THE HONG KONG POLYTECHNIC UNIVERSITY
Department of Electronic and Information Engineering

EIE3109 Mobile Systems and Application Development
Laboratory Exercise 2
Android App Development (Android Studio 4.0.1)

Objectives:

At the end of the lab exercise, students will be able to

(i) Create empty projects in Android Studio
(ii) Create AVD
(iii) Identify Android Studio's project files corresponding to the MVC model
(iv) Create user interface (UI)
(v) Customize UI elements
(vi) Connect UI elements to code
(vii) Enable event handling (OnClickListener)
(viii) Understand SharedPreferences, Content Provider, and database query

Intended Learning Outcomes to be assessed in this laboratory exercise:

Part	OC1	OC2	OC3	OC4
A-G			Yes	Yes

Table 1

*Refer to the **Appendix A** for Intended Learning Outcomes and the corresponding Rubrics*

Introduction:

This lab exercise provides an introduction to Android mobile application development using Android Studio. Students are asked to develop 2 apps using Java and realise data communication within the system. The aim of this lab is to give students a taste of mobile application development with Android Studio. Students are expected to have basic programming and mathematical skills.

Equipment:

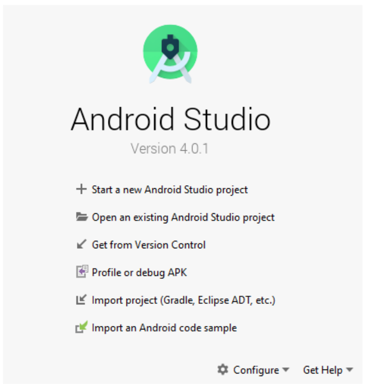
Android Studio 4.0.1

Procedure:

A. Create a Hello World project

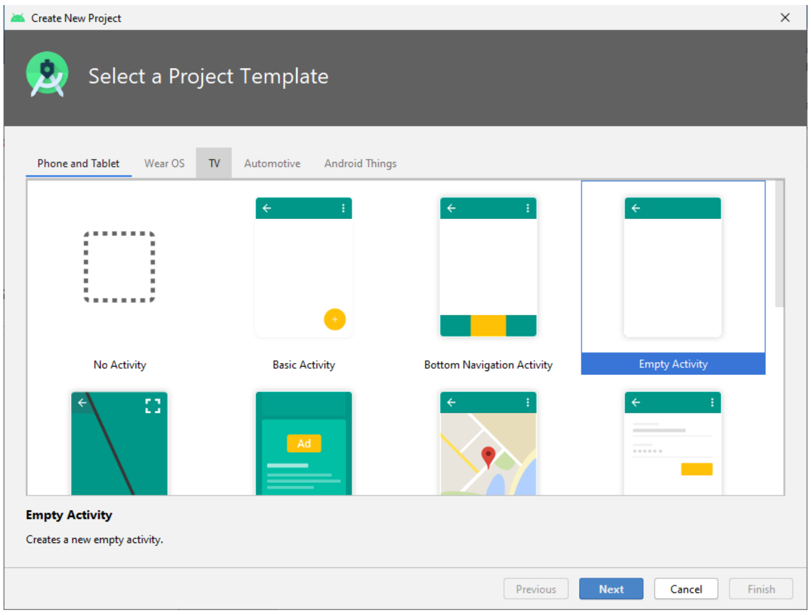
1.

Open **Android Studio** and choose **Start a new Android Studio project**.


Android Studio
Version 4.0.1
Start a new Android Studio project
Open an existing Android Studio project
Get from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

2.

Choose **Empty Activity**.

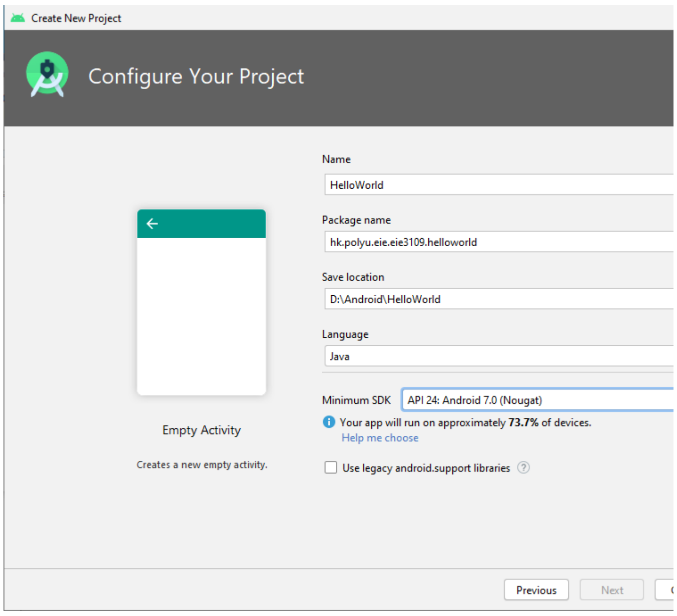


Empty Activity

Creates a new empty activity.

3.

Configure your project
Name: **HelloWorld**
Package Name:
hk.polyu.eie.eie3109.helloworld
Save Location:
C:\Temp\EIE3109\YOUR_STUDENTID\HelloWorld (if you are working at CF504)
Language: **Java**
Minimum API level: **API 24: Android 7.0 (Nougat)**



Name: HelloWorld

Package name: hk.polyu.eie.eie3109.helloworld

Save location: D:\Android\HelloWorld

Language: Java

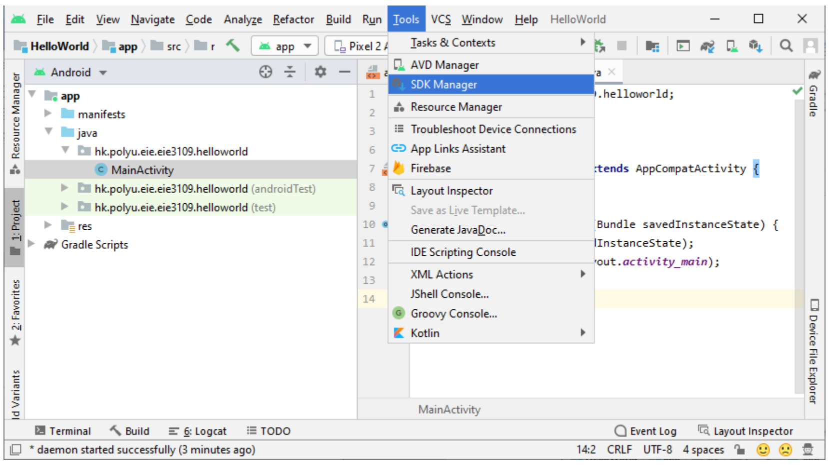
Minimum SDK: API 24: Android 7.0 (Nougat)

Your app will run on approximately **73.7%** of devices.

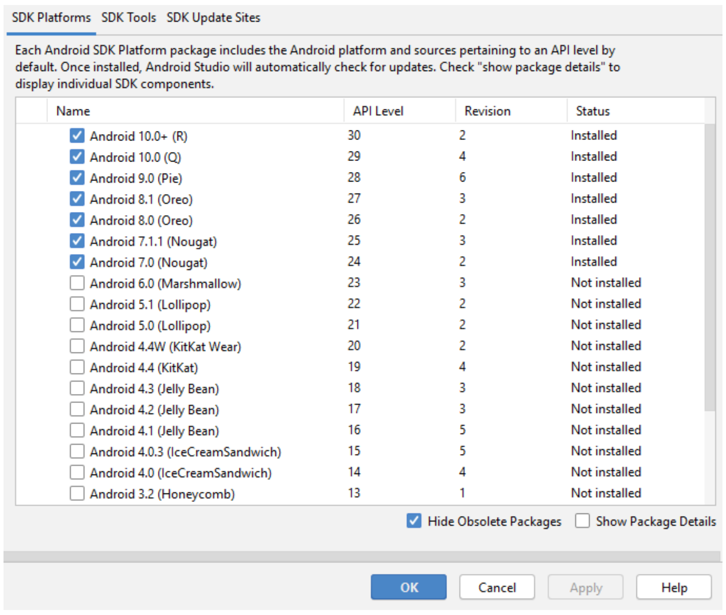
☐ Use legacy android.support libraries

4.

