# Android Basic Development Guide

## Building Student Record Keeper app.

Software Engineering – 1 (COMP-3350)

GITHUB: https://github.com/trsiddiqui/StudentRecordKeeper

Taha R. Siddiqui, Muhammad Moein Almasi
1/12/2016

Section 1

# Setting up Environment

**Fig 1. Start by Downloading Android Studio®**

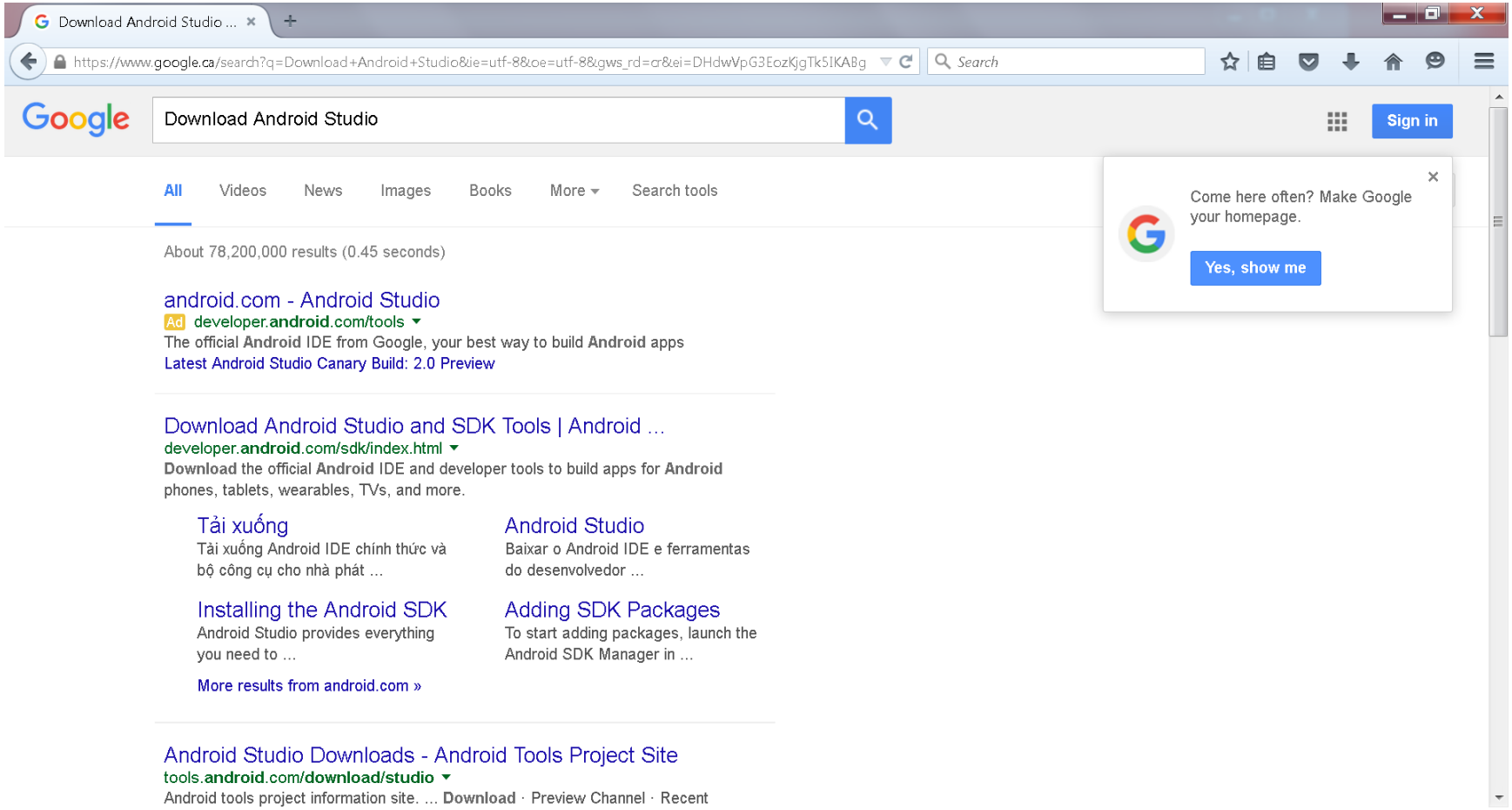


**Fig 2. Downloading Android Studio®**

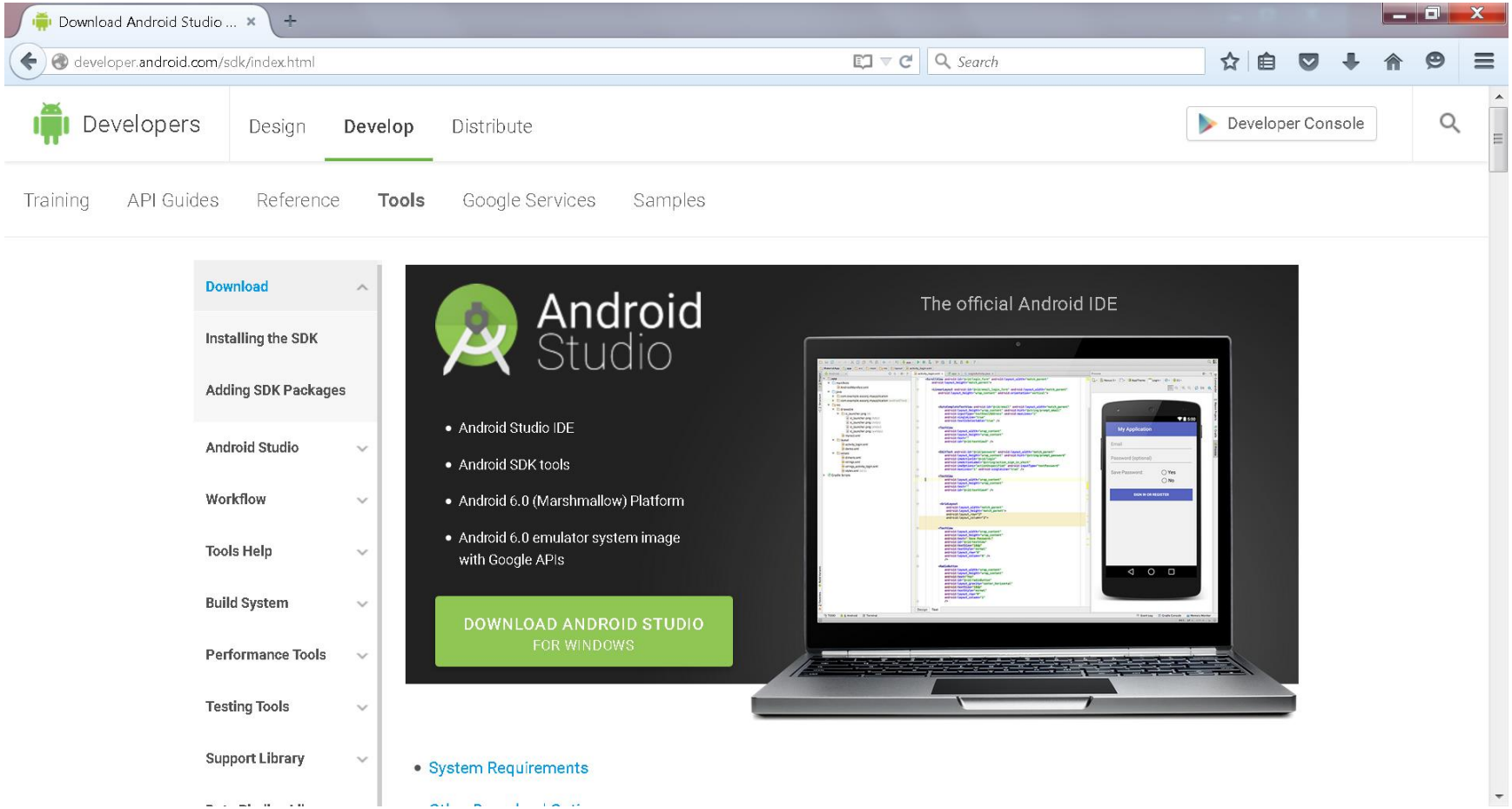


**Fig 3. Downloading Android Studio®**

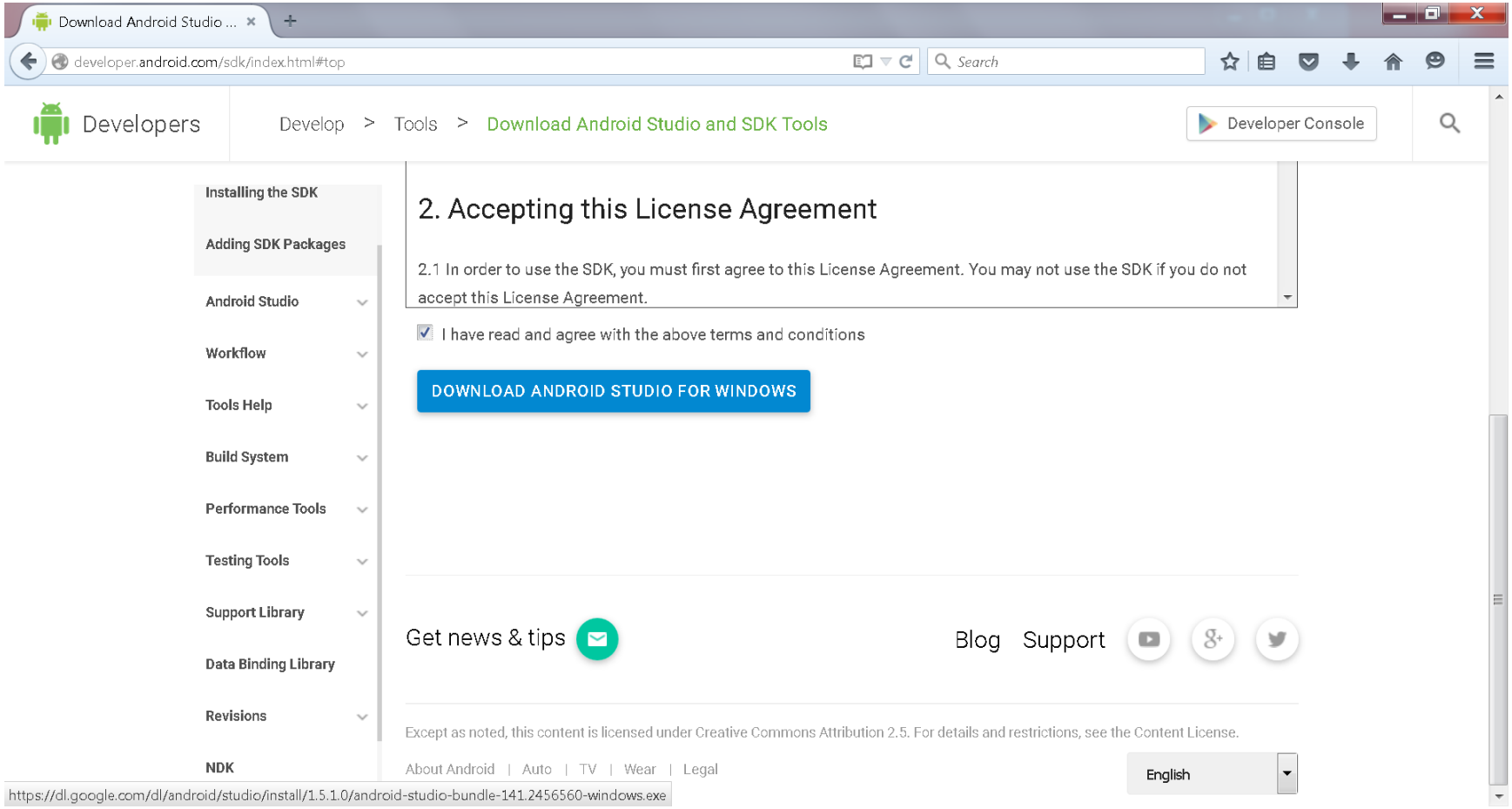


**Fig 4. Run Android Studio® setup as Administrator**

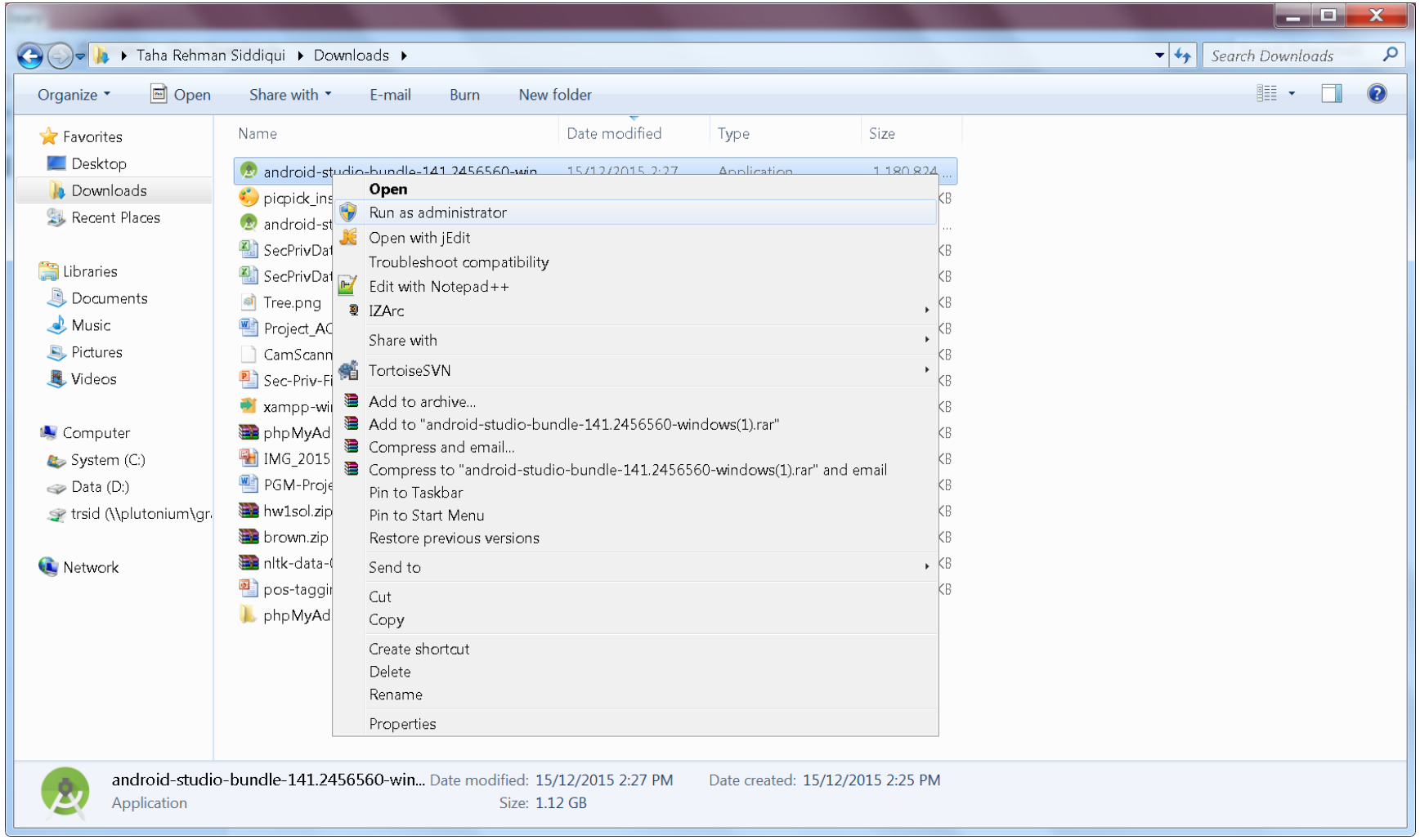


**Fig 5. Set up Android Studio® (Keep clicking Next)**

**Fig 6. Set up Android Studio® Hardware Accelerator (The allowed Custom range depends upon the memory in your system. Allocating one quarter of the total memory is recommended)**

