



Birzeit University

Faculty of Engineering and Technology

Electrical and Computer Engineering  
Department

Manual for  
ENCS515 Advance Computer Systems  
Engineering Laboratory

Designed and Updated by Eng. Rajaie Imseeh  
and Dr. Mohammed Hussein

September 2019

# Table of Experiments

EXP. No. 1. Introduction to Android Programming .....	1
EXP. No. 2. Android Layouts .....	13
EXP. No. 3. Using Intents and Notifications .....	33
EXP. No. 4. SQLite Database .....	45
EXP. No. 5. Frame Animation and Tween Animation in Android	54
EXP. No. 6. Singleton and Shared Preferences .....	68
EXP. No. 7. Fragments.....	77
EXP. No. 8. Integrating REST API into Android Application.....	88
EXP. No. 9. Spring Boot Part 1 .....	100
EXP. No. 10. Spring Boot Part 2 .....	117



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

## **EXP. No. 1. Introduction to Android Programming**

### **1. Objectives**

- ❖ Download Android Studio software.
- ❖ Create a “Hello World” Android Application.
- ❖ Design a New Application and understand the various parts of an Android.
- ❖ Install and run the application on Android Emulator and physical device.

### **2. Introduction**

The goal of this lab is to learn the fundamentals of developing Android Applications, from project creation to installation on a physical device. More specifically, you should gain the knowledge of how to use basic development tools to support the application development process, as well as the key components of an Android application itself.

## ***2.1. Activity Classes***

There are four major types of component classes in any Android application:

- **Activities:** Much like a Form for a web page, activities display a user interface for performing a single task. An example of an Activity class would be one which displays a Login Screen to the user.
- **Services:** These differ from Activities in that they have no user interface. Services run in the background to perform some sort of task. An example of a Service class would be one which fetches your email from a web server.
- **Broadcast Receivers:** The sole purpose of components of this type is to receive and react to broadcast announcements which are either initiated by system code or other applications. If you've ever done any work with Java Swing, you can think of these like Event Handlers. For example, a broadcast announcement may be made to signal that a Wi-Fi connection has been established. A Broadcast Receiver for an email application listening for that broadcast may then trigger a Service to fetch your email.
- **Content Providers:** Components of this type function to provide data from their application to other applications. Components of this type would allow an email application to use the phone's existing contact list application for looking up and retrieving an email address.

## ***2.2. Android Manifest file***

Every project has a file with this exact name in the root directory. It contains all the information about the application that Android will need to run it: item Package name used to identify the application. item List of Activities, Services, Broadcast Receivers, and Content Provider classes and all their necessary information, including permissions. item System Permissions the application must define in order to make use of various system resources, like GPS. item Application defined permissions that other applications must have in order to interact with this application. item Application profiling information. item Libraries and API levels that the application will use.

### ***2.3. R.java Class***

This is a special static class that is used for referencing the data contained in your resource files. If you open this file you will see several static inner classes for each of the resource types, as well as static constant integers within them. Notice that the names of the member variables are the same as the names of the values in your resource files. Each value in a resource file is associated with an integer ID, and that ID is stored in a member variable of the same name, within a static class named after its data type.

Android projects has 2 parts: Design and Coding. The components that are taken in design window are given a specific id. This id is given so that during multiple identical components i.e. 2 textview or 2 buttons, they can be distinctly recognized. When you do the coding part, and you want to make a component functional then you first initialize it using findViewById method. Its identical to the initialization of data variable in C language. This statement just connects designing with the coding on per component basis.

## **3. Procedure**

Before starting a new project, you need to be connected to the internet once you create a new project. In this lab you are asked to build a simple application to get the user's name from text field and then display it on label. You will be creating an Activity class which will allow the user to enter their name into a text field and press a button.

### ***3.1. Creating new Android Project in Android Studio***

Before moving on to slightly more advanced topics, now is a good time to validate that all the required development packages are installed and functioning correctly. The best way to achieve this goal is to create an Android application and compile and run it. This section will cover the creation of a simple Android application project using Android Studio. Once the project has been created, the next section will explore the use of the Android emulator environment to perform a test run of the application.

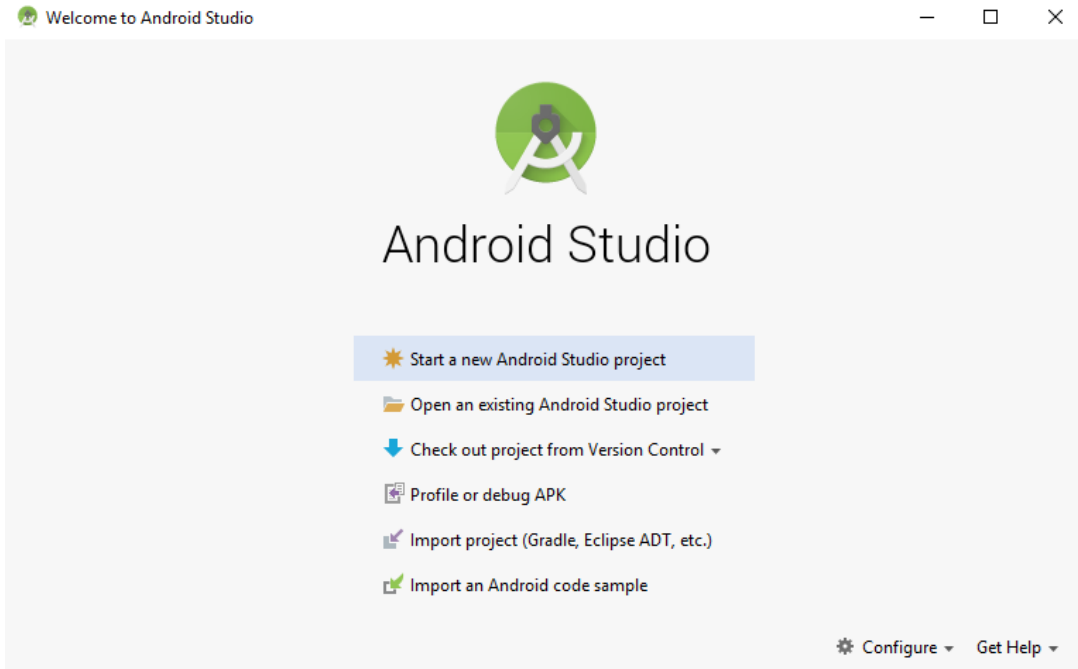


Figure 1.1 welcome to Android Studio Screen

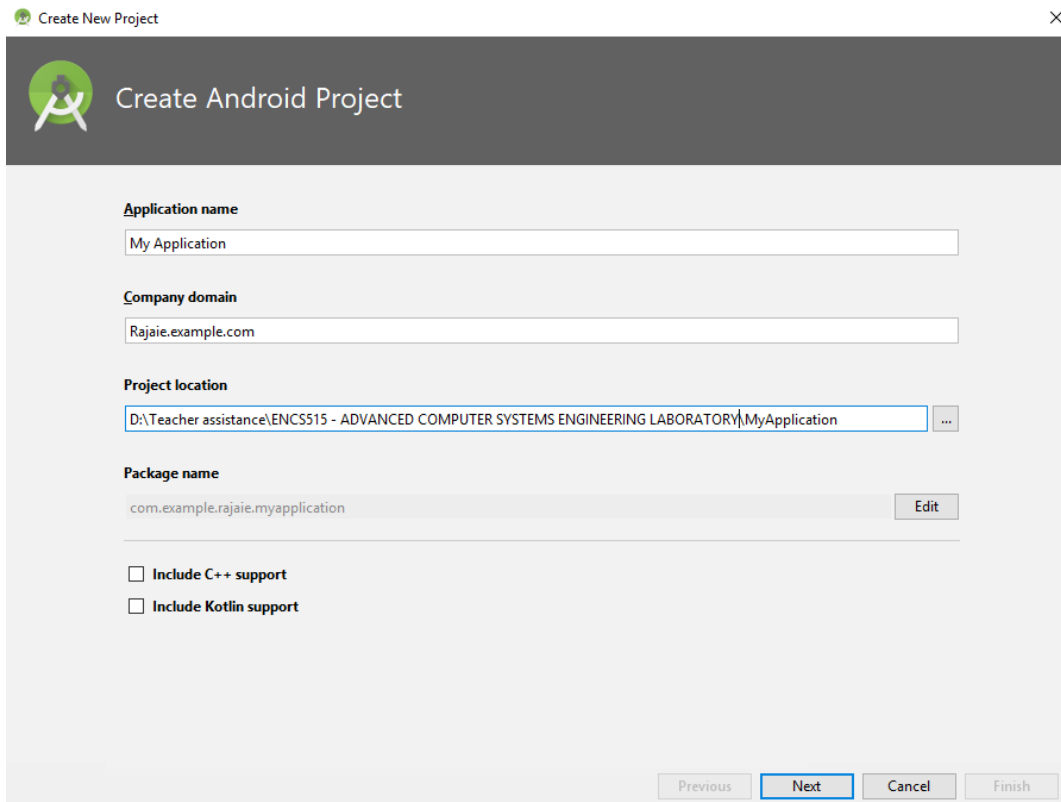


Figure 1.2 Create new Project Screen

### ***3.2. Creating a New Android Project***

The first step in the application development process is to create a new project within the Android Studio environment. Begin, therefore, by launching Android Studio so that the “Welcome to Android Studio” screen appears as illustrated in Figure 1.1.

Once this window appears, Android Studio is ready for a new project to be created. To create the new project, simply click on the Start a new Android Studio project option to display the first screen of the New Project wizard as shown in Figure 1.2.

### ***3.3. Defining the Project and SDK Settings***

In the New Project window, set the Application name field to “My Application”. The application name is the name by which the application will be referenced and identified within Android Studio and is also the name that will be used when the completed application goes on sale in the Google Play store. The Package Name is used to uniquely identify the application within the Android application ecosystem. It should be based on the reversed URL of your domain name followed by the name of the application. For example, if your domain is `www.mycompany.com`, and the application has been named “My Application”, then the package name might be specified as follows: `com.mycompany.myapplication` If you do not have a domain name, you may also use `ebookfrenzy.com` for the purposes of testing, though this will need to be changed before an application can be published: `com.ebookfrenzy.myapplication` The Project location setting will default to a location in the folder named `AndroidStudioProjects` located in your home directory and may be changed by clicking on the button to the right of the text field containing the current path setting. Click Next to proceed. On Target Android Devices screen as shown in Figure 1.3, enable the Phone and Tablet option and set the minimum SDK setting to API 15: Android 4.0.3 (IceCreamSandwich) and click on Next. The reason for selecting an older SDK release is that this ensures that the finished application will be able to run on the widest possible range of Android devices.

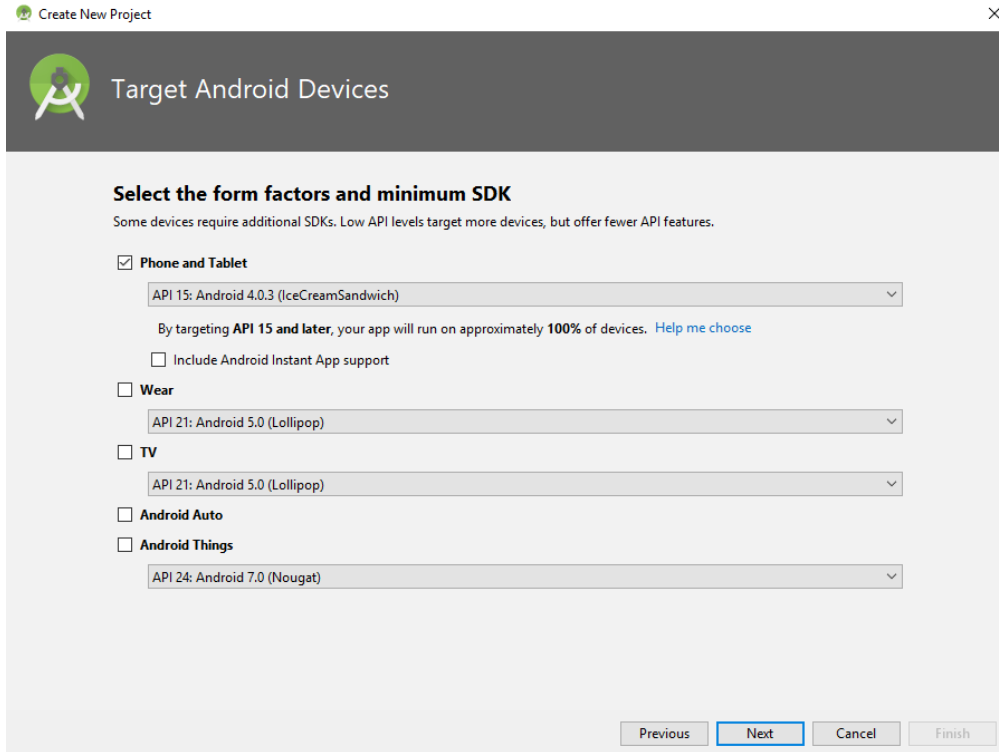


Figure 1.3 Target Android Devices Screen

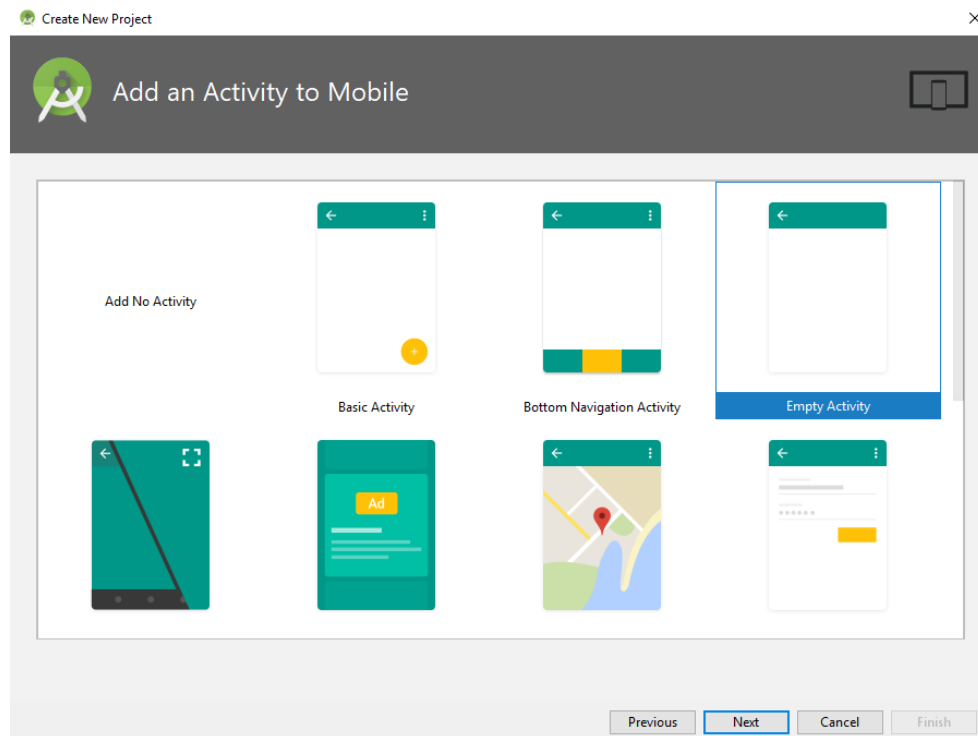


Figure 1.4 Add an Activity to Mobile Screen



### 3.4. Creating an Activity

The next step is to define the type of initial activity that is to be created for the application. A range of different activity types is available when developing Android applications. For the purposes of this example, however, simply select the option to create an Empty Activity as shown in Figure 1.4.

With the Empty Activity option selected, click Next. On the final screen as shown in Figure 1.5 name the activity and title MainActivity. The activity will consist of a single user interface screen layout which, for the purposes of this example, should be named activity\_main shown in Figure 1.5. Finally, click on Finish to initiate the project creation process.

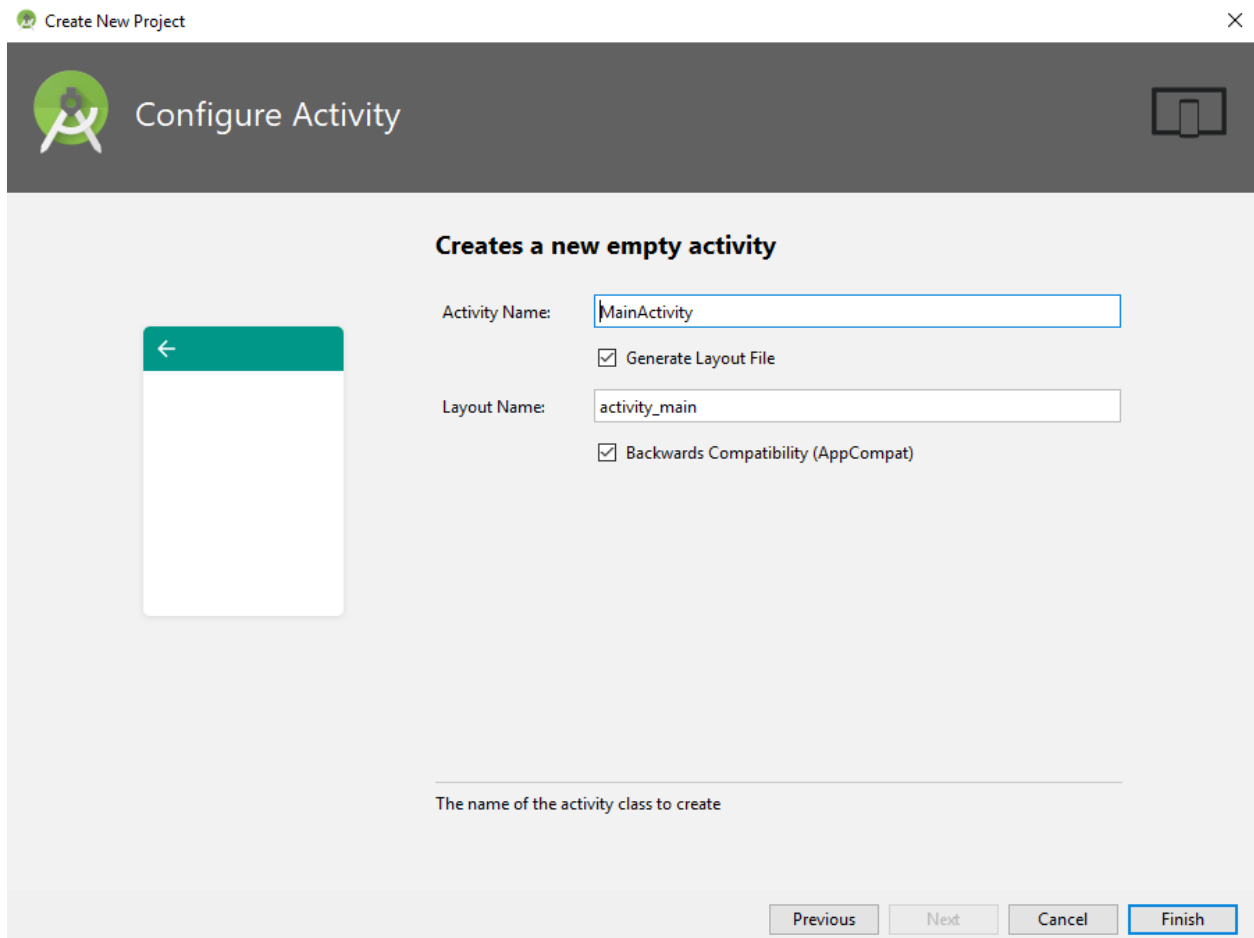


Figure 1.5 Configure Activity Screen

### 3.5. Modifying the Created Application

At this point, Android Studio has created a minimal example application project and opened the main window as shown in Figure 1.6.

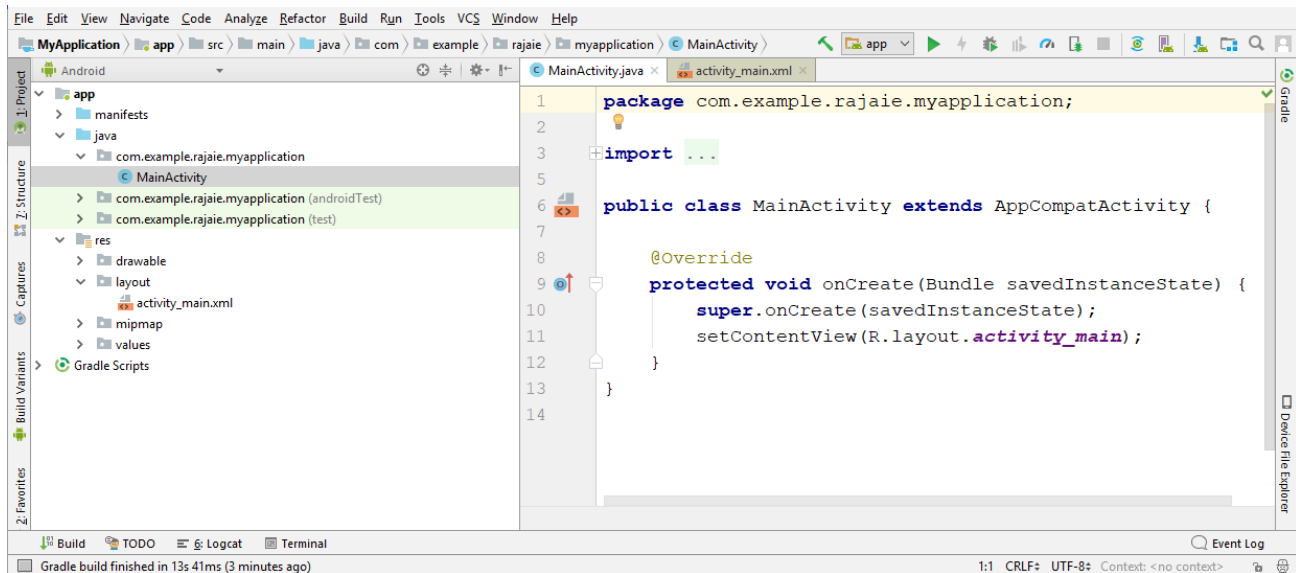
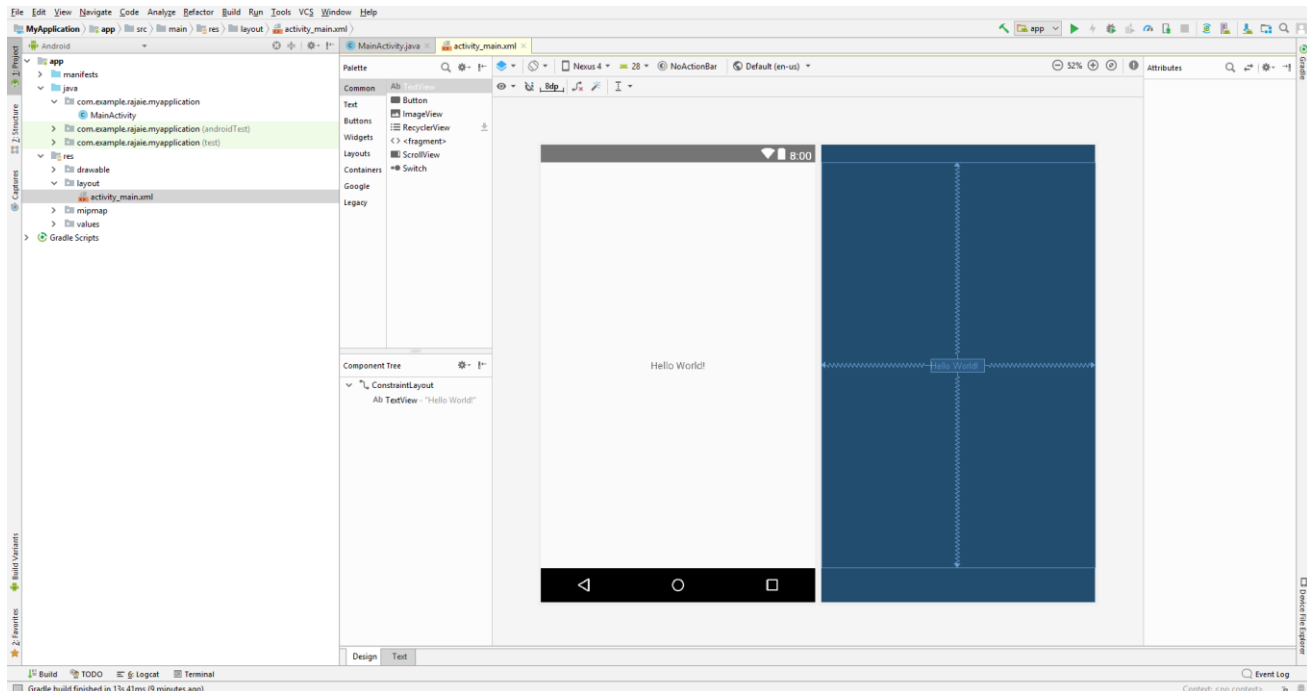


Figure 1.6 Android Studio Application screen

The newly created project and references to associated files are listed in the Project tool window located on the left-hand side of the main project window. The Project tool window has several modes in which information can be displayed. By default, this panel will be in Android mode. This setting is controlled by the drop-down menu at the top of the panel. If the panel is not currently in Android mode, click on this menu and switch to Android mode.

when finished to proceed to the HelloWorld greeting Activity. The example project created for us when you selected the option to create an activity consists of a user interface containing a label that will read “Hello World” when the application is executed.

The user interface design for our activity is stored in a file located under app, res, layout in the project file hierarchy. Using the Project tool window, locate this file as shown in Figure 1.7.