Download APIs
Tools->SDK Manager
(No need to do this part in the lab)



Download and install **API 24** or other versions than you use.
(No need to do this part in the lab)



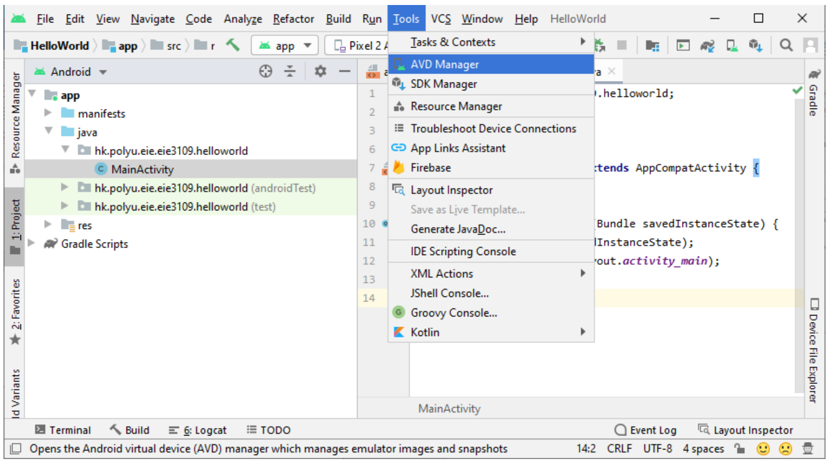
Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
✓ Android 10.0+ (R)	30	2	Installed
✓ Android 10.0 (Q)	29	4	Installed
✓ Android 9.0 (Pie)	28	6	Installed
✓ Android 8.1 (Oreo)	27	3	Installed
✓ Android 8.0 (Oreo)	26	2	Installed
✓ Android 7.1.1 (Nougat)	25	3	Installed
✓ Android 7.0 (Nougat)	24	2	Installed
Android 6.0 (Marshmallow)	23	3	Not installed
Android 5.1 (Lollipop)	22	2	Not installed
Android 5.0 (Lollipop)	21	2	Not installed
Android 4.4W (KitKat Wear)	20	2	Not installed
Android 4.4 (KitKat)	19	4	Not installed
Android 4.3 (Jelly Bean)	18	3	Not installed
Android 4.2 (Jelly Bean)	17	3	Not installed
Android 4.1 (Jelly Bean)	16	5	Not installed
Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
Android 4.0 (IceCreamSandwich)	14	4	Not installed
Android 3.2 (Honeycomb)	13	1	Not installed

✓ Hide Obsolete Packages Show Package Details

5.

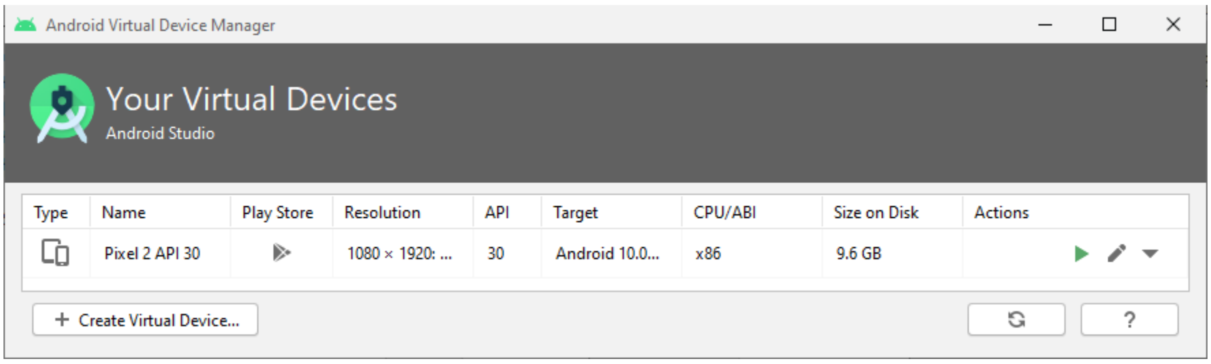
Configure AVD
Tools -> AVD Manager
(No need to do this part in the lab)



```
.helloworld;
tends AppCompatActivity {
Bundle savedInstanceState) {
InstanceState);
out.activity_main);
```

Opens the Android virtual device (AVD) manager which manages emulator images and snapshots   14:2 CRLF UTF-8 4 spaces

Click **Create Virtual Device** at a bottom to create an AVD (emulator)

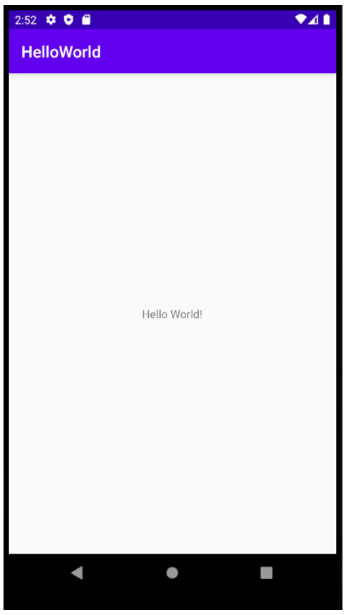


Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 2 API 30		1080 × 1920: ...	30	Android 10.0...	x86	9.6 GB	

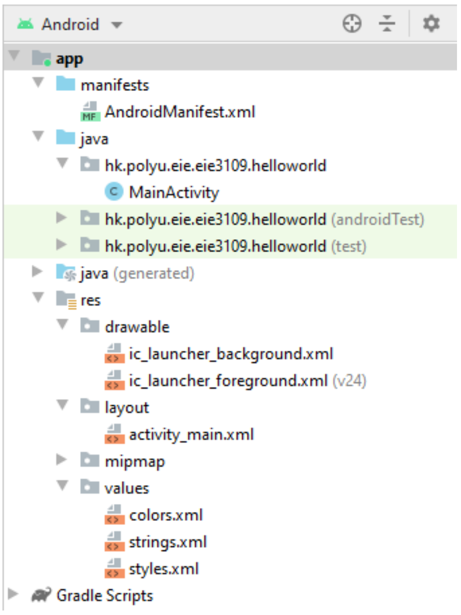
6.

Run or **Run-> Run 'app'**

Note: It will take some time for the emulator to load. DO NOT CLOSE the emulator. Otherwise, you will have to wait for it to boot up again.



B. Create Basic Interface



```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ hk.polyu.eie.eie3109.helloworld
        MainActivity
    ▶ hk.polyu.eie.eie3109.helloworld (androidTest)
    ▶ hk.polyu.eie.eie3109.helloworld (test)
  ▶ java (generated)
  ▼ res
    ▼ drawable
        ic_launcher_background.xml
        ic_launcher_foreground.xml (v24)
    ▼ layout
        activity_main.xml
    ▶ mipmap
    ▼ values
        colors.xml
        strings.xml
        styles.xml
▶ Gradle Scripts
```

On the left panel, there are different files generated by Android Studio.

AndroidManifest.xml under manifests folder contains information of which Activity to start when the program first launches and also the names of Activities in this project.

Under java->package name (hk.polyu....), an Activity class is created. Double click the **MainActivity** to show the Java source code of the Activity class.

res folder contains icons and images, layouts, and constants of the project.

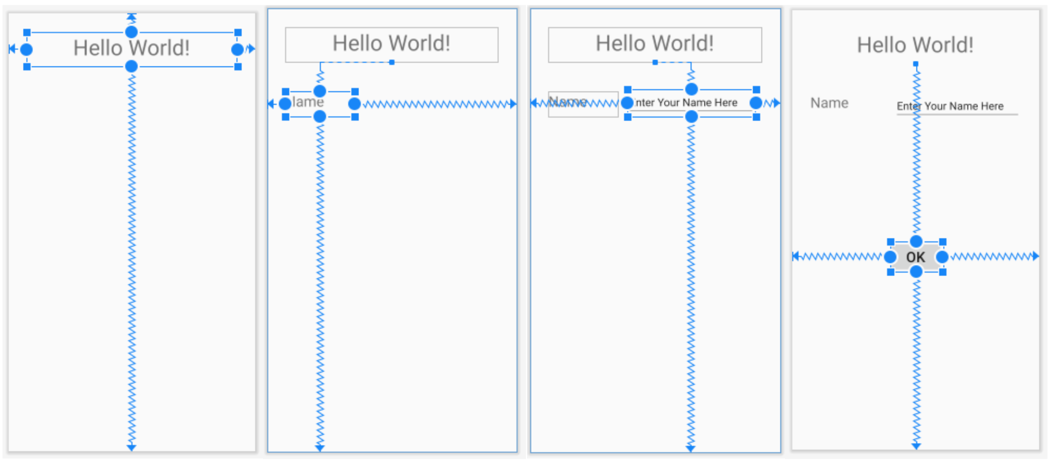
1. In **res->layout->activity_main.xml**

Move the "Hello World" **TextView**, then add a **TextView**, an **EditText** (Plain Text), and a **Button** to the layout.

Click and drag the components from the Palette to the phone.

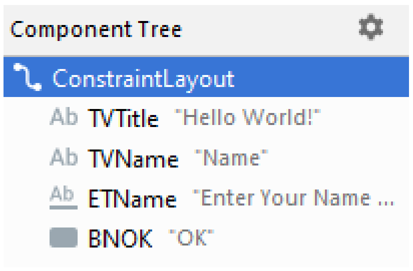
Connect the constraint (blue lines) to the border or to another component.

*If you don't see any layout, refer to Appendix B



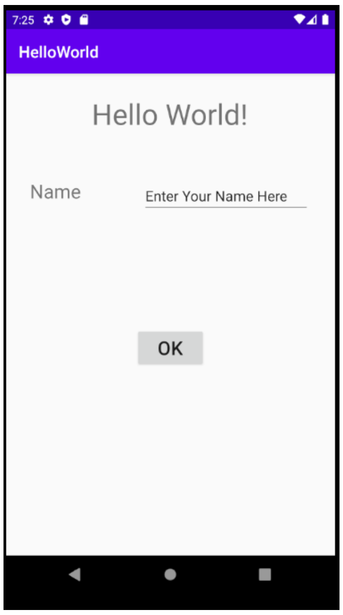
2.

For each component, change the **id**, **text**, and its **size** under **Attributes** on the right side.


Component Tree
ConstraintLayout
Ab TVTitle "Hello World!"
Ab TVName "Name"
Ab ETName "Enter Your Name ...
BNOK "OK"

3.

