



Birzeit University
Faculty of Engineering and Technology
Electrical and Computer Engineering Department
Advance Computer Systems Engineering Lab ENCS515

EXP. No. 2. Android Layouts

1. Objectives

- ❖ What Views, View Groups, Layouts, and Widgets are and how they relate to each other.
- ❖ How to declare layouts dynamically at runtime.
- ❖ Adding widgets dynamically at runtime.
- ❖ Switching between two Activities.
- ❖ How to use Events and Event Listeners.

2. Introduction

In this lab you will be learning how to use and extend the Android user interface library. In several ways, it is very similar to the Java Swing library, and in perhaps just as many ways it is different. While being familiar with Swing may help in some situations, it is not necessary. It is important to note that this lab is meant to be done in order, from start to finish. Each activity builds on the previous one, so skipping over earlier activities in the lab may cause you to miss an important lesson that you should be using in later activities.

2.1. Brief Background on View Classes

In Android, a user interface is a hierarchy composed of different View objects. The View class serves as the base class for all graphical elements, of which there are two main types:

- Widgets: Can either be individual, or groups of UI elements. These are things like buttons, text fields, and labels. Widgets directly extend the View class.
- Layouts: Provide a means of arranging UI elements on the screen. These are things like a table layout or a linear layout. Layouts extend the ViewGroup class, which in turn extends the View class.

Layouts are all subclasses of the ViewGroup class, and their main purpose is to control the position of all the child views they contain. Figure 2.1 are some of the more common layout types that are built into the Android platform.

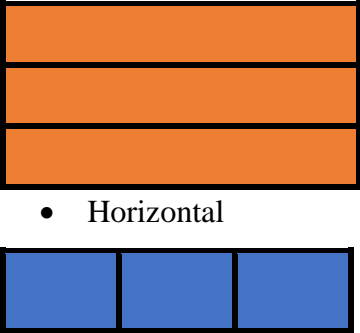
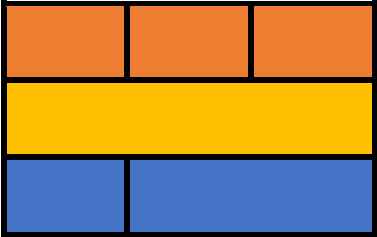
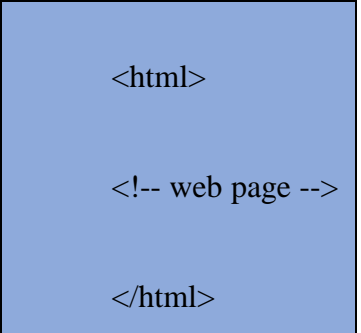
Linear Layout	Relative Layout	Web View
 <ul style="list-style-type: none">• Horizontal• Vertical		
A layout that organizes its children into a single horizontal or vertical view. It creates a scrollbar if the length of the window exceeds the length of the screen.	Enables you to specify the location of child object relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).	Displays Web Pages.

Figure 2.1 Common Layout Types

Note: Although you can nest one or more layouts within another layout to achieve your UI design, you should strive to keep your layout hierarchy as shallow as possible. Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a deep view hierarchy).

Declaring the layouts for your user interface can be done dynamically (in code), statically (via an XML resource file), or any combination of the two. In the following subsection, you will build a set of user interfaces in code. In a future lab, you will build a set of user interfaces in XML.

3. Procedure

In this lab you are asked to build a new android application that allows the user to add and show the added customers from customer's list. This is done in two separated Activities one to show all customers in the list (MainActivity) as shown in Figure 2.2.a and the other is to add new customers to the list (AddCustomerActivity) as shown in Figure 2.2.b.

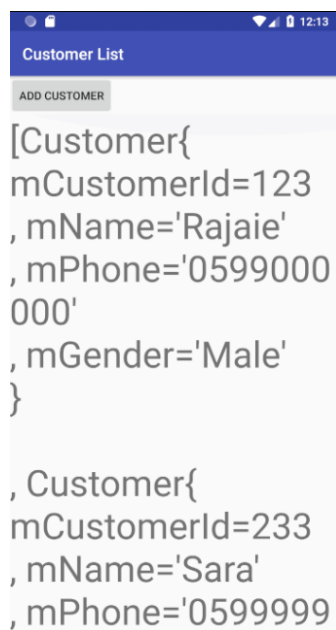


Figure 2.2.a Main Activity

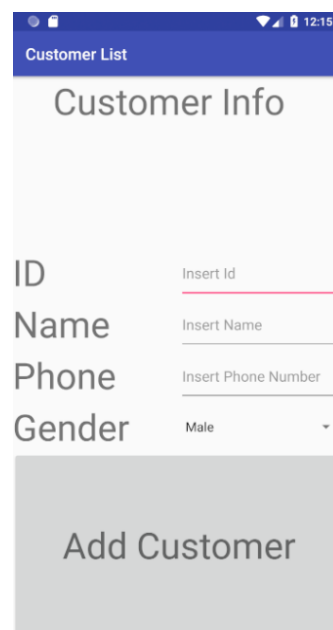


Figure 2.2.b Add Customer Activity

Figure 2.2 Application Activities

3.1. Create a new Android Project:

- In Android Studio, create a new project:
 - If you don't have a project opened, in the Welcome to Android Studio window, click Start a new Android Studio project.
 - If you have a project opened, select File > New Project.
- In the New Project screen, enter the following values:
 - Application Name: "Customer List" see Figure 2.3.
 - Company Domain: "birzeit.edu" see Figure 2.3.
- Click Next.
- In the Target Android Devices screen, keep the default values and click Next.
- In the Add an Activity to Mobile screen, select Empty Activity and click Next.