*Figure 1.7 Layout Screen*

Once located, double click on the file to load it into the User Interface Designer tool which will appear in the center panel of the Android Studio main window. In the toolbar across the top of Designer window is a menu currently set to Nexus 4 which is reflected in the visual representation of the device within the Designer panel. A wide range of other device options are available for selection by clicking on this menu.

As can be seen in the device screen, the layout already includes a label that displays a Hello World! message. Running down the left-hand side of the panel is a palette containing different categories of user interface components that may be used to construct a user interface, such as buttons, labels and text fields.

- The first step in modifying the application is to delete the “Hello World!” TextView component from the design. Begin by clicking on the TextView object within the user interface view so that it appears with a blue border around it. Once selected, press the Delete key on the keyboard to remove the object from the layout.
- The Android Studio Designer tool also provides an alternative to dragging and dropping components from the palette on to the design layout. Components may also be added by

selecting the required object from the palette and then simply clicking on the layout at the location where the component is to be placed

- Start by pressing right click on the constraint layout (which is considered as the main layout) and click on convert layout to convert it to linear Layout as shown in Figure 1.8.
- Set the orientation to Vertical by selecting the Linear layout on the left-hand of the panel and changing the orientation to Vertical from the right-side of the panel as in Figure 1.9.
- Find and drag Plain Text which exists in Text fields on the left-side of the panel as shown in Figure 1.10. Then find and drag Button and TextView Form Widgets panel on the left-side of the panel. You can see the layout for the application in Figure 1.11.

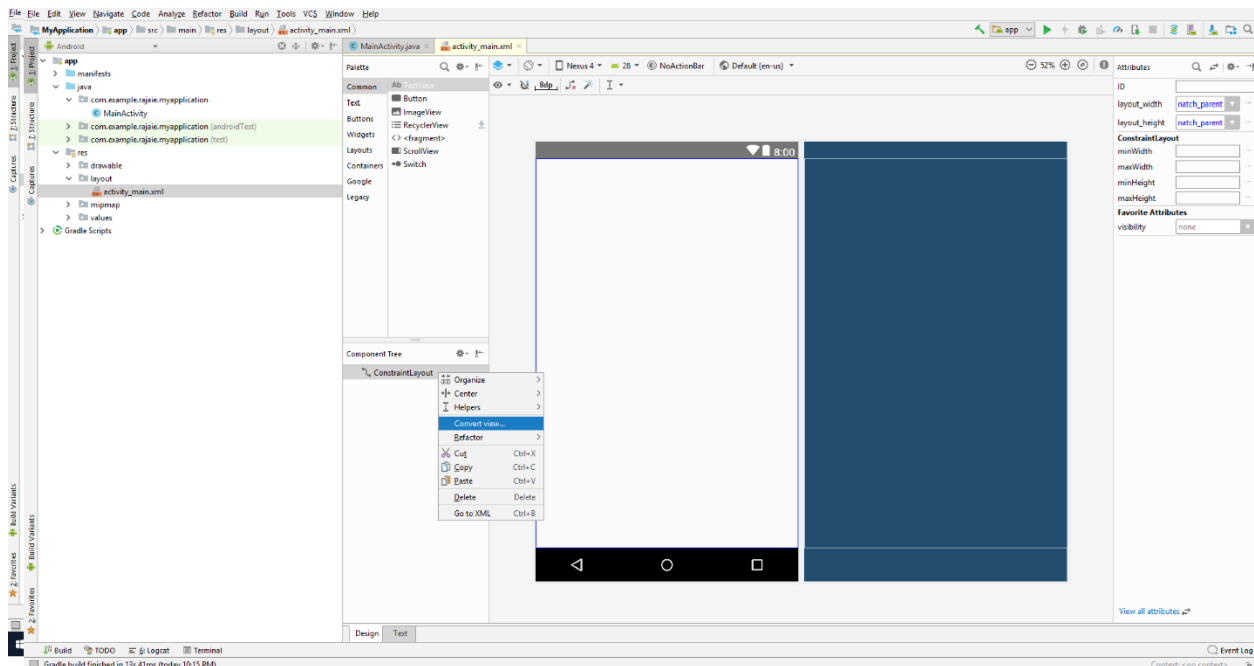


Figure 1.8 Converting the Constraint layout Screen

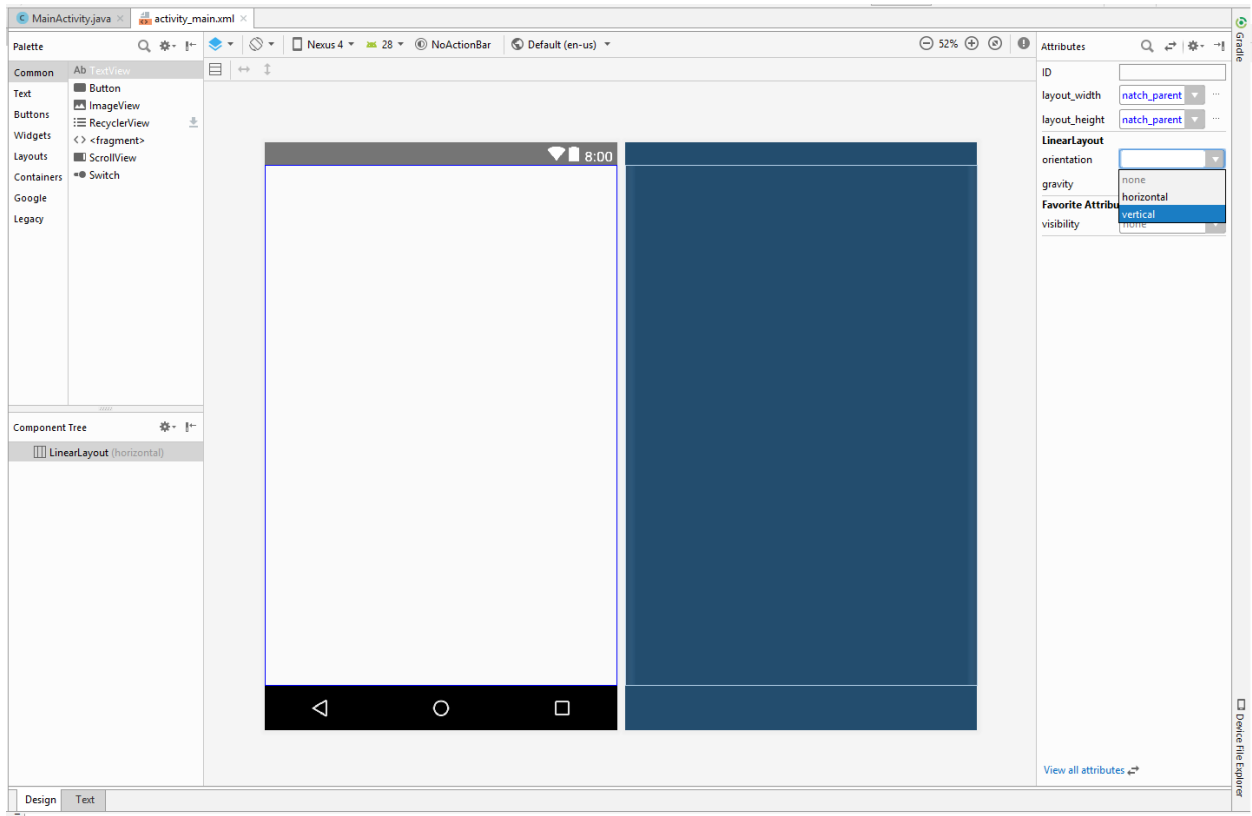


Figure 1.9 Changing the Layout to Vertical Layout Screen

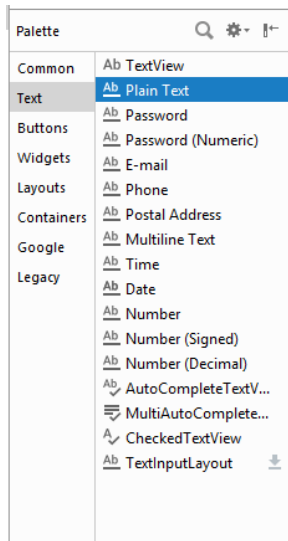


Figure 1.10 Adding Plain Text Screen



Figure 1.11 Application Layout Design

- Open MainActivity.java file. Add the following code inside the onCreate method in the MainActivity Class (don't remove the first two lines in the onCreate method) (the setContentView is for linking the MainActivity java file to the activity\_main layout using the R.java file).

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        EditText editText=(EditText) findViewById(R.id.editText);
        TextView textView=(TextView) findViewById(R.id.textView);
        textView.setText(editText.getText().toString());
    }
});
```

- Your application is ready now, you can build the application and test it by running it on a real device or on a virtual device.

## 4. Todo

Ask the Instructor or the Teacher Assistant for the todo (it will be specified at the end of the lab) you should perform the todo on your own machine and show it next lab.



**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.


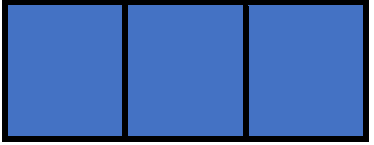
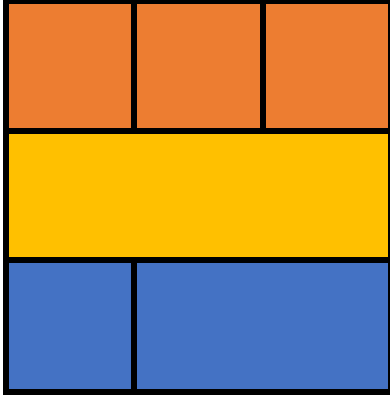
Linear Layout	Relative Layout	Web View
<ul style="list-style-type: none"> <li>• Vertical</li> </ul>  <ul style="list-style-type: none"> <li>• Horizontal</li> </ul> 		<pre data-bbox="1052 1052 1419 1457"> &lt;html&gt;  &lt;!-- web page --&gt;  &lt;/html&gt;</pre>
<p>A layout that organizes its children into a single horizontal or vertical view. It creates a scrollbar if the length of the window exceeds the length of the screen.</p>	<p>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).</p>	<p>Displays Web Pages.</p>

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.

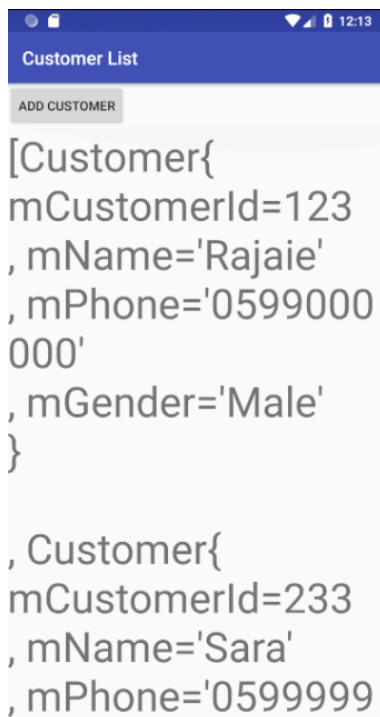


Figure 2.2.a Main Activity

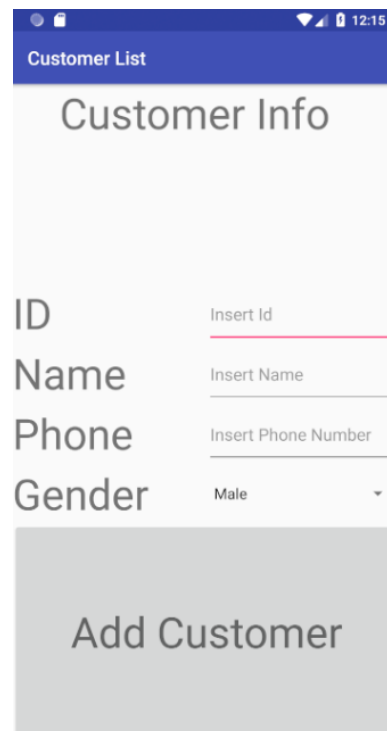


Figure 2.2.b Add Customer Activity

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.

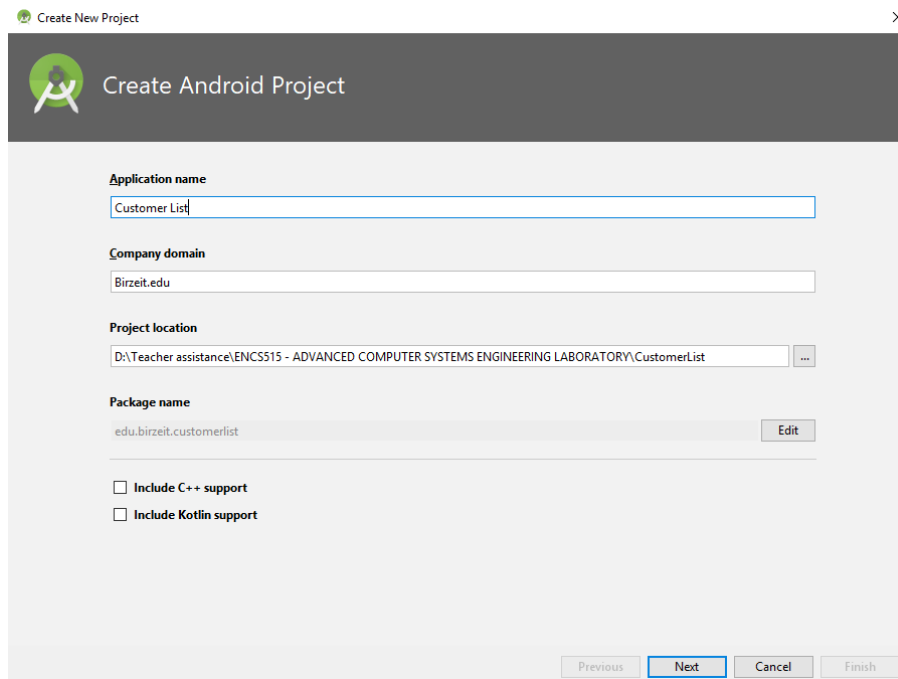


Figure 2.3 Creating new Project Screen.

### 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.

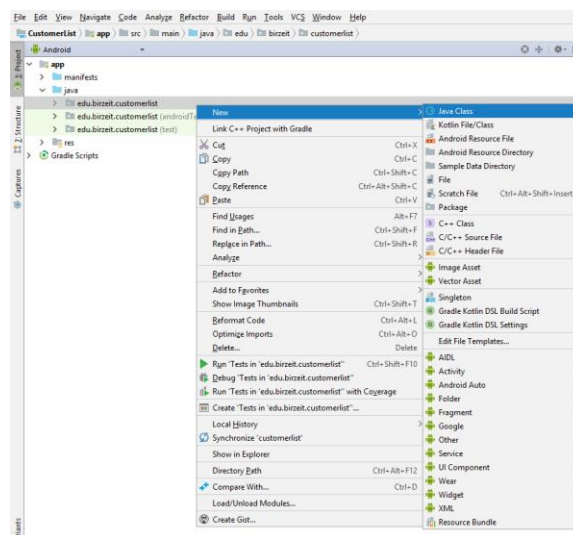


Figure 2.4 Adding New Java Class Screen

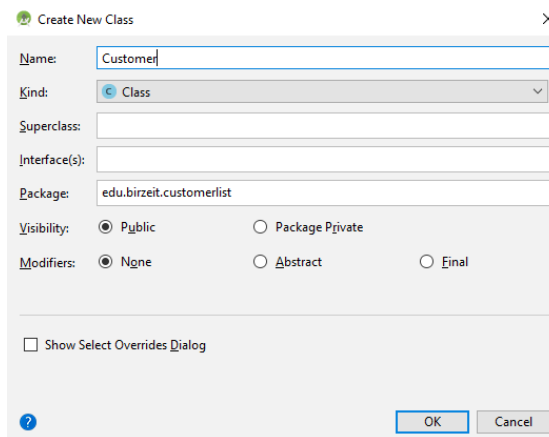
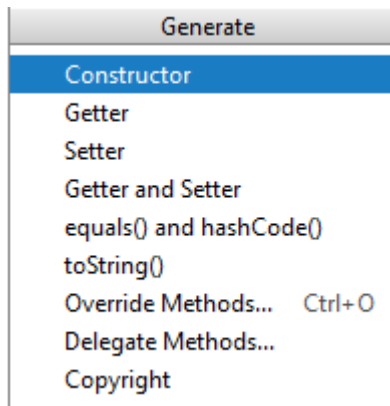
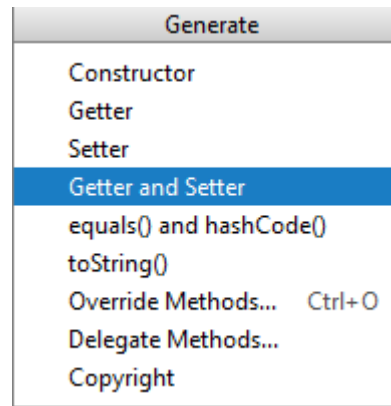


Figure 2.5 Adding Customer java Class Screen

- 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.

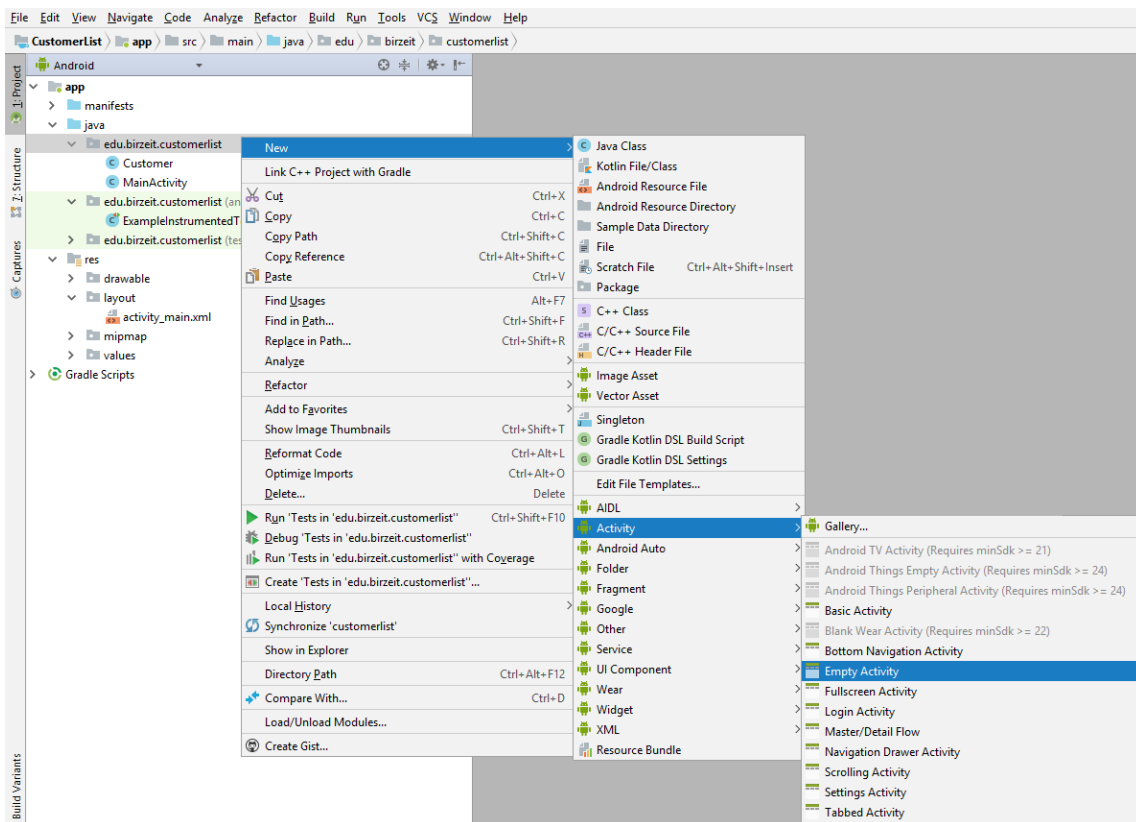


Figure 2.7 Adding Empty Activity

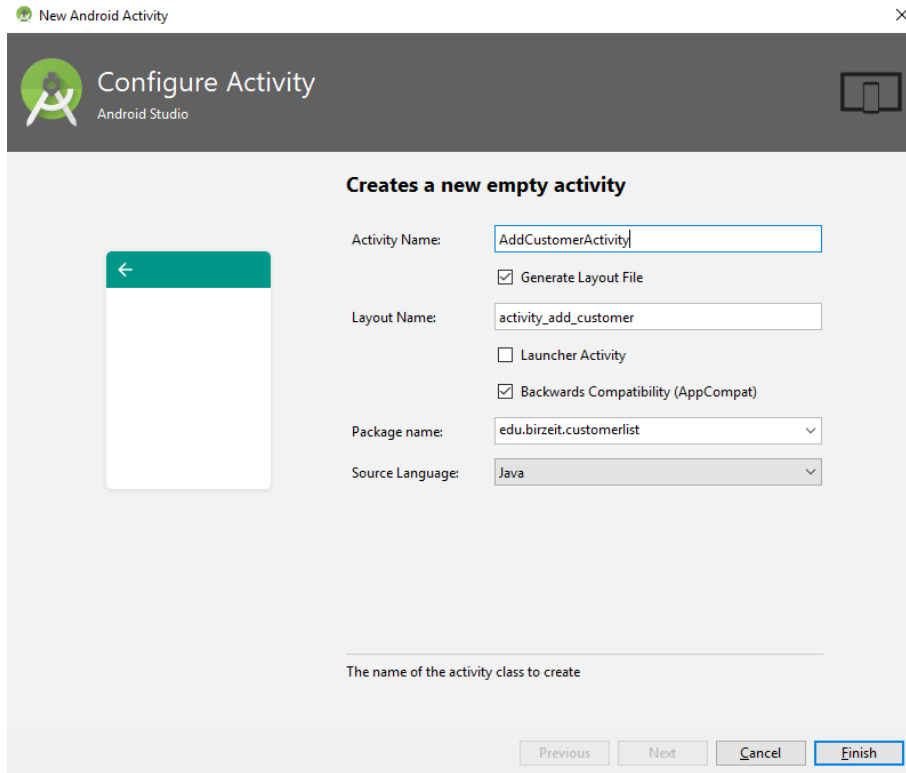


Figure 2.8 New Android Activity Screen

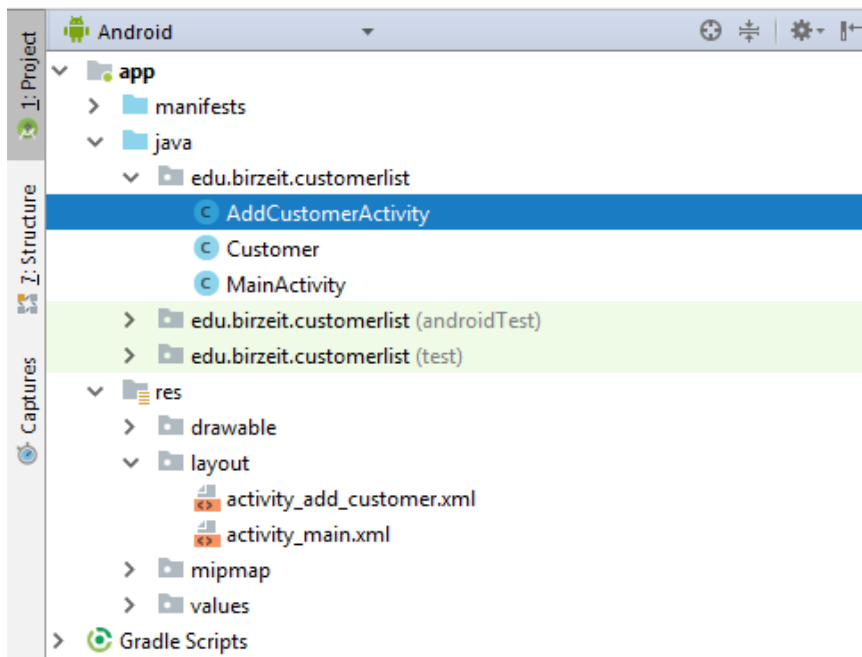
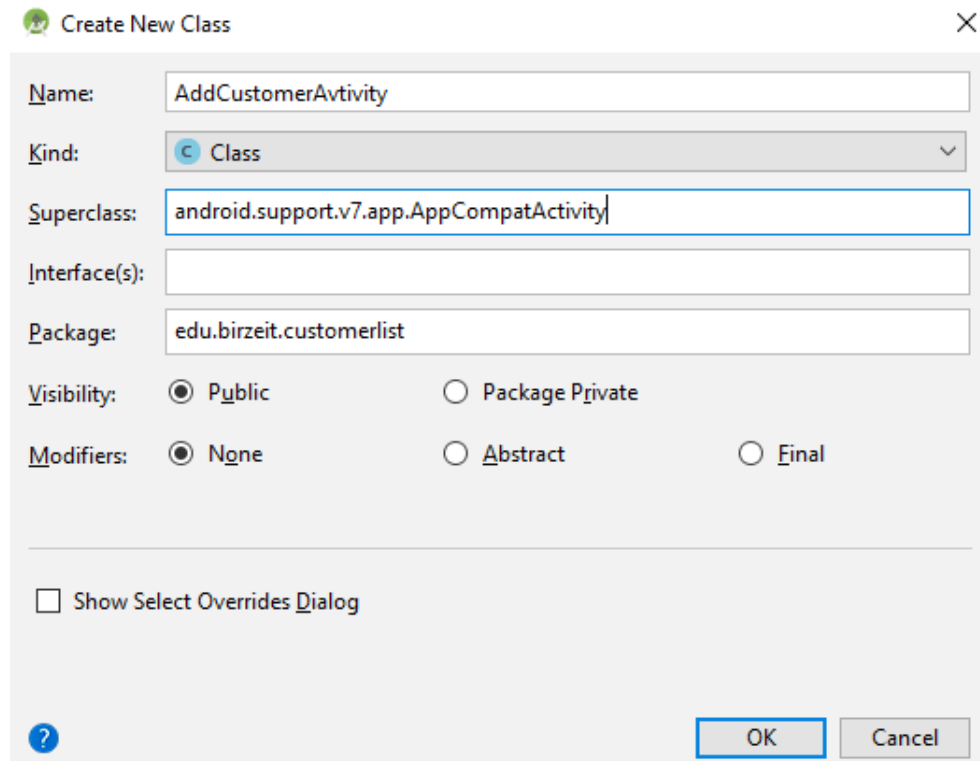


Figure 2.9 Activity java and layout

❖ 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*



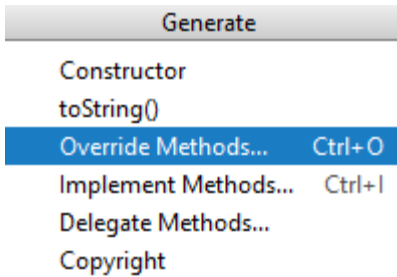


Figure 2.11.a

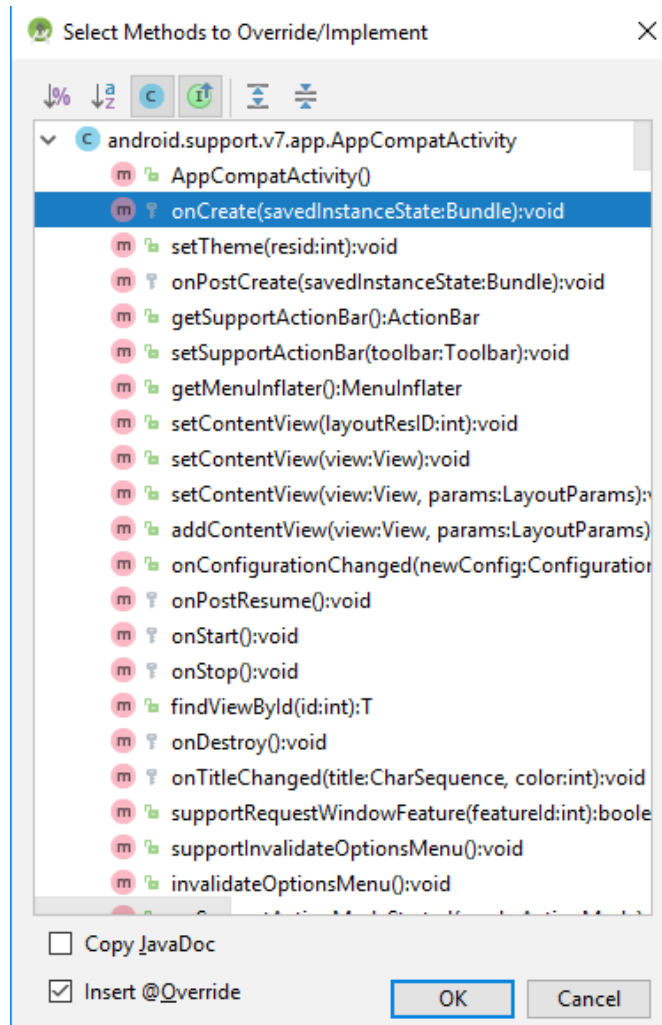


Figure 2.11.b

```
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 then 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.