Run



C. Button Event

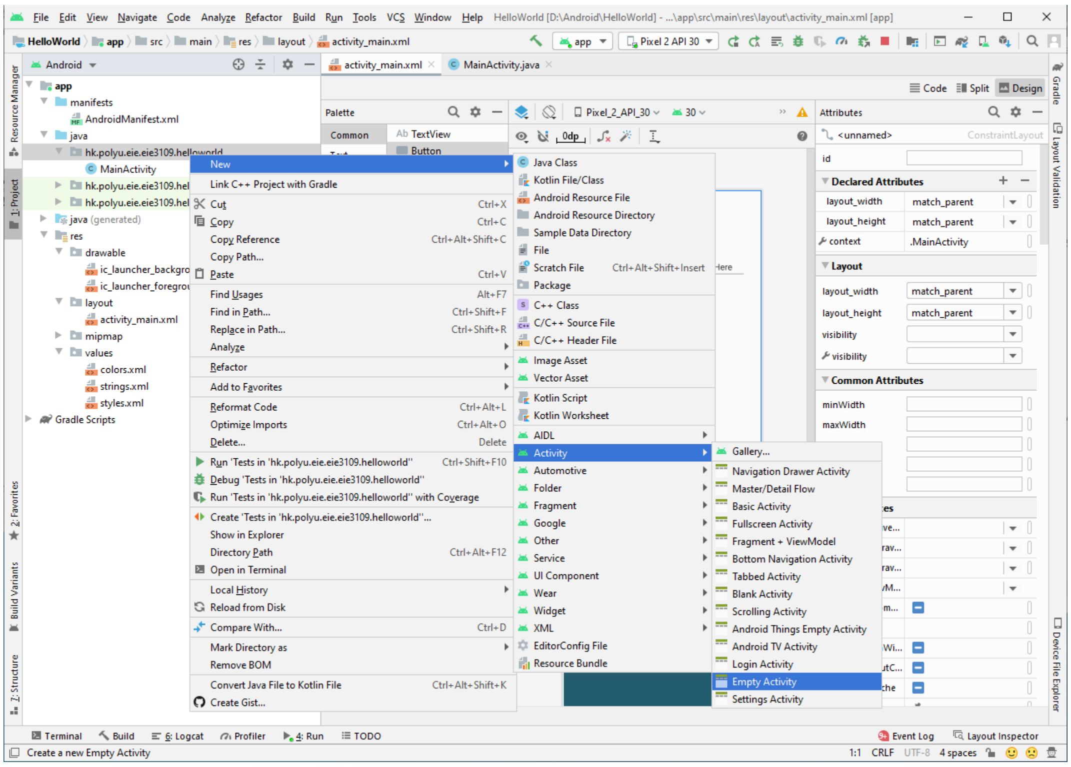
The app so far only contains the interface and cannot respond to any interaction. We will use the OK button on the MainActivity to change to another Activity.

To achieve this goal we will

1. Create a new activity,
2. Find the button in the layout file (xml),
3. Enable the OnClickListener for the button.

1. Create a new activity

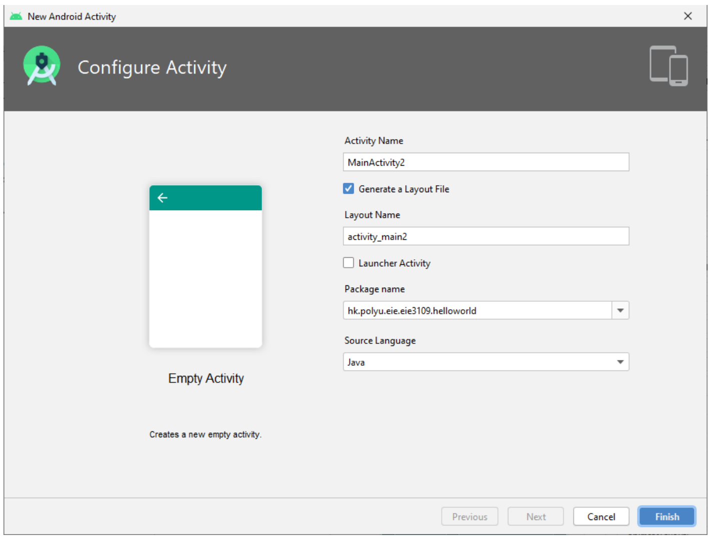
i. Right Click **hk.polyu.eie.eie3109.helloworld->New->Activity->Empty Activity**



ii.

Configure Activity.

Activity Name: **MainActivity2**
Check Generate Layout File
Layout Name: **activity_main2**
Package Name:
hk.polyu.eie.eie3109.helloworld
Source Language: **Java**


New Android Activity

Configure Activity

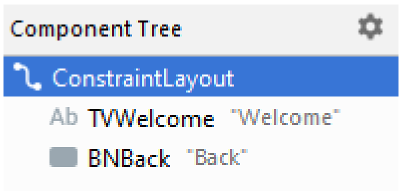
Activity Name: MainActivity2
[✓] Generate a Layout File
Layout Name: activity_main2
[ ] Launcher Activity
Package name: hk.polyu.eie.eie3109.helloworld
Source Language: Java

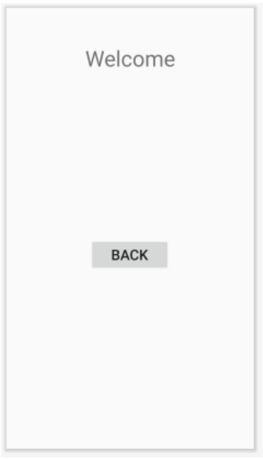
Empty Activity

Creates a new empty activity.

iii.

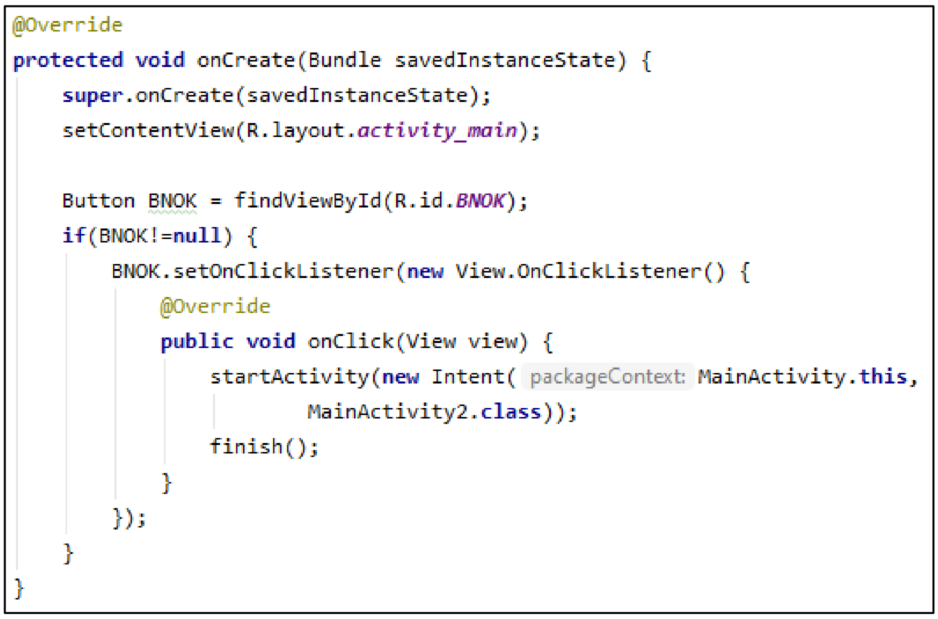
Create the interface for Main2Activity


Component Tree
ConstraintLayout
Ab TVWelcome "Welcome"
BNBack "Back"



2. In **MainActivity**,

Modify the **onCreate** method as follow: (You will need to import some libraries. Follow the hints provided by Android Studio to import the correct libraries)