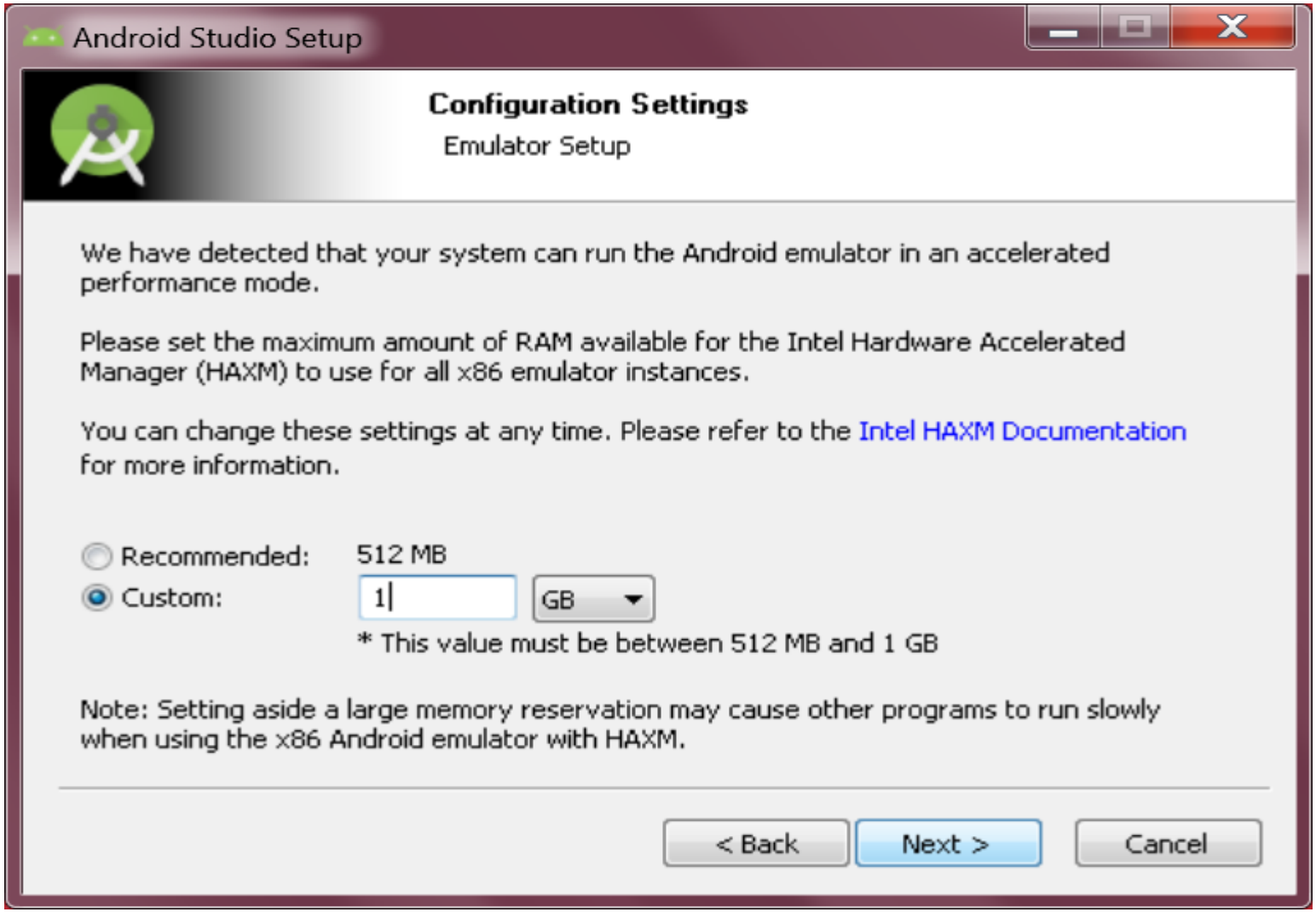


**Fig 7. Start Android Studio® (Keep clicking Next)**

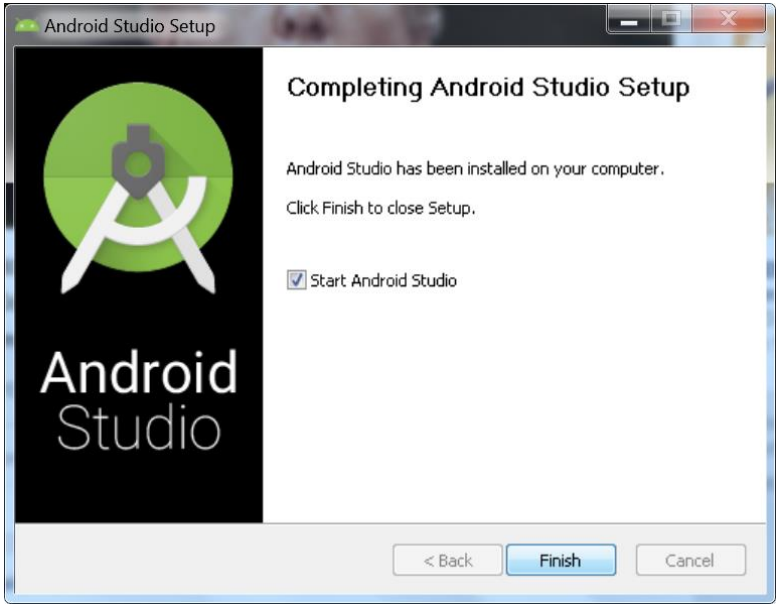



**Fig 8. Select Do not Import Settings**

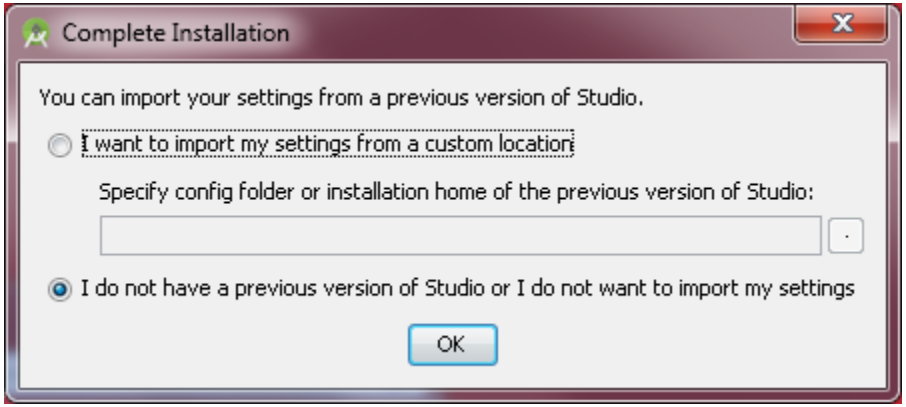



**Fig 9. Add exception to firewall (as it will attempt to download a lot of resources itself)**

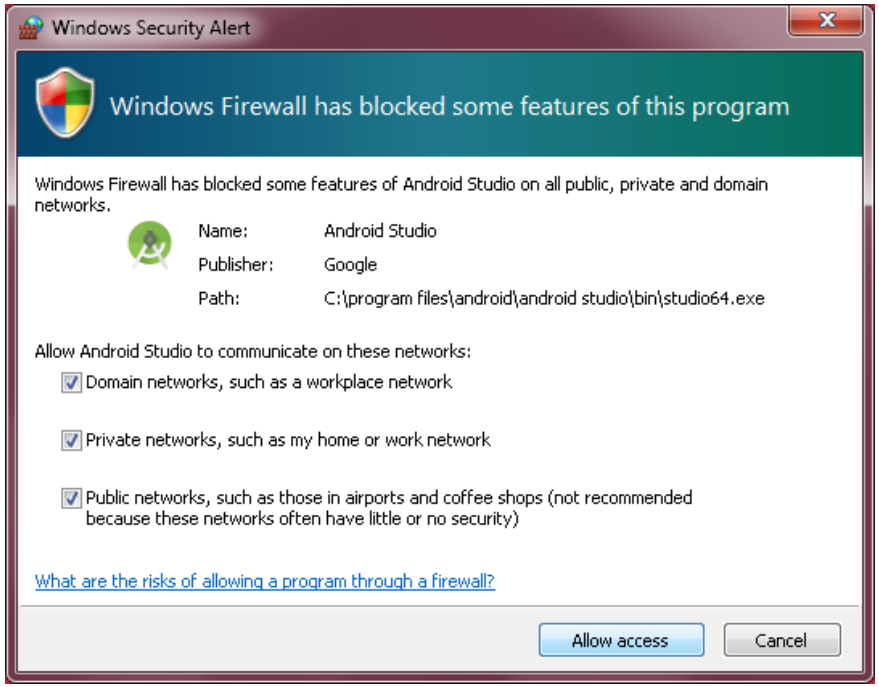