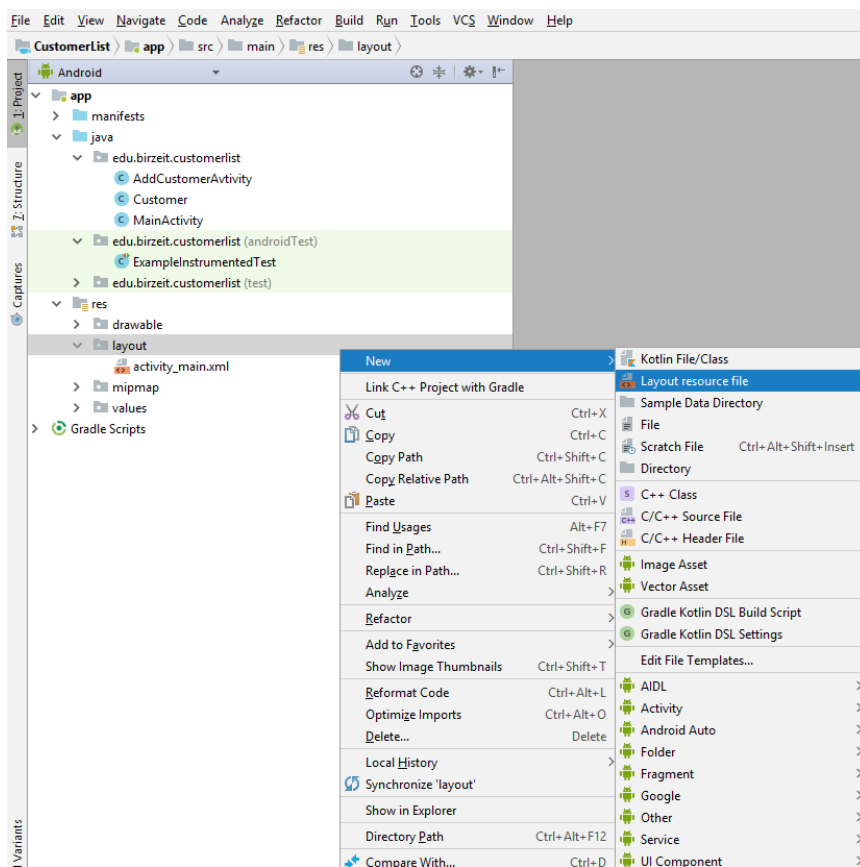


Figure 2.12 Adding New .xml Layout Screen

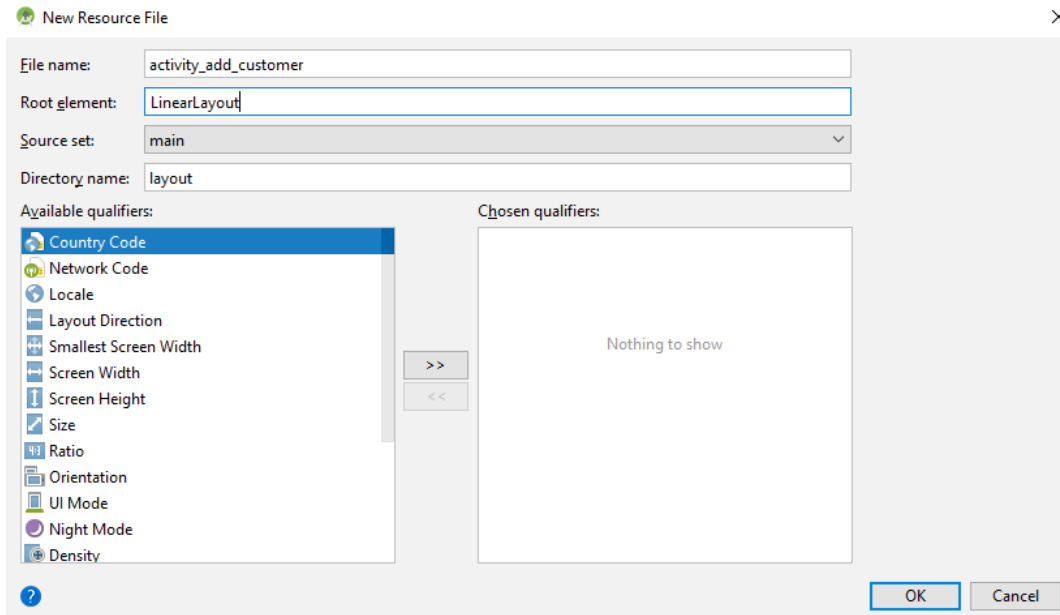


Figure 2.13 Adding activity\_add\_customer Layout Screen

```

<?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:supportsRtl="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>

```

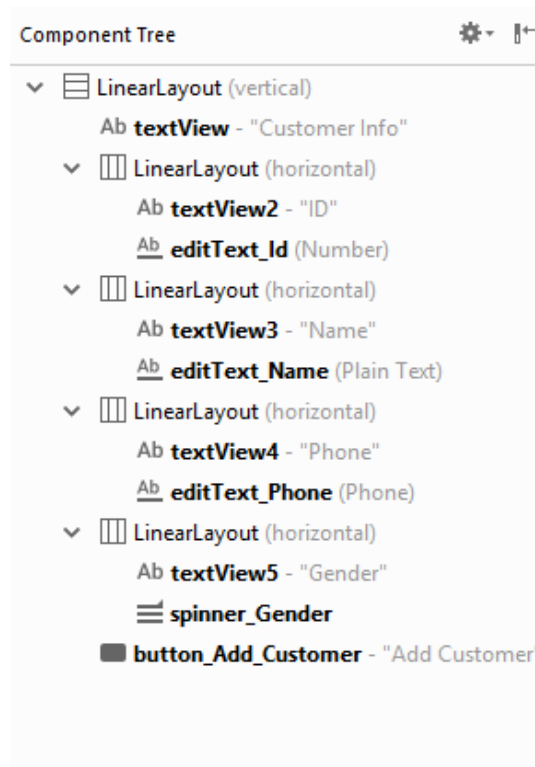
Figure 2.14 Adding the Activity in the Manifests file

### 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\_hight 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\_hight 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\_hight 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\_high 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_height` 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.



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

## **EXP. No. 3. Using Intents and Notifications**

### **1. Objectives**

- ❖ Install google play services in the Android Emulator.
- ❖ Using Intent class to apply different functionalities.
- ❖ How to create Toast Notifications.
- ❖ How to incorporate Google Maps into an application.
- ❖ How to incorporate Gmail into an application.
- ❖ How to incorporate Dial Up into an application.
- ❖ How to post notifications in the Notification Bar

### **2. Introduction**

In this lab you are going to learn how intents are coded and implemented. And the importance of intents in any android application. Also, you will learn how to make Toast messages in any application.

#### **2.1. Intents In android:**

It is a data structure that represents an operation to be performed. Intents is constructed by one component that wants some work and it is received by one activity that can perform that work. For example, you can use the intent Class to start new activity or to perform a service.

## 2.2. Intent Fields:

➤ **Action:** String represents a desired operation. You can set the action of the activity by two ways:

- By passing it in the constructor:
- By setAction method

examples of the actions that can be performed are:

- ACTION\_MAIN: start as initial activity of app
- ACTION\_DIAL: dial a number
- ACTION\_EDIT: display data to edit
- ACTION\_SYNC: synchronize device data with server

➤ **Data:** data associated with the intent. It is formatted as a uniform resource identifier (URI). For example, to dial a number, you can use the following URI.

```
Intent intent=new Intent(Intent.ACTION_DIAL,Uri.parse("tel:+1555"));
```

➤ **Category:** additional information about the components that can handle the intent. For example, CATEGORY\_LAUNCHER: can be the initial activity a task.

➤ **Component:** the component that should receive this intent. Use this field when there is exactly one component that should receive the intent.

➤ **Extras:** what additional information you need to provide (key/value pairs).

## Types of Intents

- **Explicit intents:** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.
  - **Implicit intents:** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.
- Intent resolution: when the activity to be activated using implicit intents, android tries to find activity that match the intent by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device. Android package manager is responsible for deciding which component is best suited to handle your intent. Intent resolution relies on two kind of information:
    - ◆ An Intent describing a desired operation.
    - ◆ Intents filters which describe which operations an activity can handle. It is specified either in Android Manifest file or programmatically.

### ***2.3. User notifications:***

Notifications basically are messages outside the user interface of the application, for example, if you download a book from the internet, you need to use the application while the book is downloading and let the user know when the download finishes. So, the developer should display the message to user contains that information.

In this experiment we will talk about two kinds of user notifications

### ➤ **Toast Messages**

Transitory messages that pop up on the current window. It automatically fades into and out of the view without user interaction or response.

### ➤ **Notification Area** (Status Bar Notification)

Android provides the notification area for alerting users about events. It also provides a notification drawer that user can pull down to see more detailed information about notifications. The operations on the notification area are managed by system service.

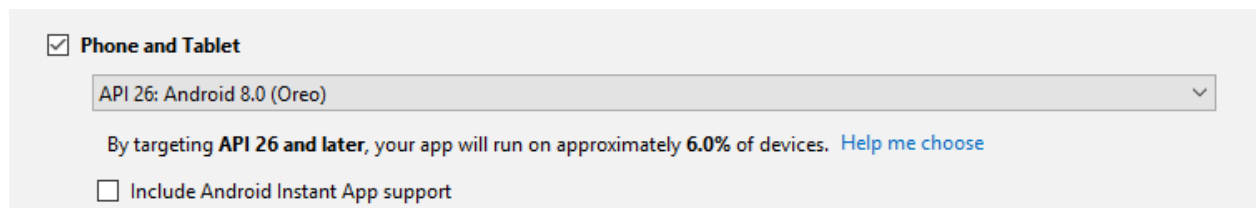
## **3. Procedure**

In this Lab you are going to create an application that has 5 buttons that will perform the following operations: open Dial (Phone) using intents, open Gmail using intents, open google maps using intents, display a notification message, display a toast message.

For displaying notification, you will create a channel to display the notification so the API for the android must not be less than 26 (Android 8.0 Oreo).

### ***3.1. Creating a new Project with API 26***

Start by Creating a new Project and change the name for “Intents And Notifications” then click on next, change the API for Phone and Tablet for API 26 (Android 8.0 Oreo) as shown in Figure 3.1. Then click next and finish.



*Figure 3.1 Create New Project API 26*

### 3.2. Adding Buttons

Convert the MainActivity root layout to LinearLayout and change the orientation to Vertical from the right-side panel. Add Five Button in the MainActivity Layout (activity\_main) and change the names and the Ids as shown in the component tree in Figure 3.2.

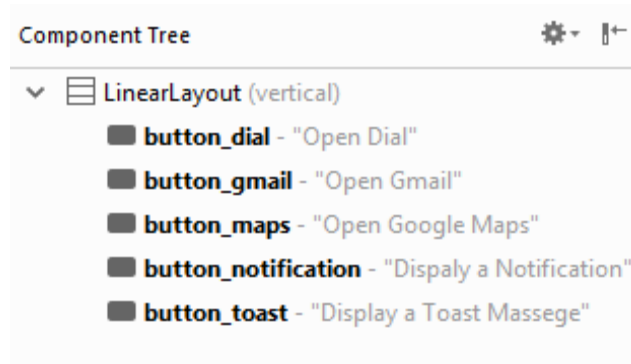


Figure 3.2 Component Tree Screen

### 3.3. Creating Listener for each Button

- Create five buttons in the MainActivity java file and find view by Id for each button as shown in the code below.

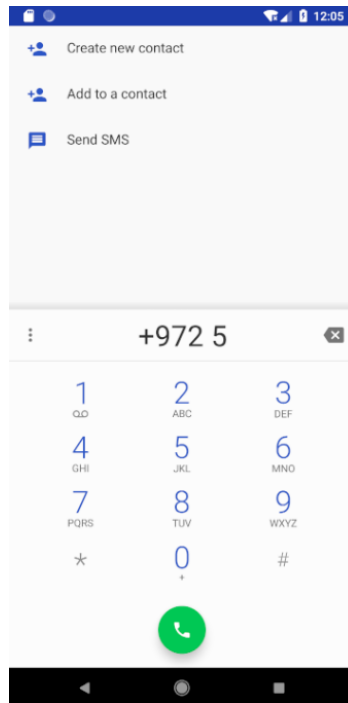
```
Button dialButton = (Button) findViewById(R.id.button_dial);
Button gmailButton = (Button) findViewById(R.id.button_gmail);
Button mapsButton = (Button) findViewById(R.id.button_maps);
Button notificationButton = (Button) findViewById(R.id.button_notification);
Button toastButton = (Button) findViewById(R.id.button_toast);
```

- Create a click Listener to dial button
  - Create a new Object of Intent this will start a new activity
  - Set the action for the intent object to Intent.ACTION\_DIAL
  - Set the data for the intent object to Uri.parse("tel:+9725")
  - Execute the intent by calling startActivity method and pass the intent to this method
  - the following code shows how these steps and Figure 3.3 shows the result when clicking the button.

```

dialButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent dialIntent =new Intent();
        dialIntent.setAction(Intent.ACTION_DIAL);
        dialIntent.setData(Uri.parse("tel:+9725"));
        startActivity(dialIntent);
    }
});

```



*Figure 3.3 Dial Screen*

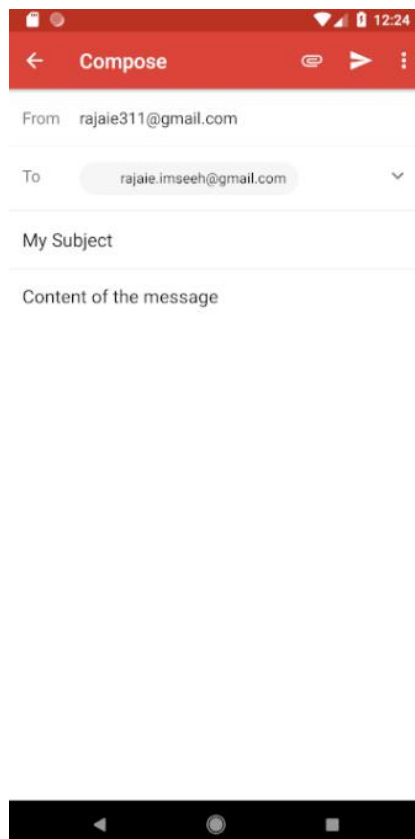
➤ Create a click Listener to gmail button

- Create a new Object of Intent call it “gmailIntent” this will start a new activity
- Set the action for the intent object to Intent.SENDTO
- Set the type for the intent object to “message/rfc822”
- Set the data for the intent object to Uri.parse(“mailto:”)
- Execute the intent by calling startActivity method and pass the intent to this method
- the following code shows how these steps and Figure 3.4 shows the result when clicking the button.



- You can set the sent to mail, subject and the text of the mail in the extras as shown in the code below
- Send the message to your email and check it to be sure that the message was sent successfully.

```
gmailButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent gmailIntent =new Intent();
        gmailIntent.setAction(Intent.ACTION_SENDTO);
        gmailIntent.setType("message/rfc822");
        gmailIntent.setData(Uri.parse("mailto:"));
        gmailIntent.putExtra(Intent.EXTRA_EMAIL,"rajaie.imseeh@gmail.com");
        gmailIntent.putExtra(Intent.EXTRA_SUBJECT,"My Subject");
        gmailIntent.putExtra(Intent.EXTRA_TEXT,"Content of the message");
        startActivity(gmailIntent);
    }
});
```



*Figure 3.4 Send Gmail Screen*

➤ Create a click Listener to Google Maps button

- Create a new Object of Intent this will start a new activity
- Set the action for the intent object to Intent.ACTION\_VIEW
- Set the data for the intent object to Uri.parse("geo:19.076,72.8777")
- Execute the intent by calling startActivity method and pass the intent to this method
- the following code shows how these steps and Figure 3.5 shows the result when clicking the button.

```
mapsButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Intent mapsIntent =new Intent();  
        mapsIntent.setAction(Intent.ACTION_VIEW);  
        mapsIntent.setData(Uri.parse("geo:19.076,72.8777"));  
        startActivity(mapsIntent);  
    }  
});
```

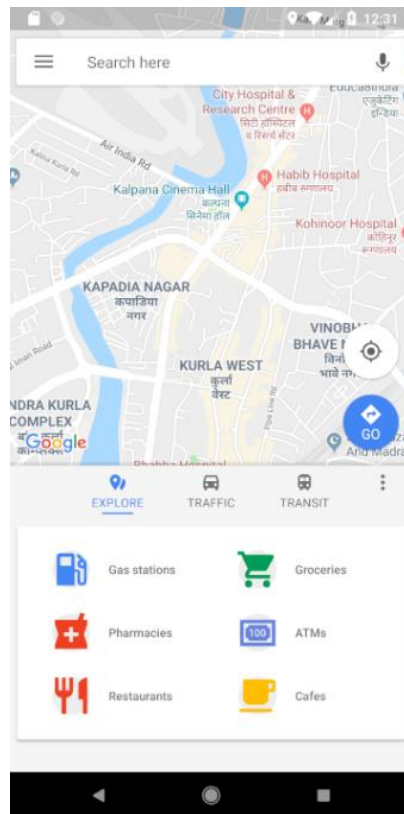


Figure 3.5 Google Maps Screen

## ➤ Notification Creation

- To create a notification, you need first to create a notification channel this can be implemented by creating a method in the main activity to create a channel. Every notification channel has a channel Id (create as global constant variable) and name it MY\_CHANNEL\_ID, a channel Name(create as global constant variable) and name it MY\_CHANNEL\_NAME and an Importance (this will determine how much the notification should interrupt the user, higher the importance in the notification, the more interruptive the notification will be).
- Using the notification manager, you will create the notification channel, the code below is the implementation of the notification channel.

```
private static final String MY_CHANNEL_ID = "my_chanel_1";
private static final String MY_CHANNEL_NAME = "My channel";

private void createNotificationChannel() {
    int importance = NotificationManager.IMPORTANCE_DEFAULT;
    NotificationChannel channel = new NotificationChannel(MY_CHANNEL_ID,
MY_CHANNEL_NAME, importance);
    NotificationManager notificationManager =
getSystemService(NotificationManager.class);
    if (notificationManager != null) {
        notificationManager.createNotificationChannel(channel);
    }
}
```

- After that you will create another method that will create the notification in this channel you created, this method will also be implemented in the main activity. You will need the following components shown in Table 3.1 to create the notification.

*Table 3.1 needed components for creating the notification*

Intent	This is for starting an activity when clicking the notification.
Pending Intent	This is created to pass it to the notification using setContentIntent.
Notification Title	The title of the notification (global constant) passed to the method
Notification Body	The Body of the notification (global constant) passed to the method
Notification small Icon	Use the mipmap ic_launcher
Property	Set to default

- Using the notification manager compat notify the created notification, this method will need an Id (create as global constant).
- The code below shows the method to create the notification.

```
private static final int NOTIFICATION_ID = 123;
private static final String NOTIFICATION_TITLE = "Notification Title";
private static final String NOTIFICATION_BODY = "This is the body of my notification";

public void createNotification(String title, String body) {
    Intent intent = new Intent(this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

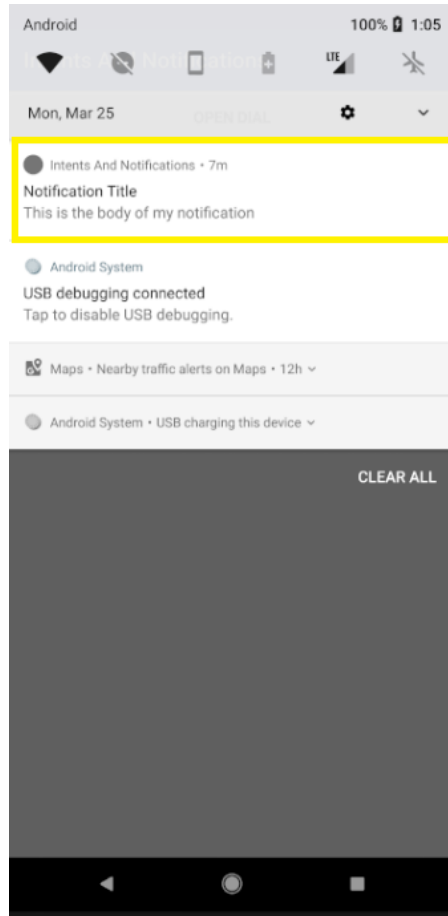
    createNotificationChannel();

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this,
        MY_CHANNEL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle(title)
        .setContentText(body)
        .setStyle(new NotificationCompat.BigTextStyle().bigText(body))
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent);
    NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);
    notificationManager.notify(NOTIFICATION_ID, builder.build());
}
```

- Create a click Listener to Notification button
- Call the createNotification method which will create the notification channel and the notification to notify it, the code below shows how to create the listener and call the createNotification method passing the title and the body to the method.

```
notificationButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        createNotification(NOTIFICATION_TITLE, NOTIFICATION_BODY);
    }
});
```

- Figure 3.6 Shows the notification created after clicking the notification button.