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

    Button BNOK = findViewById(R.id.BNOK);
    if(BNOK!=null) {
        BNOK.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new Intent( packageContext: MainActivity.this,
                        MainActivity2.class));
                finish();
            }
        });
    }
}
```

Button BNOK = findViewById(R.id.*BNOK*);

This statement will find the Button in the layout file (xml) with an id BNOK.

The **setOnClickListener** method enables the button to response to a click event.

There are only 2 lines in the **onClick** methods. It starts a new activity and then close the current activity.

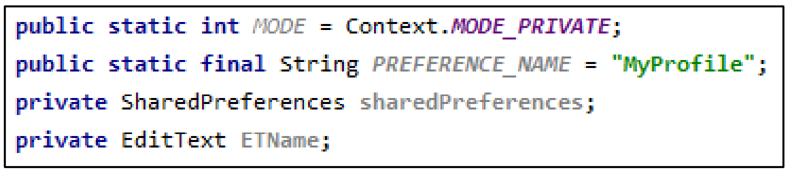
Run again and you can use the OK button to go to the new activity.

C. Pass data between Activities

SharedPreferences is a Java interface for accessing and modifying data in Android. For any particular set of preferences, there is a single instance of this class that all clients share. Modifications to the preferences must go through a **SharedPreferences.Editor** object to ensure the preference values remain in a consistent state and control when they are committed to storage. Objects that are returned from the various get methods must be treated as immutable by the application.

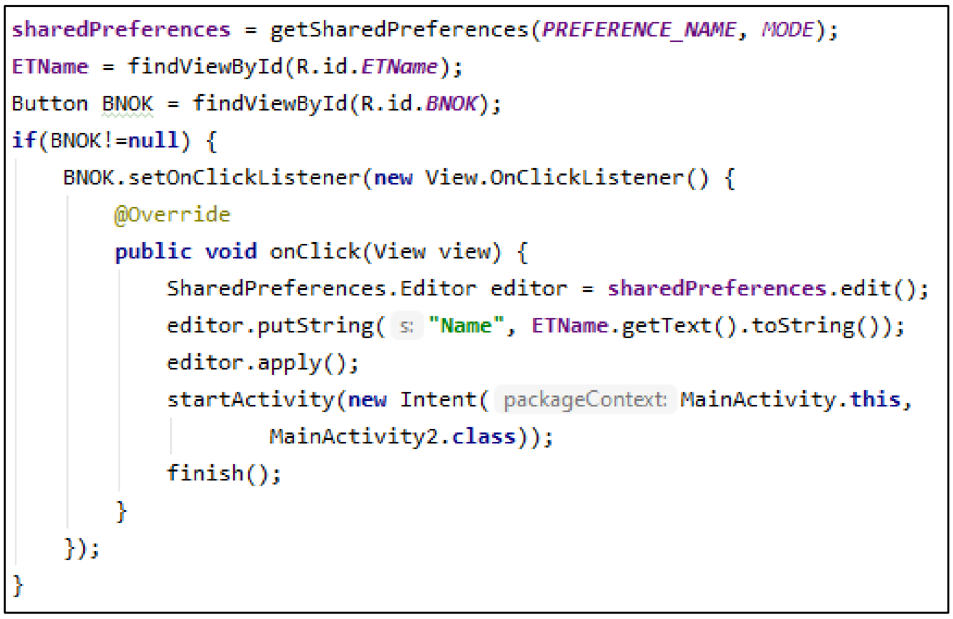
We first set up the **SharedPreferences** in **MainActivity**.

1. In **MainActivity**, set up the constants and variables as class variables for **SharedPreferences**.



```
public static int MODE = Context.MODE_PRIVATE;
public static final String PREFERENCE_NAME = "MyProfile";
private SharedPreferences sharedPreferences;
private EditText ETName;
```

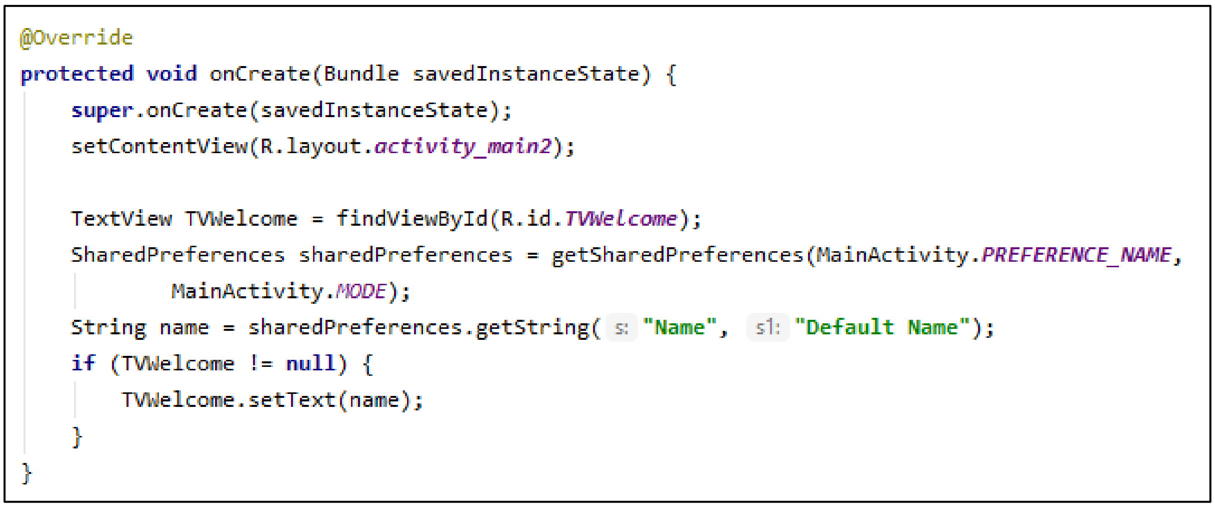
2. Save the data to SharedPreferences by using Editor and **putString** method.



```java
sharedPreferences = getSharedPreferences(PREFERENCE_NAME, MODE);
ETName = findViewById(R.id.ETName);
Button BNOK = findViewById(R.id.BNOK);
if(BNOK!=null) {
    BNOK.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            SharedPreferences.Editor editor = sharedPreferences.edit();
            editor.putString( s: "Name", ETName.getText().toString());
            editor.apply();
            startActivity(new Intent( packageContext: MainActivity.this,
                    MainActivity2.class));
            finish();
        }
    });
}
```

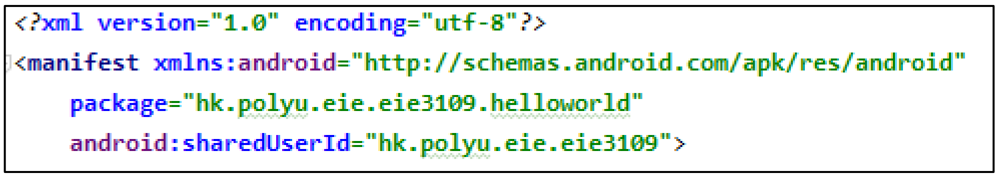
3. Load the SharedPreferences in **MainActivity2**

In the **OnCreate** method in **MainActivity2**, find the TextView, get the SharedPreferences, load the value of “Name”, and set the “Name” to the TextView.



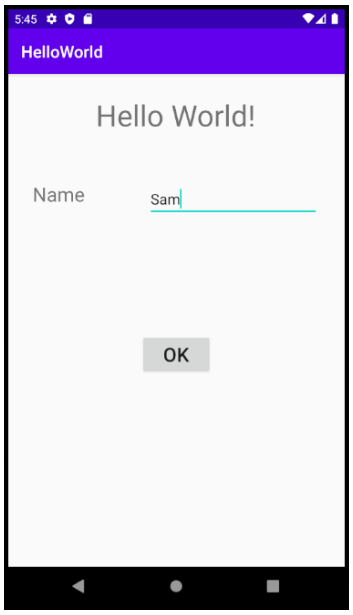
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    TextView TVWelcome = findViewById(R.id.TVWelcome);
    SharedPreferences sharedPreferences = getSharedPreferences(MainActivity.PREFERENCE_NAME,
            MainActivity.MODE);
    String name = sharedPreferences.getString( s: "Name", s1: "Default Name");
    if (TVWelcome != null) {
        TVWelcome.setText(name);
    }
}
```

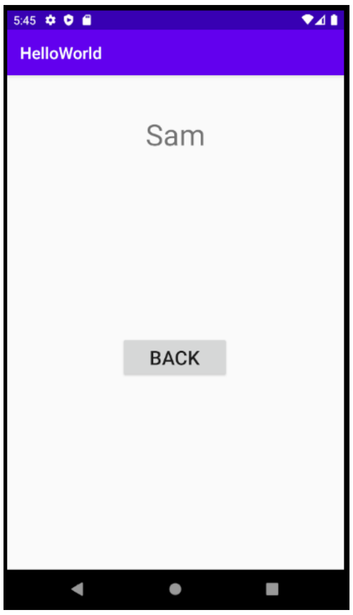
4. Add a **sharedUserID** in **AndroidManifest.xml** so that apps having the same sharedUserID can access the data using SharedPreferences. (This is for sharing data between apps in next part, no need to add sharedUserId if you are sharing data within the same app only.)



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hk.polyu.eie.eie3109.helloworld"
    android:sharedUserId="hk.polyu.eie.eie3109">
```

Run and you may need to uninstall the app first as your sharedUserId has been changed.



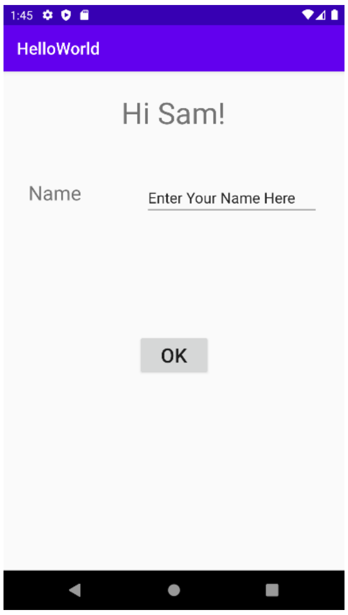


****

<u>Exercise 1</u>

1.

Load the SharedPreferences in MainActivtiy so that it reads the saved data in SharedPreferences and change the title as follow:



2.

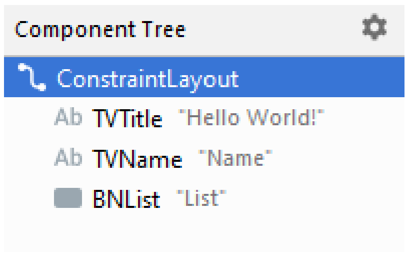
Enable the back button to go back to MainActivity.

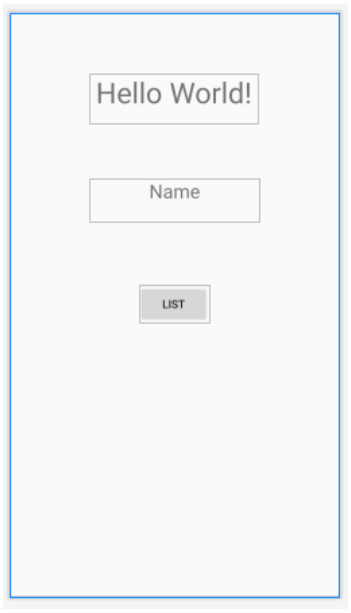
****

D. <u>Pass data between apps</u>

We will build an app that stores a to-do lists and use ShardPreferences to get data from the HelloWorld app.

1. Create a new Empty Project named "ToDoList"
2. Build the interface for MainActivity





Modify **AndroidManifest.xml** to use the same **sharedUserID**. Make sure it is **exactly the same** as the sharedUserID in the HelloWorld project.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hk.polyu.eie.eie3109.todolist"
    android:sharedUserId="hk.polyu.eie.eie3109">
```

In **MainActivity**, define the SharedPreferences constants.

```
public static final int MODE = Context.MODE_PRIVATE;
public static final String PREFERENCE_NAME = "MyProfile";
public static final String PREFERENCE_PACKAGE = "hk.polyu.eie.eie3109.helloworld";
```

Load the SharedPreferences in the OnCreate method.