**Fig 10. Auto-Downloading latest SDK**

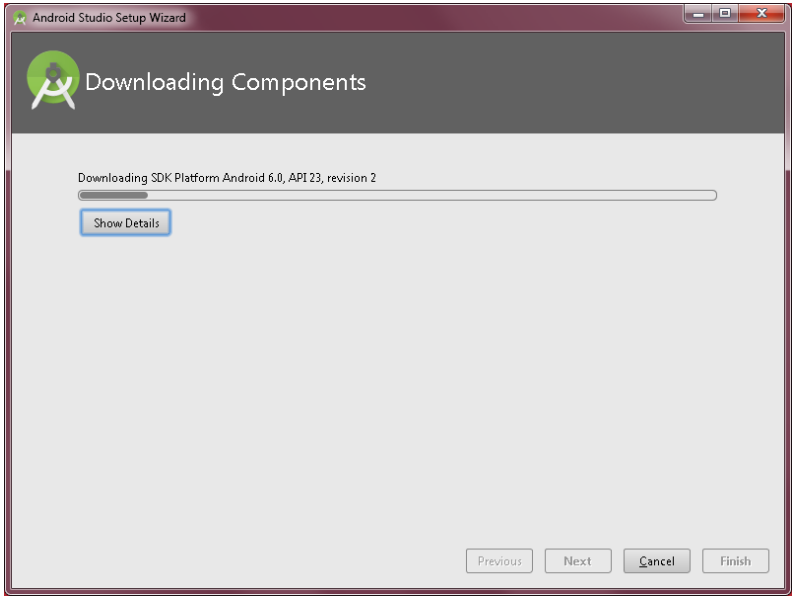



**Fig 9. You may have to retry it (happened to me both times I installed Android Studio ® on different machines)**

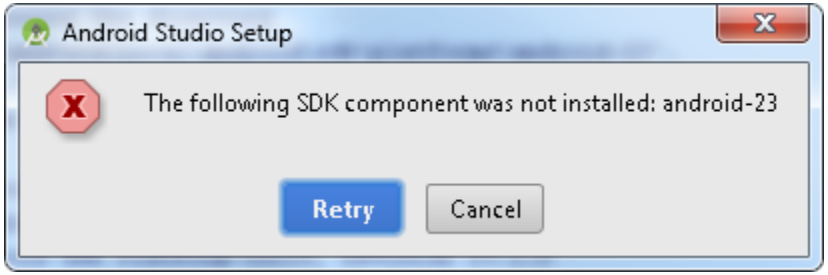


**Fig 10. Android Studio ® is ready**

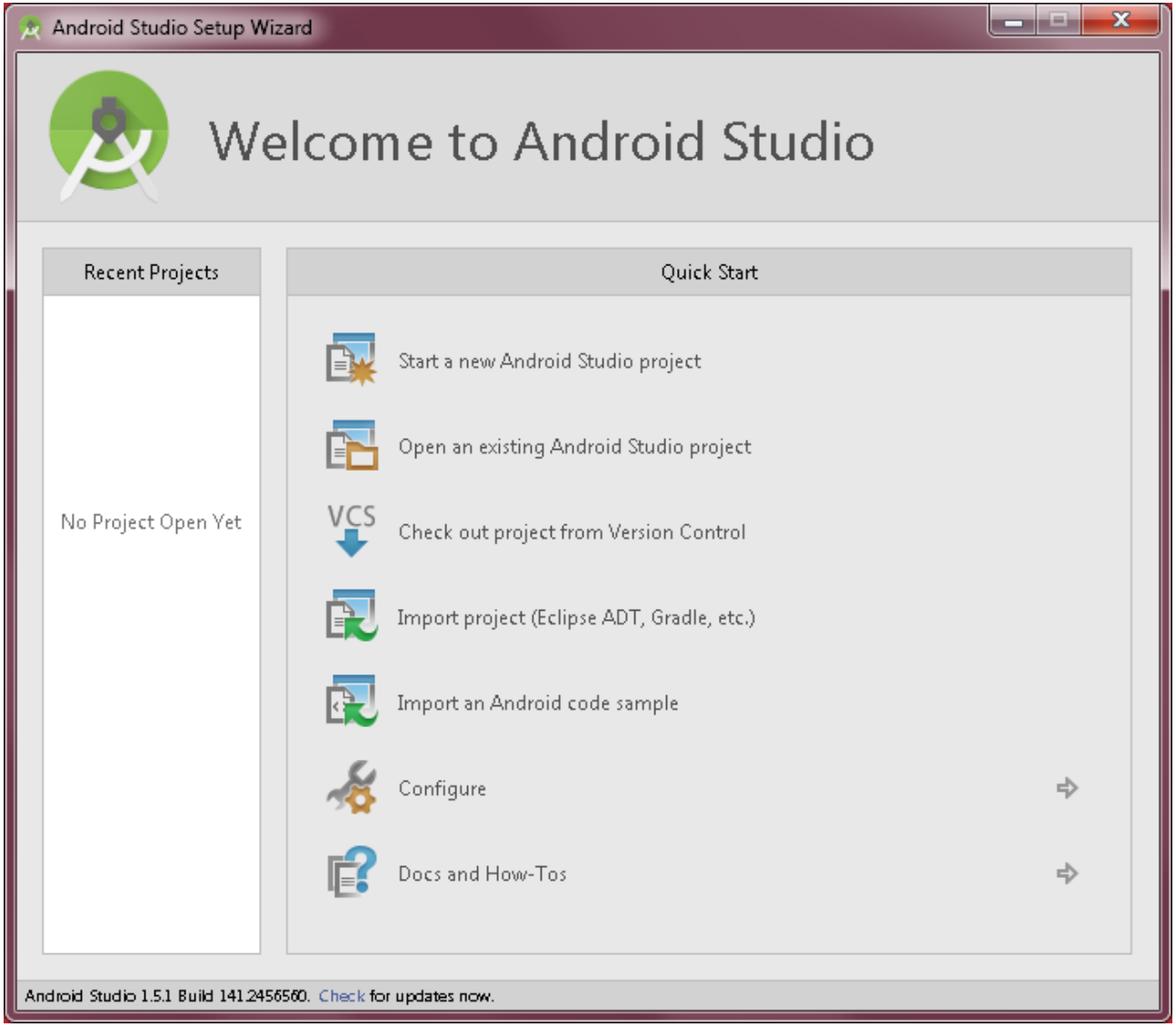
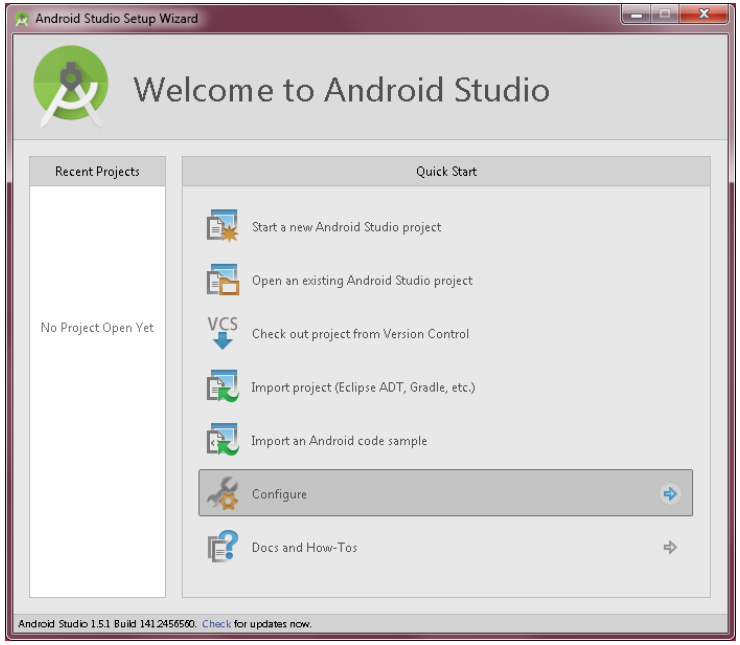
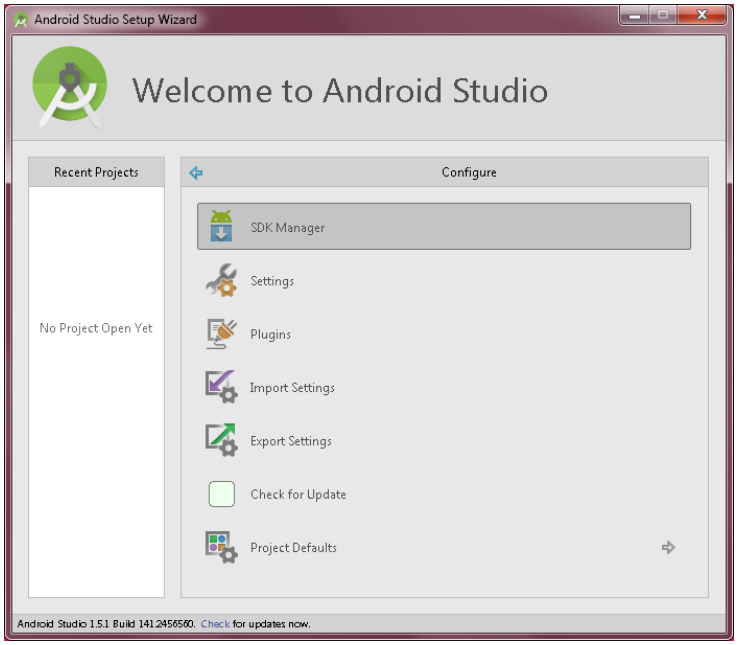
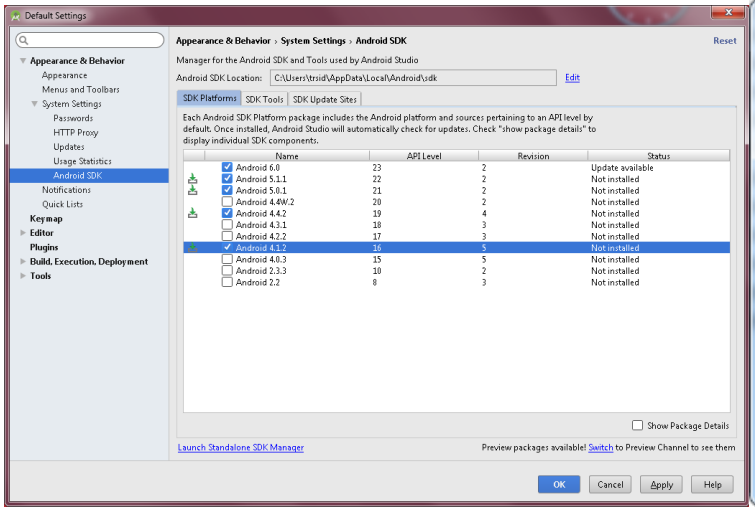