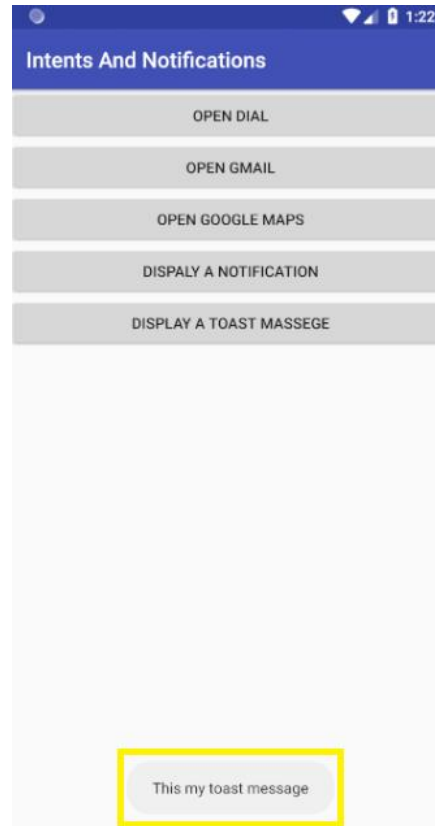


*Figure 3.6 Notification displaying Screen*

- Create a click Listener for the Toast Message button
  - To create a Toast Message, you will need to create a Toast object and pass the application context, text for the toast message, and the duration (Length short for short duration toast message and Length Long for long duration toast message).
  - Then call the method show() to show the toast message. The code below shows how to create a toast message and Figure 3.7 shows the toast message after clicking the toast message button.

```
private static final String TOAST_TEXT = "This my toast message";

toastButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast toast =Toast.makeText(MainActivity.this,
        TOAST_TEXT,Toast.LENGTH_SHORT);
        toast.show();
    }
});
```



*Figure 3.7 Toast Message Displaying Screen*

## 4. Todo

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



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

## **EXP. No. 4. SQLite Database**

### **1. Objectives**

- ❖ How to create your own database using SQLite.
- ❖ How to create custom Views from scratch to suit a specific need.
- ❖ How to create Confirmation Dialogs.

### **2. Introduction**

In this lab, you will be learning how to save data to a database for repeating or structured data, such as contact information. This experiment assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android.

#### ***2.1. Define a schema and contract***

One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database. You may find it helpful to create a companion class, known as a contract class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

A contract class is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.

## ***2.2. Create a database using an SQL helper***

Once you have defined how your database looks, you should implement methods that create and maintain the database and tables. Just like files that you save on the device's internal storage, Android stores your database in your app's private folder. Your data is secure, because by default this area is not accessible to other apps or the user. The `SQLiteOpenHelper` class contains a useful set of APIs for managing your database. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. All you need to do is call `getWritableDatabase()` or `getReadableDatabase()`.

## ***2.3. Put information into a database***

Insert data into the database by passing a `ContentValues` object to the `insert()`. The first argument for `insert()` is simply the table name. The second argument tells the framework what to do if the `ContentValues` is empty. The `insert()` methods returns the ID for the newly created row, or it will return -1 if there was an error inserting the data. This can happen if you have a conflict with pre-existing data in the database.

## ***2.4. Read information from a database***

To read from a database use the `query()` method, passing it your selection criteria and desired columns. The method combines elements of `insert()` and `update()`, except the column list defines the data you want to fetch (the "projection"), rather than the data to insert. The results of the query are returned to you in a `Cursor` object.



## ***2.5. onCreate and onResume methods in Activities***

### ➤ onCreate()

You must implement this callback, which fires when the system first creates the activity. On activity creation, the activity enters the Created state.

### ➤ onResume()

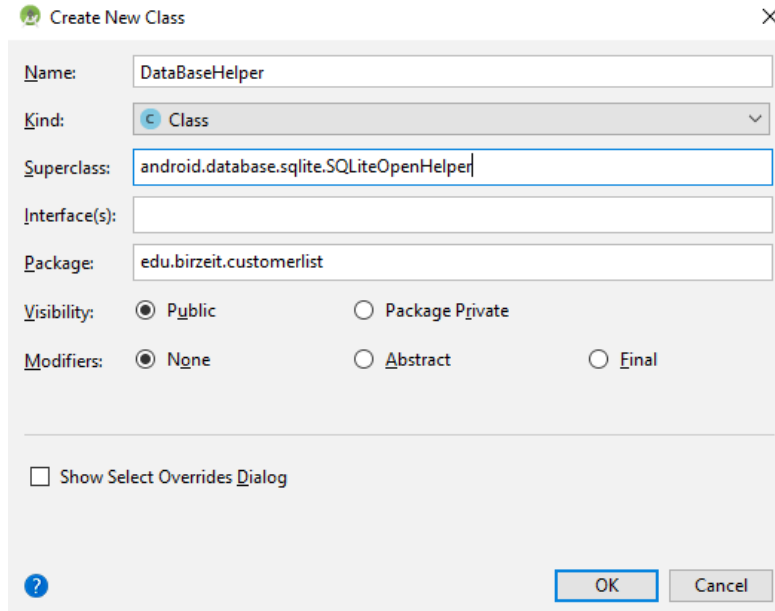
When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback. This is the state in which the app interacts with the user.

## **3. Procedure**

You are going to upgrade the second experiment to save the customers in your app's private folder using SQLiteOpenHelper API, rather than saving it in array List as you did in the second experiment. You will not change the layout of the second experiment so start by opening the second experiment project.

### ***3.1. Creating DataBaseHelper class that extends SQLiteOpenHelper Class***

Create a new class and call it DataBaseHelper by right-clicking on the package name to add a new class, fill the name to “DataBaseHelper” and the Superclass to “android.database.sqlite.SQLiteOpenHelper” and then click ok as shown in Figure 4.1.



*Figure 4.1 Adding DataBaseHelper Class*

- Add a constructor to the DataBaseHelper class, this constructor will call the super constructor which will create the data base, this constructor will take four values (Context, name of the database, factory, and the version of the data base). As shown in the code below.

```
public DataBaseHelper(Context context, String name,
    SQLiteDatabase.CursorFactory factory, int version) {
    super(context, name, factory, version);
}
```

- override the onCreate and OnUpgrade methods

As shown in Figure 4.2 override the onCreate and onUpgrade methods. onCreate will create the table and onUpgrade will update the table if needed in the future, In this lab you will implement onCreate method.

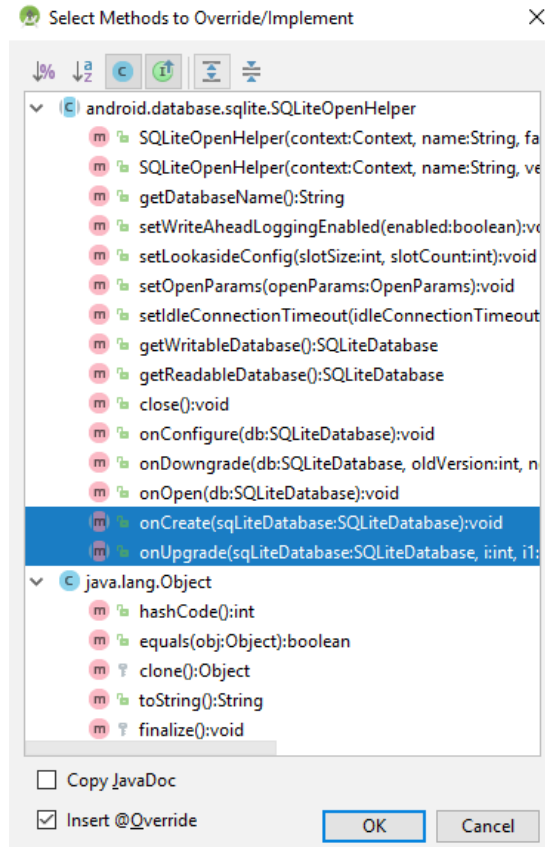


Figure 4.2 Overriding `onCreate` and `onUpgrade` methods

- implementing `onCreate` method

You will execute the query for creating a customer table, this table has ID column with prototype Long and it's the primary key, a NAME column with prototype Text, a PHONE column with prototype Text and a GENDER entry with prototype Text as shown in the code below.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    sqLiteDatabase.execSQL("CREATE TABLE CUSTOMER(ID LONG PRIMARY
KEY,NAME TEXT, PHONE TEXT,GENDER TEXT) ")
};
}
```

- implement a method to add a customer

Before inserting you will create an object from `SQLiteDataBase` which will call the `getWritableDatabase()` method, this will give access to write to the database. To add a Customer to the data base you will use the `insert` method from the `SQLiteDataBase` class. This method takes three values, the first is the table name to insert the value, the second will be set to `null` (If you specify `null`, the framework does not insert a row when there are no values) and a `ContentValues` which the entry that will be used to be added to the `DataBase`. The code below shows the method to insert an entry to the `Customer` Table.

```
public void insertCustomer(Customer customer) {
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("ID", customer.getmCustomerId());
    contentValues.put("NAME", customer.getmName());
    contentValues.put("PHONE", customer.getmPhone());
    contentValues.put("GENDER", customer.getmGender());
    sqLiteDatabase.insert("CUSTOMER", null, contentValues);
}
```

- implementing a method to get all Customers from Customer table

To get entry from the database tables, you will create an object from `SQLiteDataBase` which will call the `getReadableDatabase()` method, this will give access to read from the database. To get all customers you will execute a raw query which will select from the customer table all the entries. The returned value will be as `Cursor`. (read about `Cursor` to know how to deal with them [Link](#)). The code below shows how to get all customers from the `Customer` table in the database.

```
public Cursor getAllCustomers() {
    SQLiteDatabase sqLiteDatabase = getReadableDatabase();
    return sqLiteDatabase.rawQuery("SELECT * FROM CUSTOMER", null);
}
```

### ***3.2. Removing the ArrayList and saving the Values to the Customer Table***

- Go to the Customer Class, remove the customersArrayList (since you will not need for it)
- In the AddCustomerActivity you will replace the Customer.customersArrayList.add(newCustomer) with a new code to add to the Customer table in the data base.
- First create an object from the DataBaseHelper class. And pass the context (AddCustomerActivity.this), name of the database “EXP4”, the factory to “null” and the version to 1 as shown in the code below. Then call the insertCustomer method by passing the customer object to it.

```
DataBaseHelper dataBaseHelper =new  
DataBaseHelper (AddCustomerActivity.this, "EXP4", null, 1);  
dataBaseHelper.insertCustomer (newCustomer);
```

### ***3.3. Displaying all Customers that were added to the Customer Table.***

- In the MainActivity make the declaration of the secondLinearLayout global. (only the declaration. Do not create new object, the creation must be in the onCreate method)
- Remove the code which display the old Customer.customersArrayList.
- Override the onResume method.
- Get All customers from the Customer table form the database
- First create an object from the DataBaseHelper class. And pass the context (MainActivity.this), name of the database “EXP4”, the factory to “null” and the version to 1 as shown in the code below.
- Create a Cursor and call the getAllCustomers method the returned value will be saved in the created Cursor.

- Display All Customers returned from the database
  - Remove all old views from the secondLinearLayout by calling removeAllViews Method.
  - Display all customers returned from the Cursor in textViews
  - Add the textviews to the second Linerlayout.
  
- The code below shows how the implementation of onResume method

```
protected void onResume() {
    super.onResume();
    DataBaseHelper dataBaseHelper =new
    DataBaseHelper(MainActivity.this,"EXP4", null,1);
    Cursor allCustomersCursor = dataBaseHelper.getAllCustomers();
    secondLinearLayout.removeAllViews();
    while (allCustomersCursor.moveToNext()){
        TextView textView =new TextView(MainActivity.this);
        textView.setText(
            "Id= "+allCustomersCursor.getString(0)
            +"\nName= "+allCustomersCursor.getString(1)
            +"\nPhone= "+allCustomersCursor.getString(2)
            +"\nGender= "+allCustomersCursor.getString(3)
            +"\n\n"
        );
        secondLinearLayout.addView(textView);
    }
}
```

- The output is shown in Figure 4.3



*Figure 4.3 Experiment 4 Output*

## **4. Todo**

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



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

## **EXP. No. 5. Frame Animation and Tween Animation in Android**

### **1. Objectives**

- ❖ To provide a good understanding of how to use Frame animation and Tween animation in android on any View.
- ❖ To provide knowledge of some attributes of translate, rotate and scale tags.
- ❖ To provide familiarity with Animation Class.

### **2. Introduction**

Animations add vivacity and personality to your apps. There are two types of animations that you can do with the Animation framework of Android: Frame animation and Tween animation.

#### ***2.1. Frame animation***

Is series of frames is drawn one after the other at regular. creates an animation by showing a sequence of images in order with an AnimationDrawable.



## 2.2. Tween animation

Are simple transformations of position, size, rotation to the content of a View. Animation can be defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.

- Animation Class: Animation class is used to hold an animation which is loaded in it from the anim folder of resource directory. In the above code, the animation reference variable holds the animation xml files loaded using AnimationUtils class.
- AnimationUtils: AnimationUtils class is used to fetch an animation file from anim folder by using loadAnimation method of this class which has two arguments: First argument defines context in which the animation is to be loaded which must be the context of given activity on which the animation is to be loaded. Second argument defines the id of the animation in resources file.
- Main attributes in Tween animation
  - **xmlns:android:** This attribute must be defined in the root tag to get the schema path of this XML file.
  - **android:pivotX and android:pivotY:** Used for specifying the center point of the rotation.
  - **android:duration:** This attribute is used to define the duration in which the animation needs to be completed in milliseconds.
  - **android:fromXDelta:** This attribute is used to define the starting point of translation animation on Xaxis.
  - **android:toXDelta:** This attribute is used to define the ending point of translation animation on Xaxis. Values from -100 to 100 ending with "%", indicating a percentage relative to itself. Values from -100 to 100 ending in "%p", indicating a percentage relative to its parent. A float value with no suffix, indicating an absolute value.
  - **android:fromXScale:** Starting X size offset, where 1.0 is no change.
  - **android:toXScale:** Ending X size offset, where 1.0 is no change.
  - **android:fromYScale:** Starting Y size offset, where 1.0 is no change.
  - **android:toYScale:** Ending Y size offset, where 1.0 is no change.
  - **android:pivotX:** The X coordinate to remain fixed when the object is scaled.
  - **android:pivotY:** The Y coordinate to remain fixed when the object is scaled.

### 3. Procedure

You are going to build two different applications the first is by using frame animation and the second is by tween animation. You will need for the frame animation application several images. However, in tween one image is needed.

#### 3.1. *Frame animation Application*

Create a new project with the name “Frame Animation Application”. This project will exchange between two images. The idea behind a frame animation is simple: You’ll be cycling through a series of images very quickly, just like an old movie. The “frame” refers to a single image. Thus, the first step in creating a custom frame animation is to create a sequence of images.

You have two options here: you can use XML drawable (such as shape drawable) or actual image files. For the sake of simplicity, you are going to use the following series of PNG images. You should make sure to have images sized appropriately for different screen densities.



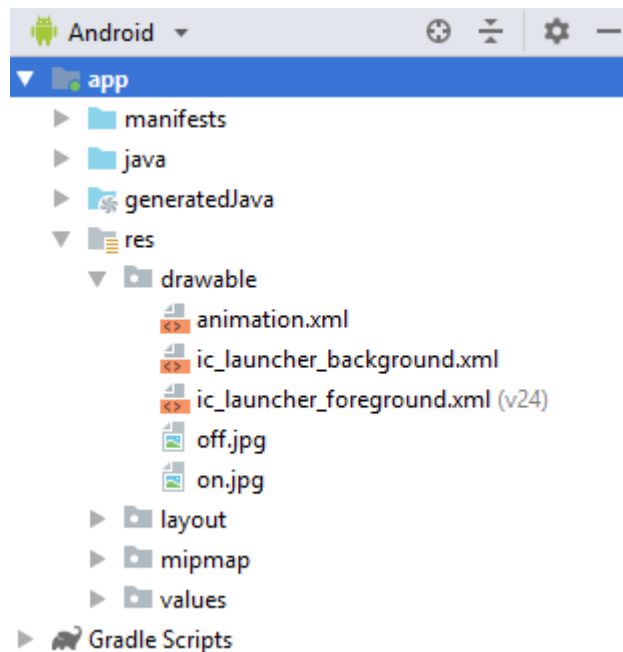
*Figure 5.1 First Image*



*Figure 5.2 Second Image*

- import those images into the res/drawable folder and call them off for Figure 5.1 and on for Figure 5.2 as shown in Figure 5.3
- Define an XML Drawable for our animation. We can use transition to make the Frame Animation. It just cycles through a sequence of provided images. Call this file “animation” and locate it in res/drawable as shown in Figure 5.3.
- The code below is to make transition between two images (on and off) this code should be in the animation.xml folder.

```
<?xml version="1.0" encoding="utf-8" ?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/off" />
    <item android:drawable="@drawable/on" />
</transition>
```



*Figure 5.3 Animation xml Code File and On and Off Images Location Screen*

- Add a button and an image view in the activity\_main layout as shown in Figure 5.4.
- Add @drawable/animation to the source compact when adding the image view as shown in Figure 5.5.

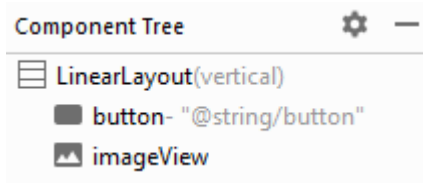


Figure 5.4 activity\_main layout

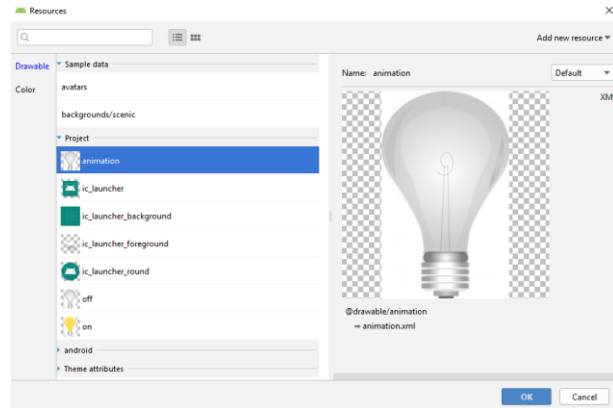
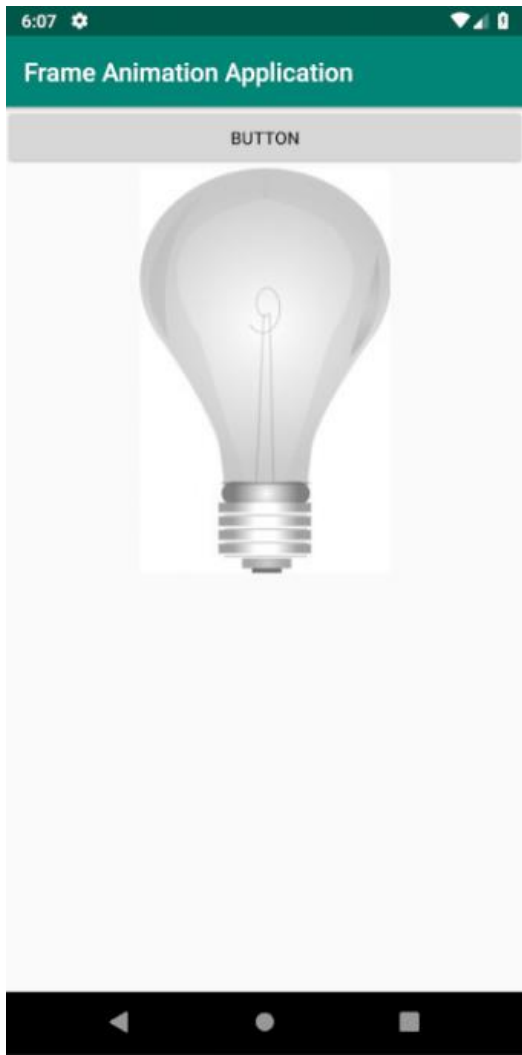


Figure 5.5 Adding the animation to the ImageView

- In the main activity java folder, you will make objects of Button and the ImageView and connect them with the button and the imageView in the layout folder as shown in the code below.
- Make a setOnClickListener method to the button.
- Make a TransitionDrawable object in the button click listener and start the transition by calling the startTransition method and pass the duration to this method in milli Second as shown in the code below. Figure 5.6 shows before clicking the button and Figure 5.7 after clicking the button.

```
Button button = (Button) findViewById(R.id.button);
final ImageView imageView = (ImageView)
findViewById(R.id.imageView);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        TransitionDrawable transitionDrawable = (TransitionDrawable)
imageView.getDrawable();
        transitionDrawable.startTransition(1000);
    }
});
```



*Figure 5.6 Before Clicking the Button*



*Figure 5.7 After Clicking the Button*

- Try to change the method from `startTransition` to `reverseTransition` and understand the difference between these methods.

### 3.2. Tween animation Application

Create a new project with the name “Tween Animation Application”. Tween Animation is defined as an animation which is used to Translate, Rotate, Scale and Alpha any type of view in Android.

- Add the image in Figure 5.8 to the res/drawable folder and call it image as shown in Figure 5.9.



Figure 5.8 Tween Image

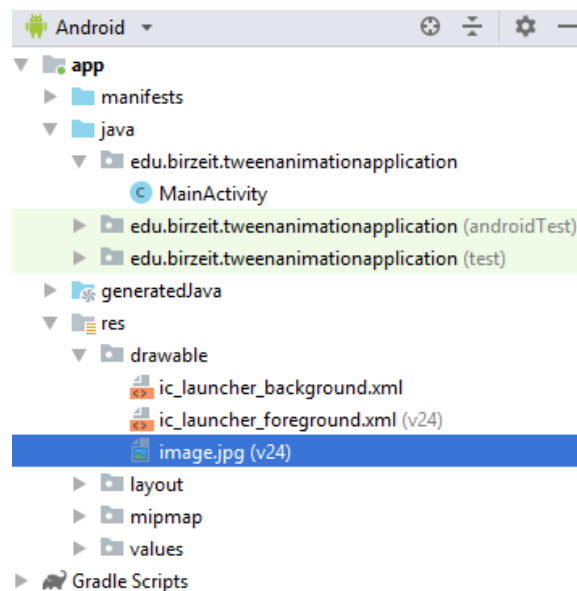


Figure 5.9 Showing drawable Directory

- In the res folder create a new Android Resource Directory as shown in Figure 5.10.

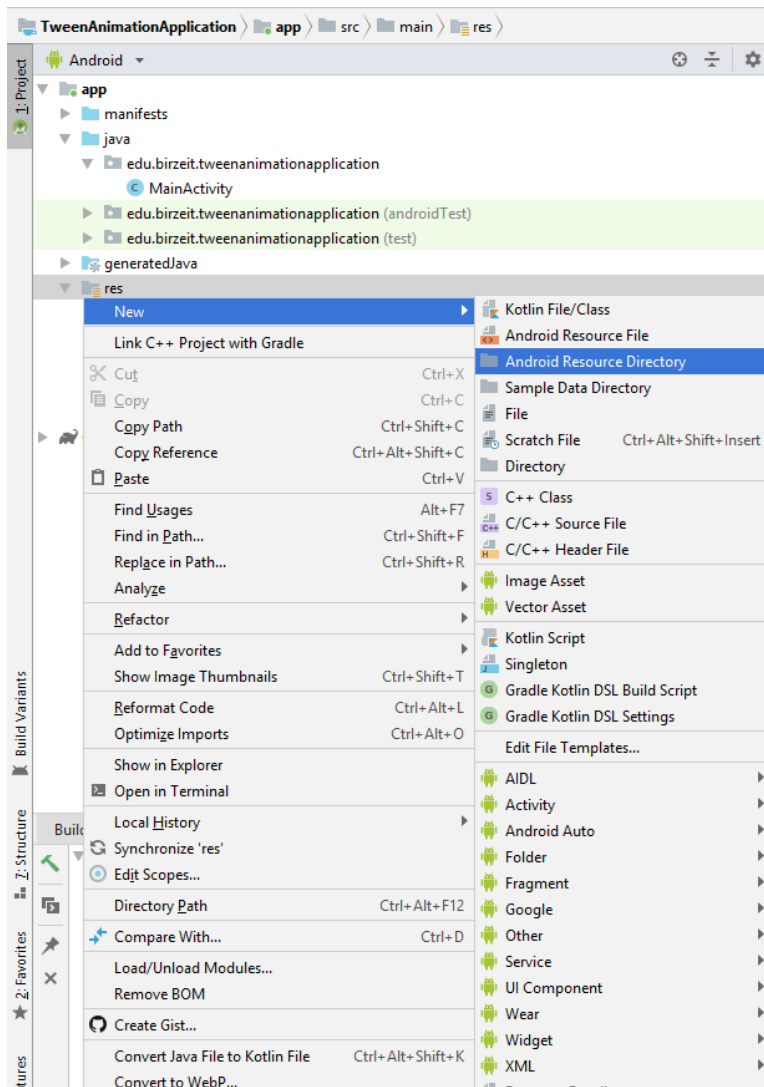
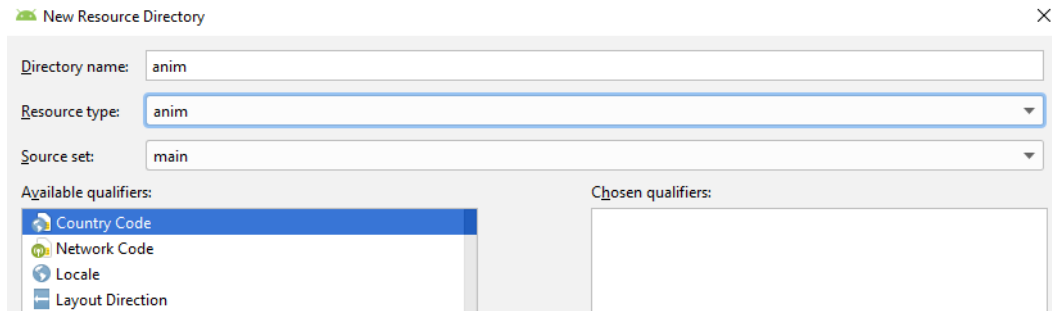


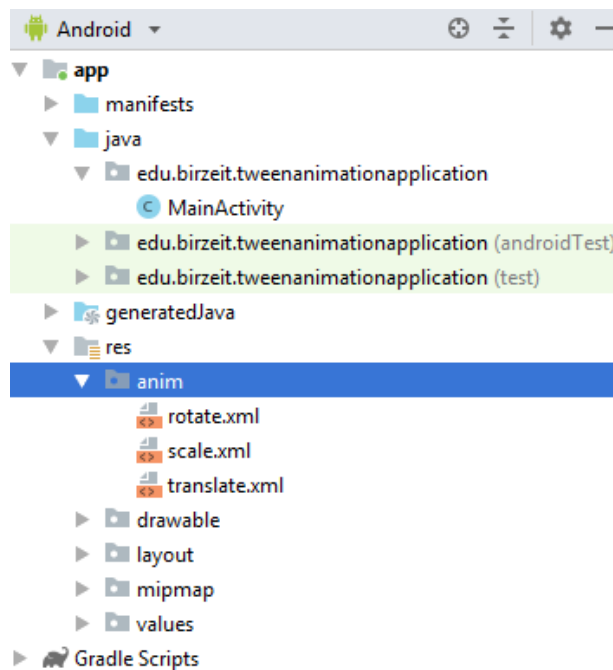
Figure 5.10 Creating Android Resource Directory Screen

- Change the Resource type to anim as show in Figure 5.11.



*Figure 5.11 New Resource Directory Screen*

- Create three Animation Resource File in the anim directory and call them rotate, scale and translate as shown in Figure 5.12.



*Figure 5.12 showing anim Directory*



- In the activity\_main layout makes the following design shown in Figure 5.13.

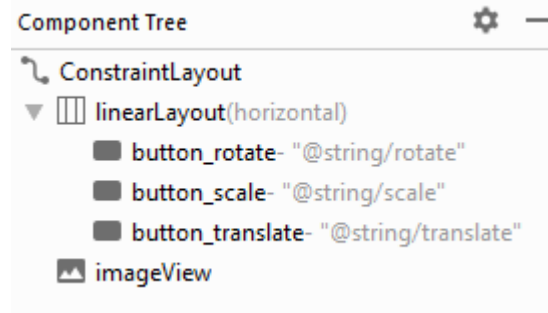


Figure 5.13 Component tree for Main Activity

- Add @drawable/image to the source compact when adding the image view as shown in Figure 5.14.

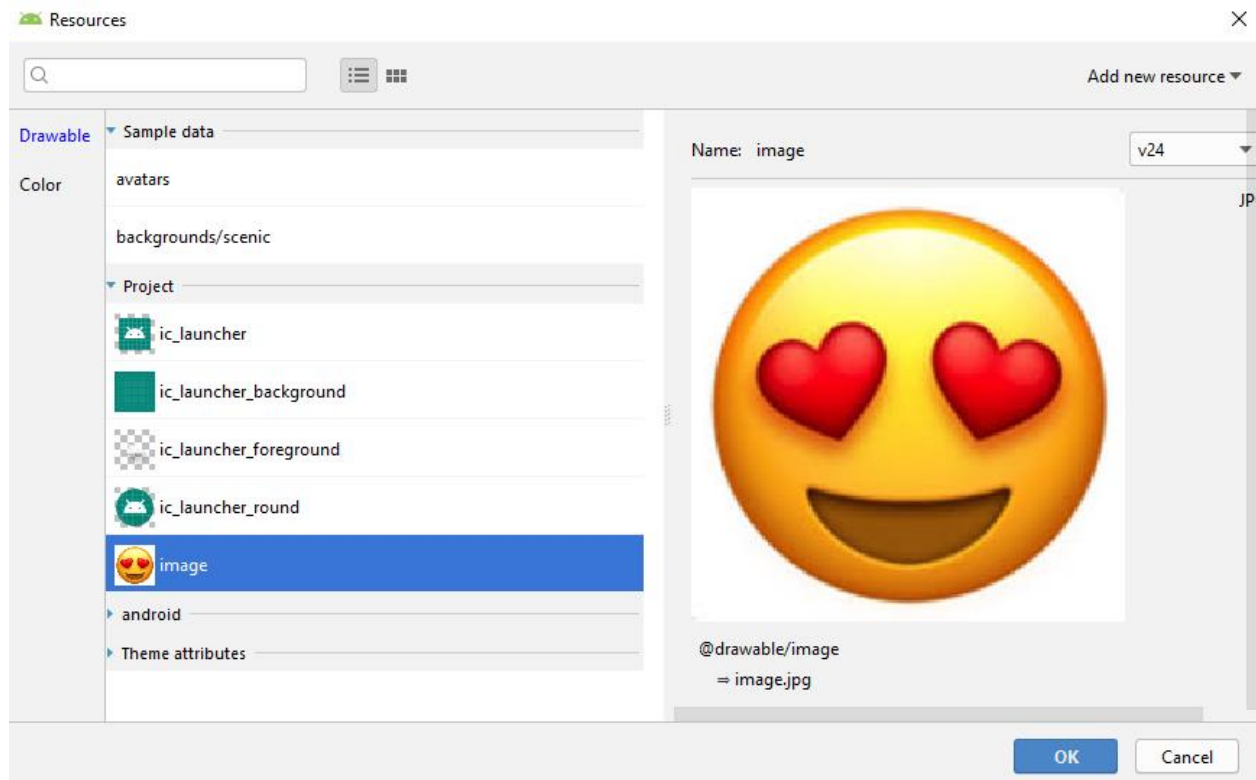


Figure 5.14 Adding the Image to the imageView

➤ Making the rotate animation (writing the code to the rotate.xml)

- Make the duration for the animation 3 seconds.
- Make the rotation 360 degree.
- Make the x coordination and the y coordination 50% (to itself).
- The following code is for the rotation.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <rotate
    android:duration="3000"
    android:fromDegrees="0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toDegrees="360" />
</set>
```