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

    TextView TVName = findViewById(R.id.TVName);

    Context c = null;
    try {
        c = this.createPackageContext(PREFERENCE_PACKAGE, CONTEXT_IGNORE_SECURITY);
        SharedPreferences sharedPreferences = c.getSharedPreferences(PREFERENCE_NAME, MODE);
        String name = sharedPreferences.getString( s: "Name",  s1: "Default Name");
        if(TVName != null) {
            TVName.setText(name);
        }
    } catch (PackageManager.NameNotFoundException e) {
        e.printStackTrace();
    }
}
```

Run and you should see the name you entered in the HelloWorld project appears in this project.

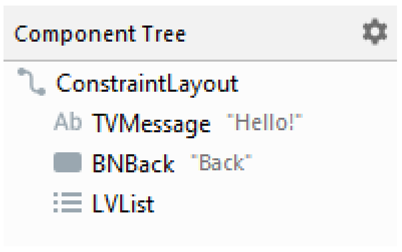
E. <u>Using ListView</u>

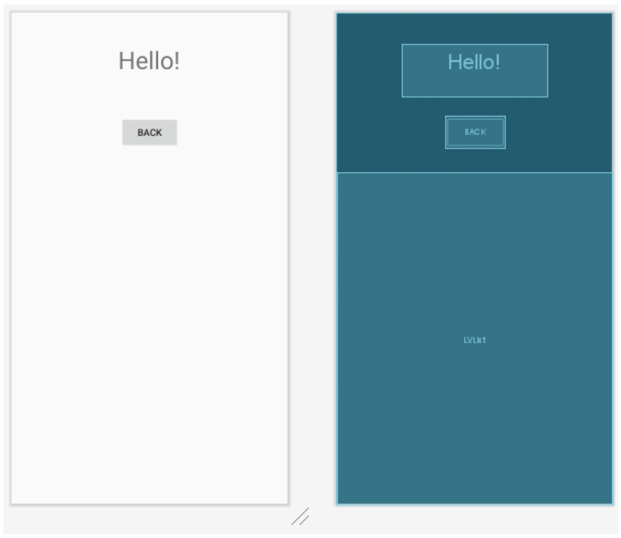
A ListView is a view that shows items in a vertically scrolling list. The items come from the ListAdapter associated with this view.

1. Create a new Activity named ListActivity
   Right click on hk.polyu.eie.eie3109.todolist->New->Activity->Empty Activity

2. Configure Activity.
   Activity Name: **ListActivity**
   Check Generate Layout File
   Layout Name: **activity_list**
   Package Name: **hk.polyu.eie.eie3109.todolist**
   Source Language: **Java**

3. Build the interface of ListActivity (ListView is under **Legacy**).


Component Tree
ConstraintLayout
Ab TVMessage "Hello!"
BNBack "Back"
LVList



4. In **ListActivity**, create 3 class variables. The String array contains the “To Do” items.

```
private ListView myToDoList;
private String[] myStringList;
private ArrayAdapter<String> myAdapter;
```

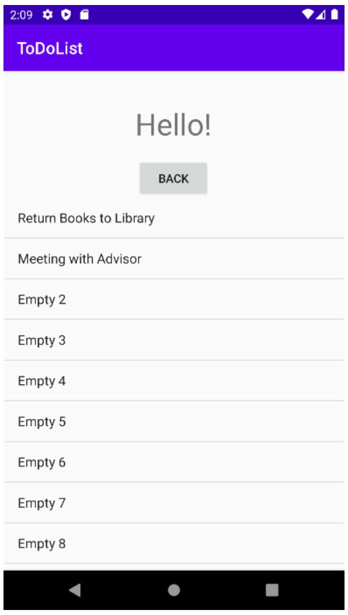
5. Modify the **OnCreate** method as follow:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_list);

    myStringList = new String[10];
    for (int i=0; i<10; i++) {
        myStringList[i] = "Empty " + i;
    }
    myStringList[0] = "Return Books to Library";
    myStringList[1] = "Meeting with Advisor";

    myToDoList = findViewById(R.id.LVList);
    myAdapter = new ArrayAdapter<String>( context: this, android.R.layout.simple_list_item_1,
            myStringList);
    myToDoList.setAdapter(myAdapter);
}
```

ListView needs an adapter. “myAdapter” contains the layout and the String array for the list item.

Enable the **List** Button in **MainActivity** to go to **ListActivity**.

Run

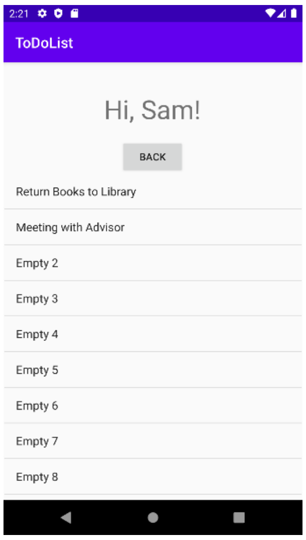


****

Exercise 2

1. Load SharedPreferences to change the Textfield

2. Enable the back button to go back to MainActivity.



Return Books to Library
Meeting with Advisor
Empty 2
Empty 3
Empty 4
Empty 5
Empty 6
Empty 7
Empty 8

****

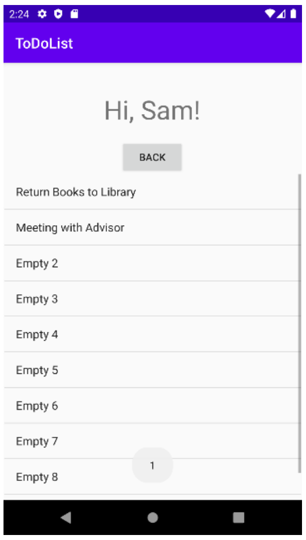
9. Toast

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout. By setting the setOnItemClickListener of the ListView, we can dislplay a Toast message when user clicks on each list item.

In the **OnCreate** method, set the **setOnItemClickListener** for **myToDoList**.

```
myToDoList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Toast.makeText(getApplicationContext(), Integer.toString(i),
                Toast.LENGTH_SHORT).show();
    }
});
```

Run and click on a list item



Return Books to Library
Meeting with Advisor
Empty 2
Empty 3
Empty 4
Empty 5
Empty 6
Empty 7
Empty 8
1

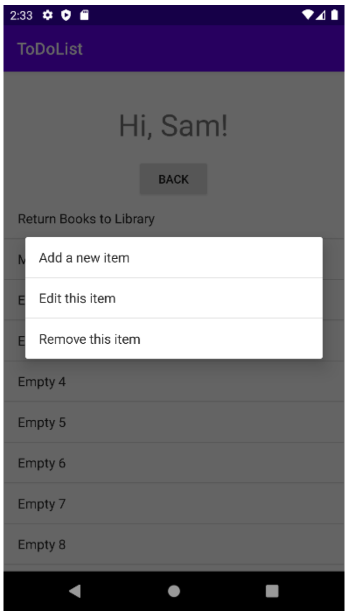
10. Dialogs

A **Dialog** is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

An **AlertDialog** is a subclass of Dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout. In the code below, we use a custom layout (list_item.xml) and build 3 choices (Add a new item, Edit this item, and Remove this item) for user to select.

```
myToDoList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        AlertDialog.Builder builder = new AlertDialog.Builder( context: ListActivity.this);
        ListView options = new ListView( context: ListActivity.this);
        options.setAdapter(new ArrayAdapter<String>( context: ListActivity.this,
                android.R.layout.simple_list_item_1,
                new String[] {"Add a new item", "Edit this item", "Remove this item"}));
        builder.setView(options);

        final Dialog dialog = builder.create();
        dialog.show();

        options.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                dialog.dismiss();
            }
        });
    }
});
```