**Fig 10. Add old SDK's (Android version 4.1, 4.2, 5.0). By default there is only 6.0**



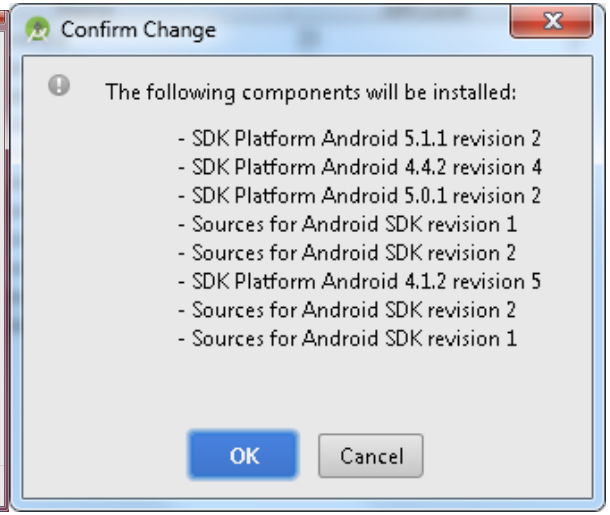
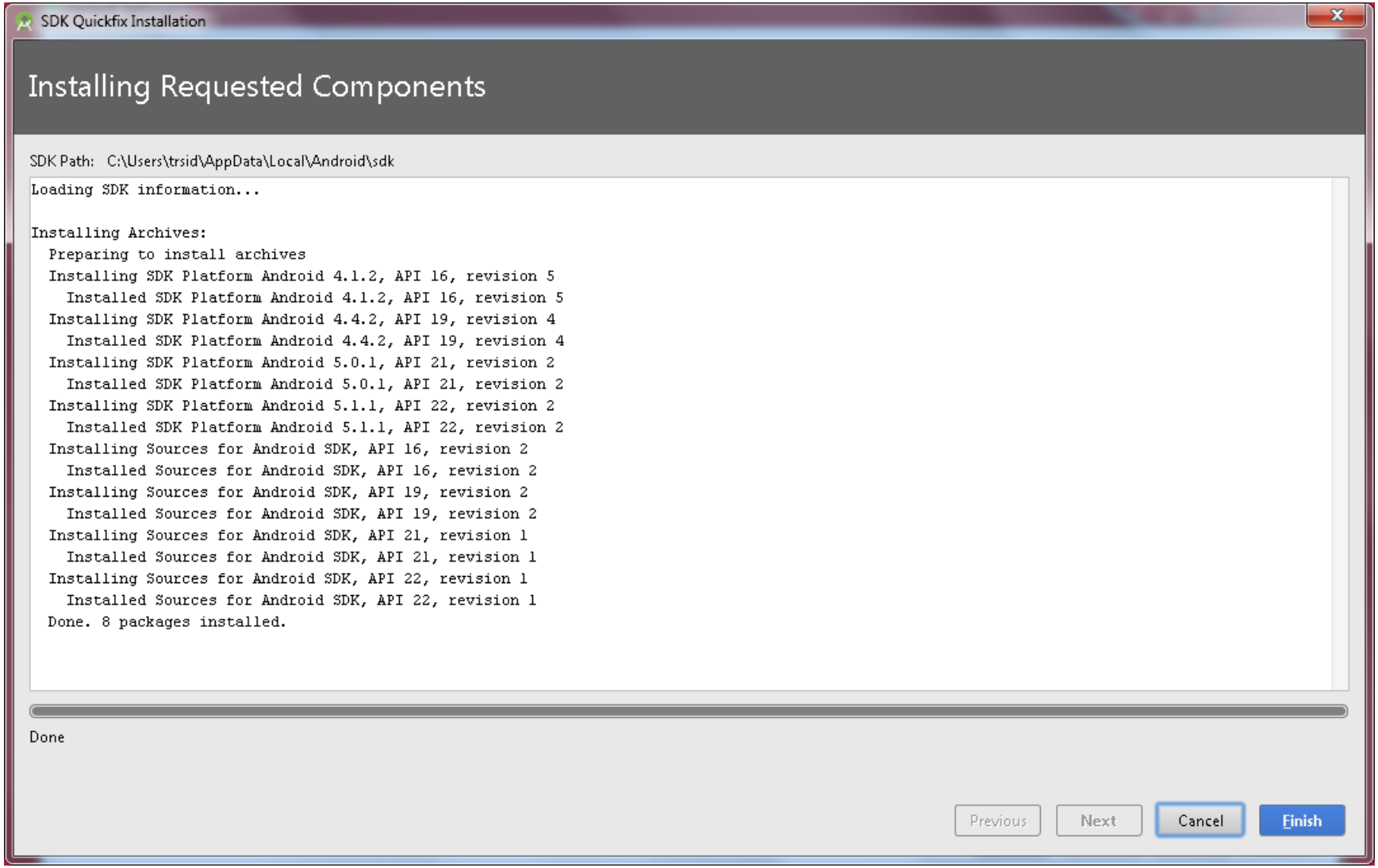



**Fig 12. Old SDK's installed**



# Section 2

# Starting Coding

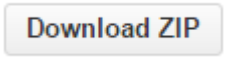
The source code written in this tutorial is available at https://github.com/trsiddiqui/StudentRecordKeeper**.** For now, you can download the source code by clicking on this button. 