➤ Making the Scale animation (writing the code to the scale.xml)

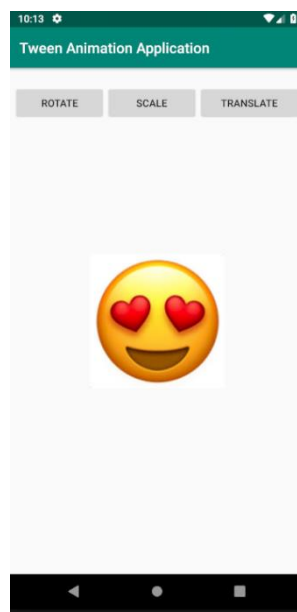
- Make the duration for the animation 2 seconds.
- Make the scale zoom in by making fromXScale=1, fromYScale=1 to toXScale=3, toYScale=3.
- Make the x coordination and the y coordination 50% (to itself).
- The following code is for the zoom in.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <scale
    android:duration="2000"
    android:fromXScale="1"
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="3"
    android:toYScale="3"/>
</set>
```

- Making the Translate animation (writing the code to the translate.xml)
  - Make the duration for the animation 1 seconds.
  - Make the translate in the x axis from 0% to 30% (to its parent view).
  - The following code is for the translate.

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate
    android:duration="3000"
    android:fromXDelta="0%p"
    android:toXDelta="30%p" />
</set>
```

- In the MainActivity.java make 3 buttons by connecting them to the three buttons in the main activity layout.
- Make an ImageView object by connecting it to the image view in the main activity layout.
- Make three setOnClickListeners to the three buttons and start the animation for each button by calling startAnimation method and passing the AnimationUtils.xml folder.
- The code below shows how to make the animation for the three buttons and the ImageView.
- The output is shown in



*Figure 5.15 Tween Animation Application*

```

Button buttonRotate=(Button) findViewById(R.id.button_rotate);
Button buttonScale=(Button) findViewById(R.id.button_scale);
Button buttonTranslate =(Button)
findViewById(R.id.button_translate);
final ImageView imageView = (ImageView)
findViewById(R.id.imageView);
buttonRotate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        imageView.startAnimation(AnimationUtils.loadAnimation(MainActivity.this,R.anim.rotate));
    }
});

buttonScale.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        imageView.startAnimation(AnimationUtils.loadAnimation(MainActivity.this,R.anim.scale));
    }
});

buttonTranslate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        imageView.startAnimation(AnimationUtils.loadAnimation(MainActivity.this,R.anim.translate));
    }
});

```

### 3.3. Try doing the following animations

- (You must know what the following code does)

- The first code

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:toAlpha="1.0" />
</set>

```

## ➤ The second Code

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator">
    <!-- Use startOffset to give delay between animations -->
    <!-- Move -->
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="300"
        android:toXDelta="75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="1100"
        android:toYDelta="70%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="1900"
        android:toXDelta="-75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="2700"
        android:toYDelta="-70%p" />
    <!-- Rotate 360 degrees -->
    <rotate
        android:duration="1000"
        android:fromDegrees="0"
        android:interpolator="@android:anim/cycle_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:startOffset="3800"
        android:toDegrees="360" />
</set>
```

## 4. Todo

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



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

## **EXP. No. 6. Singleton and Shared Preferences**

### **1. Objectives**

- ❖ Provide a simple knowledge of shared preferences in android.
- ❖ Introduce you for a new java concept about singleton classes.
- ❖ To provide the ability to save strings, integers, long, Boolean and other variable types that are commonly used in the application.

### **2. Introduction**

Shared preferences is commonly used in any android application with will give the user the ability to save the most used values in the application and any other setting locally on the device. To use the shared preferences, we only need one object to write and read the values locally, so a singleton class is the perfect use for this.

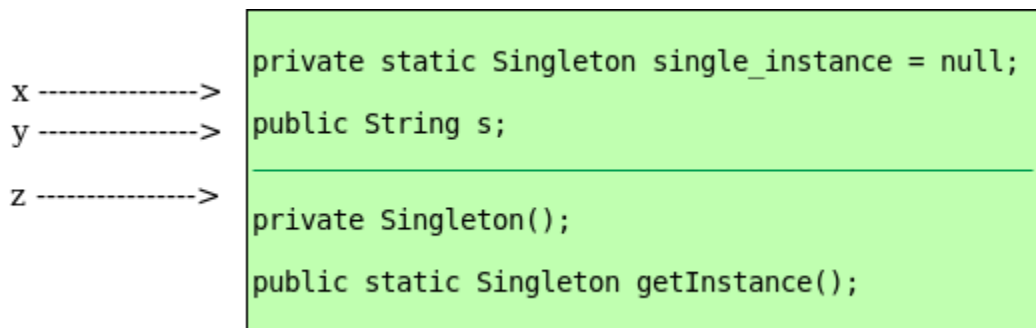
## 2.1. Singleton Class in Java

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time.

After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, it affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined. To design a singleton class:

- Make constructor as private.
- Write a static method that has return type object of this singleton class. Here, the concept of Lazy initialization is used to write this static method.

Normal class and Singleton class. Difference in normal and singleton class in terms of instantiation is that, For normal class we use constructor, whereas for singleton class we use `getInstance()` method. In general, to avoid confusion we may also use the class name as method name while defining this method.



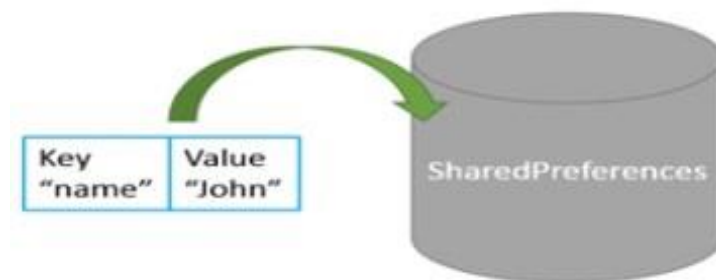
*Figure 6.1 singleton class attributes*

In the Singleton class, when we first time call `getInstance()` method, it creates an object of the class with name `single_instance` and return it to the variable. Since `single_instance` is static, it is changed from `null` to some object. Next time, if we try to call `getInstance()` method, since `single_instance` is not `null`, it is returned to the variable, instead of instantiating the Singleton class again. This part is done by if condition.

## 2.2. Shared preferences

It is used by the application to save data in key-value pairs like Bundle. Data is stored in XML file in the directory “data/data/<package name>/shared-prefs folder. Shared preferences only allows you to save primitive data types (that is, Booleans, floats, longs, ints and strings).

- There are two methods to access shared preferences:
  - `getPreferences(int mode)` : it is used if you have only 1 preference file.
  - `getSharedPreferences(String name , int mode)`: it is used if you have several files.
- The mode parameter can take several values:
  - `MODE_PRIVATE`: Only your app can access the file.
  - `MODE_WORLD_READABLE`: All apps can read the file.
  - `MODE_WORLD_WRITABLE`: All apps can write to the file.
- You can use the following steps to store the data to a shared preference file



*Figure 6.2 writing to the shared preferences*

- Get a reference to the `SharedPreferences` object.

```
sharedPreferences = getSharedPreferences (SHARED_PREF_NAME,  
Context.MODE_PRIVATE) ;
```



Where SHARED\_PREF\_NAME is a string constant of the name of the shared preferences file.

- Call the editor by using SharedPreferences object.

```
SharedPreferences.Editor editor = sharedPreferences.edit();
```

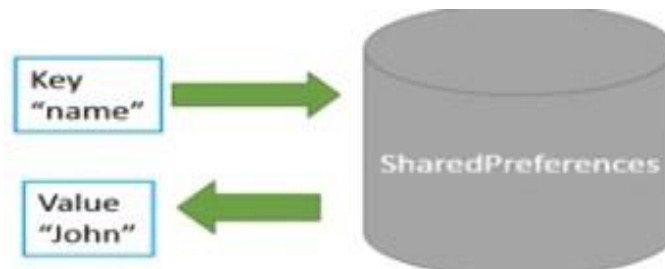
- Use the editor to add the data with a key.

```
String key = "name";  
String value = "Rajaie";  
editor.putString(key, value);
```

- Commit editor changes.

```
editor.commit();
```

- You can use the following steps to read the data from a shared preference file



*Figure 6.3 Reading From the Shared Preferences*

- Get a reference to the SharedPreferences object.

```
sharedPreferences = getSharedPreferences(SHARED_PREF_NAME,  
Context.MODE_PRIVATE);
```

- Use the key provided earlier to get data

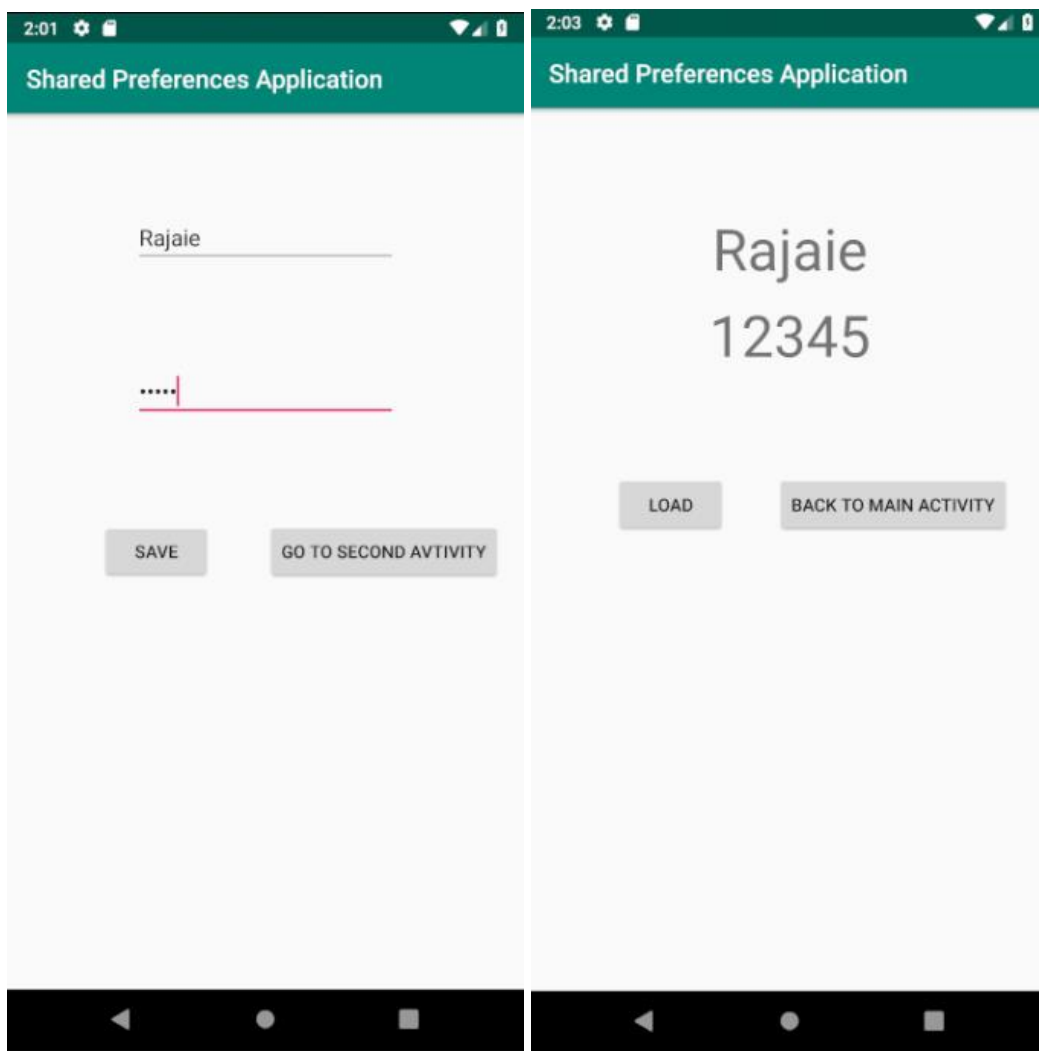
```
String name = sharedPreferences.getString("name", "noValue");
```

Note: getString method will return noValue if the data is not found.

### 3. Procedure

You will build an application that has two activities as shown in Figure 6.4, the first will save a username and password in shared preferences and the second will load these values from the shared preferences. You will access the shared preferences using singleton class called SharedPrefManager.

Create a new project and call the project “Shared Preferences Application”, use empty activity to the project, then create a new Activity call this one SecondActivity.



*Figure 6.4 SharedPreferences Application Layouts*

### 3.1. Creation of SharedPrefManager Singleton Class

Create a new java class in the main package of the project as shown in and call it SharedPrefManager, this class will access the shared preferences for reading and writing. This class will have the following

- Attribute that is called ourInstance that has a type of SharedPrefManager and initialed to null and private.
- Attribute that is called sharedPreference that has a type of SharedPreferences and initiated to null and private.
- Attribute that is called editor that is used for writing to the shared preferences has a type of SharedPreferences. Editor and initiated to null and private.
- A private constructor which will initiate the sharedPreference and the editor.
- A static method called getInstance which will return ourInstance if its not null and if its null (first time to initiate the object) it will create a new object of SharedPrefManager using the private constructor and return it.
- Public method for writing to the sharedPreferences
- Public method for reading from the sharedPreferences
- Other constant values as the sharedPreference name, shared Preference access Modes etc...

The following code shows the singleton SharedPrefManager class with two method to write and read string from the sharedPreferences

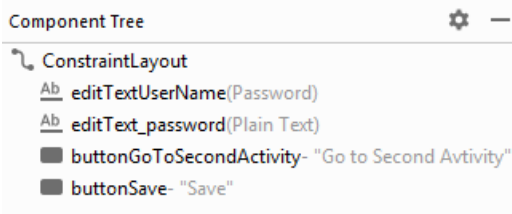


Figure 6.5 MainActivity Component Tree

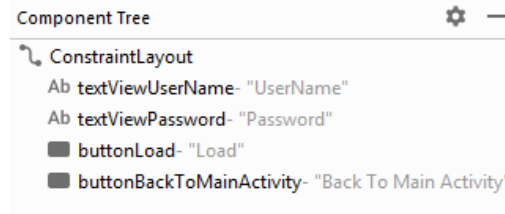


Figure 6.6 SecondActivity Component Tree

```

import android.content.Context;
import android.content.SharedPreferences;

public class SharedPrefManager {

    private static final String SHARED_PREF_NAME = "My Shared Preference";
    private static final int SHARED_PREF_PRIVATE = Context.MODE_PRIVATE;

    private static SharedPrefManager ourInstance = null;
    private static SharedPreferences sharedPreferences = null;
    private SharedPreferences.Editor editor = null;

    static SharedPrefManager getInstance(Context context) {

        if (ourInstance != null) {
            return ourInstance;
        }
        ourInstance=new SharedPrefManager(context);
        return ourInstance;
    }

    private SharedPrefManager(Context context) {
        sharedPreferences = context.getSharedPreferences(SHARED_PREF_NAME,
SHARED_PREF_PRIVATE);
        editor = sharedPreferences.edit();
    }

    public boolean writeString(String key, String value) {

        editor.putString(key, value);
        return editor.commit();
    }

    public String readString(String key, String defaultValue) {
        return sharedPreferences.getString(key, defaultValue);
    }
}

```

### 3.2. Building the MainActivity Layout and the SecondActivity Layout

Build the layout activity\_main as shown in Figure 6.4 (the component tree is shown in Figure 6.5) and the activity\_second layout as shown in Figure 6.4 (the component tree is shown in Figure 6.6)

- **Note that we used Constraint layout as the main layout (read about how to add widgets to a Constraint layout press here [Link](#) to see a tutorial about constraint layouts.**

### 3.3. Building the MainActivity Java Class

Get reference to the UserName and Password editTexts and the Save and GoToSecondActivity Buttons you added in the activity\_main layout.

Initiate a SharedPreferences by calling the static getInstance method. Make a listener to the Save button and save the values from the editTexts to the sharedPreferences using the writeString from the SharedPreferences by passing the key (e.g. "username") and the value from the edit text.

Make a listener to the GoToSecondActivity and start the SecondActivity. The code below is for the main Activity.

```
EditText editTextUserName;
EditText editTextPassword;
Button buttonSave;
Button buttonGoToSecondActivity;
SharedPreferences sharedPreferences;
Intent intent;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    editTextUserName = (EditText) findViewById(R.id.editTextUserName);
    editTextPassword = (EditText) findViewById(R.id.editText_password);
    buttonSave = (Button) findViewById(R.id.buttonSave);
    buttonGoToSecondActivity = (Button) findViewById(R.id.buttonGoToSecondActivity);
    sharedPreferences = SharedPreferences.getInstance(this);
    intent = new Intent(MainActivity.this, SecondActivity.class);

    buttonSave.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            sharedPreferences.writeString("username", editTextUserName.getText().toString());
            sharedPreferences.writeString("password", editTextPassword.getText().toString());
            Toast.makeText(MainActivity.this, "Values written to shared Preferences",
                Toast.LENGTH_SHORT).show();
        }
    });

    buttonGoToSecondActivity.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            startActivity(intent);
            finish();
        }
    });
}
```

### 3.4. Building the SecondActivity Java Class

Get reference to the UserName and Password TextViews and the Load and BackToMainActivity Buttons you added in the activity\_Second layout. Initiate a SharedPreferences by calling the static getInstance method. Make a listener to the Load button and Load the values from the sharedPreferences to the editTexts using the readString from the SharedPreferences by passing the key (e.g. "username") and the default value. Make a listener to the BackToMainActivity and start the MainActivity. The code below is for the Second Activity.

```
TextView textViewUserName;
TextView textViewPassword;
Button buttonLoad;
Button buttonBackToMainActivity;
SharedPreferences sharedPreferences;
Intent intent;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    textViewUserName = (TextView) findViewById(R.id.textViewUserName);
    textViewPassword = (TextView) findViewById(R.id.textViewPassword);
    buttonLoad = (Button) findViewById(R.id.buttonLoad);
    buttonBackToMainActivity = (Button)
findViewById(R.id.buttonBackToMainActivity);
    sharedPreferences=SharedPreferences.getInstance(this);
    intent = new Intent(SecondActivity.this,MainActivity.class);
    buttonLoad.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

textViewPassword.setText(sharedPreferences.readString("userName", "noValue"));

textViewUserName.setText(sharedPreferences.readString("password", "noValue"));
        }
    });
    buttonBackToMainActivity.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            startActivity(intent);
            finish();
        }
    });
}
```

## 4. Todo

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



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

## **EXP. No. 7. Fragments**

### **1. Objectives**

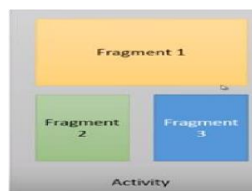
- ❖ Introduce to new concept in Android Applications.
- ❖ Simplify code and reduce latency time.

### **2. Introduction**

Tablet have larger display screen than small devices, so they can support multiple UI panes/user behavior at the same time.

A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities as shown in Figure 7.1.

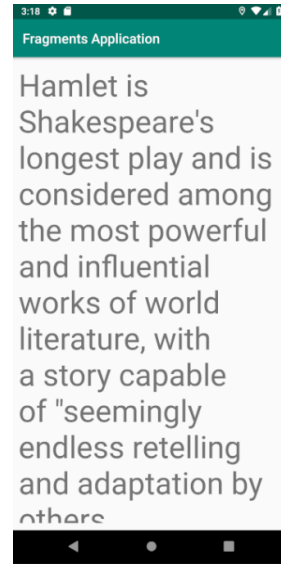
Let us take Fragments application as an example application. It uses two activities, the first one shows the titles of plays or characters and allows user to select one title as shown in Figure 7.2. The second one shows a quote from selected title as shown in the Figure 7.3.



*Figure 7.1 Multiple Fragments in the Same Activity*



*Figure 7.2 First activity shows three titles*



*Figure 7.3 Second activity shows selected title*



*Figure 7.4 Two Fragments in One Activity*



This interface is reasonable for a phone or small screen device, but it is inefficient on large device. So, it is better to use two cooperating layout units on one screen as shown in Figure 7.4. In Figure 7.4, the activity contains two fragments: first one shows three titles and second one shows selected title information.

