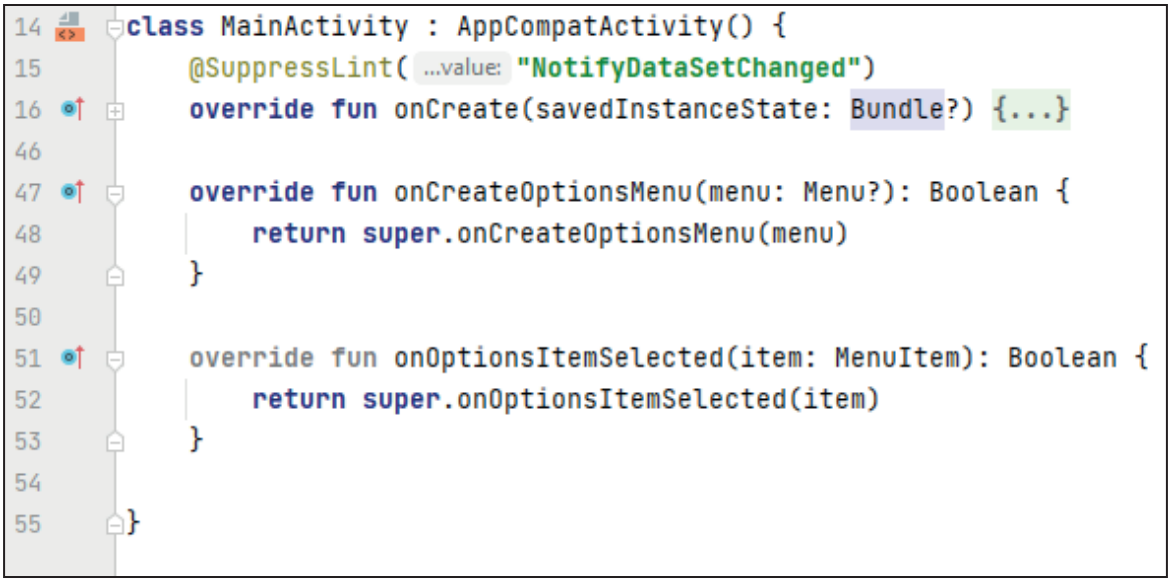
5- To add the previous search menu to your **MainActivity** file, open the **MainActivity** file, and as illustrated in the figure below, click the (-) sign in your Android Studio to hide the code of the **OveronCreate(savedInstanceState: Bundle?) .** This action increases your focus on the new code.

The figure below displays where the (-) sign is which will hide the previous code, and where you should add the new code in the next steps.

6- Add the function: **`onCreateOptionsMenu`** to your code illustrated in the following figure:



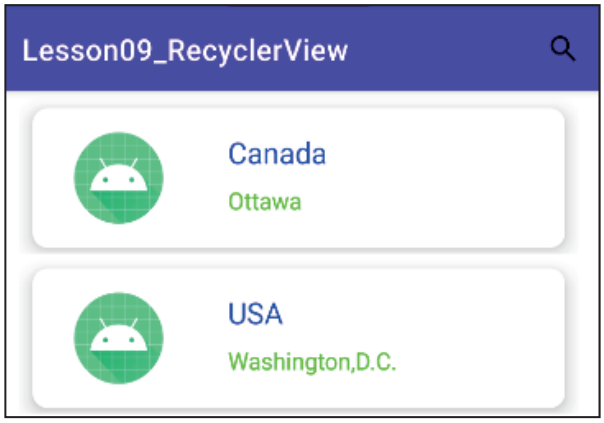
7- Add the **`onOptionsItemSelected`** function to your code as illustrated in the figure below:



8- Add the following code to your **onCreateOptionsMenu** function. This code will add the **menu.xml** file content to your **MainActivity** interface.

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {

    menuInflater.inflate(R.menu.menu,menu)
    val menuItem=menu!!.findItem(R.id.search)

    return super.onCreateOptionsMenu(menu)
}
```

Just to check if the search area has been created in your app, **Stop** then **Run** your app. You should get a search area at the top right of your app interface as illustrated in the figure below:



As you see the search icon, it is black in color. To change its color to white, open the **ic_search_black_24dp.xml** file (**res → drawable**) in the Code mode and change the attribute value:

`android:fillColor="#FF000000"` To: `android:fillColor="#FFFFFF"`

9- Open the **MainActivity** file, click the previous (+) sign to display the hidden code again, and as illustrated in the following figure, select the configuration of the **countryList** variable, right click, select **Cut**, then paste this variable configuration at the top of the **onCreate** function. This makes this variable available to use with all other functions in this file.

```
class MainActivity : AppCompatActivity() {
    @SuppressLint( ...value: "NotifyDataSetChanged")

                                                    Paste
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

         var adapter: PlaceListAdapter?=null
         var countryList:ArrayList<Place>?=null          Cut
         var layoutManager: RecyclerView.LayoutManager?=null

        countryList=ArrayList<Place>()
        layoutManager= LinearLayoutManager( context: this)
        adapter=PlaceListAdapter(countryList, context: this)
```

You will get a red underline with the **countryList** in your code in two different locations. Just add: **!!** to the variable which has a red underline only. It should be: **countryList!!**

10- Add the **`displayList`** variable to your code as illustrated in the following figure:

```
class MainActivity : AppCompatActivity() {
    @SuppressLint( ...value: "NotifyDataSetChanged")

    var countryList:ArrayList<Place>?=null
    var displayList =ArrayList<Place>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

11- Replace the **`countryList`** in the **`adapter`** variable configuration as illustrated in the figure below:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

     var adapter: PlaceListAdapter?=null

     var layoutManager: RecyclerView.LayoutManager?=null

    countryList=ArrayList<Place>()
    layoutManager= LinearLayoutManager( context: this)

    adapter=PlaceListAdapter(countryList!!, context: this)

    myRecyclerView.layoutManager=layoutManager
    myRecyclerView.adapter=adapter
```

With: **displayList** as illustrated in the figure below:

```
countryList=ArrayList<Place>()
layoutManager= LinearLayoutManager( context: this)

adapter=PlaceListAdapter(displayList, context: this)

myRecyclerView.layoutManager=layoutManager
myRecyclerView.adapter=adapter
```

12- At the end of the **onCreate** function and as illustrated in the following figure, add the following configuration:

```
displayList.addAll(countryList!!)
```

The configuration as illustrated in the following screenshot:

```
        for (i in 0..14) {
            val place=Place()
            place.CountryName = countryNameList[i]
            place.CityName = citiesNameList[i]
            countryList!!.add(place)
        }

        displayList.addAll(countryList!!)

        adapter.notifyDataSetChanged()

    }
```

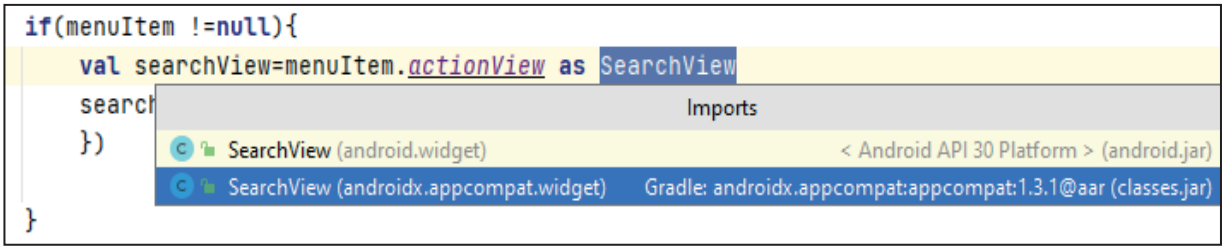
13- Add the following **if** configuration to the: **`onCreateOptionsMenu`** function

```
if(menuItem !=null){
    val searchView=menuItem.actionView as SearchView
    searchView.setOnQueryTextListener(object:SearchView.OnQueryTextListener{
     })

}
```

This function should have the following configuration:

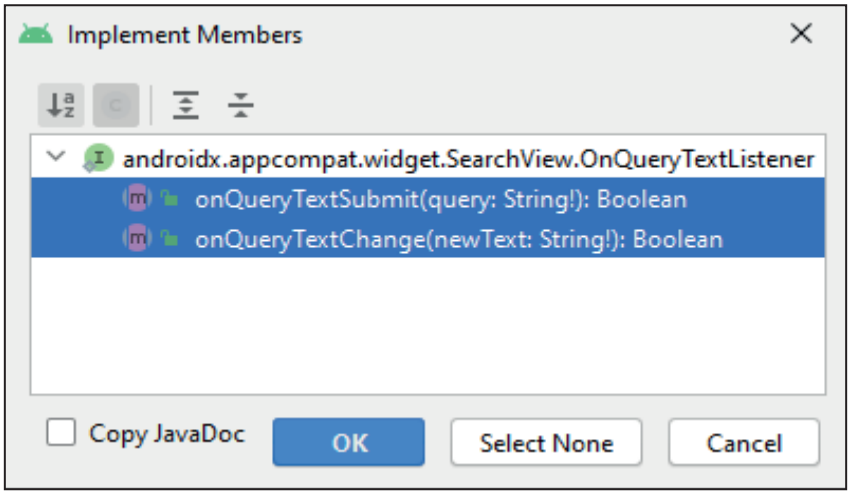
```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu,menu)
    val menuItem=menu!!.findItem(R.id.search)

    if(menuItem !=null){
        val searchView=menuItem.actionView as SearchView
        searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener{

        })

    }


    return super.onCreateOptionsMenu(menu)
}
```

In the code above, double click the **SearchView**, click the red pop-up lamp, and select **Import** as illustrated in the following figure:



Then, double click the **object**, click the red pop-up lamp, and select **Implement members**

As illustrated in the figure below, you should get a dialog box including two methods. Press and hold **Shift** key, select the two methods and click **OK**



14- For the two methods which you have added in the previous step, replace the following expression:

```
TODO("Not yet implemented")
```

with

```
return true
```

As illustrated in the figure below:

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu,menu)
    val menuItem=menu!!.findItem(R.id.search)

    if(menuItem !=null){
        val searchView=menuItem.actionView as SearchView
        searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener{
            override fun onQueryTextSubmit(query: String?): Boolean {
                return true
            }

            override fun onQueryTextChange(newText: String?): Boolean {
                return true
            }

        })
```

15- Configure the function: **onQueryTextChange** which has been added in the previous step (step number 13) with the following configuration:

```
override fun onQueryTextChange(newText: String): Boolean {

if(newText.isNotEmpty()){
        displayList.clear()
        var search = newText.lowercase(Locale.getDefault())
        countryList!!.forEach {

  if (it.CountryName!!.lowercase(Locale.getDefault()).contains(search))
{
                displayList.add(it)
            }
        }
        myRecyclerView.adapter!!.notifyDataSetChanged()
    }

    else{
        displayList.clear()
        displayList.addAll(countryList!!)
        myRecyclerView.adapter!!.notifyDataSetChanged()
    }
    return true
}
```

Double click **Locale**, click the red pop-up lamp, and select **Import**

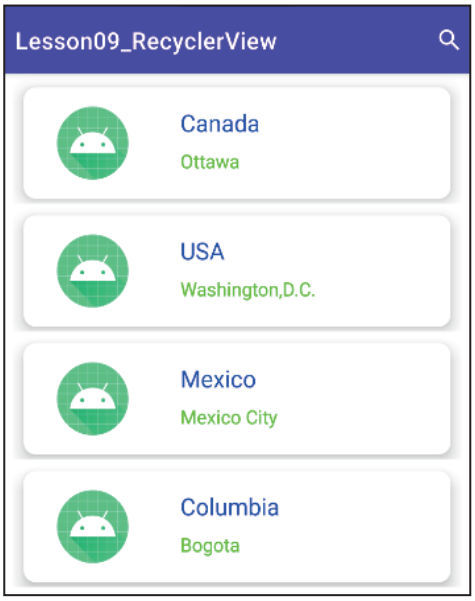
16- Add the following code at the end of the `onCreate(savedInstanceState: Bundle?)` code:

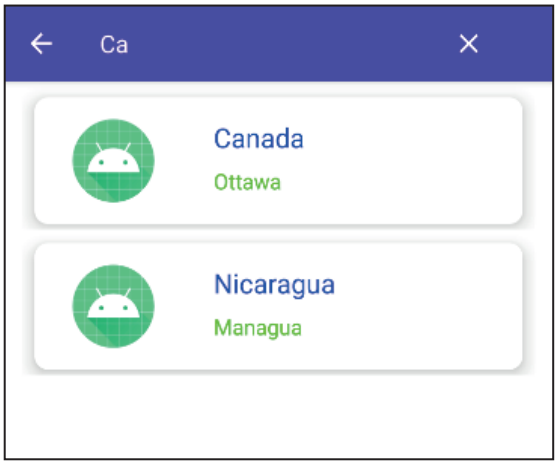
```
displayList.addAll(countryList!!)
```