**Fig 13. Start a new Project**

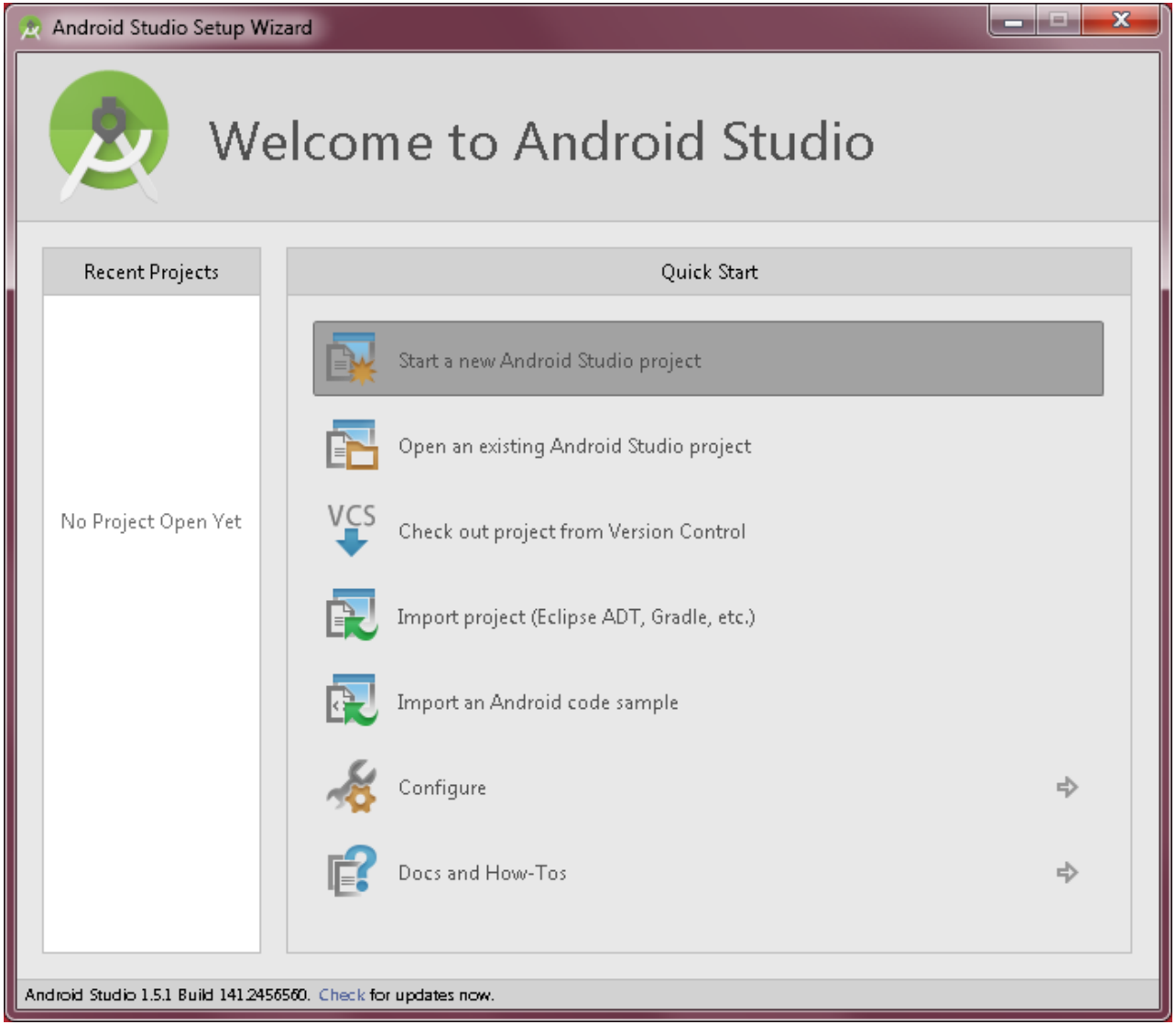


**Fig 14. Name your project**

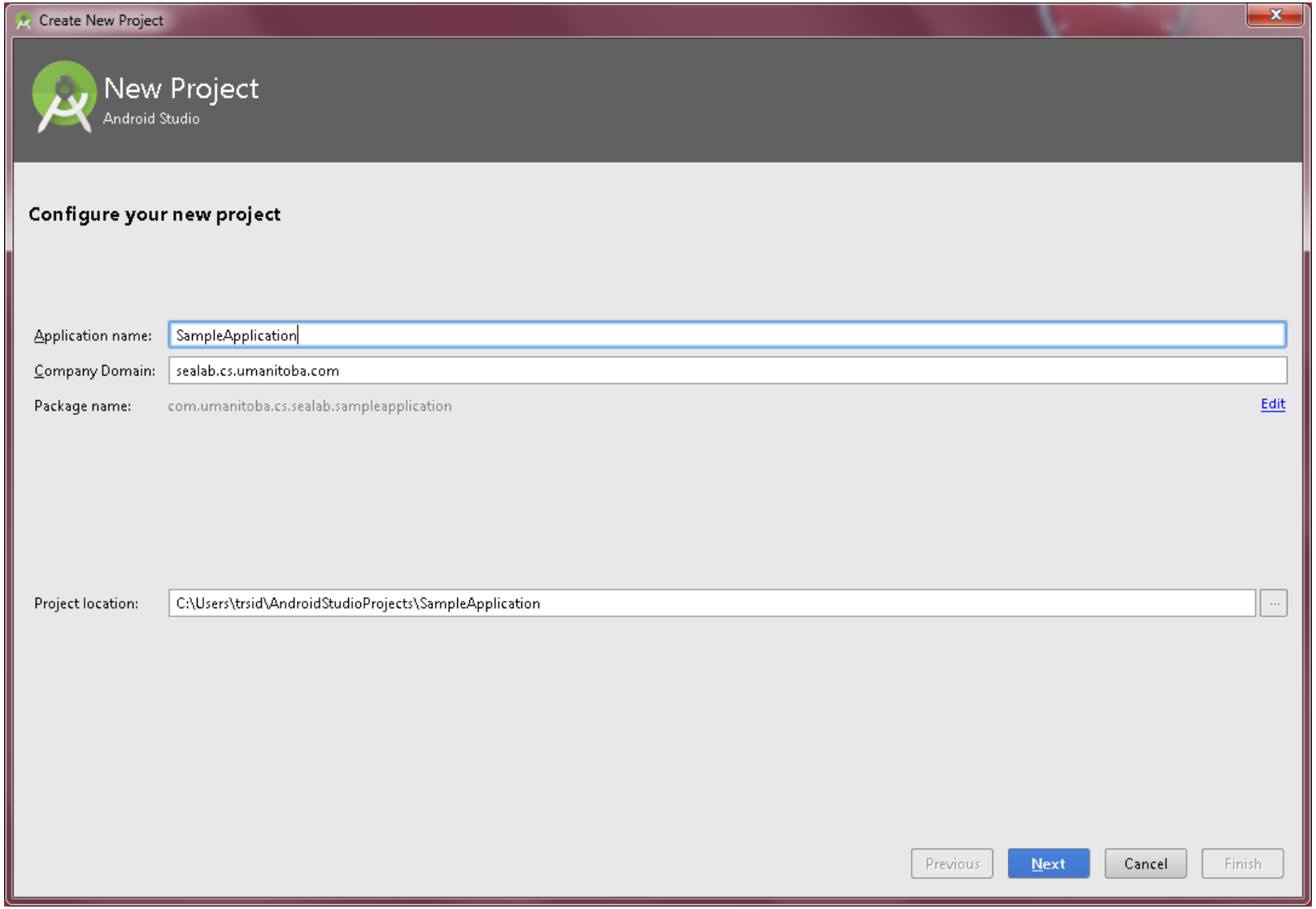


**Fig 14. Select the minimum supported version (SDK). This is the minimum version of android to be supported by your app.**

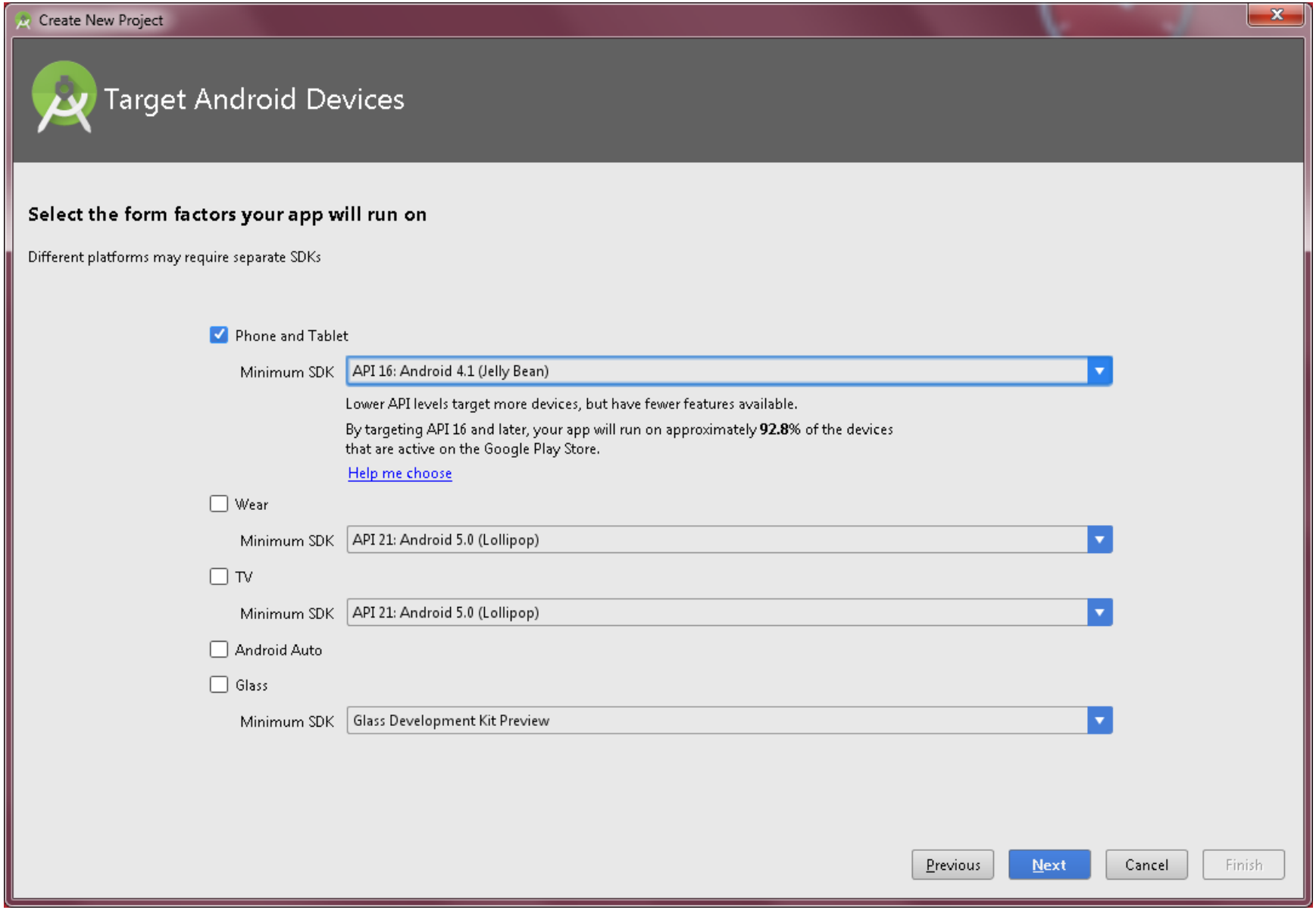
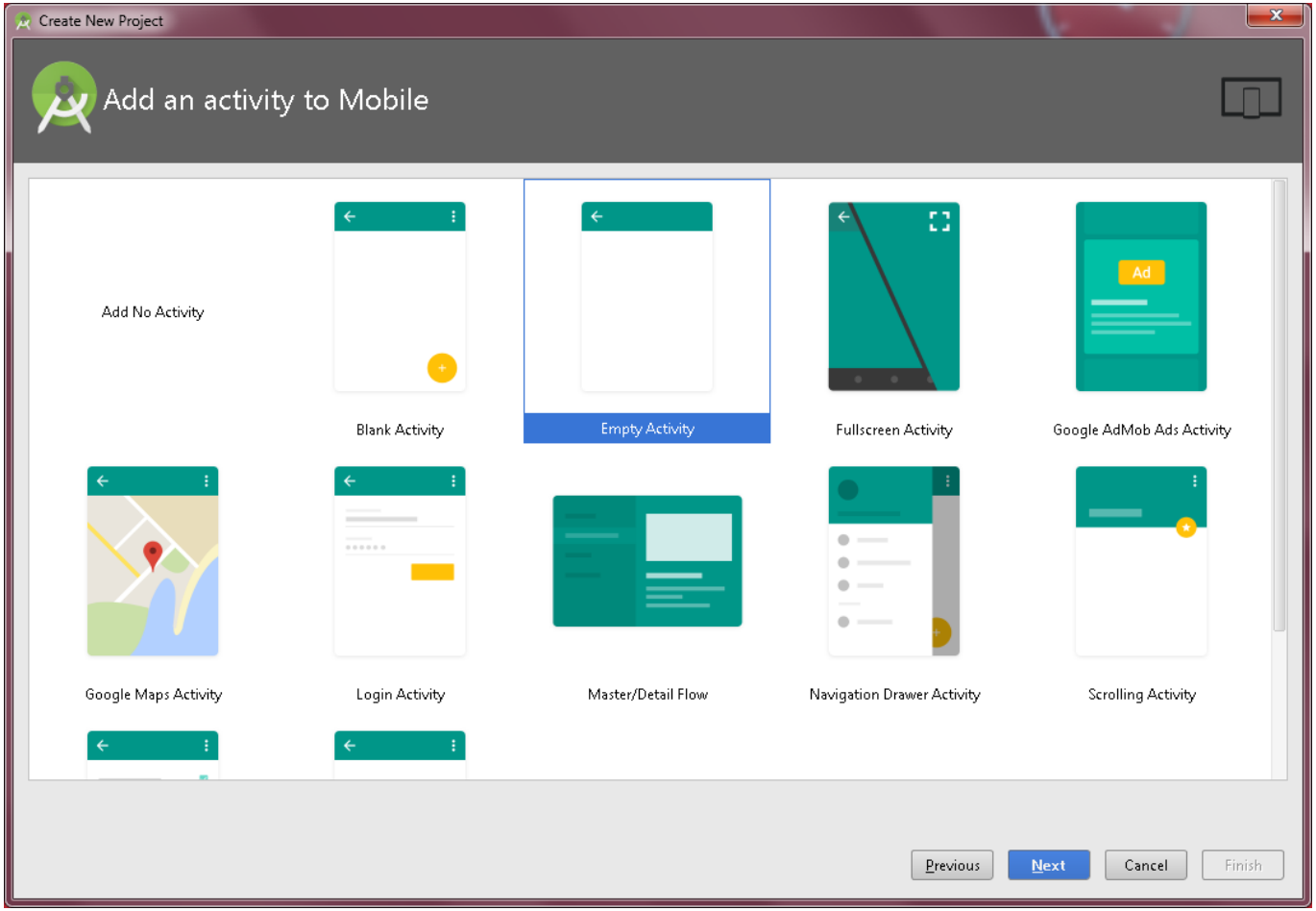