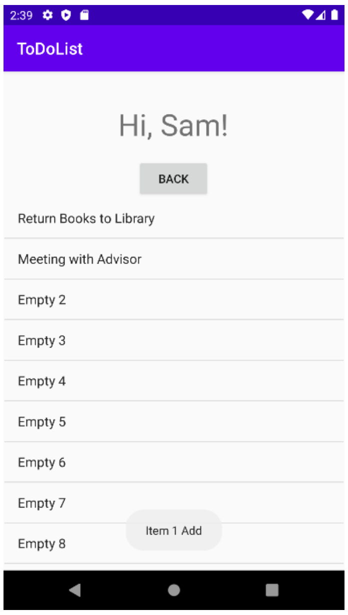
Run



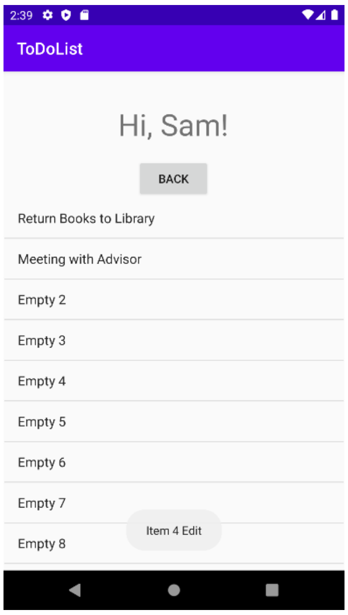
****

Exercise 3

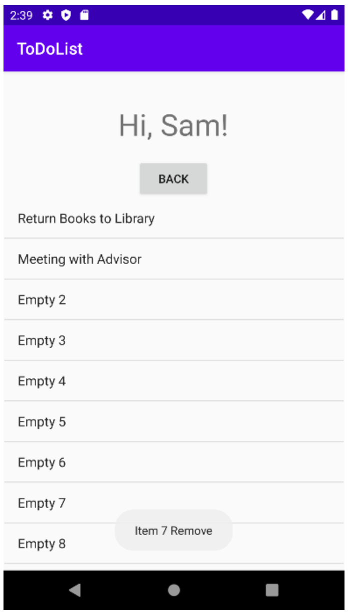
For Add, Edit and Remove, display the **item location** and the corresponding **dialog option** in a **Toast** message.



Hi, Sam!
Return Books to Library
Meeting with Advisor
Empty 2
Empty 3
Empty 4
Empty 5
Empty 6
Empty 7
Empty 8
Item 1 Add



Hi, Sam!
Return Books to Library
Meeting with Advisor
Empty 2
Empty 3
Empty 4
Empty 5
Empty 6
Empty 7
Empty 8
Item 4 Edit



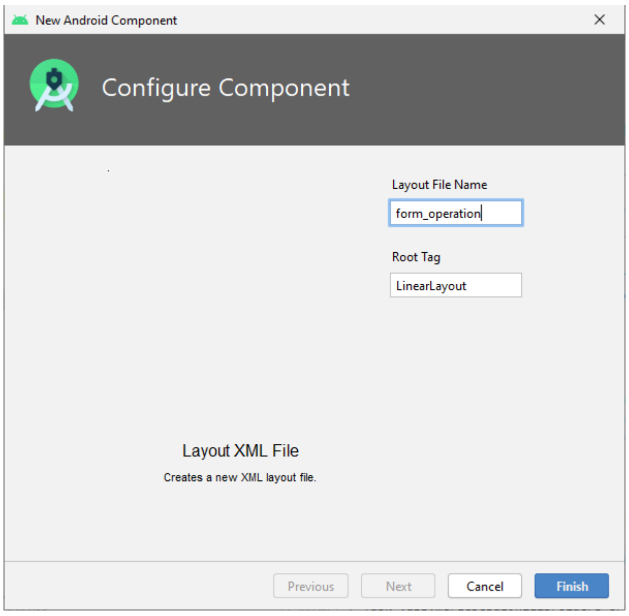
Hi, Sam!
Return Books to Library
Meeting with Advisor
Empty 2
Empty 3
Empty 4
Empty 5
Empty 6
Empty 7
Empty 8
Item 7 Remove

****

11. For the **Edit** option, we will create a **Dialog box** so that user can edit the text.

Right click layout->New->XML->Layout XML file

Layout File Name: **form_operation.xml**
Root Tag: **LinaerLayout.**



Configure Component

Layout File Name: form_operation

Root Tag: LinearLayout

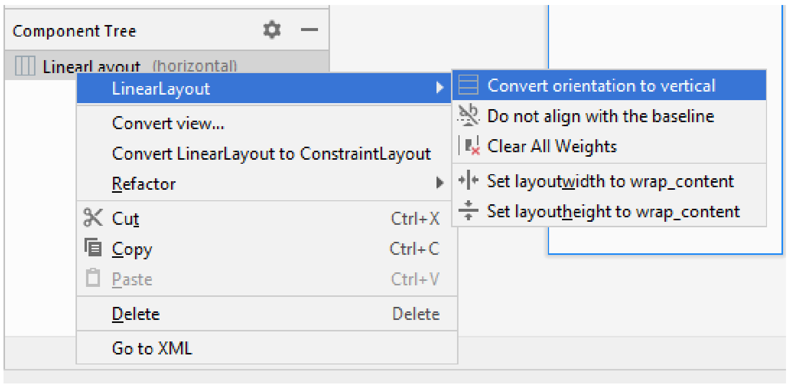
Layout XML File

Creates a new XML layout file.

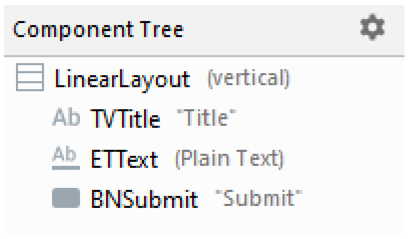
Build the interface for form_operation.xml.

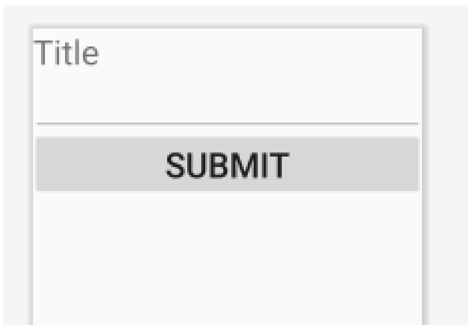
Change LinearLayout from horizontal to **vertical.**

Right click on LinearLayout under Component Tree.


Component Tree
LinearLayout (horizontal)
LinearLayout
Convert view...
Convert LinearLayout to ConstraintLayout
Refactor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Go to XML
Convert orientation to vertical
Do not align with the baseline
Clear All Weights
Set layoutwidth to wrap_content
Set layoutheight to wrap_content

Build the interface for **form_operation.xml.**





When **Edit** is clicked, set up the dialog.

```
final Dialog dialogForm = new Dialog( context: ListActivity.this);
dialogForm.requestWindowFeature(Window.FEATURE_NO_TITLE);
dialogForm.setContentView(R.layout.form_operation);
```

Set up the form and display the original text in the EditText. (You need to define item_loc to get the position)

```
TextView TVTitle = dialogForm.findViewById(R.id.TVTitle);
final EditText ETText = dialogForm.findViewById(R.id.ETText);
Button BNSubmit = dialogForm.findViewById(R.id.BNSubmit);
```

```
if(TVTitle!=null) {
    TVTitle.setText("Edit this item");
}
if(ETText != null) {
    ETText.setText(myStringList[item_loc]);
    //item_loc is the position of the item clicked on the list
}
```

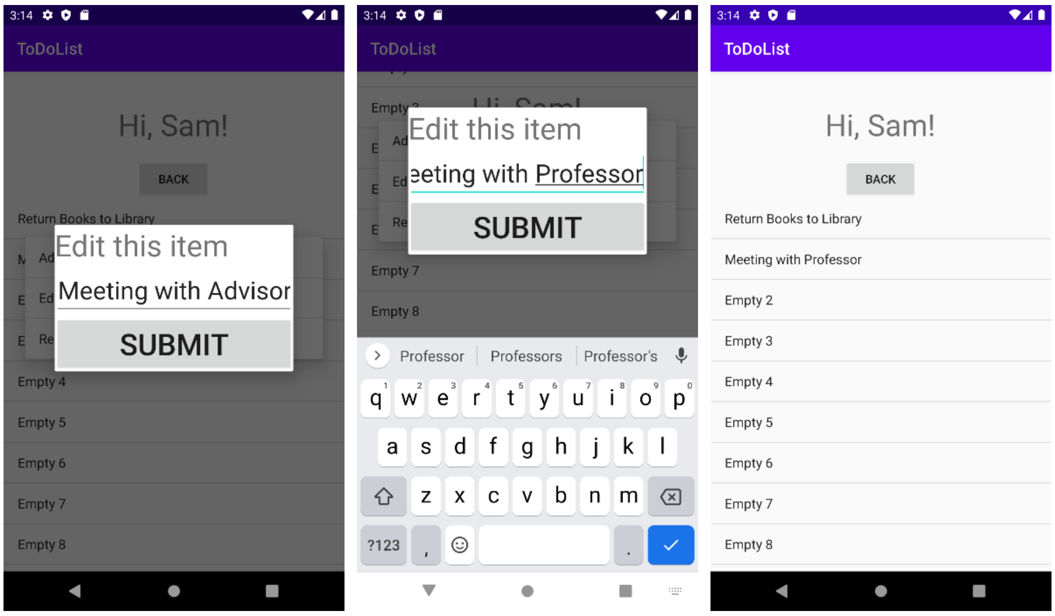
Enable the OnClickListener for the button

```
if(BNSubmit != null) {
    BNSubmit.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            myStringList[item_loc] = ETText.getText().toString();
            myAdapter.notifyDataSetChanged();
            dialogForm.dismiss();
            dialog.dismiss();
        }
    });
}
```

Then, show the dialog.

```
dialogForm.show();
```

Run



****

Exercise 4

1. String array is a fixed size array. By using **ArrayList**, the array size can vary. Replace the String array with ArrayList. Read the documentation to find out how to use ArrayList.
2. Enable **Add** and **Remove** functions using ArrayList.
3. Use **SharedPreferences** to store and load the List.