As illustrated in the following figure:

```
    val countryNameList:Array<String> = arrayOf("Canada","USA","Mexico

    val citiesNameList:Array<String> = arrayOf("Ottawa","Washington,D.

    for (i in 0..14) {
        val place=Place()
        place.CountryName = countryNameList[i]
        place.CityName = citiesNameList[i]
        countryList!!.add(place)
    }

    displayList.addAll(countryList!!)  <-- Here

    adapter.notifyDataSetChanged()

}
```

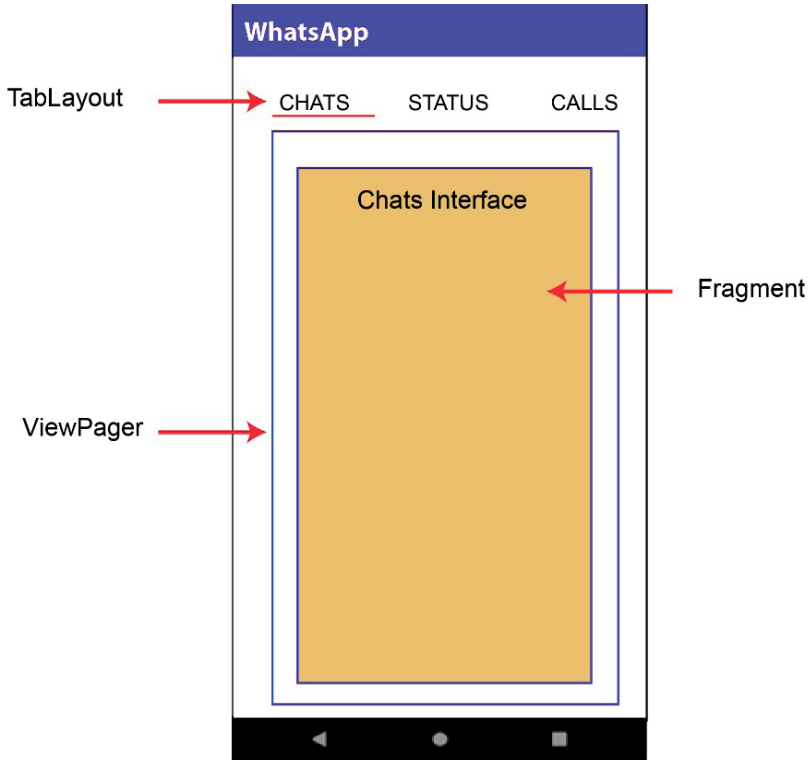
17- Now, your app is ready for test. Stop, then Run your app. Type **Ca** in the search area. You should get all the countries' names containing **ca**. To get full view of your search result, hide your emulator keyboard panel. The run results are as follows:





TabLayout and ViewPager

If the content of your app can be divided into two or more different categories or roles, you may select to display your app contents in two or more different tabs as illustrated in the following figure:



The **`TabLayout`** widget is the main container or the parent for these tabs and it provides a horizontal layout to display tabs and each tab content can be represented by a fragment.

A ***Fragment*** is a piece of an application's user interface or behavior and has its own layout that can be placed in an activity.

Example:
In this example, you will first create an Android app using the **TabLayout** widget in its navigation technique, this is similar to the figure above including **TabLayout** widget, then you will add a ViewPager2 widget which will include three fragments. The main role of each of these fragments is to represent the content of each TabLayout tab of your app.

The steps are as follows:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

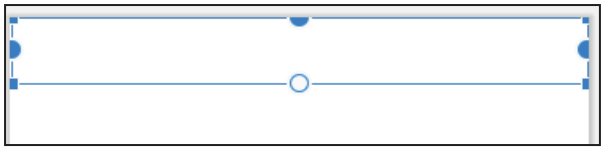
3- Type: **Lesson09_TabLayout** for the application name, then click **Finish**.

4- Before you start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson09_TabLayout)** file and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**
And click the **Sync Now**. The configuration should be as follows:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** file in Design mode, and then **delete** the "Hello World!" text.

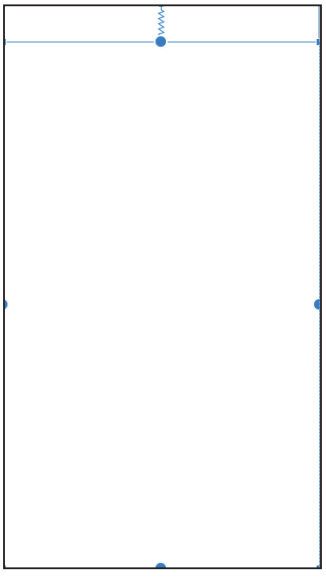
6- From the **Palette** panel (Containers), add a **TabLayout** widget to your activity interface using drag and drop technique, set its constrains and set its location at the top of your interface as illustrated in the following figure:



Set the **id** attribute value for this **TabLayout** widget with: **tabLayout**

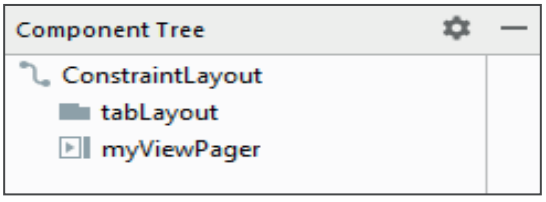
7- The content of TabLayout tabs should be displayed in the **activity_main.xml** file. You need to add a **ViewPager2** widget to your **activity_main.xml** file. This **ViewPager2** widget allows you to display the tab contents and navigate between them with a horizontal finger swipe.