**Fig 15. Select the main activity (notes below the screenshot)**



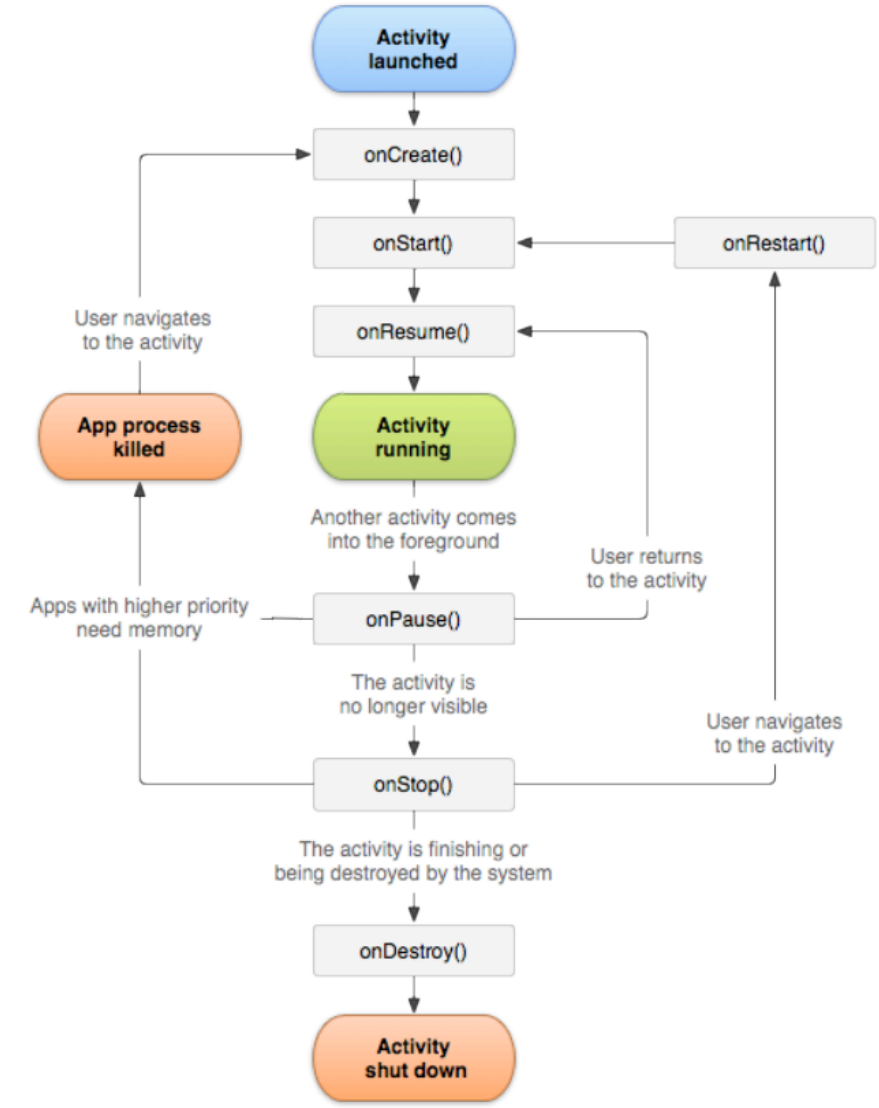
# Activity

An Activity is similar to a window on a regular desktop graphical interface, though it takes up the entire display. When your app starts, one Activity will be displayed. This activity needs to be configured as the Launcher (or main) activity. It acts as the starting point for your Android application, similar to the main() method in a desktop Java or C++ console application. You create a subclass of Activity and set that subclass as the Launcher activity. It will be the first thing that appears when your application starts.

Since we are developing a Student Record Keeper, this main activity should show the list of students in the system, with action (buttons) to perform other operations like Add, Update, Delete etc.

An Activity has methods that are called automatically when certain events occur. For example, onCreate() is called once when the activity is created for the first time. To execute your own code once when an activity is first created, you override this method (if it is not already overridden), call super.onCreate() at the start, and then add your own code. Code in the Launcher Activity's onCreate() is a lot like code in main(): it is executed exactly once at launch. For instance, when opening an application you might want to check if certain database already exists in the mobile system and if not then create. There are other events that occur in various situations, as defined by the Android Activity life cycle. The other events are used when your Activity enters certain states that are required on an Android device. Because it has less memory and storage than a desktop computer, Android will put your Activity into other states (paused or stopped) when other activities come to the front. A good Android app will free resources in these states so that the front most one has more available resources (such as memory or processing). This should be done on onPause(), which is called when another activity is called from this activity. For our tasks, as long as your app and all of its activities work on the tablet, you do not need to worry about anything else that is running, but when you are dealing with operations that consume more memory than normal you should free it before moving to another. The opposite of onCreate() is onDestroy(). Here you should free or close anything you created that needs to be explicitly closed, or you may want to delete any temporary files you might have created. ***A flow chart of Android Activity Life Cycle is shown on the next page***. This life cycle shows all the method that are called during certain events, such as calling another activity or closing the activity etc.

# Activity(cont.): Android Activity Life Cycle (1)



# Activity(cont.)

Important note: Activities are typically arranged in a stack: the user starts out with your Launcher Activity, then the user does something in the user interface that creates another activity, which may create another, and so on. When the user is done, they use the back button on the device to go back to the previous activity and so on.

Activities are created and launched through an Intent, which sends a message telling your app to do something; in this case, launch another activity. The system launches the activity for you. The new activity will be pushed onto the stack, and is created, started, and resumed, as shown in the life cycle diagram earlier. The activity that created the new one (one below the top of the stack) is paused and stopped, but not destroyed and when you comes back to it the method *onResume* is called. The new activity (top of the stack) can be quit explicitly with a call to the finish() method, or using the back button(automatically handled). When finished, the top activity will be paused, stopped, destroyed, and popped off the stack. The activity now at the top of the stack (the one the user went back to) will be restarted, started, and resumed. It can then create another new activity, and so on, until it is finished.

Below is the code for moving to a new activity.

```
Intent intent = new Intent(<thisActivity,eg. MainActivity>.this,<activityToDirectTo, eg StudentDetail>.class);
intent.putExtra("student_Id", 0);
startActivity(intent);
```

Notice the second line where data is placed in intent object itself so that the activity to be opened receives the data with the command of opening. This data can be retrieved by the receiving activity in the *onCreate* method as below.

```
@Override protected void onCreate(Bundle savedInstanceState) {
      Intent intent = getIntent();
      int _Student_Id =intent.getIntExtra("student_Id", 0);
}
```

You can also animate the transition from one activity to another after calling *startActivity* or in the *finish()* method.

```
overridePendingTransition(R.anim.abc_fade_in, 100000);
```

**Fig 16. Let's create an activity, name MainActivity for now**

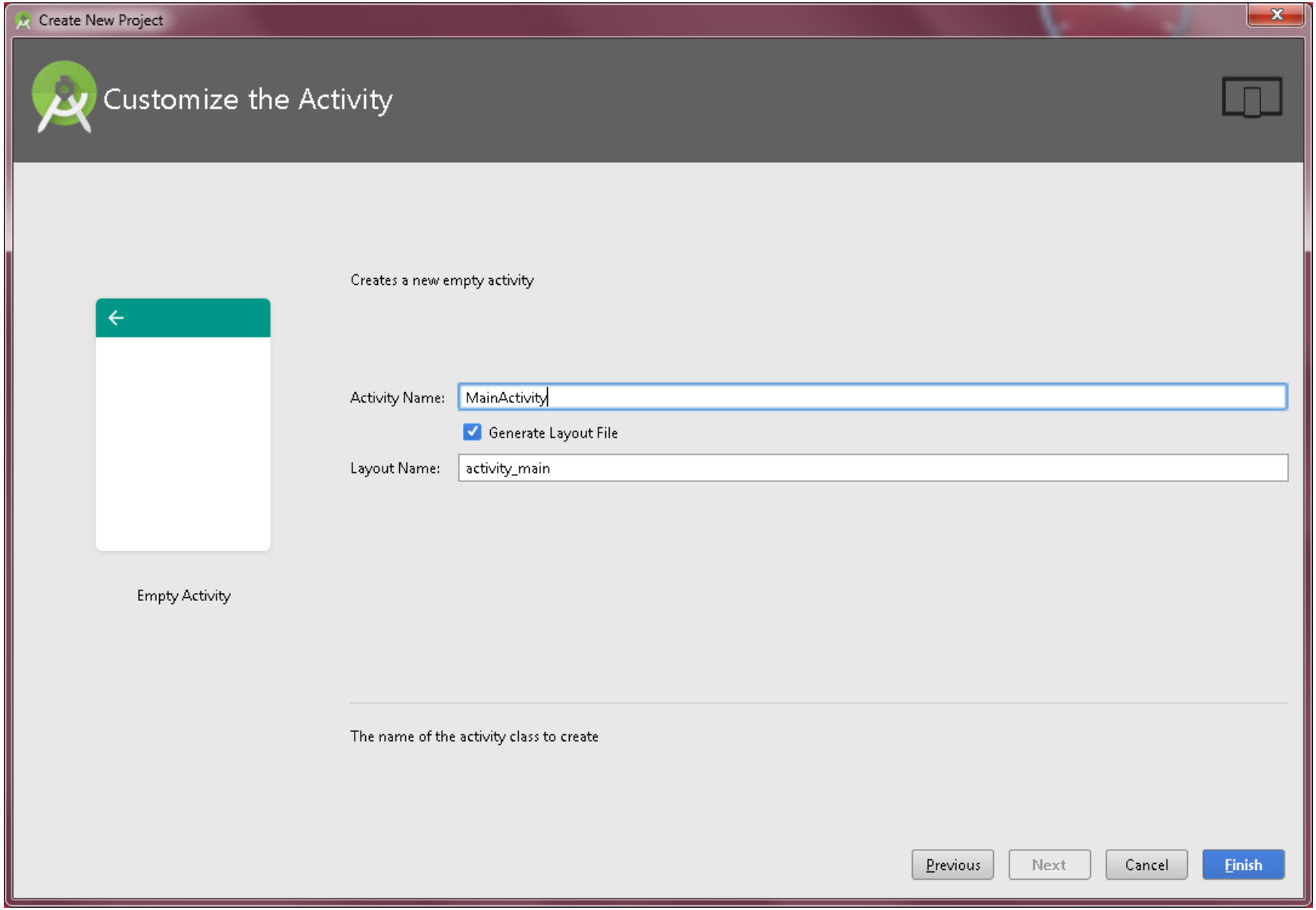
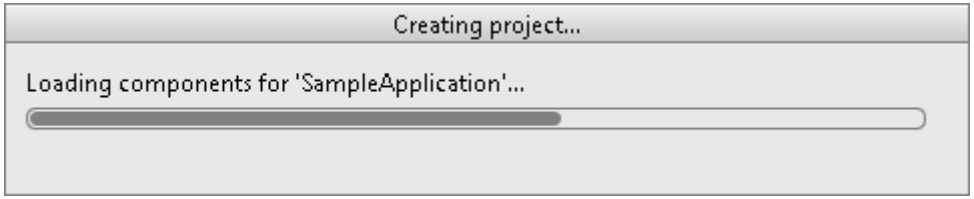