****

F. Content Provider

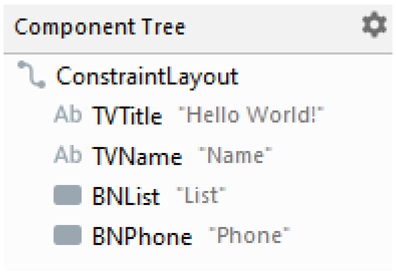
Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.

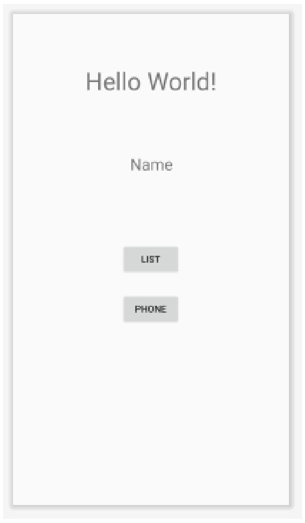
When you want to access data in a content provider, you use the ContentResolver object in your application's Context to communicate with the provider as a client. The ContentResolver object communicates with the provider object, an instance of a class that implements ContentProvider. The provider object receives data requests from clients, performs the requested action, and returns the results.

You don't need to develop your own provider if you don't intend to share your data with other applications. However, you do need your own provider to provide custom search suggestions in your own application. You also need your own provider if you want to copy and paste complex data or files from your application to other applications.

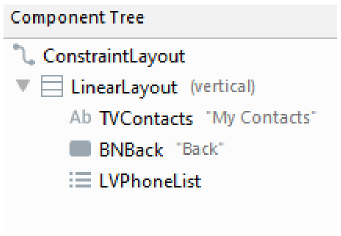
Android itself includes content providers that manage data such as audio, video, images, and personal contact information. You can see some of them listed in the reference documentation for the android.provider package. With some restrictions, these providers are accessible to any Android application.

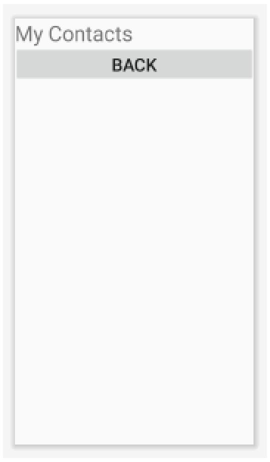
1. Add a new Button on **MainActivity**.


Component Tree
- ConstraintLayout
  - Ab TVTitle "Hello World!"
  - Ab TVName "Name"
  - BNList "List"
  - BNPhone "Phone"

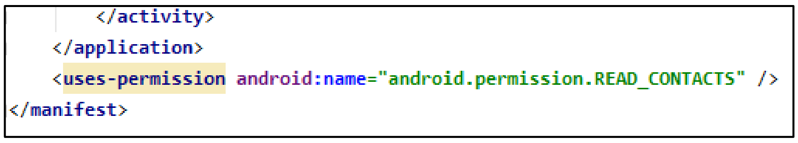


2. Create an **Empty Activity** named **PhoneActivity** and build the interface.


Component Tree
- ConstraintLayout
  - LinearLayout (vertical)
    - Ab TVContacts "My Contacts"
    - BNBack "Back"
    - LVPhoneList

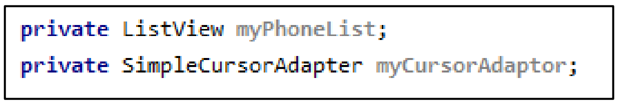


3. In **AndroidManifest.xml**, give the uses-permission to read the phone book.



```
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_CONTACTS" />
</manifest>
```

4. Create 2 class variables in **PhoneActivity**.



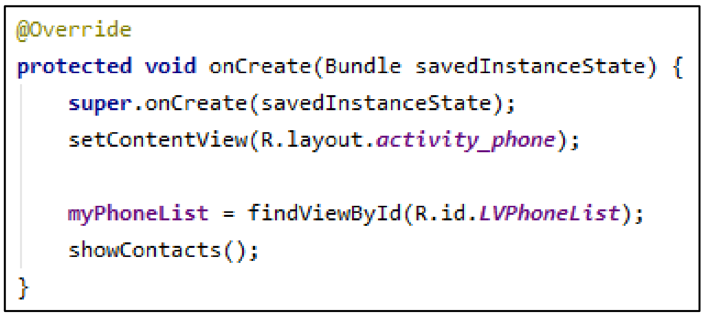
```
private ListView myPhoneList;
private SimpleCursorAdapter myCursorAdaptor;
```

5. Create two methods in **PhoneActivity**.

These two methods are for newer Android APIs (23 or above) that require user to grant permission to the app to read the contacts information on the phone. You may copy and paste the code to your java file.

```
private void showContacts() {
    // Check the SDK version and whether the permission is already granted or not.
    if (checkSelfPermission(Manifest.permission.READ_CONTACTS) != PackageManager.PERMISSION_GRANTED) {
        requestPermissions(new String[]{Manifest.permission.READ_CONTACTS}, 100);
        //After this point you wait for callback in onRequestPermissionsResult(int, String[], int[]) overriden method
    } else {
        //Code to query the phone numbers
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                       int[] grantResults) {
    if (requestCode == 100) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Permission is granted
            showContacts();
        } else {
            Toast.makeText(this, "Until you grant the permission, we cannot display the names",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

6. In the **OnCreate** method, find the ListView and call the showContacts method.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_phone);

    myPhoneList = findViewById(R.id.LVPhoneList);
    showContacts();
}
```

7. Query the phone book.

In SQL language,

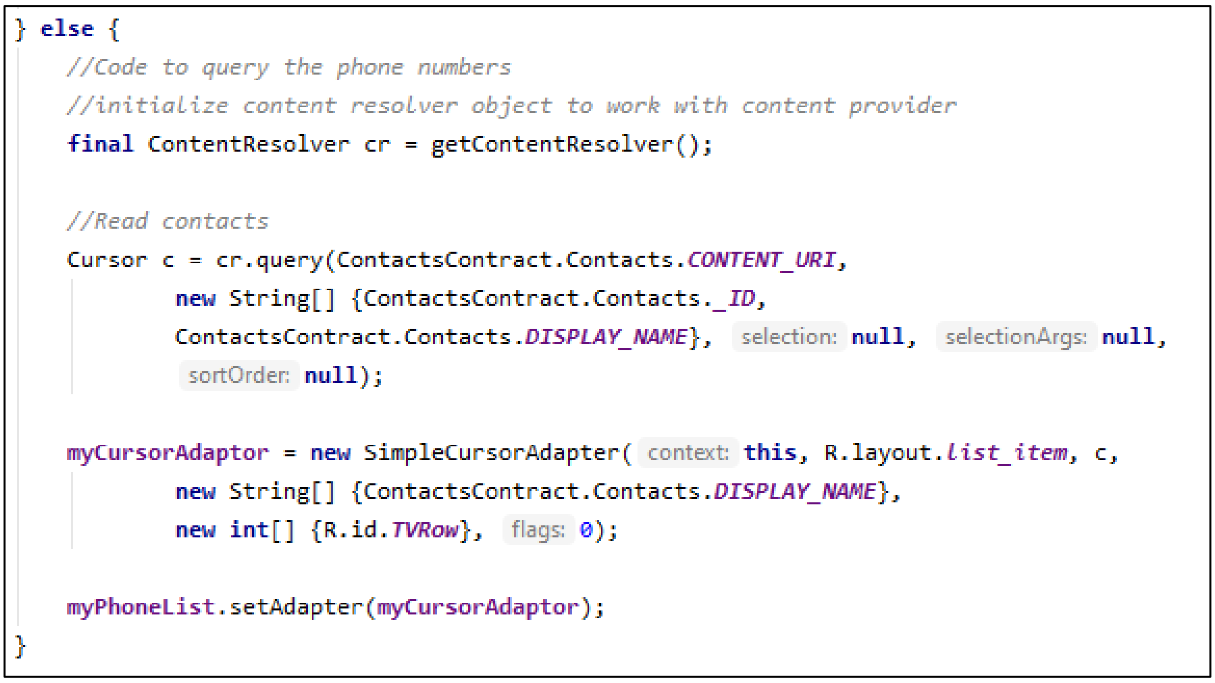
SELECT *ID, DISPLAY_NAME, AS_PHONE_NUMBER*
FROM *Contacts.CONTENT_URI*

In Java, we can use the query method:

```
query (Uri uri,        // FROM Table
String[] projection, //SELECT Columns to Return
String selection,       //WHERE clause
String[] selectionArgs,  //WHERE clause value
                                          substitution
String sortOrder)         //SORT BY
```

The return result of a query method is in Cursor type.