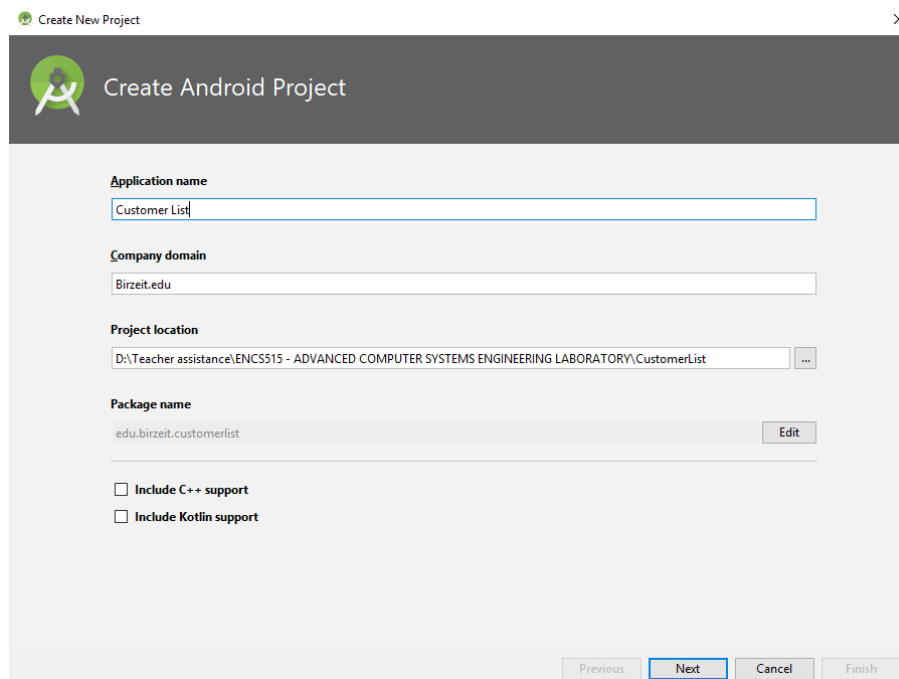


Figure 2.3 Creating new Project Screen.

3.2. Creating a Customer Model

At the beginning you are asked to build a customer class to enable you to create objects from customer class. This will hold the data about the customer.

- Right click on the java package at the left hand of the panel and add new java class as shown in Figure 2.4 (be careful to follow the java notation (class name capital letter)) as isolated in Figure 2.5.

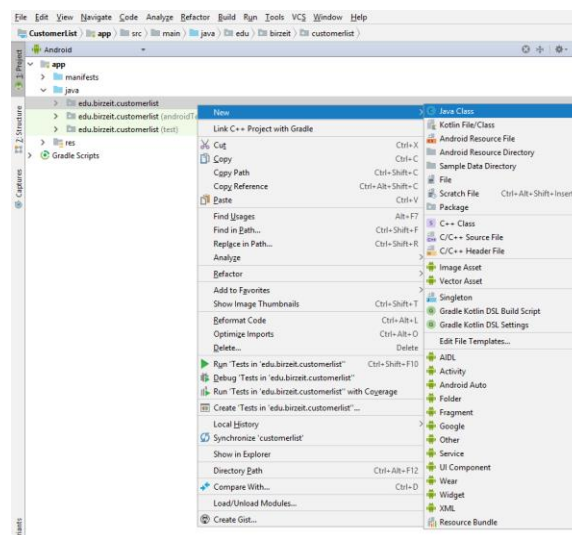


Figure 2.4 Adding New Java Class Screen

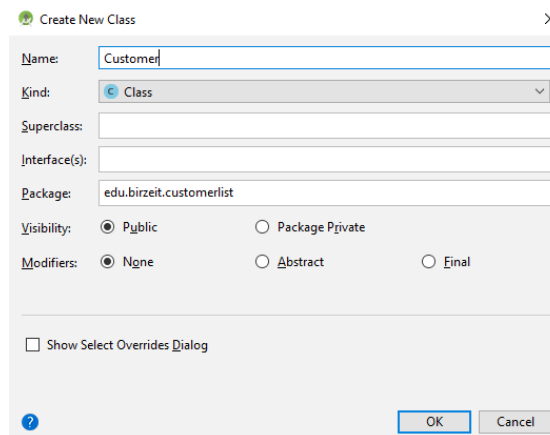


Figure 2.5 Adding Customer java Class Screen

- Add the following attributes to Customer class:
 - mCustomerID (Long): private unique ID number for customer.
 - mName (String): private holds the customer name.
 - mPhone (String): private holds the phone number.
 - mGender (String): private holds the gender type.
- Create empty constructor and a constructor with fields (press Alt + Insert on the keyboard) as shown in Figure 2.6.a .