**Fig 17. Project is Loading**



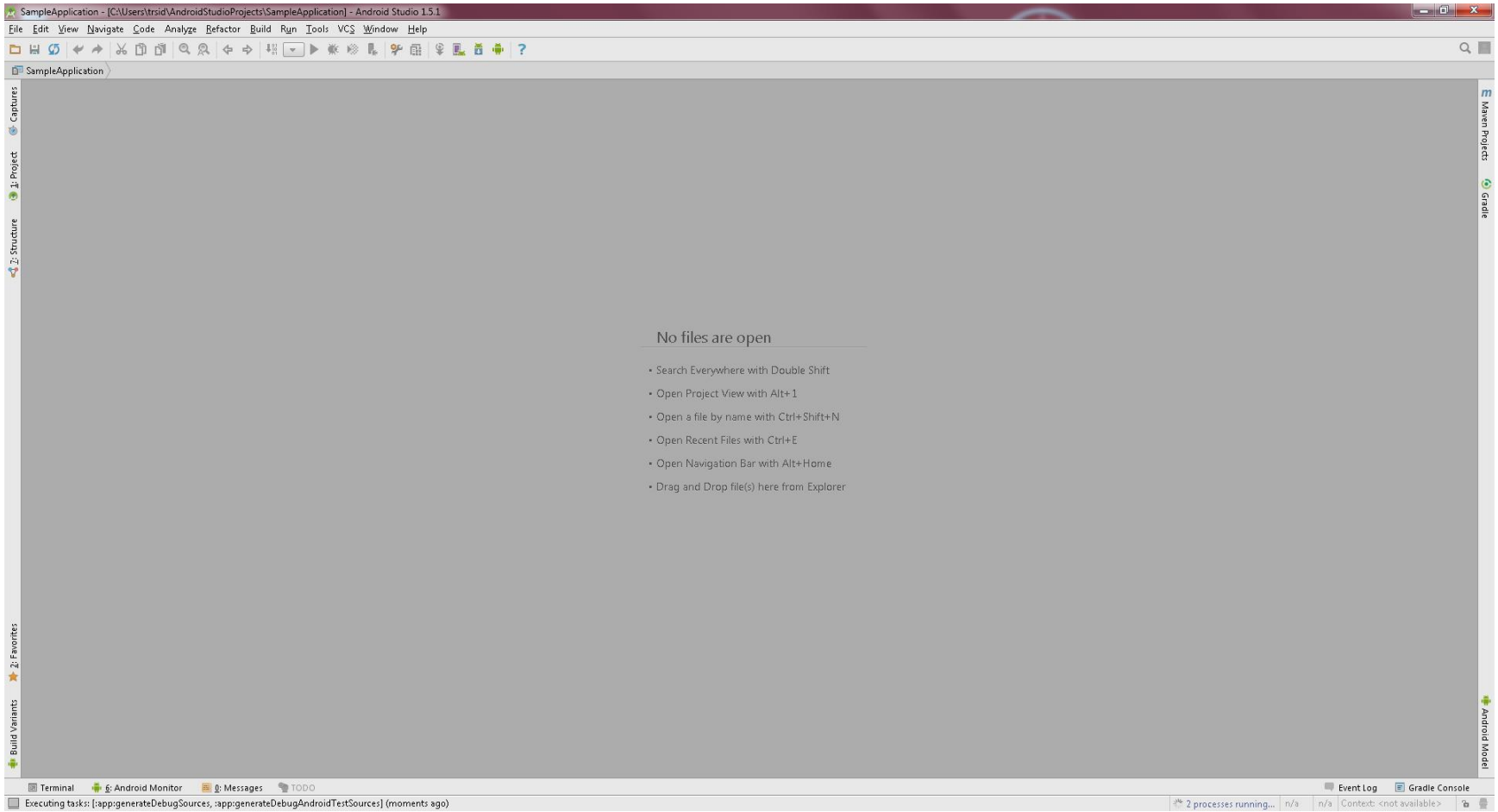
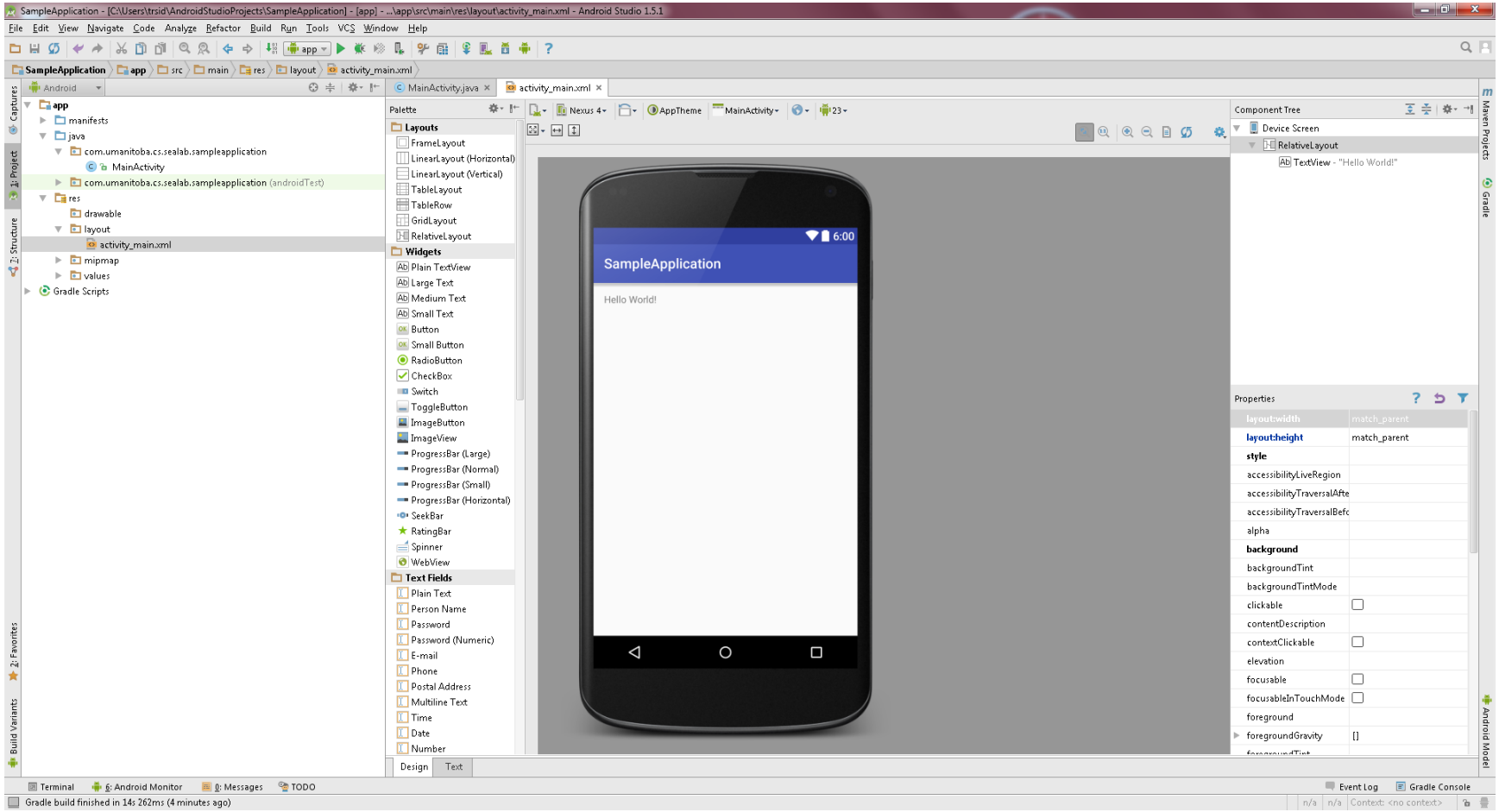



**Fig 18. Project Ready. Notice the file structure on the left. Your java code is going to be in java folder, and their corresponding views in layout folder. This project is ready to run as an application (on a mobile or emulator), although the only thing it shows yet is the text Hello World.**



# Views

A view (layout) is a file that describes the widgets in an Activity, and their relative positions. You can find layout files in your project under res → layout. They are XML files, where XML is a language used to describe structured data. If you open a layout file, you can view and edit it in its original XML (text) format, or through the graphical Preview cum editor.

Laying out your user interface in the preview editor gives you a good quick start. Sometimes, you may want to use the text format to easily copy and paste widgets, such as to bring examples in from a web site. Once you get comfortable with the text view, it's very likely that you will stick to text view. You can also switch between the two views by clicking on the tabs (Design, Text) at the bottom of the window pane.

# Menu

Android apps also have a menu accessible through the three dots located in the ActionBar (title bar) at the top of each Activity. In this example application, the menus are empty (hence not shown), but you can add items like Settings, About, or other actions relevant to your app. Each Activity can have its own menu, or you can reuse them. The menu options are found in res > menu XML files, and the code for handling them is in onCreateOptionsMenu and onOptionsItemSelected methods in the Activity. Don't forget to enable the HasOptionsMenu flag in onCreate method of the Activity. For more information visit [3].

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/sample_action"
          android:showAsAction="ifRoom|withText"
          android:title="Do Something" />
</menu>
```

Note: Menu can only be contained in Fragments. A Fragment is a type of widget that contains nested layouts and other widgets, but it is not used on its own. Instead, a Fragment is a reusable group of widgets, that you can place into different activities, to avoid copy and paste. For example, the student editing area (Name, Age, Email address) that appears in the StudentsActivity is only used in one Activity, but if it needed to be re-used in another Activity (perhaps a separate screen to edit detailed student information), then the widgets and associated code could be moved into a Fragment and a corresponding class. This Fragment would then be placed in both activities, and instantiated when the Activity is created. Find further about Fragment at [4].

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ……
    setHasOptionsMenu(true);
    ……
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.sample_action) {
        //Do Some Action
    }
    return true;
}
```

**Fig 19. Edit the activity_main.xml under layout, this is the view associated with the action MainActivity, and is the main page where we are going to show the List of students. Remove the Hello World TextView. TextView is a control used to show text like Label.**