From the **Palette** panel (Containers), add a **ViewPager2** widget to your activity interface using the drag and drop technique, set its constrains and set its location as illustrated in the following figure:



Set the **id** attribute value for this **ViewPager2** widget with: **myViewPager**

You should have the following component tree structure:

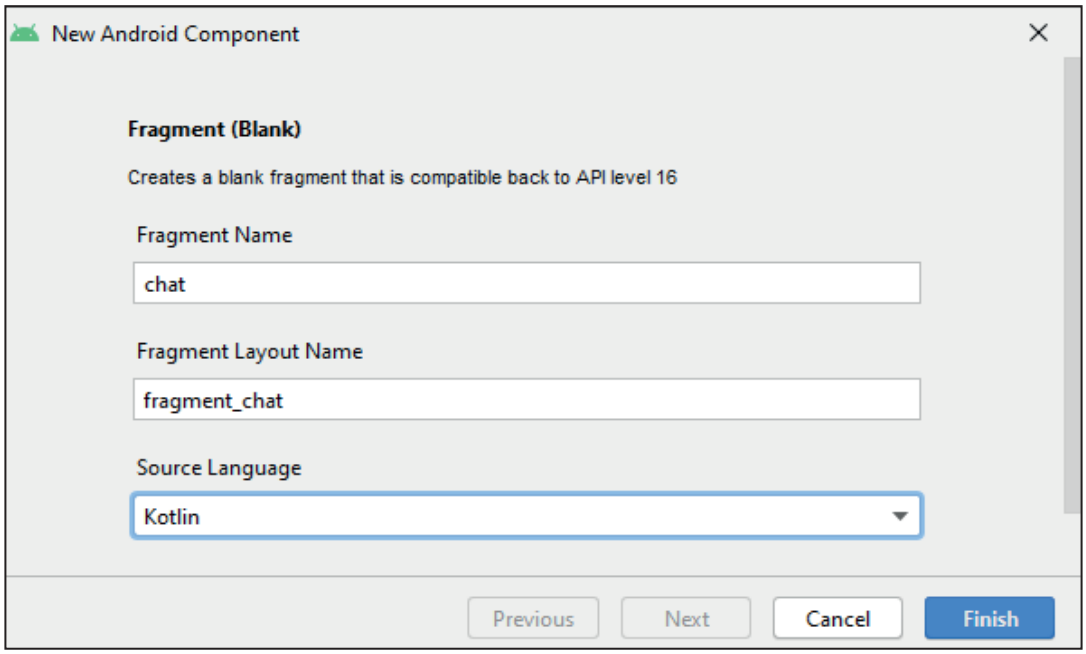


8- Open your **activity_main.xml** in the **Code** mode and you should have the following code:

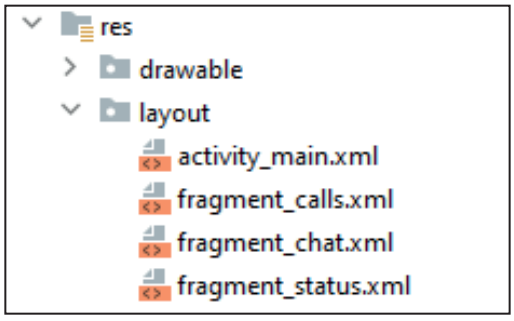
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
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

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="409dp"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="SpeakableTextPresentCheck" />

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/myViewPager"
        android:layout_width="411dp"
        android:layout_height="683dp"
        android:layout_marginTop="48dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/tabLayout"
        app:layout_constraintVertical_bias="0.0">

    </androidx.viewpager2.widget.ViewPager2>

</androidx.constraintlayout.widget.ConstraintLayout>
```

9- Now, you should add the content for your TabLayout tabs in your interface. Because each tab should have its content or its interface, you should create an activity interface for each tab. This tab interface will be presented by a **Fragment**.
In this example, you will create three fragments (chat, status, and calls). To do that, from your app project files structure panel, right click **layout → New → Fragment → Fragment (Blank)**

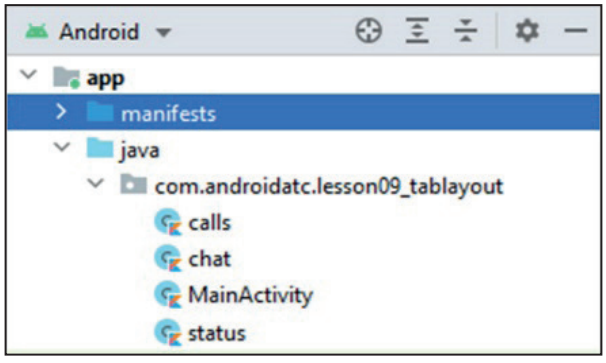
You should get the dialog box below. As illustrated in the dialog box below, type **chat** for the **Fragment Name**, and click **Finish**



10- Repeat the same previous step to create **status** and **calls** fragments. You should have the following layout files structure:



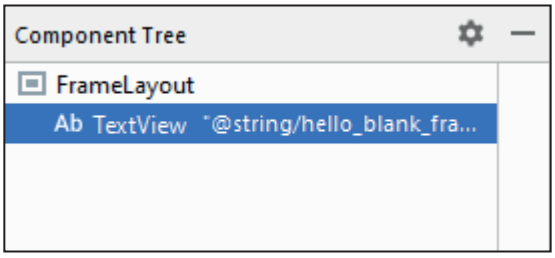
Also, three Kotlin files have been created as illustrated in the following figure:



In each of these fragments you may add what content you need. For example, in the chat fragment, you may add a **RecyclerView** widget with a **SearchView** widget configuration. This gives you a similar design to the CHATS tab in the WhatsApp app.

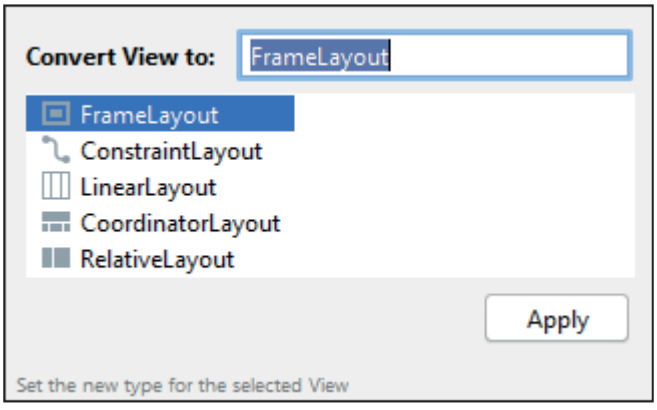
However, in this example, we will add a simple content because we will focus on the design and development concept.

11- Open the **fragment_chat.xml** file in the **Design** mode. You should have the following **Component Tree**:

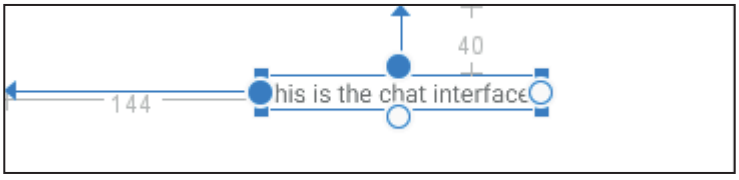


Select the **TextView** widget, and **Delete** it.

12- In the **Component Tree**, right click the **FrameLayout** , and select **Convert view…**. As illustrated in the following figure, select **ConstraintLayout**, and then click **Apply**



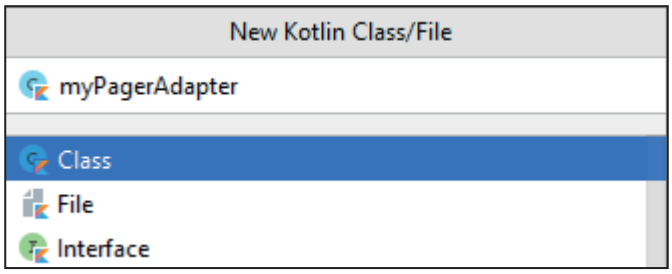
13- Open the **fragment_chat.xml** file in the **Design** mode, add a **TextView** widget to your fragment interface, set its constraints, and set its **text** attribute value to: **This is the chat interface.**



14- Repeat the last three steps to add a **TextView** widget to the **fragment_status.xml** and **fragment_calls.xml** files. The **text** attribute values will be: **This is the status interface** and **This is the calls interface** respectively.

15- Now, you should create a file is called fragment adapter. The main role of this file or class is to make connections between the **TabLayout** tabs and your fragment files (tabs contents).

To create your adapter class, Right click the **com.androidatc.lesson09_tablayout** container → **New** → **Kotlin Class/File**. As is illustrated in the figure below, for the file name type: **myPagerAdapter**, and press **Enter**