First initialize **content resolver**, and then query the database using a **cursor** object and set up the **adapter**, finally set the adapter for the ListView.



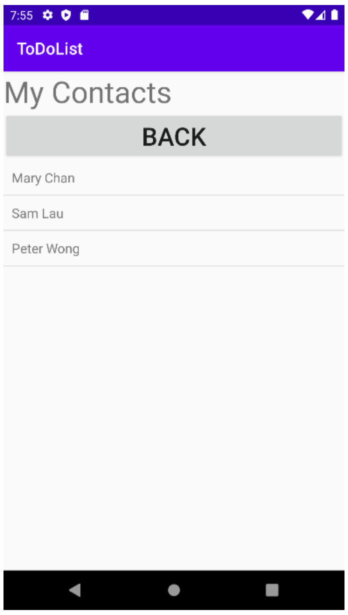
```
} else {
    //Code to query the phone numbers
    //initialize content resolver object to work with content provider
    final ContentResolver cr = getContentResolver();

    //Read contacts
    Cursor c = cr.query(ContactsContract.Contacts.CONTENT_URI,
            new String[] {ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME}, selection: null, selectionArgs: null,
             sortOrder: null);

    myCursorAdaptor = new SimpleCursorAdapter( context: this, R.layout.list_item, c,
            new String[] {ContactsContract.Contacts.DISPLAY_NAME},
            new int[] {R.id.TVRow}, flags: 0);

    myPhoneList.setAdapter(myCursorAdaptor);
}
```

Note:
list_item is a layout xml.
Create this layout in
res->**layout**->**list_item.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/TVRow"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:padding="10dp"
   android:textSize="16sp" >
</TextView>
```

Enable the **Phone** button in **MainActivity** to go to PhoneActivity.

Create a few contacts in the emulator and run.



****

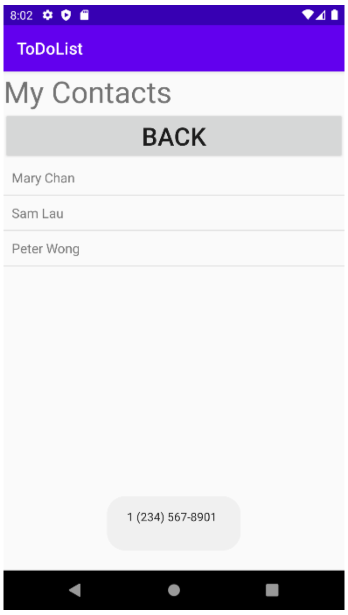
Exercise 5

1. Display the person's phone number on a Toast message when the name is clicked. i.e. implement the OnItemClickListener.

```
myPhoneList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {

    }
});
```

2.

Enable the Back button to go back to MainActivity.

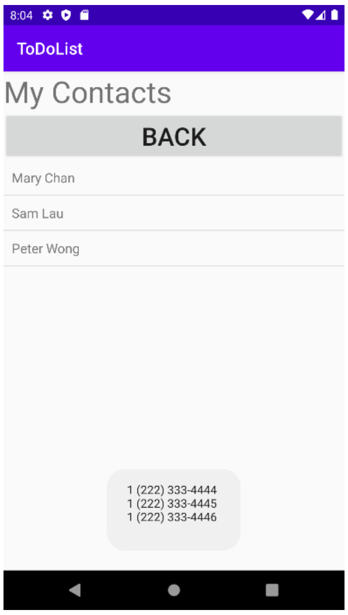


****

Assessment:

Basic: Complete Exercise 1-5

Extra feature: Display all phone numbers of a contact



Mary Chan
Sam Lau
Peter Wong

```
1 (222) 333-4444
1 (222) 333-4445
1 (222) 333-4446
```

Submission:

1. Zip the 2 projects individually and upload to Blackboard.
2. Make a video demonstration.

Assessment:

Technical (100%) – Basic (50%) and Extra features (20%), and video demonstration (30%)

~~END~~

Reference:

Android developer: https://developer.android.com/index.html

Appendix A: Subject Intended Learning Outcomes and Rubrics

Upon completion of the subject, students will be able to:

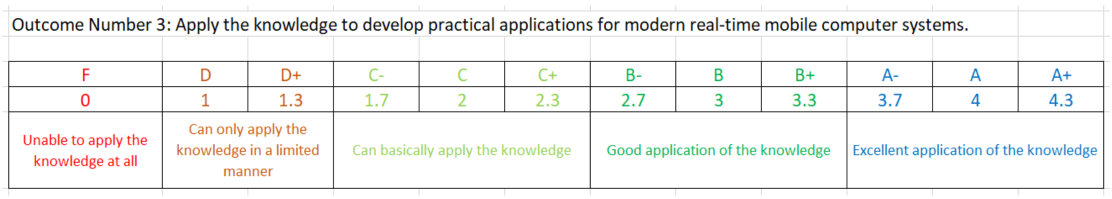
Category A: Professional/academic knowledge and skills

1. Understand the structure of real-time operating systems for modern mobile computer systems.
2. Understand the programming techniques and tools for developing software that is run in modern mobile computer systems
3. Apply the knowledge to develop practical applications for modern real-time mobile computer systems.

Category B: Attributes for all-roundedness

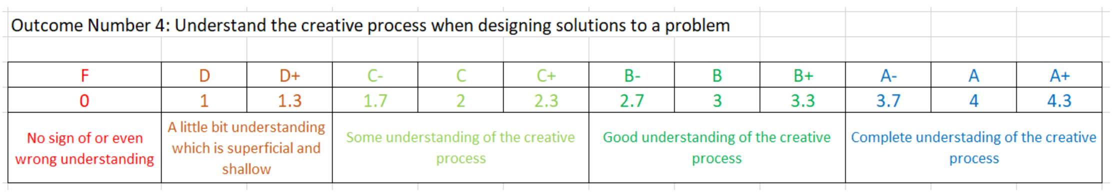
4. Understand the creative process when designing solutions to a problem

Rubrics for this laboratory



Outcome Number 3: Apply the knowledge to develop practical applications for modern real-time mobile computer systems.

F	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
0	1	1.3	1.7	2	2.3	2.7	3	3.3	3.7	4	4.3
Unable to apply the knowledge at all	Can only apply the knowledge in a limited manner		Can basically apply the knowledge			Good application of the knowledge			Excellent application of the knowledge		



Outcome Number 4: Understand the creative process when designing solutions to a problem

F	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
0	1	1.3	1.7	2	2.3	2.7	3	3.3	3.7	4	4.3
No sign of or even wrong understanding	A little bit understanding which is superficial and shallow		Some understanding of the creative process			Good understanding of the creative process			Complete understading of the creative process		

Appendix B

If the layout does not show correctly as Fig. B-1, follow the steps below.

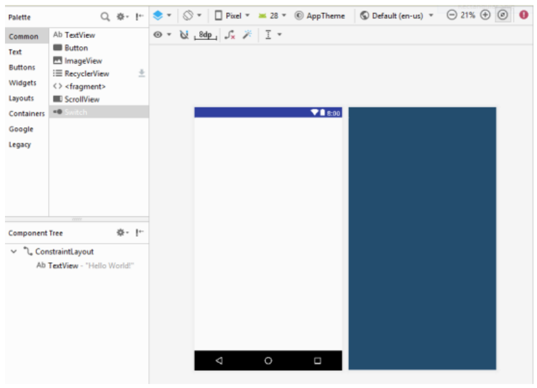
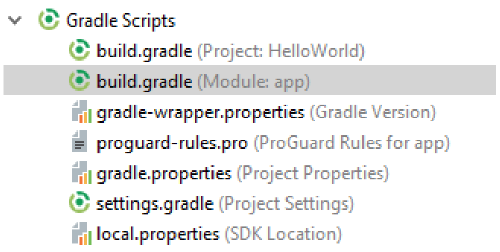


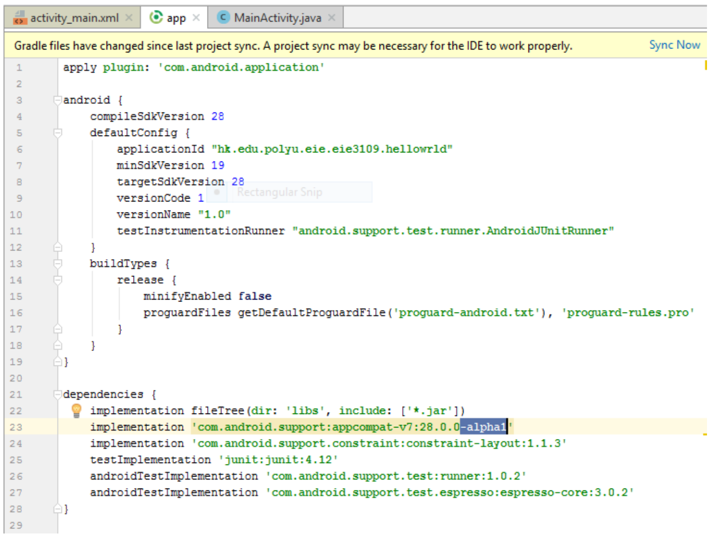
Fig. B-1

Open Module:app


Gradle Scripts
build.gradle (Project: HelloWorld)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Add "-alpha1" to the highlighted line

Then click "Sync Now" at the top right



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "hk.edu.polyu.eie.eie3109.hellowrld"
        minSdkVersion 19
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0-alpha1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```