2 Android Layouts

Introduction

In this lab we will be learning how to use and extend the Android user interface library. In a number of 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.

Objectives

At the end of this lab you will be expected to know:

- 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.
- How to build web app in webView

Procedure

For this lab we will be creating a "Customer List" application. It is a simple app that allows a user to view and edit a list of customer. All tasks for this lab will be based off of this application. Over the course of the lab you will be iteratively refining and adding functionality to the Customer List app. With each iteration you will be either improving upon the previous iteration's functionality, or you will be implementing the same functionality in a different way.

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. Below are some of the more common layout types that are built into the Android platform.

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

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.

Customer List Application

For this lab we will be creating a "Customer List" application. It is a simple app that allows a user to view and edit a list of customer.

Create a new Android Project:

1. 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**.
2. In the New Project screen, enter the following values:
   - Application Name: "Customer List"
   - Company Domain: "birzeit.edu"
3. Click **Next**.
4. In the **Target Android Devices** screen, keep the default values and click **Next**.
5. In the **Add an Activity to Mobile** screen, select **Empty Activity** and click **Next**.
6. In the **Customize the Activity** screen, enter the following values:
   - Activity Name:" CustomerMain"
   - Layout Name: "customer_main"

Creating and Filling Business Entities:

To begin development, you will need to create **Customer** class which is used to hold data about customer.

-Add **Customer.java** class under **edu.birzeit.customerlist** package which exists in **java** folder.

-Add the following properties to **Customer** class:

1. CustomerID (Long): unique ID number for customer.
2. Name (String): holds the customer name.
3. Phone (String): holds the phone number.
4. Gender (String): holds the gender type.

```
package edu.birzeit.customerlist;
public class Customer {
    public long CustomerID;
    public String Name;
    public String Phone;
    public String Gender;
    public long getCustomerID() {return CustomerID;}
    public void setCustomerID(long customerID) {CustomerID = customerID;}
    public String getName() {return Name;}
    public void setName(String name) {Name = name;}
    public String getGender() {return Gender;}
    public void setGender(String gender) {Gender = gender;}
    public String getPhone() {return Phone;}
    public void setPhone(String phone) {Phone = phone;}
}
```

Task:

- Override **toString()** function to return the customer information as a string, but each of customer properties should be written in separated line. **Hint**: use "**\n**" to add new line.
- Override **equals(Object objCust)** function to compare between two customers. The compression should be done using name, phone, and age. Hint: use "**instanceof**" to check whether the passed Object **(objCust)** is instance of **Customer**.

Viewing Customers using Simple List

Work done in this section will be limited to the **CustomerMain.java** file. **CustomerMain** is an Activity class which displays a vertical scrollable list of all the customers.

In previous experiment , 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 have to 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 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.main)** from **onCreate** method.
- Define the linear layouts with Vertical Orientations.

```
LinearLayout objFirstLinearLayout  = new LinearLayout(this);
LinearLayout objSecondLinearLayout= new LinearLayout(this);
objFirstLinearLayout .setOrientation(LinearLayout.VERTICAL);
objSecondLinearLayout.setOrientation(LinearLayout.VERTICAL);
```

- Add a button to the **First Linear Layout** with "**Add Customer**" as a text. The height and width of the button should be wrapped to the button content.

```
Button btnAddCustomer = new Button(this);
btnAddCustomer.setText("Add Customer");
btnAddCustomer.setLayoutParams(new LinearLayout.LayoutParams
(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
objFirstLinearLayout .addView(btnAddCustomer);
```

- Build a new instance of **ScrollView** in order to make **Second Linear Layout** scrollable by adding the linear layout to **ScrollView** object.

```
ScrollView objScrollView = new ScrollView(this);
objScrollView.addView(objSecondLinearLayout);
```

- Add the built **scrollview** object to **First Linear Layout**.

```
objFirstLinearLayout .addView(objScrollView);
```