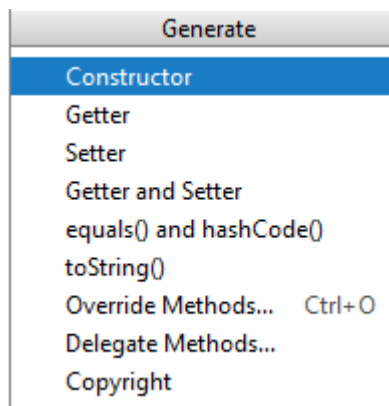


Figure 2.6.a Adding Constructor

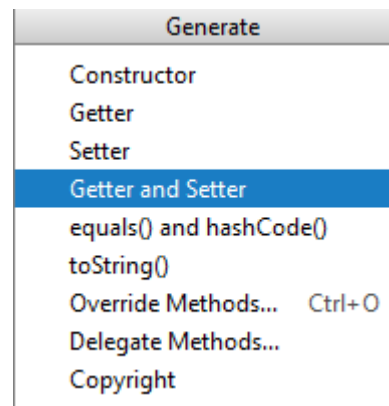


Figure 2.6.b Adding Getters and Setters

Figure 2.6 adding Constructor, Getters and Setters Screen

- Create setters and getters to all attribute's fields (press Alt + Insert on the keyboard) as shown in Figure 2.6.b.
- Override toString method by the same way as the constructor and the getters and setters are added.
- Add a static Array List of Customers in the Customer Class where you will save the added Customers.
- The code for all attributes, constructors, setters, getters and toString methods are shown below.

```

package edu.birzeit.customerlist;

import java.util.ArrayList;

public class Customer {
    public static ArrayList<Customer> customersArrayList=new ArrayList<Customer>();
    private long mCustomerId ;
    private String mName;
    private String mPhone;
    private String mGender;

    public Customer() {

    }
    public Customer(long mCustomerId, String mName, String mPhone, String mGender)
    {
        this.mCustomerId = mCustomerId;
        this.mName = mName;
        this.mPhone = mPhone;
        this.mGender = mGender;
    }
    public long getmCustomerId() {
        return mCustomerId;
    }
    public void setmCustomerId(long mCustomerId) {
        this.mCustomerId = mCustomerId;
    }
    public String getmName() {
        return mName;
    }
    public void setmName(String mName) {
        this.mName = mName;
    }
    public String getmPhone() {
        return mPhone;
    }
    public void setmPhone(String mPhone) {
        this.mPhone = mPhone;
    }
    public String getmGender() {
        return mGender;
    }
    public void setmGender(String mGender) {
        this.mGender = mGender;
    }
    @Override
    public String toString() {
        return "Customer{" +
            "\nmCustomerId=" + mCustomerId +
            "\n, mName='" + mName + '\'' +
            "\n, mPhone='" + mPhone + '\'' +
            "\n, mGender='" + mGender + '\'' +
            "\n}\n\n";
    }
}

```

3.3. Creating new Activity (Add Customer Activity)

There is two different ways to add a new activity one is simple and the other is standard you can choose any way you feel it is easier:

❖ Simple Way

- The first way to add an activity is by right-clicking on the package where you want to add a new activity and click on Activity then on Empty Activity as shown in Figure 2.7. the New Android Activity screen will appear, change the Activity name to AddCustomerActivity as shown in Figure 2.8. This will add a java class and a layout (.xml) as shown in Figure 2.9. and it will also add the activity to the manifests file as shown in Figure 2.14.