```
package com.androidatc.lesson09_tablayout

class myPagerAdapter {
}
```

Then add the following code:

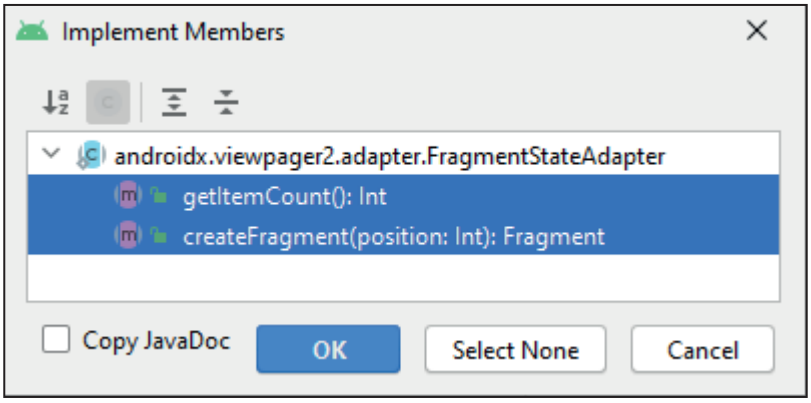
```
class myPagerAdapter(fragmentManager:FragmentManager, lifecycle:
Lifecycle):FragmentStateAdapter(fragmentManager, lifecycle) {
}
```

16- Double click the **FragmentManager** class, click the red pop-up lamp, and select **Import**.

17- Double click the **Lifecycle** class, click the red pop-up lamp, and select **Import**.

18- Double click the **FragmentStateAdapter** class, click the red pop-up lamp, and select **Import**.

19- Double click the class name: **myPagerAdapter**, click the red pop-up lamp, and select**: Implement Members**. As illustrated in the following figure, select the two methods, and click **OK**.



You should have the following code:

```
Class myPagerAdapter(fragmentManager: FragmentManager,
lifecycle:Lifecycle):FragmentStateAdapter
(fragmentManager,lifecycle){


    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

    override fun createFragment(position: Int): Fragment {
        TODO("Not yet implemented")
    }
}
```

20- Replace the: **`TODO("Not yet implemented")`** in the **`getItemCount()`** method with: **`return 3`**

Here, number **3** represents the number tabs index which you have in your app (0, 1, and 2).

21- Replace the: **`TODO("Not yet implemented")`** in the **`createFragment`** method with the following code:

```
return when (position) {
            0 -> { return chat()
            }
            1 -> { return status()
            }
            else -> {return calls()}
        }
```

The full code of your **`myPagerAdapter`** adapter file should be as follows:

```
package com.androidatc.lesson09_tablayout

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.lifecycle.Lifecycle
import androidx.viewpager2.adapter.FragmentStateAdapter

class myPagerAdapter(fragmentManager: FragmentManager, lifecycle:
Lifecycle):FragmentStateAdapter(fragmentManager, lifecycle) {


    override fun getItemCount(): Int {
        return 3
    }

    override fun createFragment(position: Int): Fragment {
        return   when (position) {
            0 -> { return chat()
```

```
            }
            1 -> { return status()
            }
            else -> {return calls()}
        }
}}
```

22- Open the Chat, Status and Calls Kotlin files (fragments), and **remove** the extra code. Each file must have the **onCreateView** class configuration only. The configuration in each file should be as follows (only the gray highlighted fragment file name is different):

```
class chat : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
     return inflater.inflate(R.layout.fragment_chat, container,
false)
    }

}
```

The **status** fragment file should be as follows:

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

The **calls** fragment file should be as follows:

```
class calls : Fragment() {

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

23- Now, open the **MainActivity** file to create the connection between the fragments, tabs, and the adapter.
As illustrated in the code below, add the configurations of the following three variables (**`TabLayout, viewPager2, adapter`**)

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val tabLayout=findViewById<TabLayout>(R.id.tabLayout)
        val viewPager2=findViewById<ViewPager2>(R.id.myViewPager)
        val adapter=myPagerAdapter(supportFragmentManager,lifecycle)
```

In the code above, double click the **myPagerAdapter** class, click the red pop-up lamp, and select **Import**.

24- Then add the configuration below for your adapter:

```
viewPager2.adapter = adapter
```

25- Add the **TabLayoutMediator** class configurations to your **MainActivity** code as illustrated in the following code:

```
TabLayoutMediator(tabLayout,viewPager2){tab,position->
    when(position){
        0->{
            tab.text="Chat"
        }
        1->{
            tab.text="Status"
        }
        2->{
            tab.text="Calls"
        }
    }
}.attach()
```

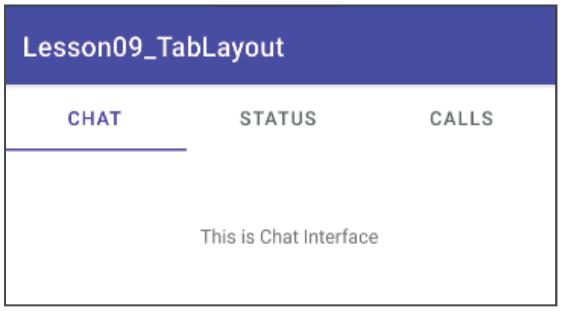
This mediator class is used to link your **TabLayout** with your **ViewPager2** classes. The mediator will synchronize the **ViewPager2**'s position with the selected tab when a tab is selected, and the **TabLayout'**s scroll position when the user drags the **ViewPager2**.

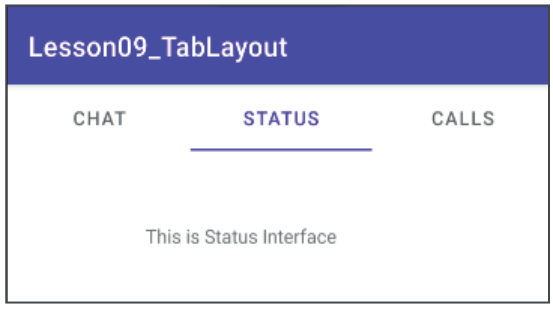
The role of the **`attach()`** method is linking the **TabLayout** and the **ViewPager2** together.

The full code of your **MainActivity** file is as follows:

```
package com.androidatc.lesson09_tablayout

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.viewpager2.widget.ViewPager2
import com.google.android.material.tabs.TabLayout
import com.google.android.material.tabs.TabLayoutMediator

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val tabLayout=findViewById<TabLayout>(R.id.tabLayout)
        val viewPager2=findViewById<ViewPager2>(R.id.myViewPager)
        val adapter=myPagerAdapter(supportFragmentManager,lifecycle)

        viewPager2.adapter=adapter

        TabLayoutMediator(tabLayout,viewPager2){tab,position->
            when(position){
                0->{
                    tab.text="Chat"
                }
                1->{
                    tab.text="Status"
                }
                2->{
                    tab.text="Calls"
                }
            }
        }.attach()

    }

}
```

26- Run your app. You should get the following results:

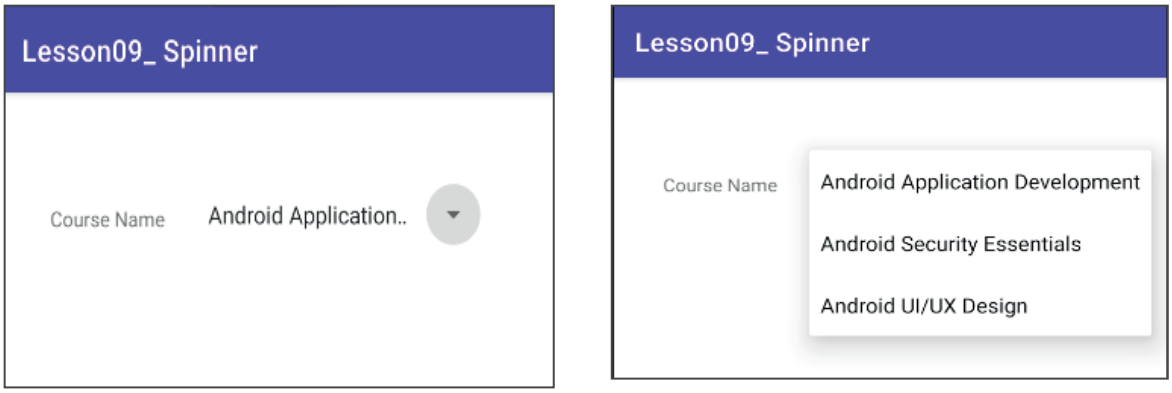




Spinner

Spinner widget provides a quick way to select one value from a set of choices. In the default state, a spinner shows its currently selected value (default value). Touching the spinner displays a dropdown menu with all other available choices from which the user can select a new one.

The following figure displays what the Spinner widget looks like before and after touching:



Example:
In the following example, you will create a small Android app using a Spinner widget to ask the app users to select one choice of the drop down list items. The steps are as follows:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lesson09_ Spinner** for the application name, then click **Finish**.

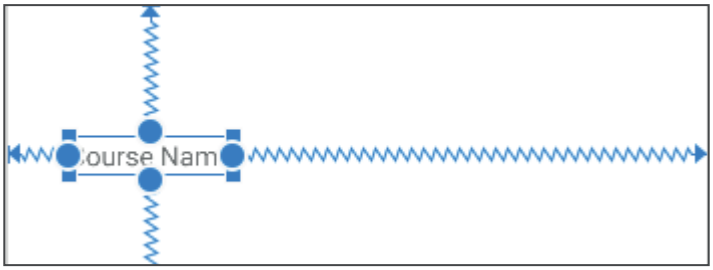
4- Before you start with typing the Kotlin code in your app, check the **build.gardle (Module: Lesson09_ Spinner.app)** file and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**
And click the **Sync Now**. The configuration should be as follows:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Open the **activity_main.xml** file in **Design** mode and then **delete** the "Hello World!" text.

6- From the **Palette** panel (Text), add a **TextView** widget to your activity interface, set its constraints, set its **text** attribute value to: **Course Name**, and set its **textSize** attribute value to **16sp**.

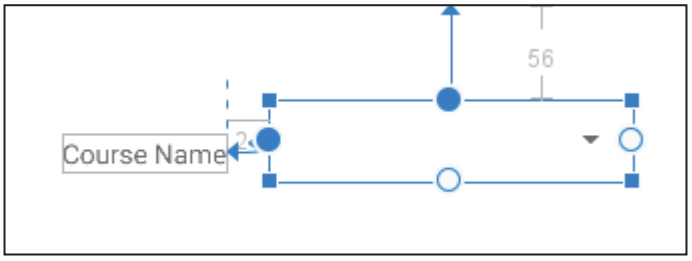
You should have the following design:



7- From the **Palette** panel (Containers), add a **Spinner** widget to your activity interface using drag and drop technique, set its constraints, and set its attributes values as follows:

id: spinnerCourse	**layout_width**: 214dp	**layout_height**: 48dp

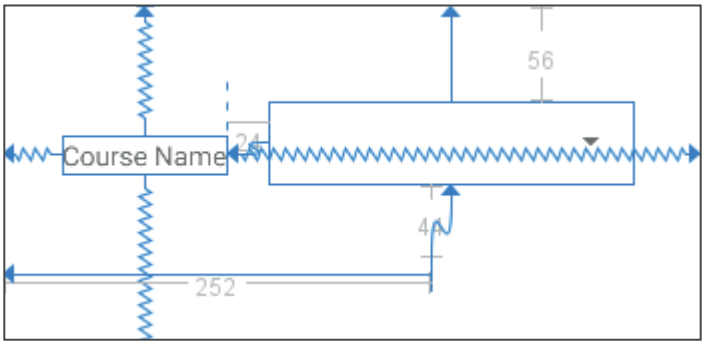
You should have the following design:



8- Later, you need to have a **TextView** widget to display your app user's selection result. Add a **TextView** widget directly below your Spinner widget, set its constraints, delete its **text** attribute value, and set its attributes values as follows:

id: course_Result	**layout_width**: wrap_content	**layout_height**: wrap_content

Your activity should have the following design:



9- Open the **MainActivity** file and configure the two variables below. The **`mySpinner`** variable represents the **Spinner** drop down list, and the **`courseName`** variable represents the drop down list choices which will be represented as array elements.