- Finally, set the **First Linear Layout** as a main content view for **CustomerMain** Activity.

```
setContentView(objFirstLinearLayout);
```

Task:

Till now , the add customer button has not implemented yet. However, you have to
test the above code. To see that follow, the following steps:

- Define a new list of **Customer** as a data member in **CustomerMain** class.
- Create a method to initialize the defined customer list for specified number of customers (e.g 100) .
- For each customer in the list, create **TextView** object and use the **toString** method to display the customer information using the created **TextView** object.
- Then, add the created **TextView** object to **Second Linear Layout** as a view.
- Run the application on the emulator (See Figure 1).
- Once the application is tested correctly, **clear** the added code for this Task.

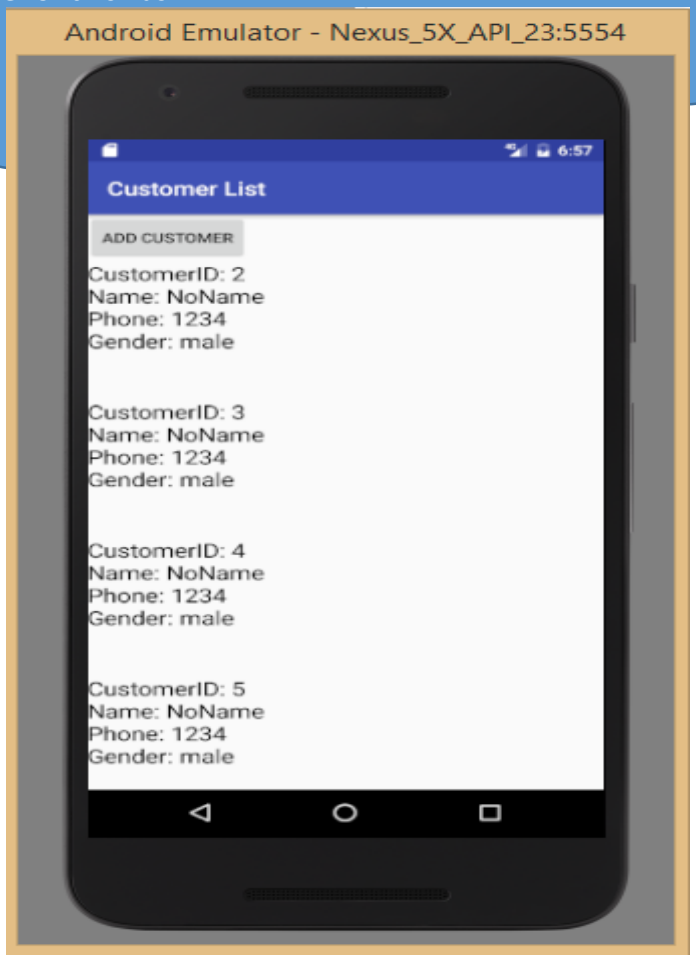

Figure 1

Adding Customers:

In this task, you will give the user the ability to enter their own customers by displaying new activity which contains all required customer information that should be entered. In order to accomplish this task, the following steps show the main requirements of this task:

- Add a listener to "**Add Customer**" button using **setOnClickListener** method.
- Upon **click** on button, the "**add customer activity**" should be appeared.
- Create new XML file in order to design the layout which will be used in "**add customer activity**".
- Validate the input information about new customer e.g(Not empty name).
- Return back to Customer View Activity once the customer is added successfully.

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

- Create a new XML file, where the name of XML is "**add_customer.xml**".
- Use the Graphical Layout for "**add_customer.xml**" to design the UI for **add_customer** activity as shown in Figure 2.