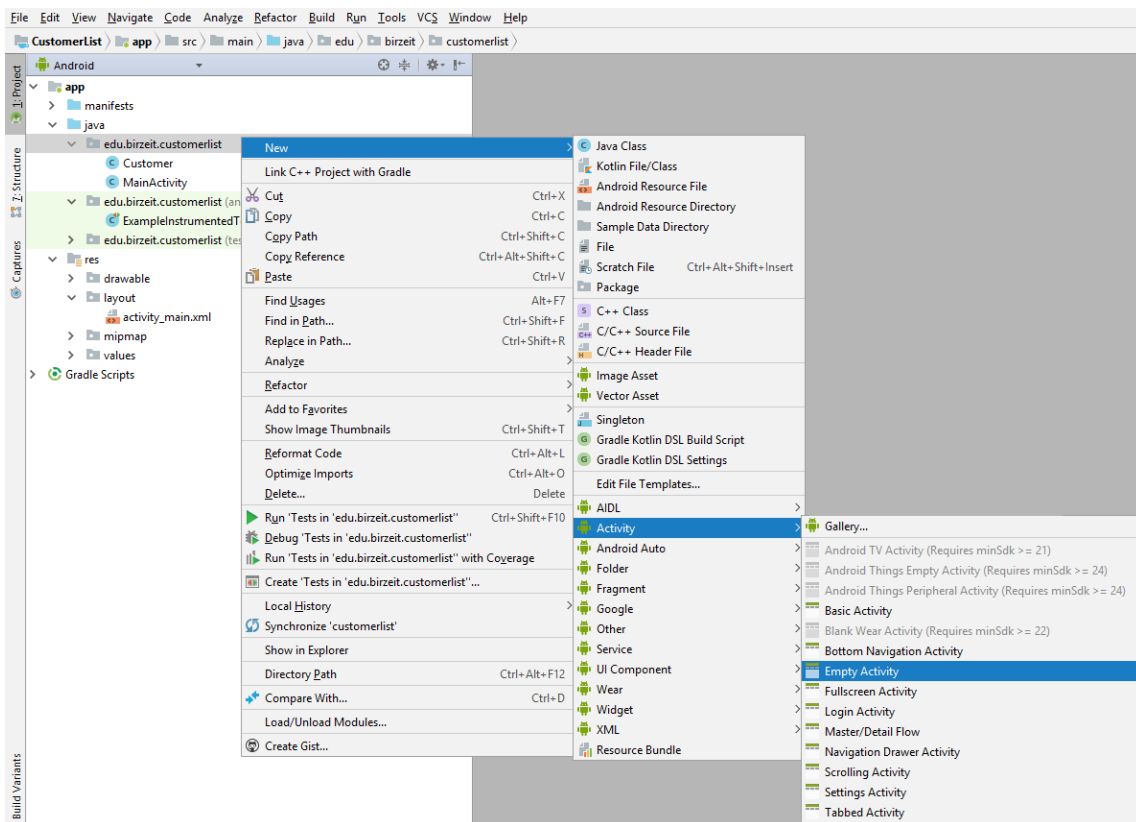


Figure 2.7 Adding Empty Activity

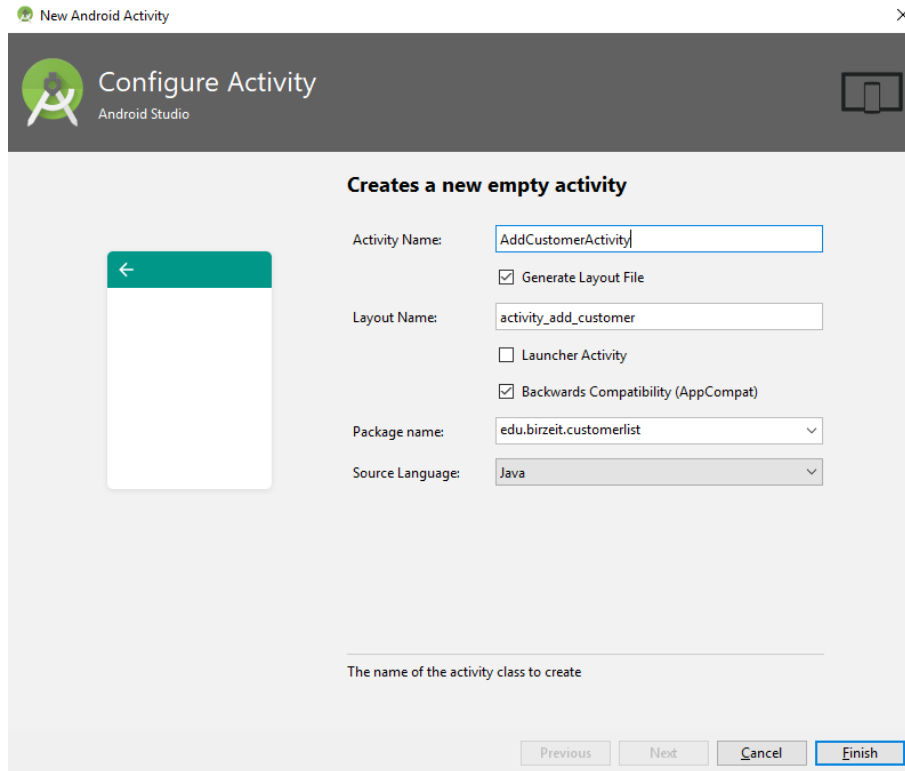


Figure 2.8 New Android Activity Screen

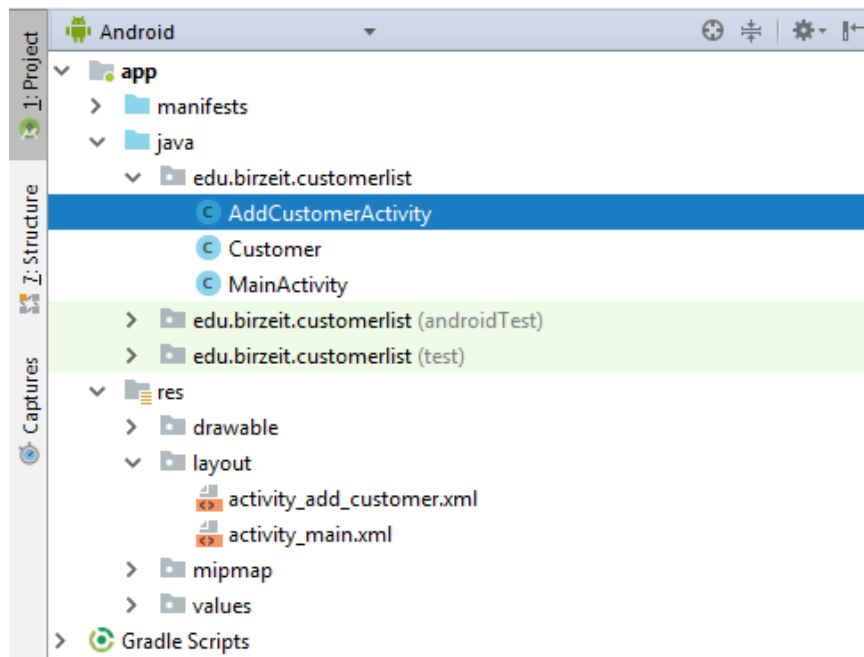


Figure 2.9 Activity java and layout

❖ Standard way

➤ The second way is by adding each component separately:

- Start by adding a java class called AddCustomerActivity by making its super class AppCompatActivity as shown in Figure 2.10 and override the onCreate method from AppCompatActivity (press Alt + Insert on the keyboard) as shown in Figure 2.11.a and then click on override methods and onCreate method as shown in Figure 2.11.b the result will be as shown in Figure 2.11.c.