## ***2.1. There are two general ways to add fragments to an activity:***

- Declare it statically in the activity layout file

Attach the fragment inside the activity through XML using `<fragment>` tag

Example: to add `FirstFragment` in the main activity we can use the following XML code in the layout of the main activity.

- Add it programmatically using the fragment manager

We will add the fragment to main activity using java. Every activity has its own `FragmentManager`, it maintains references to all fragments inside the activity. You can get access to `FragmentManager` through `getFragmentManager()` method. You can use `findFragmentById()` or `findFragmentByTag()` methods to get reference to a particular fragment. Changes to UI in terms of adding, removing and replacing fragments are conducted as `Fragment Transactions`. You must begin a transaction, then add, remove or replace a fragment and finally you have to commit the transaction to see the effect on your activity.

## ***2.2. Inter fragment communication in android:***

How to communicate between `AFragment` and `BFragment` without making a dependencies between the two fragments?

- Define an Interface in the `BFragment` class and implement it within the Activity.

- Implement the Interface: The activity that hosts AFragment and BFragment must implement the interface defined in the BFragment class.
- The BFragment captures the interface implementation during its onAttach() or onActivityCreated() lifecycle methods and can then call the Interface methods in order to communicate with the Activity.

### ***2.3. Fragment Transactions:***

If you want to add, replace, remove, attach or detach a fragment, you have to use the transactions by the following steps:

- get a new object from your fragment.
- begin the transaction by using beginTransaction method in the fragment manager.
- add/remove/replace/attach/detach the fragment.
- commit the transaction

## **3. Procedure**

In this lab you will design two different applications, the first is to add two fragments and make a communication between the fragments, where the second one is to how to add, remove, attach, detach and replace fragments dynamically.

### ***3.1. First Application***

In this application we will have two fragments (FirstFragment and SecondFragment) where the FirstFragment will send a string to the SecondFragment containing the number of times the button in the first fragment is clicked and the SecondFragment will display the string in a text view as shown in Figure 7.10. Create new project and change the application name to “Fragments Communication Application”.

➤ Adding fragments

Add two fragments as shown in Figure 7.5 and call them FirstFragment and SecondFragment as shown in Figure 7.6 and uncheck the *include fragment factory methods?* and *include interface callbacks?* Check boxes then click on finish.

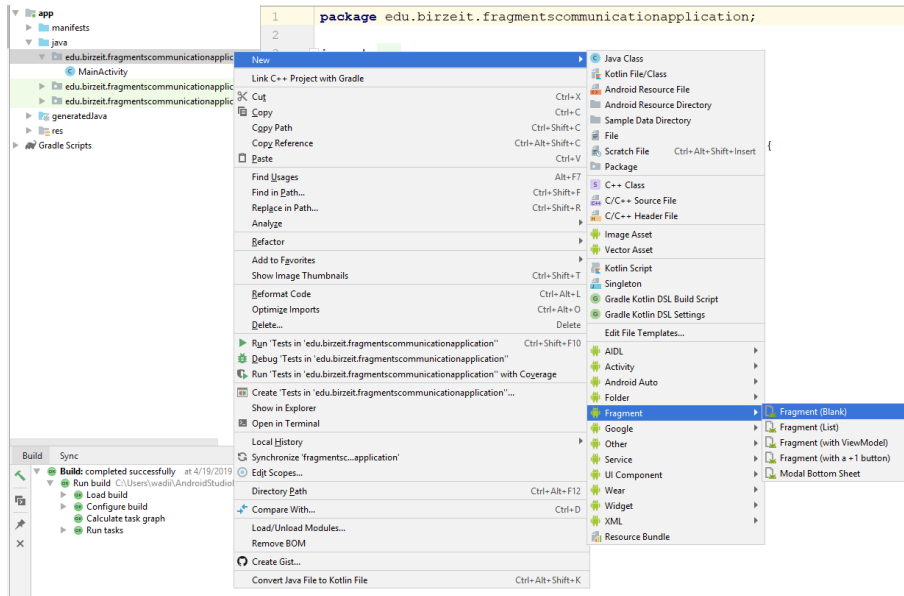


Figure 7.5 Adding Frsrgment Screen

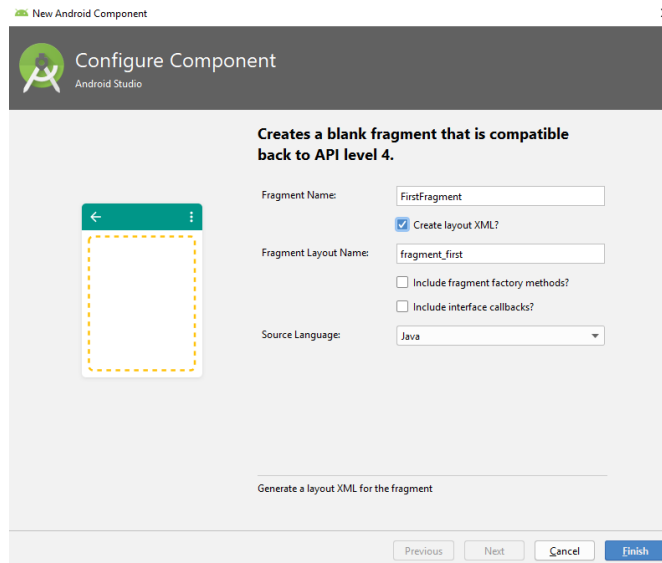
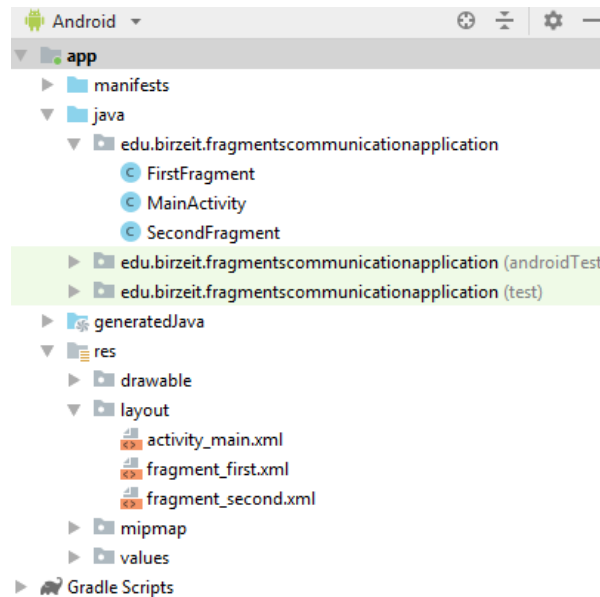


Figure 7.6 Fragment Android Component

When creating fragments, this will create a java class for the fragment and a layout as when creating an activity, after creating the fragments the project folders should look as in Figure 7.7.



*Figure 7.7 project classes and layouts*

➤ Design the fragments layouts

- In the fragment\_first layout start by adding a button and making the root layout height to wrap\_content. You also can change the background color of the root layout to dark holo\_green\_dark by adding int following code to the root layout xml code.

```
android:background="@android:color/holo_green_dark"
```

- In the fragment\_second layout start by adding a textview and making the root layout height to wrap\_content. You also can change the background color of the root layout to dark holo\_blue\_light by adding int following code to the root layout xml code.

```
android:background="@android:color/ holo_blue_light"
```

the fragments should look as in Figure 7.8.a for the first fragment and Figure 7.8 for the second fragment.



Figure 7.8.a First Fragment layout with button



Figure 7.8.b Second Fragment layout with textView

Figure 7.8 fragments layouts

➤ Adding fragments to the Main Activity

Convert the constraint layout into linear vertical layout and then add the following:

- Text view and change the text to “Main Activity”
- From the palette on the left side hand of the window choose <fragment> (drag and drop the fragment to the activity\_main layout). A fragment popup screen will show up, choose the FirstFragment as shown in Figure 7.9. this will connect the added fragment to the FirstFragment. Change the id to fragment1.
- Repeat the previous step for the SecondFragment. Change the id to fragment2

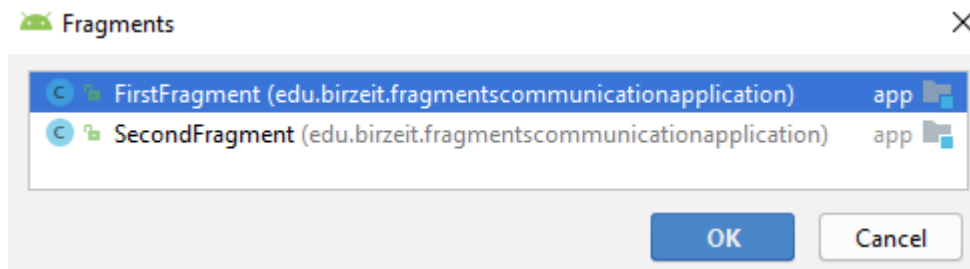


Figure 7.9 adding Fragment screen

## ❖ Making the communication between the fragments

### ➤ Define an Interface in the SecondFragment class and implement it within the Activity.

- In the second fragment java class make an interface and call it communicator.
- Add an abstract method in the communicator and call it respond. This method will be implemented in the Main activity, this method will take a string to be displayed in the text view. As shown in the following code.
- Inside SecondFragment class, define a method to display the data on a Text View. As shown in the following code.

```
interface communicator {
    public void respond(String data);
}

public void changeData(String data){
    TextView textView = (TextView) getActivity().findViewById(R.id.textView);
    textView.setText(data);
}
```

### ➤ Implement the Interface: The activity that hosts FristFragment and SecondFragment must implement the interface defined in the SecondFragment class.

- Implement the SecondFragment.communicator in the main activity.
- Override the respond method in the Main activity.
- Get a reference to the second fragment from the layout.
- Call the change data method in the second fragment.
- The following code shows the override respond method in the main activity.

```
@Override
public void respond(String data) {
    SecondFragment secondFragment =
    (SecondFragment) getSupportFragmentManager().findFragmentById(R.id.fragment2);
    secondFragment.changeData(data);
}
```

➤ Calling the respond method from the FirstFragment

- In the first fragment override the `onActivityCreated` method
- Make an instance of the Second Fragment communicator
- Add a click Listener to the button
- And send the text containing the number of times the button has been clicked to be displayed in the SecondFragment
- The following code shows the First Fragment `onActivityCreated` method.

```
@Override
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    final SecondFragment.communicator communicator =
(SecondFragment.communicator) getActivity();
    Button button = (Button) getActivity().findViewById(R.id.button);
    final int[] i = {0};
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            i[0]++;
            communicator.respond("the button is clicked "+ i[0] +" times");
        }
    });
}
```

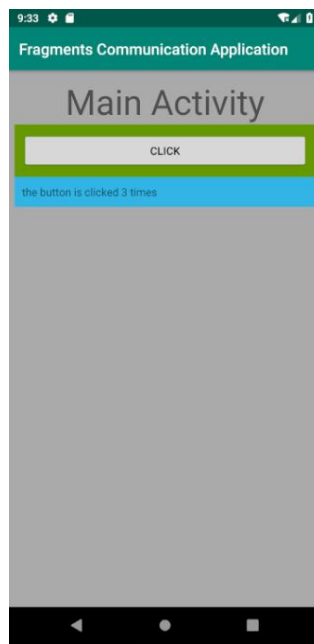


Figure 7.10 Fragment communication Application

### 3.2. Second Application

You will design an application that can add, remove, attach, detach and replace fragments using fragment transaction. These will be done dynamically in the java code of the main activity. Figure 7.11 shows the design of the application before and after adding the fragments.

Start by creating a new project and change the name of the project to “Fragments Transaction Application”. Add two Fragments (FirstFragment and Second Fragment) as we did in the previous section and add a text view in each fragment and change the text of each one to indicate the fragment name (e.g. text view in the First fragment change the text to “This Is The First Fragment”). Also change of the root layout for each fragment layout. The code below shows the FirstFragment Layout.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".FirstFragment"
    android:background="@android:color/holo_green_dark">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="This Is The First Fragment"

        android:textAppearance="@style/TextAppearance.AppCompat.Display2" />
</FrameLayout>
```

- We will show how to make a transaction for adding a fragment and you should do the remove, attach, detach and replace.
- To add a fragment dynamically you will use fragment manager to make a fragment transaction to add a new fragment and after you finish you should commit the transaction you made. The code below shows how to add a the FirstFragment when clicking on a button.



```

Button buttonAddF = findViewById(R.id.add_f);
final FirstFragment firstFragment = new FirstFragment();
final FragmentManager fragmentManager = getSupportFragmentManager();

buttonAddF.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.root_layout, firstFragment, "FristFrag");
        fragmentTransaction.commit();
    }
});

```

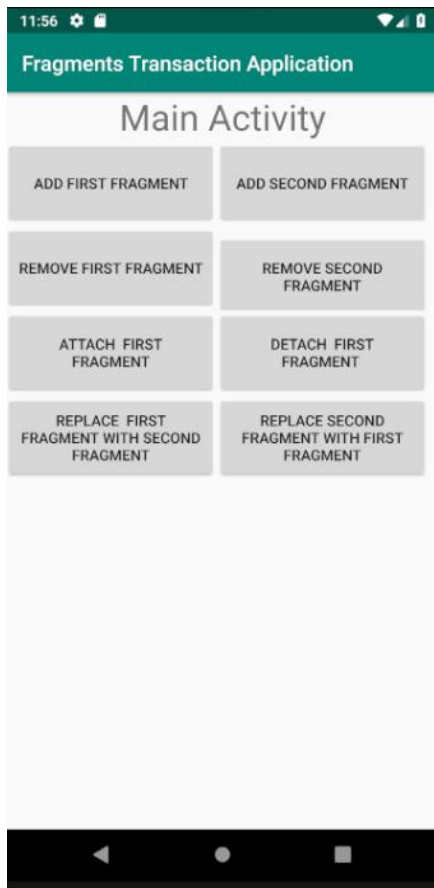


Figure 7.11 before adding the fragments



Figure 7.11 after adding the fragments

Figure 7.11 Second application Layout

## 4. Todo

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



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

## **EXP. No. 8. Integrating REST API into Android Application**

### **1. Objectives**

- ❖ Integrate Android apps with RESTful web services.
- ❖ How to access data from RESTful web services using simple GET and POST requests.
- ❖ How to parse JSON object to java object.

### **2. Introduction**

REST describes a set of architectural principles by which data can be transmitted over a standardized interface (such as HTTP (Hyper-Text Transfer Protocol)). The acronym REST (Representational State Transfer), this basically means that each unique URL is a representation of some object.

Exposing a system's resources through a RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way.

All the user interface actions are managed by the main thread, so you should not block this thread by a process needs a few of seconds to be executed. If you do that, the UI components will freeze, and you will receive ANR (application not responding) error. One of the ways to solve this error is using AsyncTask class.

AsyncTask is an abstract class which allows you to perform long/background operations and show its result on the main thread.

To integrate an Android app with a RESTful web service, you'll need to make calls over the network. You can choose from a few different HTTP clients, some that are included with the Android SDK, and some open-source libraries that are available from various organizations.

URLConnection: It is an abstract class used to send and receive data over the web .It is included in Android SDK.

To find any resource you need on the Internet, you can use the URL (Uniform Resource Locator) class. The constructor of this class takes string parameter of the following structure:

protocol://host:port/path (E.g.: (<http://www.mocky.io/v2/5cbc5efb320000641080d86b>))

In this experiment, we will request the API from the server using HttpURLConnection class. So we will build a simple android application to get a JSON object from the server .We will parse the JSON to java object using JSONObject class. Finally, we will print the data in a textview component. In this experiment we will get a JSON of the following structure:

```
[
  {
    "name": "",
    "age": "",
    "id": ""
  }
]
```

### 3. Procedure

#### ➤ Designing the Main Activity Layout

Create a new Project and call it “REST Application”, create the basic UI and application structure. This application will consist of one activity (Main Activity) to get the data from the server, and to display the results. To implement the UI of the application, you will use the layout\_main so that we can arrange our components. You will use the constraint layout for the design as shown in Figure 8.1, the code below shows xml file of the main activity.

```

<?xml version="1.0" encoding="utf-8" ?>
<android.support.constraint.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">
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Main Activty"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Display3"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.1" />

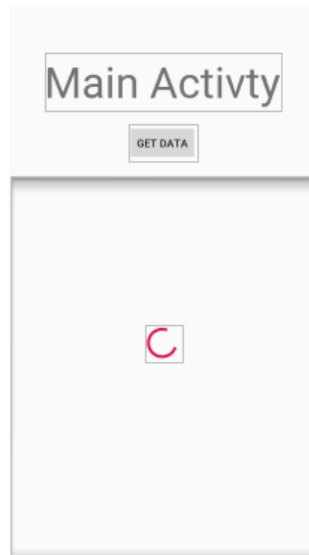
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="160dp"
    android:layout_marginEnd="8dp"
    android:text="Get Data"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<LinearLayout
    android:id="@+id/layout"
    android:layout_width="395dp"
    android:layout_height="507dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:background="@android:drawable/gallery_thumb"
    android:orientation="vertical"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button"></LinearLayout>

<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:visibility="gone"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.62" />

</android.support.constraint.ConstraintLayout>

```



*Figure 8.1 REST Application Layout*

➤ Creating Model class and Jason to Array List class:

Getting the data from the URI will return a Json array which we will parse this array into Array list using a special class. So, we will need a class to present the Json Object and a class to parse the Json Array into Array List

- Model Class

You will create a class which will have the same attributes as the Json Object has.

Name this class “Student”. This class will have 3 attributes:

private int ID

private String name

private Double age

Create two constructors (empty and with attributes) for this class. Then create setters and getters for all attributes.

Override toString method as we did in EXP. No. 2 sec 3.2 the following code shows the Student class.

```

public class Student {
    private int ID;
    private String name;
    private Double age;

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getAge() {
        return age;
    }

    public void setAge(Double age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
            "\nID= " + ID +
            "\nname= " + name +
            "\nage= " + age +
            +'\n'+'}'+'\n';
    }
}

```

- **Json To Model Class**

Create a new class and call it “StudentJasonParser” this class will convert the Json object we got from the REST API to Array List of the type Student. The following code shows how to convert the JSON List into Array List.

```

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class StudentJasonParser {

    public static List<Student> getObjectFromJason(String jason) {
        List<Student> students;
        try {
            JSONArray jsonArray = new JSONArray(jason);
            students = new ArrayList<>();
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = new JSONObject();
                jsonObject = (JSONObject) jsonArray.get(i);
                Student student = new Student();
                student.setID(jsonObject.getInt("id"));
                student.setName(jsonObject.getString("name"));
                student.setAge(jsonObject.getDouble("age"));

                students.add(student);
            }
        } catch (JSONException e) {
            e.printStackTrace();
            return null;
        }
        return students;
    }
}

```

#### ➤ HTTP manager Class

Add a new Java Class and call it “HttpManager”, this class is responsible for requesting a Http request to get the Json Array from the URL we will pass to this class. you are now going to implement the REST API call. Add the getData method which is called in doInBackground method. This will first create a URL based on the string that is passed into the method. It will next open the connection and create a BufferedInputStream to receive the results from the call. The code below is for the HttpManager class.

```

import android.util.Log;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpManager {

    public static String getData(String URL) {
        BufferedReader bufferedReader = null;
        try {
            URL url = new URL(URL);
            HttpURLConnection httpURLConnection =
(HttpURLConnection) url.openConnection();
            bufferedReader = new BufferedReader(new
InputStreamReader(httpURLConnection.getInputStream()));
            StringBuilder stringBuilder = new StringBuilder();
            String line = bufferedReader.readLine();
            while (line != null) {
                stringBuilder.append(line + '\n');
                line = bufferedReader.readLine();
            }
            return stringBuilder.toString();
        } catch (Exception ex) {
            Log.d("HttpURLConnection", ex.toString());
        }
        return null;
    }
}

```

#### ➤ Invoking the API call

You will be making the API call in a separate thread, which is always a good practice since the user interface will not be blocked while the call is being made. This is especially important in mobile devices that may drop network connections or experience high network latency. To accomplish this, you will implement an AsyncTask class. We should add a new class called ConnectionAsyncTask. We will override three methods, onPreExecute() doInBackground(String... params) and onPostExecute(String s). onPreExecute is executed at first before starting executing the doInBackground in the context of the UI thread. On the other hand the doInBackground executes in a separate thread so we will call the REST API and parse the results in this thread. onPostExecute executes in the context of the UI thread so we will use this method to display the results. We can access the UI components in methods that work in UI thread.



So, we should show/hide the ProgressBar in these two methods. The code below shows the ConnectionAsyncTask class code.

```
import android.app.Activity;
import android.os.AsyncTask;

import java.util.List;

public class ConnectionAsyncTask extends AsyncTask<String, String,
String> {

    Activity activity;

    public ConnectionAsyncTask(Activity activity) {

        this.activity = activity;
    }

    @Override
    protected void onPreExecute() {

        ((MainActivity) activity).setButtonText("connecting");
        super.onPreExecute();
        ((MainActivity) activity).setProgress(true);
    }

    @Override
    protected String doInBackground(String... params) {

        String data = HttpManager.getData(params[0]);

        return data;
    }

    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
        ((MainActivity) activity).setProgress(false);
        ((MainActivity) activity).setButtonText("connected");
        List<Student> students =
StudentJsonParser.getObjectFromJason(s);
        ((MainActivity) activity).fillStudents(students);
    }
}
```

## ➤ Implementing the Main Activity

In main activity we will get reference to the button and the root layout. The event handler of this button creates a new object of Connection AsyncTask class so that we connect to the server asynchronously, also we will make a method which will displays the users from array list. which will be called in the onPostExecute method in the ConnectionAsyncTask class. The code below is for the MainActivity.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.TextView;

import java.util.List;

public class MainActivity extends AppCompatActivity {
    Button button;
    LinearLayout linearLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setProgress(false);

        button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ConnectionAsyncTask connectionAsyncTask = new
ConnectionAsyncTask(MainActivity.this);
connectionAsyncTask.execute("http://www.mocky.io/v2/5b4e6b4e320002c
009c2a44");
            }
        });

        linearLayout = (LinearLayout) findViewById(R.id.layout);
    }

    public void setButtonText(String text) {
        button.setText(text);
    }

    public void fillStudents(List<Student> students) {
        LinearLayout linearLayout = (LinearLayout)
findViewById(R.id.layout);
    }
}
```

```

        linearLayout.removeAllViews();
        for (int i = 0; i < students.size(); i++) {
            TextView textView = new TextView(this);
            textView.setText(students.get(i).toString());
            linearLayout.addView(textView);
        }
    }

    public void setProgress(boolean progress) {
        ProgressBar progressBar = (ProgressBar)
        findViewById(R.id.progressBar);
        if (progress) {
            progressBar.setVisibility(View.VISIBLE);
        } else {
            progressBar.setVisibility(View.GONE);
        }
    }
}

```

### ➤ Adding the internet Permission

In android, to give your application the ability to use internet, special permission should be added in the manifest.xml file, open that file and add the following code before the application tag

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

### ➤ For API more than 28

- By default, the Http requests are disabled due to low security. And since we will make a Http request, you should enable this.
- To Enable Http requests add a new resource directory by right clicking on the res director and add new Android Resource Directory as shown in Figure 8.2. Then change the Resource Type in the new Resource Directory Screen to xml as shown in Figure 8.3.
- In the new xml Directory add a new Android Resource File and call it network\_security\_config and add the code below in this file.

```

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <base-config cleartextTrafficPermitted="true">
    <trust-anchors>
      <certificates src="system" />
    </trust-anchors>
  </base-config>
</network-security-config>

```

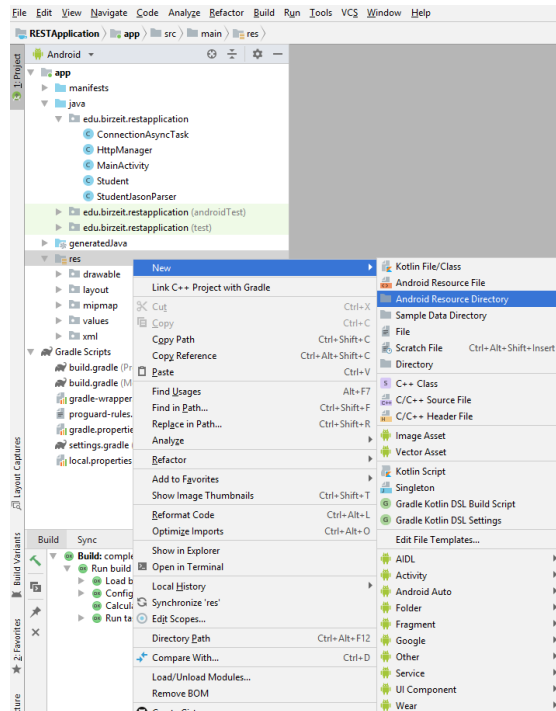


Figure 8.2 Adding new Resource Directory

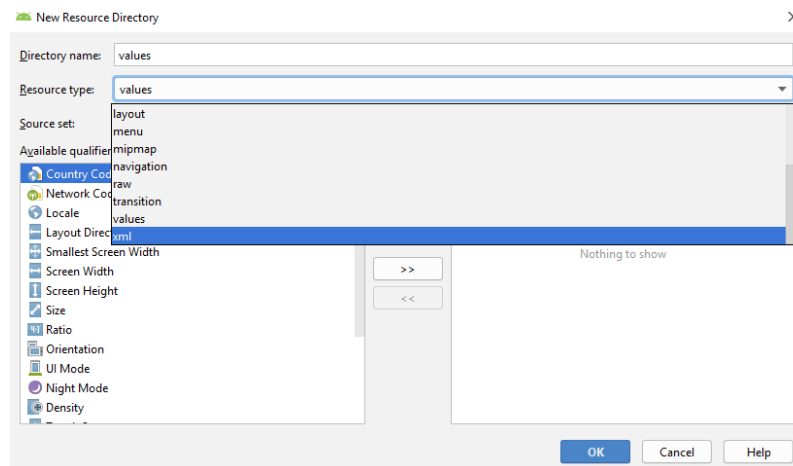


Figure 8.3 New Resource Directory Screen

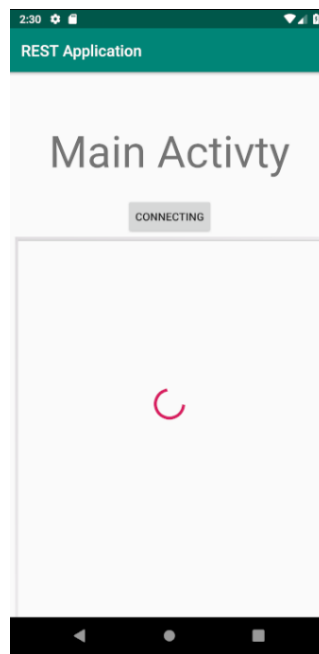
- In the Manifest File and in application header set the networkSecurityConfig to the file we added.

```
android:networkSecurityConfig="@xml/network_security_config"
```

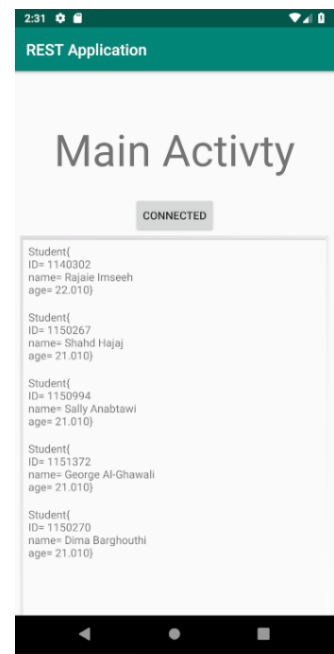
the output of the Application is shown in Figure 8.4



*Figure 8.4.a Before Clicking the Button*



*Figure 8.4. While Fetching the Data form the URL*



*Figure 8.4.c After Fetching the Data and Displaying it*

*Figure 8.4 REST Application Output*

## 4. Todo

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



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

## **EXP. No. 9. Spring Boot Part 1**

### **1. Requirements**

- ❖ Knowledge of the java programming language
- ❖ Basic Understanding of MVC (Model View Controller) architecture
- ❖ Basic Knowledge of Maven tool
- ❖ Basic Idea about Spring framework (preferably Spring MVC)
- ❖ Knowledge of java ORM (Object Role Modeling) models is extremely helpful but not required
- ❖ Spring Tool Suite software (STS) (For windows 64-bit download this [Link](#))
- ❖ Postman software (For windows 64-bit download this [Link](#))

### **2. Objectives**

- ❖ Understanding the fundamentals of Spring Boot framework
- ❖ Create rest services using Spring MVC

### **3. Introduction**

Spring is an enterprise java framework which lets you write enterprise java applications, spring framework is widely used due to It's useful features , spring strongly introduces the concept of inversion of control specifically dependency injection which helps you wire your associations in a way which reduces the dependency between the entities.