```
val mySpinner:Spinner = findViewById(R.id.spinnerCourse)

val courseName =listOf <String>("Android Application Development",
"Android Security Essentials", "Android UI/UX Design")
```

Note: The choices you provide for the Spinner can come from any source, but must be provided through a SpinnerAdapter as you will see later in this example, such as an **ArrayAdapter** if the choices are available in an array or a CursorAdapter if the choices are available from a database query.

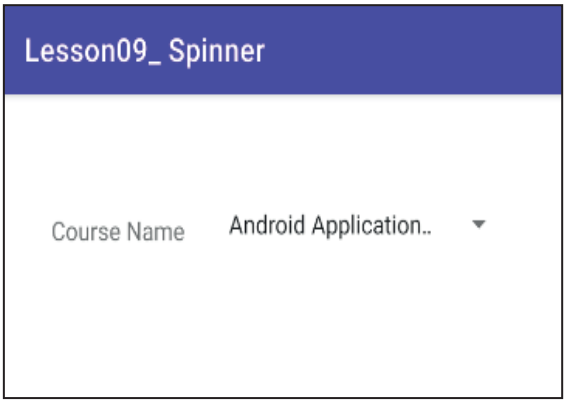
10- Now, you want to build a connection between this spinner drop down list (**mySpinner**) with the choices list (**courseName**). You may do that using the following **Spinner Adapter** configuration:

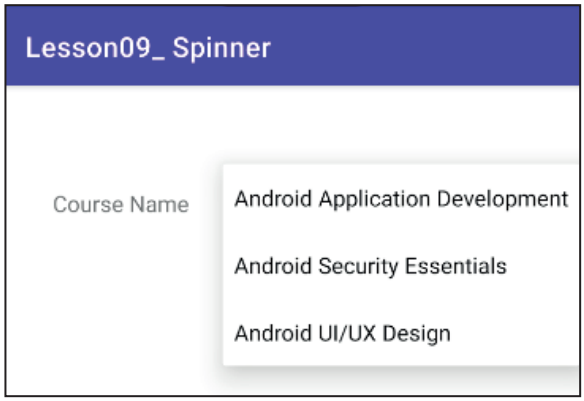
```
valcourseListAdapter = ArrayAdapter(this,android.R.layout.simple_
spinner_item,courseName)

courseListAdapter.setDropDownViewResource(android.R.layout.
simplespinner_dropdown_item)

mySpinner.adapter = courseListAdapter
```

The full code of the **MainActivity** file is as follows:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mySpinner:Spinner = findViewById(R.id.spinnerCourse)
        val courseName =listOf <String>("Android Application
Development","Android Security Essentials","Android UI/UX Design")

        val courseListAdapter = ArrayAdapter(this,android.R.layout.
simple_spinner_item,courseName)

        courseListAdapter.setDropDownViewResource(android.R.layout.
simple_spinner_dropdown_item)

        mySpinner.adapter = courseListAdapter

    }
}
```

11- **Run** your app to test if your app until this step is working fine. You should get the following:



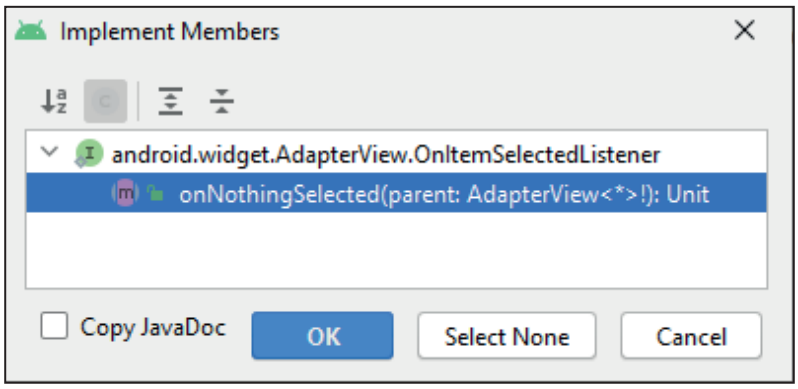


Now, you will configure your app to display the app user selected choice in the TextView widget which is located below the spinner drop down list and which has **id: course_Result** as illustrated in the next steps.

12- The following configuration explains the event which will happen when the user selects an item from the drop-down? The Spinner object receives an on-item-selected event.

```
mySpinner. onItemSelectedListener =    object:AdapterView.
OnItemSelectedListener{
    override fun onItemSelected(
        parent: AdapterView<*>?,
        view: View?,
        position: Int,
        id: Long) {
        TODO("Not yet implemented")
    }
}
```

Double click **object**, click the red pop-up lamp, and select **Implement Members**, you should get the dialog box below, click **OK**.