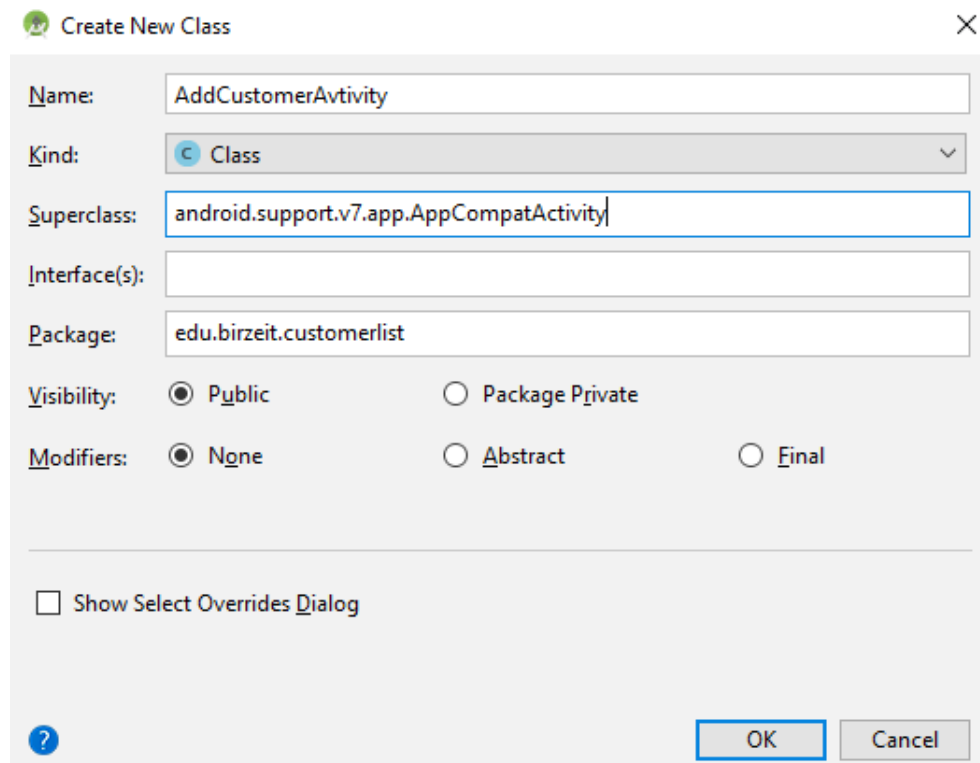


Figure 2.10 Adding AddCustomerActivity Class Screen

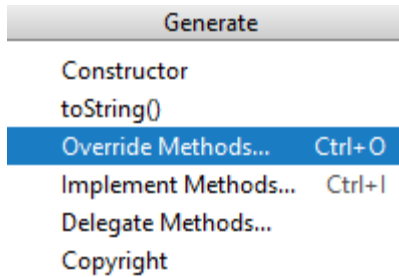


Figure 2.11.a

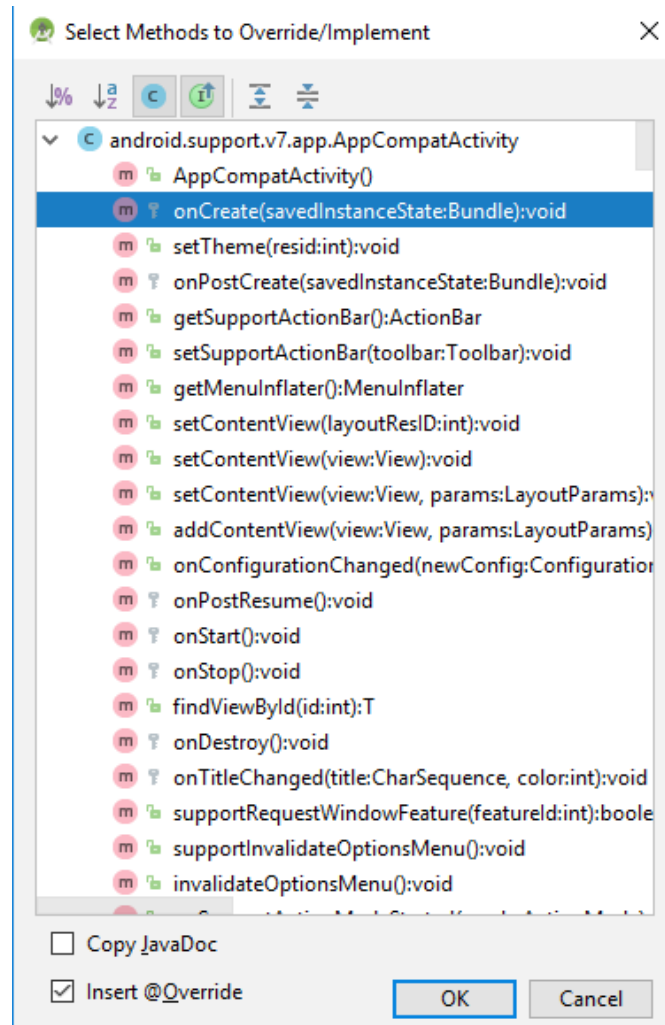


Figure 2.11.b

```
public class AddCustomerAvtivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Figure 2.11.c

Figure 2.11 Overriding OnCreate Method in Add Customer Activity.

- Then add the xml file in the res/layout package which is called activity_add_customer by right-clicking on the layout package and the by clicking new Layout resource file as shown in Figure 2.12. The new resource file screen will appear you can change the root element (root layout to LinearLayout) as shown in Figure 2.13
- After that you should add the activity in the manifests file as shown in Figure 2.14.