Spring boot was introduced to make creating spring applications easier, spring boot was built above Spring MVC, Spring MVC is a java MVC framework which helps you build MVC architecture using defined features, It wires up all the MVC components in a nice way which helps developers develop flowless and easily tested scenarios.

Spring Boot makes it easy to create stand-alone, production-grade based Applications that you can simply run without even the need of servlet container, also Spring Boot contains default configuration which probably will be enough for most of the cases which makes it easy to start development without the need to diving into complex configurations ( as in Spring , Spring MVC), also being stand-alone with embedded servlet container means the configuration of the server are the embedded with the application configurations, which makes it easier to deploy in different machines without worrying about configuring the servlet container.

Keep in mind that spring boot mainly bootstrap the development of spring MVC projects so actually spring MVC will be the one getting the job done.

## **4. Procedure**

### ***4.1. Downloading and running the STS software***

After downloading the STS software, extract the downloaded file as in Figure 9.1, and run the software in

```
spring-tool-suite-3.9.8.RELEASE-e4.11.0-win32-x86_64>sts-bundle>sts-3.9.8.RELEASE>STS.exe
```

as shown in Figure 9.2. the software panels are shown as in Figure 9.3.

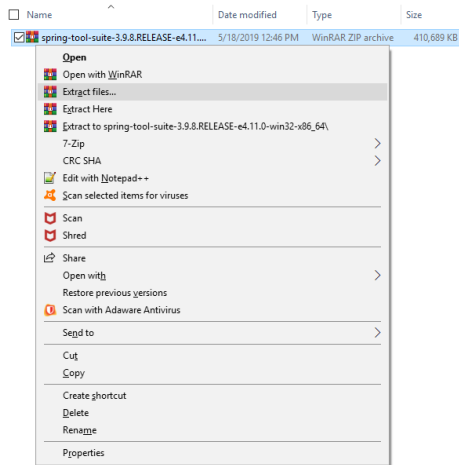


Figure 9.1 Extracting STS software

Name	Date modified	Type	Size
configuration	3/26/2019 1:04 AM	File folder	
dropins	3/26/2019 1:04 AM	File folder	
features	3/26/2019 1:04 AM	File folder	
p2	3/26/2019 1:03 AM	File folder	
plugins	3/26/2019 1:04 AM	File folder	
readme	3/26/2019 1:04 AM	File folder	
.eclipseproduct	3/26/2019 1:05 AM	ECLIPSEPRODUCT...	1 KB
artifacts.xml	3/26/2019 1:04 AM	XML Document	256 KB
eclipsec.exe	3/26/2019 1:02 AM	Application	120 KB
license.txt	3/26/2019 12:17 AM	Text Document	12 KB
open-source-licenses.txt	3/26/2019 1:04 AM	Text Document	1,586 KB
<input checked="" type="checkbox"/> STS.exe	3/26/2019 1:02 AM	Application	408 KB
STS.ini	3/26/2019 1:04 AM	Configuration sett...	1 KB

Figure 9.2 STS software



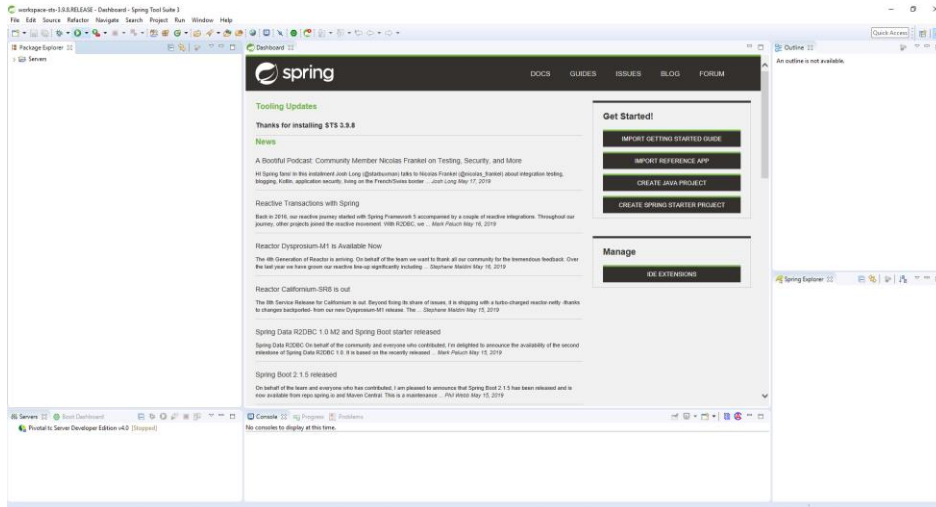


Figure 9.3 STS software panels screen

## 4.2. Setup Spring Boot Project

Spring boot aims to bootstrap the development of spring MVC application, spring project suffers of complex configurations which spring boot solves in a very nice way. There are many ways to create a spring boot project, we will introduce one way but keep in mind this is not the only way and not the best, it depends on the taste of developer and the environment you are working on.

### ➤ STS (Spring Tool Suite):

You can use spring tool suite to create spring boot application in simple steps.

- Creating new Spring Boot starter Project as shown in Figure 9.4.a and Figure 9.4.b.
- After choosing Spring Boot starter Project a new screen as shown in Figure 9.5 will appear. Change the name of the application to “FirstApplication”. Make sure all other attributes are as shown in Figure 9.5.a The click next
- Search for web in the search slot and add the web dependency for the application as shown in Figure 9.5.b. Then click finish.

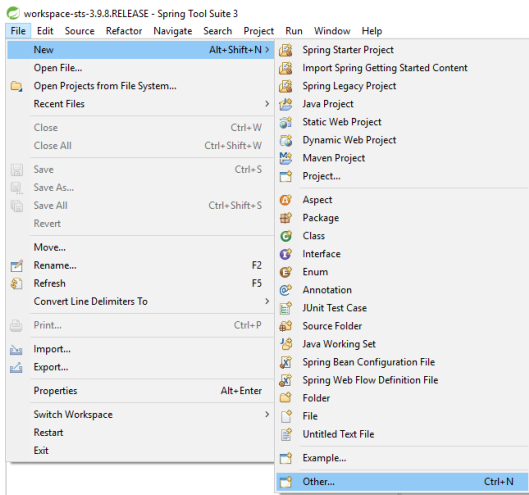


Figure 9.4.a New Other Project

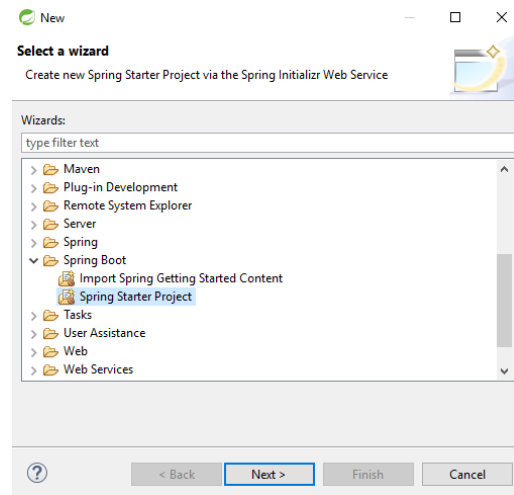


Figure 9.4.b New Spring Starter Project

Figure 9.4 Creating New SpringBoot Project

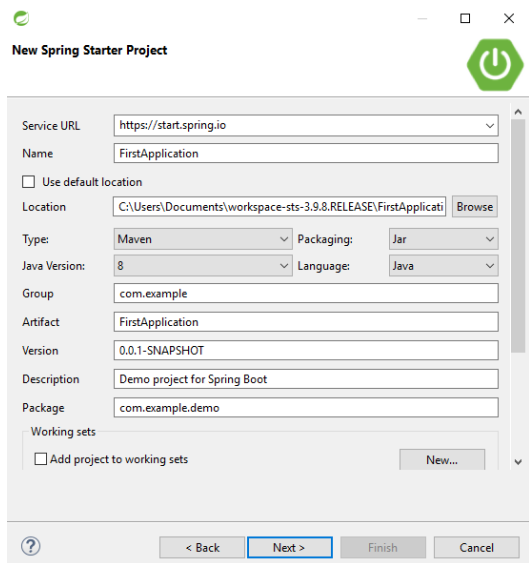


Figure 9.5.a Application name and other attributes

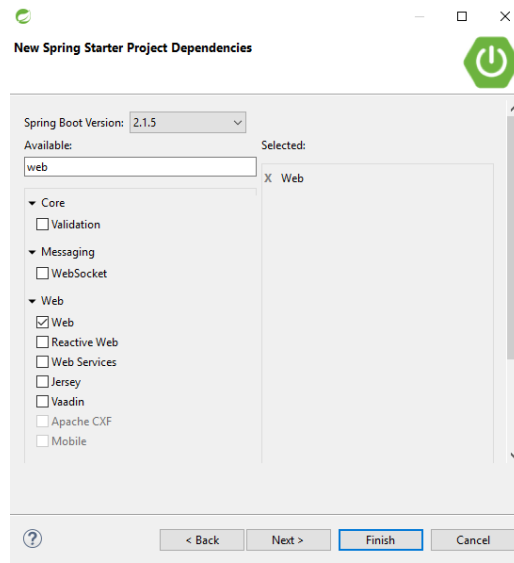


Figure 9.5.b Application dependencies

Figure 9.5 New Spring Starter Project Screen

### 4.3. Create Project Structure

As we mentioned before spring boot is just an upper framework built above Spring MVC so spring MVC will do all the work, to start building our application we need to build a structure for our MVC components, there are many structures and trends to build MVC application, in this walk through we are aiming to build RESTful API's without worrying about the views.

REST (Representational state transfer): briefly is a standard of communication between client and server using textual representation of the resources using http stateless ( each request does not know about the other request) requests, resources are presented using text wrapped inside http message, each resource is defined by URL and operations are defined through sub URL's and https methods (**Post for create new, Put for update, Get for getting resources, Delete for delete resources and others -search for them-**). In Spring MVC you will have multi layers as:

- Controllers will play the role of the rest api's, business logic will not be included in controllers instead it will be wrapped in another layer called business services to keep the controllers layer as thin as possible, controllers will be mapped by URL's to access methods inside the controller.
- Services are the layer which contains most of the business logic, services will be injected inside controllers to be used when URL's mapped by controllers are accessed, services are by default singleton, which means whenever trying to instantiate the service you will get the same object, this is useful as the transfer of state between all the components is guaranteed, though it introduces a risk with multithreading systems which requires using thread safe operations.
- Models layer will contain our entities which represents the resources in our system.

You will create for packages under the demo package calling them (Controllers, Services and Models as shown in Figure 9.6.

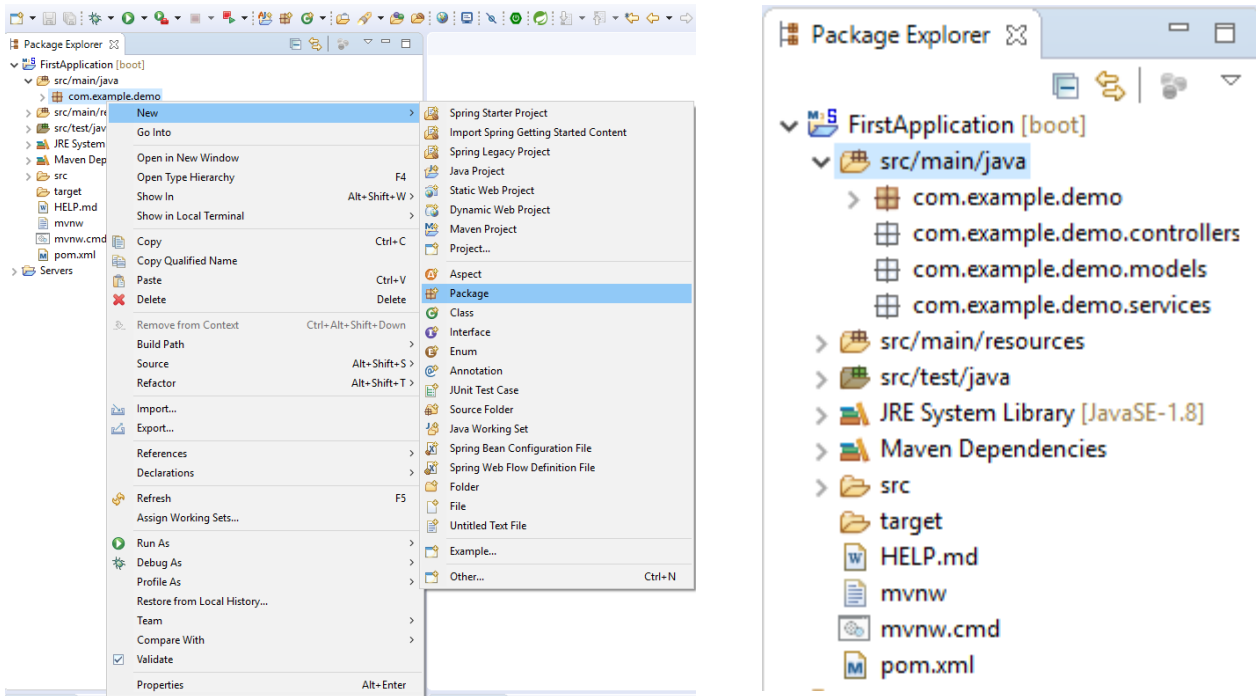


Figure 9.6 Adding Project Packages

➤ Models

Models are the entities of our application, in this walk through we will be implementing a User entity which has a name, a username and an email, models in spring boot basically are normal classes, so in models package go ahead and create a class User and add its attributes. Notice that all attributes should be private and should be accessed using accessors (getters and setters) to follow encapsulation standards.

```

package com.example.demo.models;

public class User {

    private String name;
    private String userName;
    private String email;

    public User() {
        super();
        // TODO Auto-generated constructor stub
    }

    public User(String name, String userName, String email) {
        super();
        this.name = name;
        this.userName = userName;
        this.email = email;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "User [name=" + name + ", userName=" + userName +
", email=" + email + "]";
    }
}

```

## ➤ Services

Services will take care of most of the job, they will be called by controllers and used to get or manipulate data, in services package create UserService class.

You can notify spring about service using “@Service” annotation, remember that spring will take care of initializing all predefined components so you will not need to initialize service manually or apply singleton constraints.

For now, let us create a static list containing users’ data (later we will get this data from database), then create a method which returns this list.

```
package com.example.demo.services;

import java.util.ArrayList;
import java.util.Arrays;

import org.springframework.stereotype.Service;

import com.example.demo.models.User;

@Service
public class UserService {

    private ArrayList<User> userList = new
ArrayList<User>(Arrays.asList(
        new User("Rajaie", "rajaie111", "Rajaie@gamil.com"),
        new User("moath", "moath111", "m@hotmail.com")
    ));

    public ArrayList<User> getUserList() {
        return this.userList;
    }
}
```

#### ➤ Controllers

Controllers are the API’s access when requesting a URL, to create a controller in packages controllers create a class UserController. To tell spring that your class is a controller you need to annotate it with “@RestController” annotation, when application starts spring will register classes annotated with this annotation as controller, this annotation in addition to define a controller tells spring that this controller is following rest standard which means a textual response will be returned instead of object. To direct the request made to a URL you need to map each URL to a method in a controller, this is done using annotation “@RequestMapping”, by providing the URL to this annotation method will be executed and the returned object/List of objects will be parsed ( by default) to JSON using a built in library( JACKSON).

```

package com.example.demo.controllers;

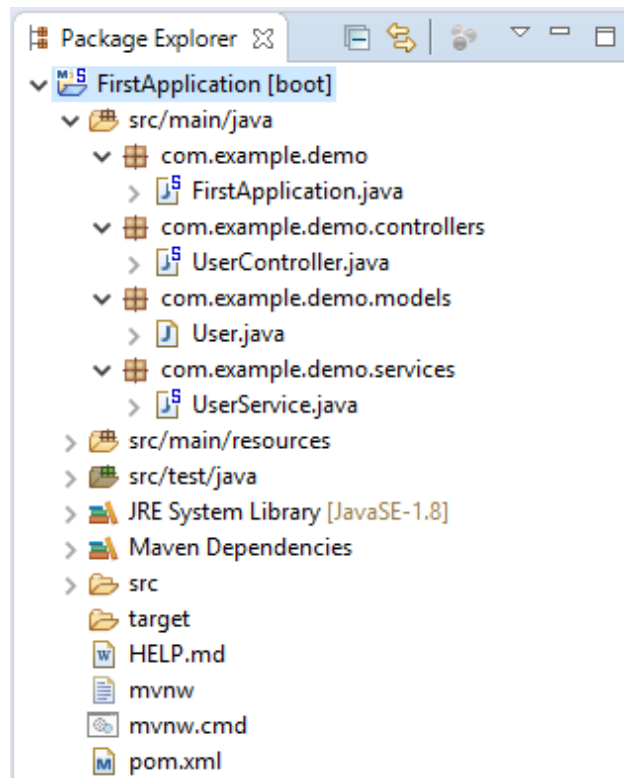
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @RequestMapping("/users")
    public String getAllUsers() {
        return "hello";
    }
}

```

The project packages and classes should look like as in Figure 9.7



*Figure 9.7 project packages and classes*

Run the application by clicking right click on the main project package and choose “Run as” > “Spring Boot App” as shown in Figure 9.8.

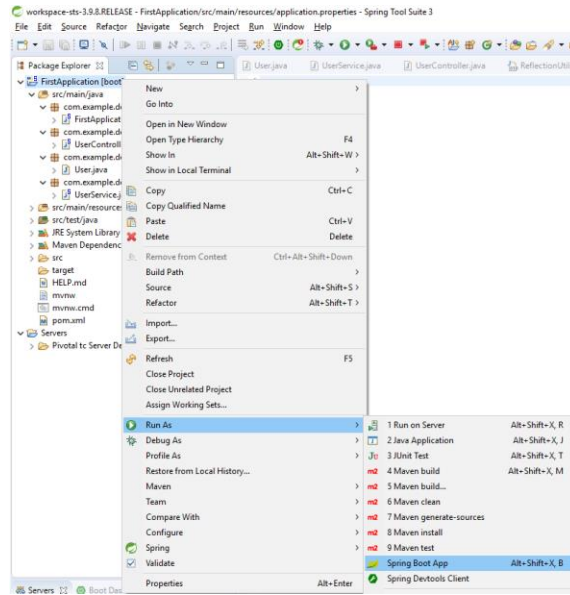


Figure 9.8 Run Spring Boot Application

When requesting localhost:8080/users on any web browser this api will return hello (String will not be parsed to JSON for sure) as shown in Figure 9.9.

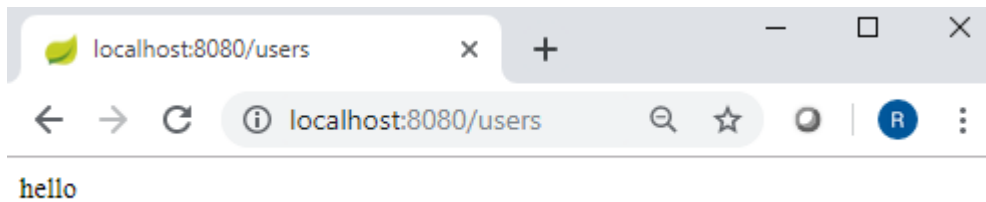


Figure 9.9 Web Browser request

In a normal way controllers use services so we need a reference for our user service in the controller, on the startup of the application spring will initialize the singleton service bean (if you do not know what beans are please do some reading about spring beans) so we do not need to initialize service, we can wire the service to our controller using “@Autowired” annotation which wires the service bean to the reference defined in our controller.

```
@Autowired
UserService service;
```



That is all what we need to use UserService, so now we can use the user service to return all the user data by calling getUserList method we created in the UserService class.

```
package com.example.demo.controllers;

import java.awt.List;
import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

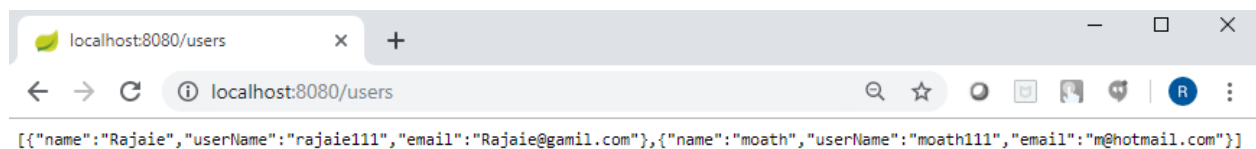
import com.example.demo.models.User;
import com.example.demo.services.UserService;

@RestController
public class UserController {

    @Autowired
    UserService service;

    @RequestMapping("/users")
    public ArrayList<User> getAllUsers() {
        return service.getUserList();
    }
}
```

Run the application as mentioned above and if you try to access localhost:8080/users you will get a JSON array representing all the users as shown in Figure 9.10.



*Figure 9.10 Web Brouser Request for all users*

This request is GET request, without specifying the HTTP method of a specific URL all requests will come to our getAllUsers() method, you can try to do a post request using PostMan Extension in chrome and you will get the same response.

To Specify a different method for POST, PUT you can pass a parameter to the “@RequestMapping” annotation defining the method using a predefined enumeration in spring.

```
@RequestMapping(method=RequestMethod.POST, value="/users")
    public boolean addUser(@RequestBody User user) {

    }
```

URL is passed use value keyword, to receive the object to be created you need to get it using “@RequestBody”, this annotation will tell spring to map the payload json included in the request body to the Entity User and create object user.

To add the user to the array list which is in the UserService we create a method that adds a user to the arraylist in the UserService class.

```
public boolean addUserToUserList (User user) {
    return userList.add(user);
}
```

Then in the UserController class in the addUser method we call this method that adds the user to the array list.

```
@RequestMapping(method=RequestMethod.POST, value="/users")
    public boolean addUser(@RequestBody User user) {
        return service.addUserToUserList(user);
    }
```

After running the application as mentioned above we can test the application using the PostMan software.