You should have the following code:

```
mySpinner.onItemSelectedListener = object:AdapterView.
OnItemSelectedListener{
    override fun onItemSelected(
        parent: AdapterView<*>?,
        view: View?,
        position: Int,
        id: Long) {
        TODO("Not yet implemented")
    }

    override fun onNothingSelected(parent: AdapterView<*>?) {
        TODO("Not yet implemented")
    }
}
```

13- Add a new variable: **userCourse** at the top of your MainActivity code with a null value as illustrated in the following code:

```
class MainActivity : AppCompatActivity() {
    var userCourse :String=""
```

14- Replace the first gray highlighted ***TODO("Not yet implemented")*** in the code of the ***onItemSelectedListener*** method with the following code:

```
userCourse = parent?.getItemAtPosition(position).toString()
```

Until this step, your full code of the MainActivity file is:

```
class MainActivity : AppCompatActivity() {

    var userCourse :String=""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mySpinner:Spinner = findViewById(R.id.spinnerCourse)
        val courseName =listOf<String>("Android Application
Development","Android Security Essentials","Android UI/UX Design")

        val courseListAdapter =
ArrayAdapter(this,android.R.layout.simple_spinner_item,courseName)

courseListAdapter.setDropDownViewResource(android.R.layout.simple_
spinner_dropdown_item)
```

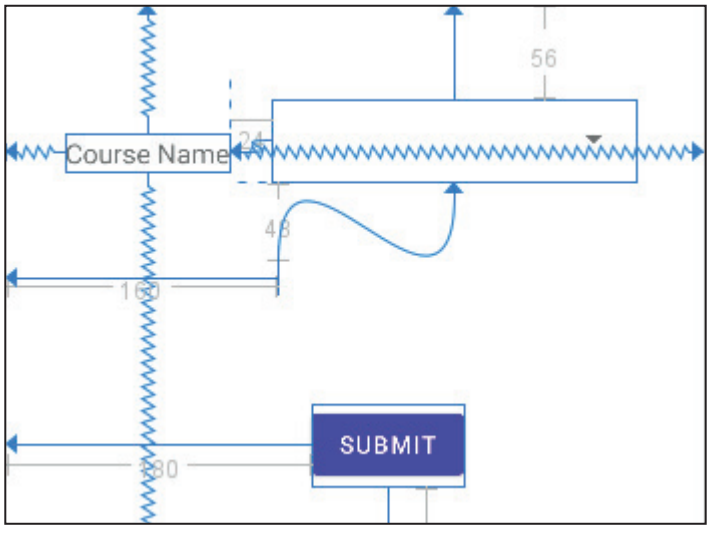
```
        mySpinner.adapter = courseListAdapter

        mySpinner.onItemSelectedListener = object:AdapterView.
OnItemSelectedListener{
            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long) {
       userCourse = parent?.getItemAtPosition(position).toString()

            }

            override fun onNothingSelected(parent: AdapterView<*>?)
{
                TODO("Not yet implemented")
            }
        }

    }
}
```

15- Now, you should add a Submit button to your interface to get your app user selection choice.

Open the **activity_main.xml** file in the Design mode, add a **Button** widget, set its constraints, and set its attributes values as follows:

id:submitId	**text**: Submit

The design of your activity interface should look like the following figure:



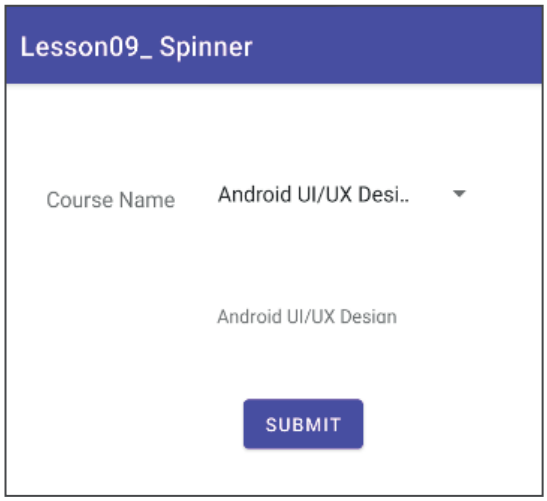
16- Open **MainActivity** file, and add the following code to your **SUBMIT** button:

```
submitId.setOnClickListener{

    course_Result.text= userCourse
}
```

The full code of the **MainActivity** file is as follows:

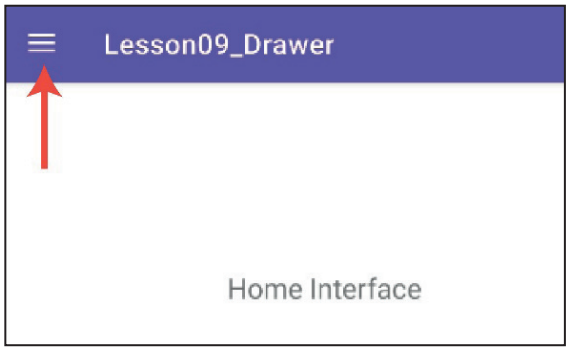
```
class MainActivity : AppCompatActivity() {

    var userCourse :String=""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mySpinner:Spinner = findViewById(R.id.spinnerCourse)
        val courseName =listOf<String>("Android Application
Development","Android Security Essentials","Android UI/UX Design")

        val courseListAdapter = ArrayAdapter(this,android.R.layout.
simple_spinner_item,courseName)

        courseListAdapter.setDropDownViewResource(android.R.layout.
simple_spinner_dropdown_item)

        mySpinner.adapter = courseListAdapter

        mySpinner.onItemSelectedListener =
objectAdapterView.OnItemSelectedListener{

            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long) {
       userCourse = parent?.getItemAtPosition(position).toString()

            }
           override fun onNothingSelected(parent: AdapterView<*>?) {
                TODO("Not yet implemented")
            }
        }

        submitId.setOnClickListener{
            course_Result.text= userCourse
        }
    }
}
```

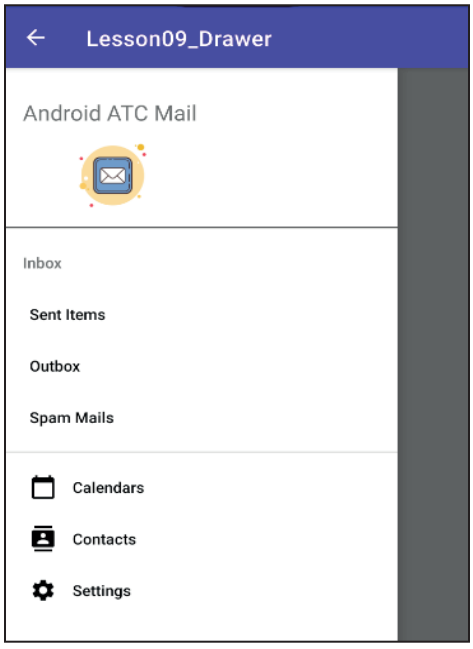
17- **Stop** your app, then **Run** it again. Select your course, and tap the **SUBMIT** button. You should get the following run result:



Drawer

In apps that use Material Design, there are two primary options for navigation: tabs and drawers. When there is insufficient space to support tabs, drawers provide a handy alternative.





The lab of this lesson will discuss step by step adding and configuring the drawer menu to an Android app.