- Finally, you should link the java file with the layout by adding setContentView(R.layout.activity_add_customer) in onCreate method.

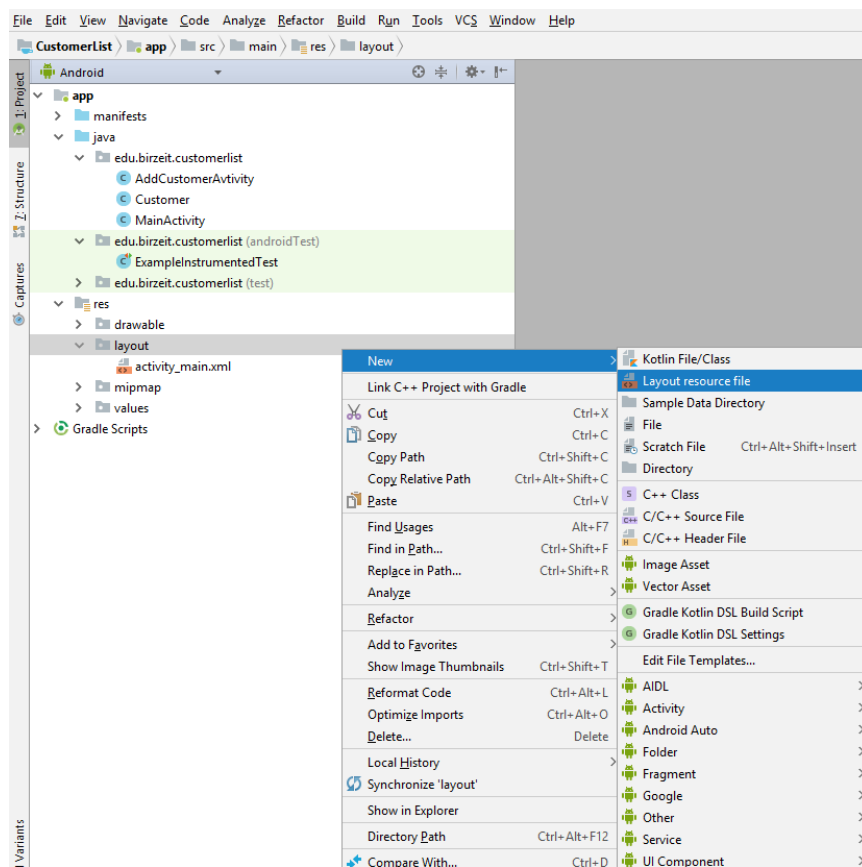


Figure 2.12 Adding New .xml Layout Screen

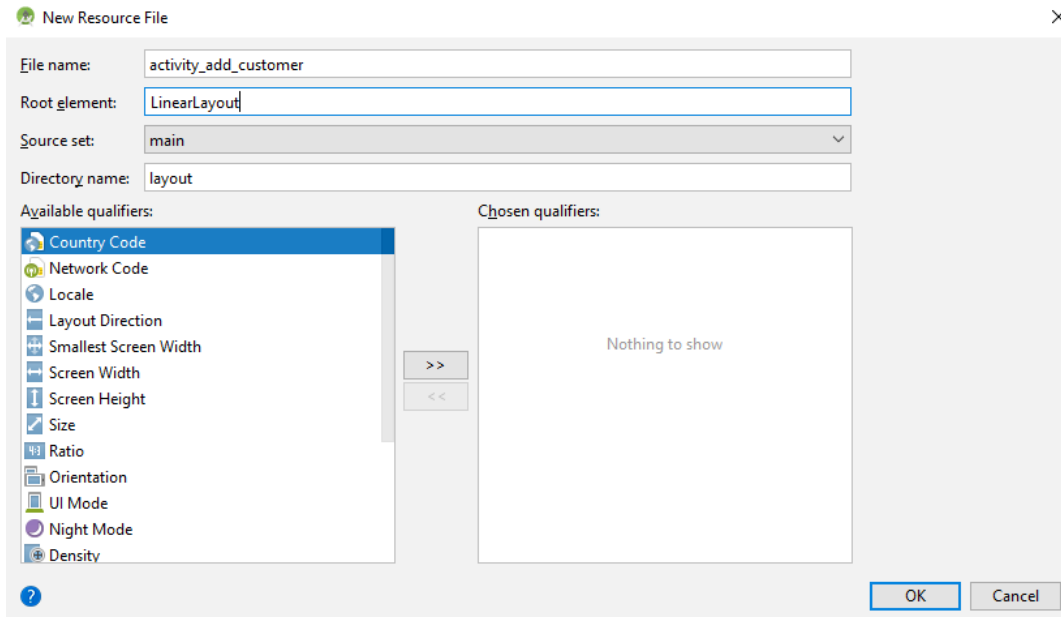


Figure 2.13 Adding `activity_add_customer` Layout Screen

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.birzeit.customerlist">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Customer List"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AddCustomerAvtivity"></activity>
    </application>

</manifest>
```

Figure 2.14 Adding the Activity in the Manifests file

3.4. Making the activity_add_customer Layout (statically using .xml or drag and drop)

In this part you are going to design the layout shown in Figure 2.2.b this can be designed using the drag and drop.