- Create a get method and add the “<http://localhost:8080/users>” URL to this request. This is for getting all users since it is a get method

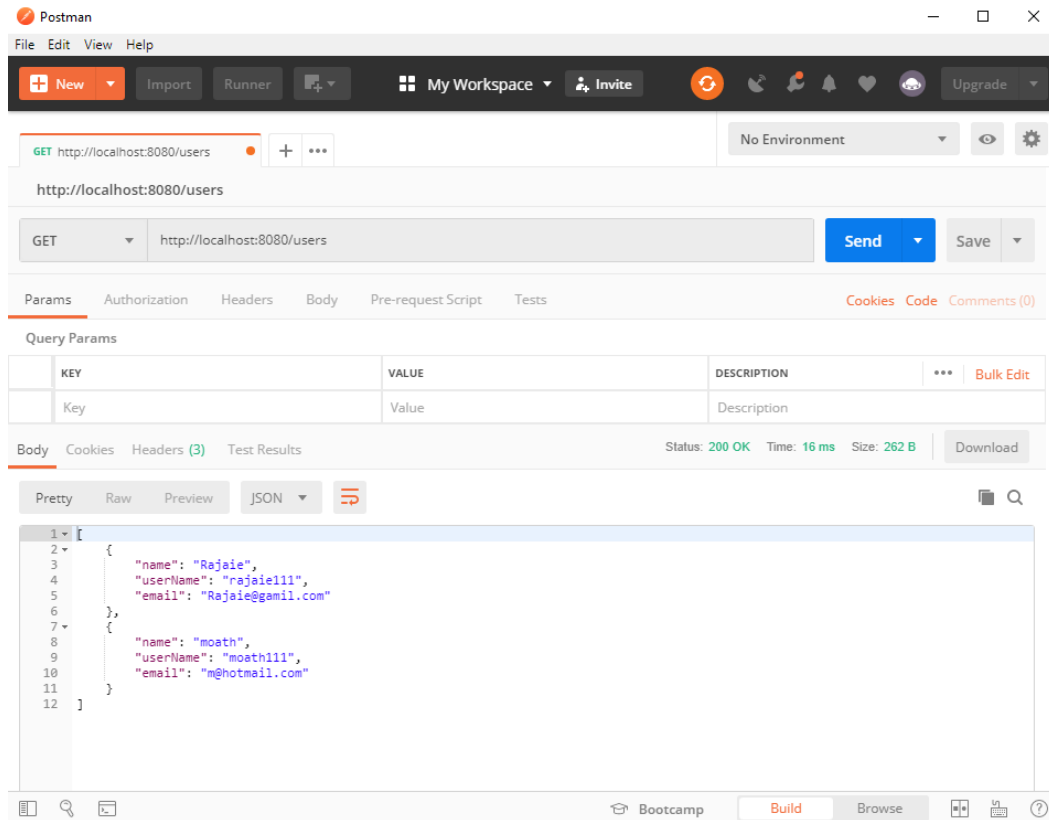


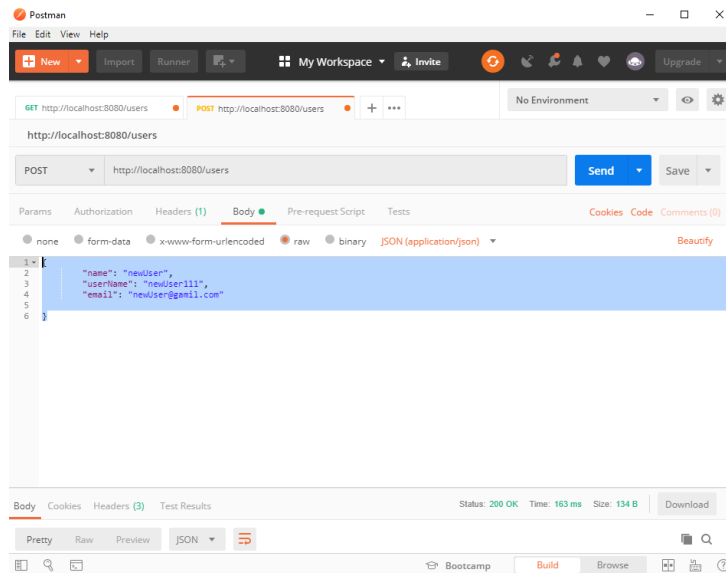
Figure 9.11 PostMan Request for all users

- Create a Post Method that will add a user to the ArrayList using PostMan software, the user to be inserted will be in the body of the request. So click on the body button in the PostMan Software and choose raw, change the input from text to JSON (application/json) and insert the new user we want to add as a json object, this is shown in Figure 9.12.

```

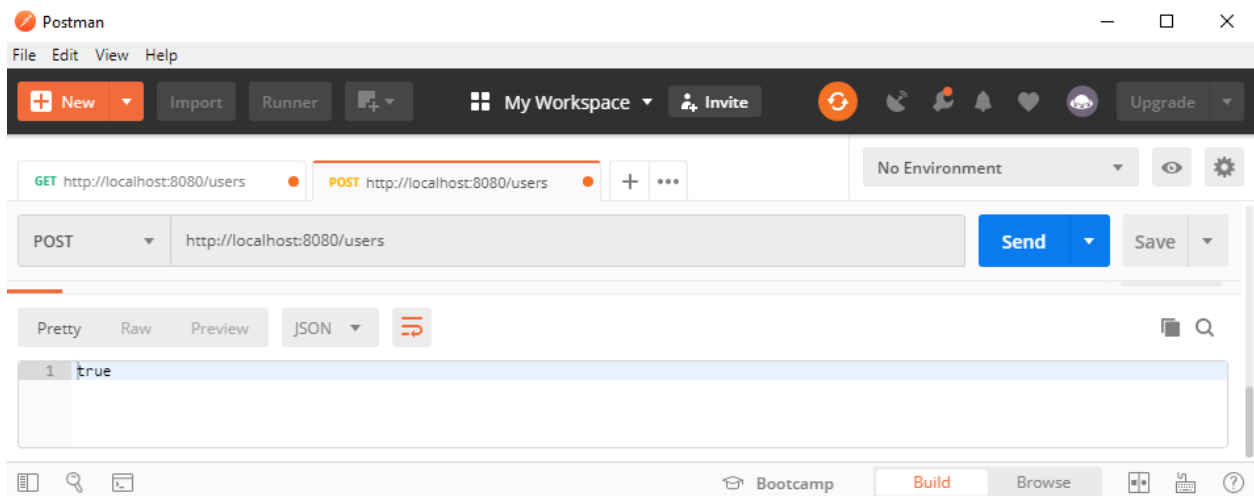
{
    "name": "newUser",
    "userName": "newUser111",
    "email": "newUser@gamil.com"
}

```



*Figure 9.12 PostMethod to Add User*

After we click on Send a true value must be returned indicating that the user was added successfully as shown in. We can test if the user is added by calling the get method for getting all users as shown in Figure 9.14.



*Figure 9.13 User was Added Successfully*

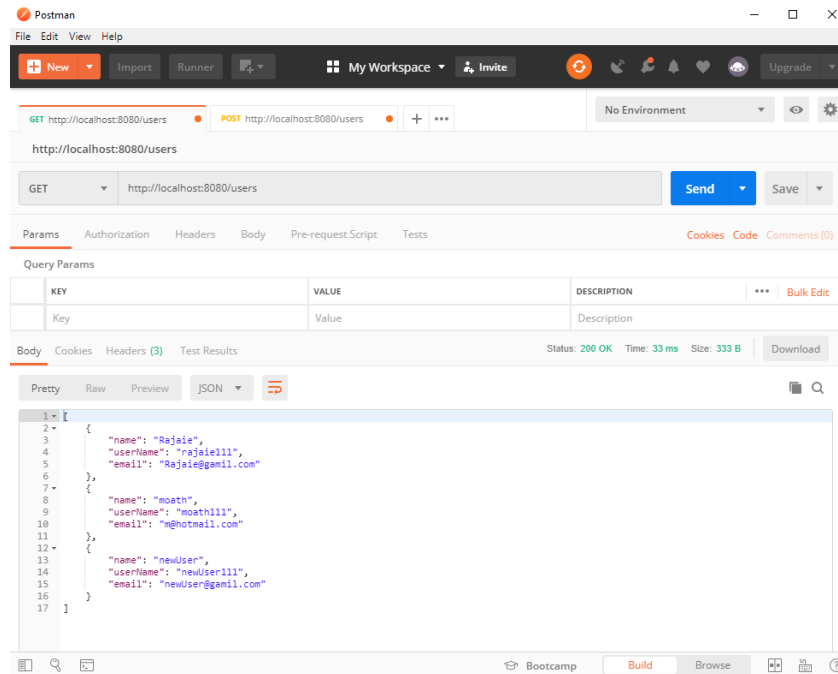


Figure 9.14 Get All Users with the New User

When delete a resource we usually specify the identifier of the resource to be deleted in the URL, for example if we want to remove an object and its identifier is “test” then the URL will be “localhost:8080/users/test” and the method will be DELETE, In our controller we can specify mapping for “users/test” but this will not be practical as we will need a mapping for each object we have, instead we can tell spring that “test” is a variable using “@PathVariable” annotation, value between { } will be stored in name variable.

```

@RequestMapping(method=RequestMethod.DELETE, value="/users/{userName}")
    public boolean deleteUser(@PathVariable String userName) {
        //delete with username from the UserService arrayList
    }

```

To do a PUT (update) we usually send the updated object in the payload JSON and the identifier of the object as a path variable

```

@RequestMapping(method = RequestMethod.PUT, value="/users/{userName}")
    public boolean updateUser(@RequestBody User user, @PathVariable
    String userName) {
        //update with username in the UserService arrayList
    }

```

So for now we walked through the basic api's in spring MVC and we saw how easily we built the application using spring boot, for now we can manipulate in memory data, data is not persisted yet which means if you restart the application all updated on data will be removed, in Next experiment we will walk through spring data jpa which is a framework to support persisting our java object in database so we will have a serialized data, also we will take a look at deployment the spring boot application using the built in servlet container and using an external servlet container (Tomcat).

## **5. Todo**

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



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

## **EXP. No. 10. Spring Boot Part 2**

### **1. Requirements**

- ❖ Knowledge of the java programming language
- ❖ Basic Understanding of MVC (Model View Controller) architecture
- ❖ Basic Knowledge of Maven tool
- ❖ Basic Idea about Spring framework (preferably Spring MVC)
- ❖ Basic Knowledge of relational databases (like MYSQL)
- ❖ Knowledge of java ORM (Object Role Modeling) models is extremely helpful but not required
- ❖ Spring Tool Suite software (STS) (For windows 64-bit download this [Link](#))
- ❖ Postman software (For windows 64-bit download this [Link](#))
- ❖ MYSQL server/Workbench (For windows 64-bit download this [Link](#))

### **2. Objectives**

- ❖ Understanding the fundamentals of Spring Data JPA framework
- ❖ Create mapping between java models and relational database models
- ❖ Connect and perform CRUD operations to MYSQL database using Spring Data JPA framework

### 3. Introduction

- ORM

ORM (Object relational mapping) is a technique to convert data between models within object-oriented languages to serialized data to be stored (In relational database in our case). ORM models presents many advantages, the main advantage is the decoupling to the database connector, databases queries vary between one another, so ORM takes care of this difference.

- JPA

JPA (Java Persistence API) is a specification or standard which provides java developers with ORM to manage relational database (map entity classes to sql tables), after configuring the mapping it's sent to some framework to handle it.

- Spring Data JPA

Spring Data JPA will be the framework to handle the mapping as we mentioned, it basically helps implementing repositories to retrieve and manipulate data in database. Spring Data JPA is built on top of Hibernate, hibernate is a very commonly used framework for ORM in java.

### 4. Procedure

Go through EXP. No. 9 Spring Boot Part 1 experiment steps 4.1 and 4.2 above and create a new project.

#### *4.1. Adding Dependencies:*

To run this application, we will use new dependencies as the JPA dependence and the MySQL dependency. These dependencies have some functions which will help us reduce the code and simplifies it.



### ➤ JPA Dependency

As we saw for Spring MVC Spring Boot gives us a meta dependency which is wrapper for all the dependencies we need for Spring Data JPA, to add the dependency go to pom.xml file and add the following dependency. This dependency contains all you need to create the needed repositories.

```
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

### ➤ MYSQL Dependency

This dependency will help us connecting and accessing the data base to add this dependency go to pom.xml file and add the following dependency.

```
<dependency>  
<groupId>mysql</groupId>  
<artifactId>mysql-connector-java</artifactId>  
</dependency>
```

After adding the dependencies, the pom.xml file will be as shown in the following code.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>DataBase</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>DataBase</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>

```

## 4.2. Configurations

We need some configuration for Hibernate (which JPA is built in top of) along with configuration to connect to database, keep in mind that spring Boot went far to provide an embedded database for development purposes, but this will not work for production purposes.

Configurations are set in `src/main/resources/application.properties` file we saw in the previous experiment, these configuration will be per database, which means configurations will change if the database changes and most likely this will be the only change, compared to JDBC templates this a big jump for database abstraction.

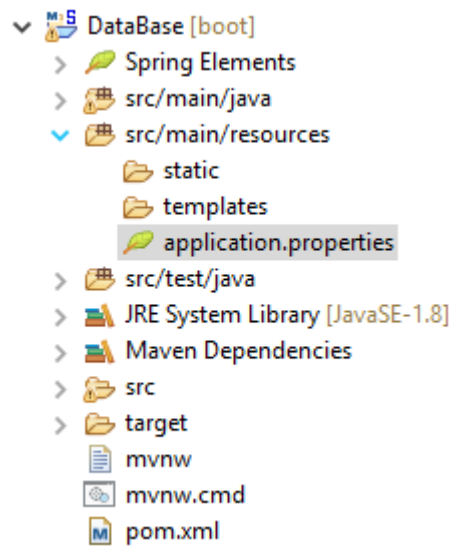


Figure 10.1 Application Properties

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/MyDB?useSSL=false
spring.datasource.username=databaseName
spring.datasource.password=databasePass
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

- server.port: this property overrides the default port Spring Boot will run on (8080).
- spring.datasource.url: this will be the url for the database connection.
- \*useSSL=false: is used to disable SSL connection since database will be running on the same machine, in case database is running on different machine we need to configure certificate for both sides.
- spring.datasource.username: the username for the database(locally or in different machine)
- spring.datasource.password: the password for the database.
- spring.jpa.hibernate.ddl-auto: this option tells hibernate to recreate the database on each run, for production purposes we usually use update does not create.
- spring.jpa.hibernate.naming-strategy: this will define the default name strategy of the database tables and columns.
- spring.jpa.properties.hibernate.dialect: this defines the language which will be uses to talk to database.

### 4.3. *ORM mapping:*

The idea of JPA is to create mapping which makes it possible to convert java model to relational model, in this experiment we will take a quick look at the mapping using annotation based configuration, many enterprise application use xml based configuration but annotation is the new trend and most of the new projects are trying to eliminate the use of xml configurations. The structure of the project will be as in 4.3. In this project we will add a new package called repositories in the com.example.demo package.

- Models and Tables:

- ◆ Tables

In ORM classes are mapped to database table, this is achieved using **@Entity** annotation on the class definition.

#### ◆ Id

Relational databases assign a primary key to each record in which it will be identified by, Spring data JPA uses the primary key to realize if a record needs to be added or updated, to specify a primary key we use `@Id` annotation, this annotation applies primary key to the attribute annotated, this applies not-null and unique constraints as well.

Primary keys are usually auto generated, there are many ways to generate them, here for simplicity we will be using auto increment for simplicity, we use `@GeneratedValue(strategy = GenerationType.AUTO)`, this strategy is by default set.

#### ◆ Columns

Columns by default are scanned and persisted by JPA, we can use `@Column` annotation to apply more specific properties, we can specify the name of the column, the uniqueness and the ability to be null. To avoid persisting a column in the database table we can use `@Transient` to tell JPA to ignore a specific attribute.

#### ◆ User Class

In this walk through we will be implementing a User entity which has an Id and User Name, the Id is the main key which will be generated automatically, models in spring boot basically are normal classes, so in `com.example.demo.models` package go ahead and create a class User and add its attributes. Notice that all attributes should be private and should be accessed using accessors (getters and setters) to follow encapsulation standards.

```

package com.example.demo.models;

import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.*;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="user_id")
    private Integer id=null;

    @Column(nullable = false, unique = true)
    private String userName;

    public User(Integer id, String userName) {
        super();
        this.id = id;
        this.userName = userName;
    }

    public User() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
}

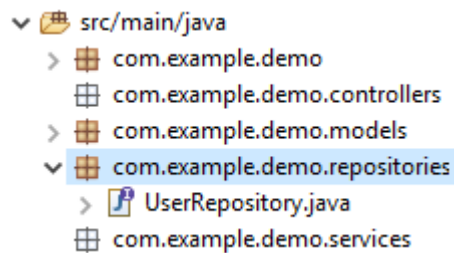
```

➤ JPA repositories

JPA introduces interface-based models for accessing the database, implementation is not required, all we need is to create custom repositories interfaces and extend predefined interfaces provided by Spring data JPA to be able perform operations on database. will create a repository interface for user, this interface should extend a repository interface in JPA, we can extend CrudRepository or JpaRepository. JpaRepository provides additional functionality including pagination, yet if you are going to need only simple CRUD operations then CrudRepository is recommended to keep the separate of concerns.

Creating repository interface is simple, we only need to create an interface in the com.example.demo.repositories as shown in Figure 10.2 and extends one of the spring jpa repositories, for user we will create UserRepository as shown in the code below.

JpaRepository takes generic types, the first type is the entity which is User in our case, the second one is the type of the id, in our previous demonstration of the mapping relation we specified the type of the id to be int which is a primitive data type, JpaRepository requires the type to be serializable, for that we can use Long or Integer which are wrapper classes for the primitive types long and int, so **we will need to change the type in the entity also to Integer**.



*Figure 10.2 User Repository*

```
package com.example.demo.repositories;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.models.User;

public interface UserRepository extends JpaRepository<User, Integer> {

    }

```

## ➤ Service

Just that simple we have a DAO layer now, in the previous experiment we discussed that service layer should take care of controlling the DAO, so we need to inject our created DAO to the service. Spring JPA will take care of initializing the DAO, so we only need to wire it to our service, note that no need for any XML definition or annotation for the interface to be marked as a bean, extending the JPA/CRUD repositories is enough. You will create a UserService class in the com.example.demo.services package you created, and make an object of the repository you created, this interface is connected to the database which allows to perform queries on the database. Don't forget to add the @Autowired annotation before creating the repository object and @Service before the class definition to notify spring about service as shown in the following code.

```
package com.example.demo.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.demo.repositories.UserRepository;

@Service
public class UserService {

    @Autowired
    UserRepository userRepository;
}
```

Now we can add a new methods to get all users and add a user to the database using the UserRepository object by using findAll and save method respectively. Don't forget to import the User class. These methods will be used by the controller which will be created next.

```
import com.example.demo.models.User;

public List<User> getAll() {
    return userRepository.findAll();
}
public String addUser(User user) {
    userRepository.save(user);
    return "success";
}
```



## ➤ Controller (APIs)

In a normal way controllers use services so we need a reference for our user service in the controller, on the startup of the application spring will initialize the singleton service bean (if you do not know what beans are please do some reading about spring beans) so we do not need to initialize service, we can wire the service to our controller using “@Autowired” annotation which wires the service bean to the reference defined in our controller.

That is all what we need to use UserService, so now we can use the user service to return all the user data by calling getUserList method we created in the UserService class.

```
package com.example.demo.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.models.User;
import com.example.demo.services.UserService;

@RestController
public class UserController {

    @Autowired
    UserService service;

    @RequestMapping("/users")
    public List<User> getAllUsers(){
        return service.getAll();
    }

    @PostMapping("/users")
    public String addOne(@RequestBody User user) {
        return service.addUser(user);
    }
}
```

Before running the application run the MySQL workbench and connect to the database, then you can run the application as we did in the previous experiment and test the methods you created.

## 4.4. Extra Things You May Do

### ➤ Delete

To delete object, we can use delete API which takes the id of the object to be deleted, we can write our own custom delete methods same as get methods.

```
public String deleteUser(Integer userId) {
    userRepository.deleteById(userId);
    return "success";
}
```

### ➤ Multiple Tables and relational database

When building applications, we are not writing data to single table, in our data model we often have relationship between entities. We will talk about different associations:

- One to One
- One to Many
- Many to Many

### ➤ One to One

one to one entity association is a relationship between two entities where the two side of relation maps to only one object in the other side. For this relation, we will use user/office relationship in which every user has an office and every office can be for only one user, we will create office class with some attributes.

```
package com.example.demo.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="office")
```

```

public class Office {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column (name="office_id")
    private Integer id=null;

    @Column (name="office_roomCode")
    private String roomCode;
}

```

To build the association we will create a field for user inside office, to map it to User entity we will use @OneToOne annotation, we will add cascade type persist, this means when persisting office object, the user object added to it will be persisted as well, then we will add @JoinColumn which is the column that will be used to join the two entities.

```

@OneToOne (cascade=CascadeType.PERSIST)
@JoinColumn(name="user_id",referencedColumnName="user_id")
private User user;

```

Name property in @JoinColumn will define the foreign key column in office table, referencedColumnName will define the primary key column in the target entity.

This is a unidirectional relation, we can access user from office, but we cannot access office from user, to make the relation bidirectional we can apply some changes.

First we need to add variable to hold the reference of office in user object, then we will annotate it with @OneToOne annotation, also we will give mappedBy = "user", here we are telling jpa that this reference is mapped by user variable in office.

```

@OneToOne (mappedBy="user")
private Office office;

```

So now we have a bidirectional relationship where we can access office from user and user from office.

### ➤ One to Many

one to many is an association between two entities where one entity has reference to more than one entity while the other entity has reference to only one entity.

For this relation we will use user/account relation, one user can have many accounts while and account can belong to only one user. First, we will create account class with some attributes and accessors.

```
package com.example.demo.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="account")
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column (name="account_id")
    private Integer id=null;

    @Column (name="account_codeNumber")
    private String codeNumber;

}
```

In User class we will add a collection of account, then we will use `@OneToMany` annotation to map the relation, then we will use `@JoinColumn`, this may be confusing because in one to one relationship this annotation was used in the owning entity (the entity which will have the foreign key) but in this case its clear that Account should have the foreign key which is the id of the user and not the opposite (User will have list of foreign keys since user can have list of accounts).

Currently this is a unidirectional relation (we will not have reference for user inside account object) so that leaves us with only the User to be able to specify the Join Column, JPA is smart enough to know that the Account is the owning entity.

Also, we will specify the cascade type to be persist so accounts get persisted along with the user, we will add additional attribute in the `@JoinColumn` which is `nullable= false`, this means each persisted account should be referenced by a user.

```
@OneToMany(cascade = CascadeType.PERSIST)
@JoinColumn(name = "user_id", nullable = false)
private List<Account> accounts = new ArrayList<>();
```

Until now the relation is bidirectional, we can access accounts from user, but the opposite is not possible. To make the relation unidirectional first we will add a variable of type user inside Account class, now we will use @ManyToOne annotation, many accounts to one user , now we will specify the joincolumn using annotation @JoinColumn, note that we no longer need to specify the join column in user table so we will remove it, also we need to specify the variable user mapped by.

So, in User class we will have

```
@OneToMany(cascade = CascadeType.PERSIST, mappedBy="user")
@JoinColumn(name = "user_id", nullable = false)
private List<Account> accounts = new ArrayList<>();
```

In Account class, we will have

```
@ManyToOne
@JoinColumn(name="user_id")
private User user;
```

So now we have a bidirectional relation where we can access any side of the relation from the other side.

#### ➤ Join Table:

Join table can be used as an alternative method to map one to many relationship, in many cases in one to many relationship, relationship could be not mandatory, association between two entities could occur but it's not a must, in this case a junction table is used to avoid having null values in the foreign key columns where the relation is not there, junction tables are used mainly for many to many mapping but we will come to this later.

Junction table called also a join table, is a median table which holds the relationship between two tables, it will have a foreign key for both the tables and the 2 keys are the primary key of this table.

To demonstrate join tables, we will use relation user/car, note that every user can have many cars but a car can only belong to a one user, but a user also can have no car, to map this relation in jpa we will use @JoinTable annotation.

After creating car entity we will create a list of type car in user, we will put @OneToMany annotation since the relationship is one to many, the we will add @JoinTable, we need to specify

the name of the junction table which will be user\_car, then we will specify the join column using attribute joinColumn for the user side and the inverseJoinColumns for the car side.

```
@OneToMany(cascade = CascadeType.PERSIST)
@JoinTable(name = "user_car", joinColumns=@JoinColumn(name="user_id"),
           inverseJoinColumns=@JoinColumn(name="car_id"))
private List<Account> cars = new ArrayList<>();
```

### ➤ Many to Many

many to many relationships are relationships where each entity can map to many entities in the other side of the relation, we will use user/course relationship, a user can be enrolled in many courses and course can have many users.

First, we will demonstrate a unidirectional relationship, we will have to pick the owning side of the relation, lets pick the user, so basically we should have a list of courses inside user, then we will use @ManyToMany annotation, we will also use @JoinTable annotation as we did in the previous section.

```
@ManyToMany(cascade = CascadeType.PERSIST)
@JoinTable(name = "user_course", joinColumns=@JoinColumn(name="user_id"),
           inverseJoinColumns=@JoinColumn(name="course_id"))
private List<Course> courses = new ArrayList<>();
```

So now we have a unidirectional relation where we can access courses from user, to make the relation bidirectional, we will not be changing anything on the user, in course entity we will establish an association.

In course entity we will create a list of users, again we will use @ManyToMany annotation, with mapped by attribute. So now we have a bidirectional relationship where we can access any side of the relation.

```
@ManyToMany( cascade= CascadeType.PERSIST ,mappedBy = "user")
private List<User> users;
```

## 5. Todo

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