Figure 2

- Use the following IDs for EditText, Spinner ,and Button views: **btnAddNewCustomerm ,txtCustomerID, txtCustomerName, txtCustomerPhone , and spnGender.**
- Return back to **CustomerMain** class and define a public and static list of **Customer**.

```
public static ArrayList<Customer> commonLstCustomer = new
ArrayList<Customer>();
```

- Add **AddCustomerActivity.java** class under **edu.birzeit.customerlist** package which exists in **java** folder.
- Inherit the **Activity** class for the added class (**AddCustomerActivity**).

```
public class AddCustomerActivity extends Activity
```

- Override the **OnCreate** method in **AddCustomerActivity** class and set the **add_customer.xml** to be the content view of this activity

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.add_customer);
    }
```

- In **OnCreate** method, find and initialize the **spnGender** for defined list of data. Depending on the requirements, the list should have two options: **Male** and **Female**.

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

- In OnCreate method, find the remaining views: **txtCustomerID**, **txtCustomerName**, and **txtCustomerPhone** by using **findViewById** method in order to extract and build **Customer** object.

```
final EditText txtCustomerID =(EditText)
findViewById(R.id.txtCustomerID);
final EditText txtCustomerName =(EditText)
findViewById(R.id.txtCustomerName);
final EditText txtCustomerPhone =(EditText)
findViewById(R.id.txtCustomerPhone);
```

- Find the **btnAddNewCustomer** button view. Then, impalement **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 **commonLstCustomer** list which exists in **CustomerMain** class. Once the customer has been added successfully, the activity should disappear.

```
        Button btnAddNewCustomer = (Button) findViewById(R.id.btnAddNewCustomer);
        btnAddNewCustomer.setOnClickListener(new  View.OnClickListener() {
            public void onClick(View v) {
                String customerID= txtCustomerID.getText().toString();
                String customerName=txtCustomerName.getText().toString();
                String customerPhone=txtCustomerPhone.getText().toString();
                String customerGender= spnGender.getSelectedItem().toString();
                Customer objCustomer = new Customer();
                //Check fields content
                if(customerID.equals(""))
                {
                    objCustomer.CustomerID=0;
                }
                else
                {
                    objCustomer.CustomerID=Long.parseLong(customerID);
                }
                if(customerName.equals(""))
                {
                    objCustomer.Name="No Name";
                }
                else
                {
                    objCustomer.Name=customerName;
                }
                if(customerPhone.equals(""))
                {
                    objCustomer.Phone="No Phone Number";
                }
                else
                {
                    objCustomer.Phone=customerPhone;
                }
                objCustomer.Gender=customerGender;
                //add customer object to commonCustomerList in CustomerMain
                CustomerMain.commonLstCustomer.add(objCustomer);
                //hide the activity
                finish();
                //Start CustomerMain Activity
                Intent myIntent=new Intent(AddCustomerActivity.this,
                        CustomerMain.class);
                AddCustomerActivity.this.startActivity(myIntent);
            }
        });
```

- The implementation of **AddCustomerActivity** class has finished. However, you must declare the "**AddCustomerActivity**" activity in **AndriodManifest.xml** File because this file describes the components of the application such as activities. To do that , put an activity tag under application tags like this:

  "<activity android:name=".AddCustomerActivity"></activity>"

- Now, return back to **CustomerMain** class in order to add listener for **AddCustomer** button which is used to show add customer activity (**AddCustomerActivity**). The following code shows how you can add listener and how the other activity can be activated:

```
btnAddCustomer.setOnClickListener( new OnClickListener() {
public void onClick(View v) {
Intent myIntent = new Intent(CustomerMain .this,
AddCustomerActivity.class);
CustomerMain.this.startActivity(myIntent);
finish();
}
});
```

- To display the customers that exist in **commonLstCustomer** list, 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 section. The following code shows a suggested implementation:

```
for(Customer objCust : commonLstCustomer)
{
    TextView txtCustomerInfo = new TextView(this);
    txtCustomerInfo.setText(objCust.toString());
    objSecondLinearLayout .addView(txtCustomerInfo);
}
```

Building Web Apps in WebView

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" />
```

ToDo

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