- Convert the root layout into a vertical layout to add the widgets and layouts inside it.
- Add a textview under the LinearLayout (vertical) and change the Text in the text View to “Customer Info” from the right-side panel attributes.
- Add LinearLayout (horizontal) under the LinearLayout (vertical) then add textview and a plaintext (EditText) inside the LinearLayout (horizontal).
 - Change the Horizontal LinerLayout layout_high from match_parent to wrap_content.
 - Change the text in the textview to “Id”.
 - Change the EditText ID in the right-panel Attributes to “editText_Id”.
- Remove the text in the EditText and add in the hint “Insert Id”.
- Add LinearLayout (horizontal) under the LinearLayout (vertical) then add textview and a plaintext (EditText) inside the LinearLayout (horizontal).
 - Change the Horizontal LinerLayout layout_high from match_parent to wrap_content.
 - Change the text in the textview to “Name”.
 - Change the EditText ID in the right-panel Attributes to “editText_Name”.
 - Remove the text in the EditText and add in the hint “Insert Name”.
- Add LinearLayout (horizontal) under the LinearLayout (vertical) then add textview and a plaintext (EditText) inside the LinearLayout (horizontal).
- Change the Horizontal LinerLayout layout_high from match_parent to wrap_content.
 - Change the text in the textview to “Phone”.
 - Change the EditText ID in the right-panel Attributes to “editText_Phone”.
 - Remove the text in the EditText and add in the hint “Insert Phone Number”.

- Add LinearLayout (horizontal) under the LinearLayout (vertical) then add textview and a Spinner inside the LinearLayout (horizontal) (if you can't find the spinner in the right-side panel palette search for it).
 - Change the Horizontal LinerLayout layout_height from match_parent to wrap_content.
 - Change the text in the textview to "Gender". This will be defined in the java code.
 - Change the spinner ID in the right-panel Attributes to "spinner_Gender".
- Add a Button under the root vertical layout. Change the Text to "Add Customer" and the ID to "button_Add_Customer" from the right-side panel Attributes.
- Figure 2.15 shows the Component Tree of the activity_add_customer layout which appears in the left-side panel.

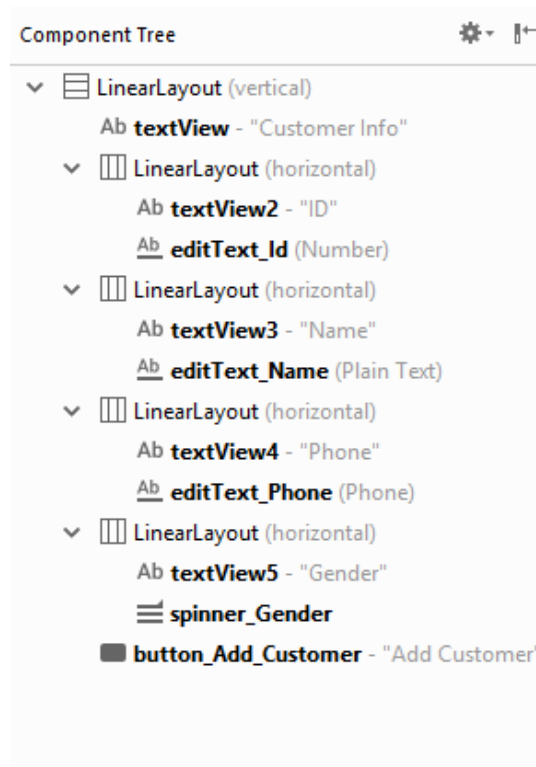


Figure 2.15 Add Customer Activity Layout Component Tree

- **Note that you can change the attributes of the widgets as you wish (changing the size of the text, the color, the alignment, etc....).**

3.5. Implementing Add Customer Activity Java Class

In this task, you will give the user the ability to enter their own customers by displaying Add Customer Activity which contains all required customer information that should be entered as you designed in the previous step. In order to accomplish this task, the following steps show the main requirements of this task that are implemented in AddCustomerActivity.

- Define the gender spinner you added in the layout.
- Add a listener to “Add Customer” button using `setOnClickListener` method.
- Validate the input information about new customer e.g. (Not empty name).
- Return to “MainActivity” once the customer is added successfully.

After reading the main requirements of this task, now you can begin implementation. The following procedure shows how the above requirements are implemented:

- In `OnCreate` method, find and initialize the `spinner_Gender` for defined list of data. Depending on the requirements, the list should have two options: Male and Female the code below shows how to add options in the spinner.

```
String[] options = { "Male", "Female" };  
final Spinner genderSpinner =(Spinner)  
findViewById(R.id.spinner_Gender);  
ArrayAdapter<String> objGenderArr = new  
ArrayAdapter<>(this, android.R.layout.simple_spinner_item, options);  
genderSpinner.setAdapter(objGenderArr);
```

- In `OnCreate` method, find the remaining `EditTexts`: `editText_Id`, `editText_Name` and `editText_Phone` by using `findViewById` method in order to extract and build Customer object. Don't forget to cast to `EditText`. The following code shows how to get the `EditTexts`.

```
final EditText idEditText =  
(EditText) findViewById(R.id.editText_Id);  
final EditText nameEditText =  
(EditText) findViewById(R.id.editText_Name);  
final EditText phoneEditText =  
(EditText) findViewById(R.id.editText_Phone);
```


Find the button_Add_Customer using findViewById method. Then, implement onClick method by using setOnClickListener. When the user clicks on the button, the customer information should be converted to Customer object in order to add it to customersArrayList which exists in Customer class. Once the customer has been added successfully, the activity should disappear. The following code is how to implement the setOnClickListener method.

```
Button addCustomerButton = (Button) findViewById(R.id.button_Add_Customer);
addCustomerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Customer newCustomer =new Customer();

        if(idEditText.getText().toString().isEmpty()) newCustomer.setmCustomerId(0);
        else newCustomer.setmCustomerId(Long.parseLong(idEditText.getText().toString()));
        if(nameEditText.getText().toString().isEmpty()) newCustomer.setmName("No Name");
        else newCustomer.setmName(nameEditText.getText().toString());
        if(phoneEditText.getText().toString().isEmpty()) newCustomer.setmPhone("No Phone");
        else newCustomer.setmPhone(phoneEditText.getText().toString());

        newCustomer.setmGender(genderSpinner.getSelectedItem().toString());
        Customer.customersArrayList.add(newCustomer);
        Intent intent=new Intent(AddCustomerActivity.this,MainActivity.class);
        AddCustomerActivity.this.startActivity(intent);
        finish();
    }
});
```

3.6. Implementing Main Activity Java class

Work done in this section will be limited to the MainActivity.java file. MainActivity is an Activity class which displays a vertical scrollable list of all the customers as shown in Figure 2.2.a.

In previous section, you defined layouts for your interface using XML, but there are plenty of instances where it is still necessary to do it at run-time in code. For instance, you may want to dynamically change the layout based on some type of user input. Mainly, in this section, you must define layouts and add widgets at runtime in code.

For customers view, you should define two vertical linear layouts, where the first linear layout should have button to add new customer which directs the user to AddCustomerActivity and the second linear layout should be added to first linear layout in order to display the available customers to be as a list. Follow the following procedure to see how that can be done:

- Remove this line `setContentView(R.layout.activity_main)` from `onCreate` method.
- Define first and second Layouts and Button as shown in the following code.

```
LinearLayout firstLinearLayout=new LinearLayout(this);
Button addButton =new Button(this);
LinearLayout secondLinearLayout=new LinearLayout(this);
```

- Define a scrollview to make the display list scrollable

```
ScrollView scrollView=new ScrollView(this);
```

- Set the orientation of the firstLinearLayout and the secondLinearLayout to Vertical. See the following code.

```
firstLinearLayout.setOrientation(LinearLayout.VERTICAL);
secondLinearLayout.setOrientation(LinearLayout.VERTICAL);
```

- Set the text of the addButton to “Add Customer” and the `layout_width` and `layout_hight` to `wrap_content`. See the following code.

```
addButton.setText("Add Customer");
addButton.setLayoutParams(new
LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,ViewGroup
.LayoutParams.WRAP_CONTENT));
```

- Add the addButton to the firstLinearLayout and the secondLinearLayout to the scrollView, and add the scrollView to the firstLinearLayout as shown in the code below.

```
firstLinearLayout.addView(addButton);
scrollView.addView(secondLinearLayout);
firstLinearLayout.addView(scrollView);
```

- Finally, set the First Linear Layout as a main content view for the MainActivity.

```
setContentView(firstLinearLayout);
```

- Add `onClickListener` for `addButton` which is used to show `AddCustomerActivity`. The following code shows how you can add listener and how the other activity can be activated.

```
addButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Intent intent = new  
Intent(MainActivity.this, AddCustomerActivity.class);  
        MainActivity.this.startActivity(intent);  
        finish();  
    }  
});
```

- Finally, to display the customers that exist in `customersArrayList`, you should write code in `onCreate` method in order to work through on that list to add customer information on the second linear layout that you already defined it in previous steps. The following code shows a suggested implementation:

```
for(Customer objCustomer : Customer.customersArrayList) {  
    TextView txtCustomerInfo = new TextView(this) ;  
    txtCustomerInfo.setTextAppearance(R.style.TextAppearance_AppCompat_Display2);  
    txtCustomerInfo.setText(objCustomer.toString());  
    secondLinearLayout .addView(txtCustomerInfo);  
}
```

4. Building Web Apps in WebView (Optional Section)

If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using `WebView`. The `WebView` class is an extension of Android's `View` class that allows you to display web pages as a part of your activity layout. It does not include any features of a fully developed web browser, such as navigation controls or an address bar. All that `WebView` does, by default, is show a web page. A common scenario in which using `WebView` is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Within your Android application, you can create an Activity that contains a `WebView`, then use that to display your document that's hosted online.

Another scenario in which WebView can help is if your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a WebView in your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout. Instead, you can design a web page that's tailored for Android devices and then implement a WebView in your Android application that loads the web page.

Adding a WebView to Your Application:

- To add a WebView to your Application, simply include the <WebView> element in your activity layout. For example, here's a layout file in which the WebView fills the screen:

```
<WebView
android:id="@+id/webview"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />
```

- To load a web page in the WebView, use loadUrl(). For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("https://ritaj.birzeit.edu");
```

Before this will work, however, your application must have access to the Internet. To get Internet access, request the INTERNET permission in your manifest file:

```
<uses-permission android:name="android.permission.INTERNET" />
```

5. Todo

This part will be given to you by the teacher assistant in the lab time.