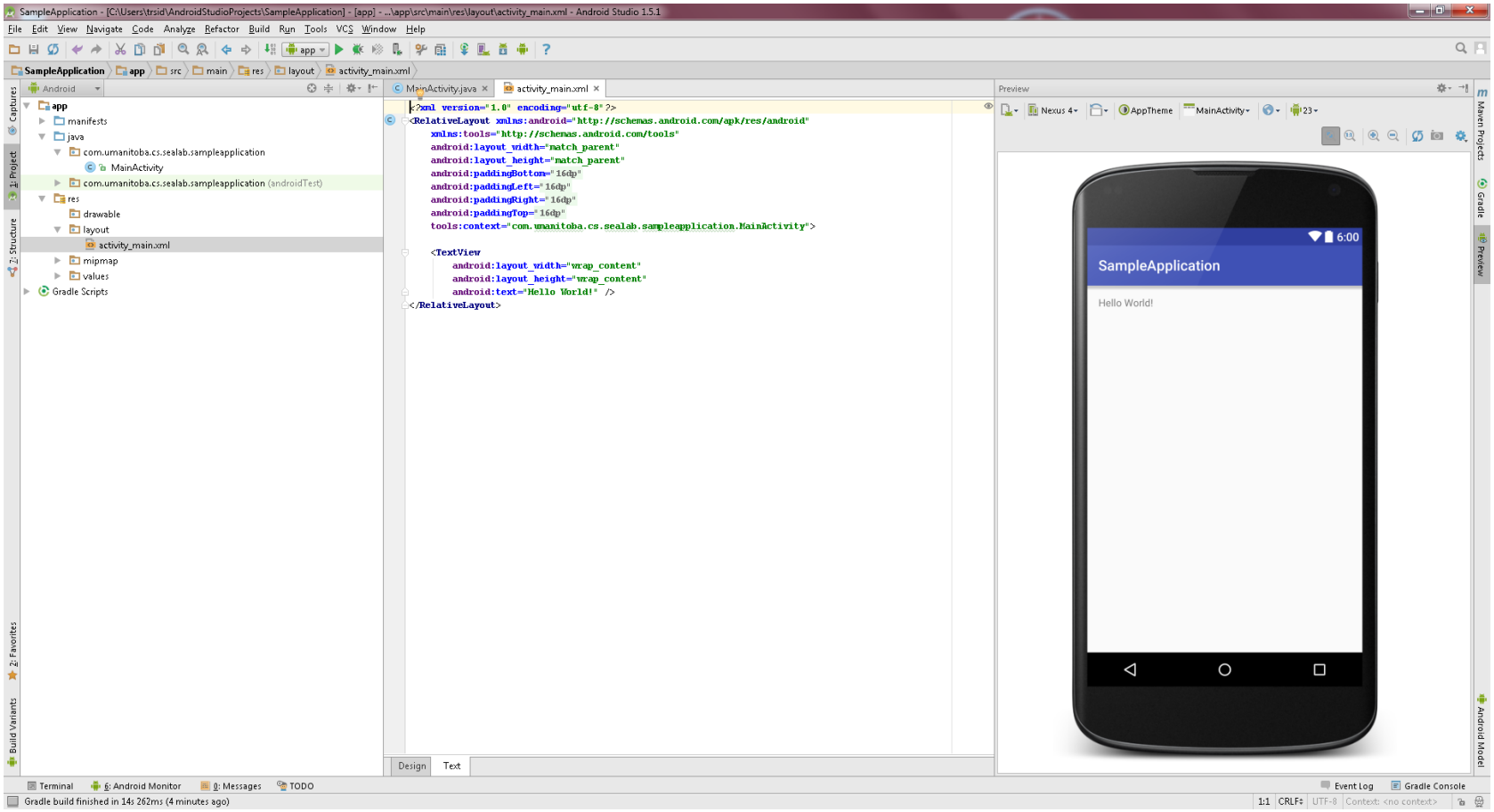
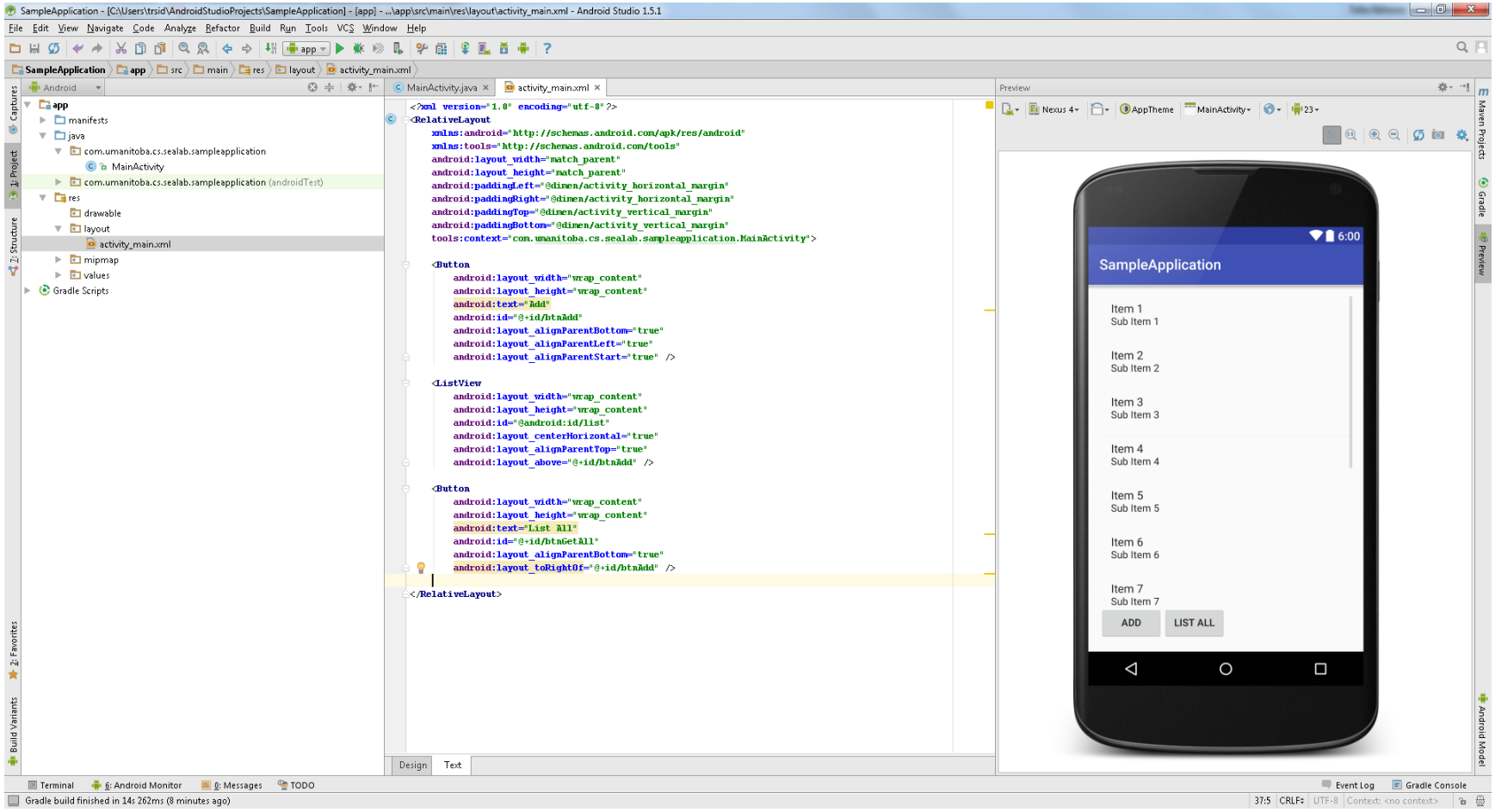


**Fig 20. Add controls to your view**



**Code 1. activity_main.xml (All the code files are also directly attached in the end of this tutorial)**

```
<RelativeLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:paddingBottom="@dimen/activity_vertical_marg
    android:paddingLeft="@dimen/activity_horizontal_marg
    android:paddingRight="@dimen/activity_horizontal_mar
    android:paddingTop="@dimen/activity_vertical_margin"
    android:label="Student Records Application"
    tools:context="com.umanitoba.cs.sealab.sampleapplica

<TextView
    android:id="@+id/student_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="3dip"
    android:paddingTop="3dip"
    android:textSize="36sp"
    android:textStyle="bold"
    android:text="Students"
    android:layout_alignTop="@android:id/list"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
<!--<Button-->
    <!--android:layout_width="wrap_content"-->
    <!--android:layout_height="wrap_content"-->
    <!--android:text="Add"-->
    <!--android:id="@+id/btnAdd"-->
    <!--android:layout_alignParentBottom="true"-->
    <!--android:layout_alignParentLeft="true"-->
    <!--android:layout_alignParentStart="true" />-->
<android.support.design.widget.FloatingActionButton
    android:id="@+id/btnAdd"
    android:background="@android:drawable/screen_background_dark"
    android:layout_width="56dp"
    android:layout_height="56dp"
    android:scaleType="fitXY"
    android:layout_gravity="bottom|end"
    android:src="@android:drawable/ic_menu_add"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
```

Notice that the text property of Button is ***@string/listAll***. This is another way of specifying texts where you maintain a list of strings in a separate file and use their indexes. From the left panel, open the file res → values → strings.xml. You will find the below text.

```
<resources>
    <string name="app_name">SampleApplication</string>
</resources>
```

Now add another string here as below

```
    <string name="listAll">List All</string>
```

This becomes very useful when you are developing multilingual applications (If you manually add text to each widget, you will have to change the text of all the widgets manually if you need to change the language of the app). String resources are automatically generatable too.

```
    android:layout_alignParentEnd="true" />

<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@android:id/list"
    android:paddingLeft="6dip"
    android:paddingTop="50dip"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/btnShowFileAction" />

<!--<Button-->
    <!--android:layout_width="wrap_content"-->
    <!--android:layout_height="wrap_content"-->
    <!--android:text="List All"-->
    <!--android:id="@+id/btnGetAll"-->
    <!--android:layout_alignParentBottom="true"-->
    <!--android:layout_toRightOf="@+id/btnAdd" />-->

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="File Read/Write Activity Button"
    android:id="@+id/btnShowFileAction"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
</RelativeLayout>
```

Notice on the top, the layout file you open contains a particular kind of layout: ***RelativeLayout*** in this case. There are other layouts, such as grid and Linear. Linear layouts can be horizontal or vertical, and arrange widgets in a single row or column. Grid layouts allow you specify a number of both rows and columns. Relative layouts place widgets adjacent to each other (e.g. left-of and below; *layout_toRightOf*), or relative to the layout itself (e.g. right and bottom; *layout_alignParentBottom*). Because layouts are also widgets, you can nest them to build your interface.[5]

All widgets have properties that determine how they appear and behave. The *id* gives each widget a unique identifier, used later in code. Layout Parameters determine how the widget will appear in its containing layout, such as *layout_alignBottom* in a relative layout, or row or column spans in a grid layout. There are also properties specific to each widget type. For buttons, one very convenient property is OnClick, which lets you name a method that will be called when the button is clicked. In your code, the method must be located in the class corresponding to the

Activity, and it must accept a View as a parameter. If you use this property to specify a method for click action then that method will be called. If not, then for all the buttons in the view should be one onClick method in the corresponding Activity class. Eg. **public void** onClick(View view). Since it is taking View as an input, you can then differentiate between the widgets using the *id* property of the view received in the method.

You can also add *listeners* (events) to your widgets in the corresponding action class. Look at the examples below.

Example 1.

```
Button btnSave =(Button)findViewById(R.id.btnSave);
btnSave.setOnClickListener(new Button.OnClickListener(){
    public void onClick(View v)
    {
        //Do Something
    }
});
```

In the 1st line we got hold of the Button with id btnSave. On the second line we set a listener Button.OnClickListener to the button so that when the button is clicked, this listener will get the control and pass it to the method defined in it.

Example 2.

```
EditText editStudentEmail =(EditText)findViewById(R.id.editTextEmail);
Button btnSave =(Button)findViewById(R.id. btnSave);
editStudentEmail.addTextChangedListener(new TextWatcher(){
    public void beforeTextChanged(CharSequence s, int start, int count, int after){}
    public void onTextChanged(){CharSequence s, int start, int before, int count}
    public void afterTextChanged(Editable s){
        btnSave.setEnabled(IsEmailValid(editStudentEmail.getText().toString()));
    }
});
```

Again, in the 1st and 2nd line we got hold of the editText(box) with id editTextName (this is the textbox you will be typing email in your application). In the 3rd line, we added a TextChangedListener for the textbox which will get control whenever Text is changed inside the textbox. Here the listener is an *anonymous* object that implements the TextWatcher interface (anonymous because instead of declaring a TextWatcher object and using it inside the addTextChangedListener method, we directly initialized an anonymous object with the keyword new and no name). The TextWatcher class implements 3 methods as defined above (beforeTextChanged, onTextChanged & afterTextChanged) and function

as to what their name suggests. *afterTextChanged* is called when the text has been changed inside a textbox, hence after user has typed in the email address in the textbox, we will verify if it is a valid email address and enable the save button if it is.

Note: Anonymous objects are only suggested only when you are not supposed to reuse them.

**Fig 21. Add a new activity for inserting the data of new students.**

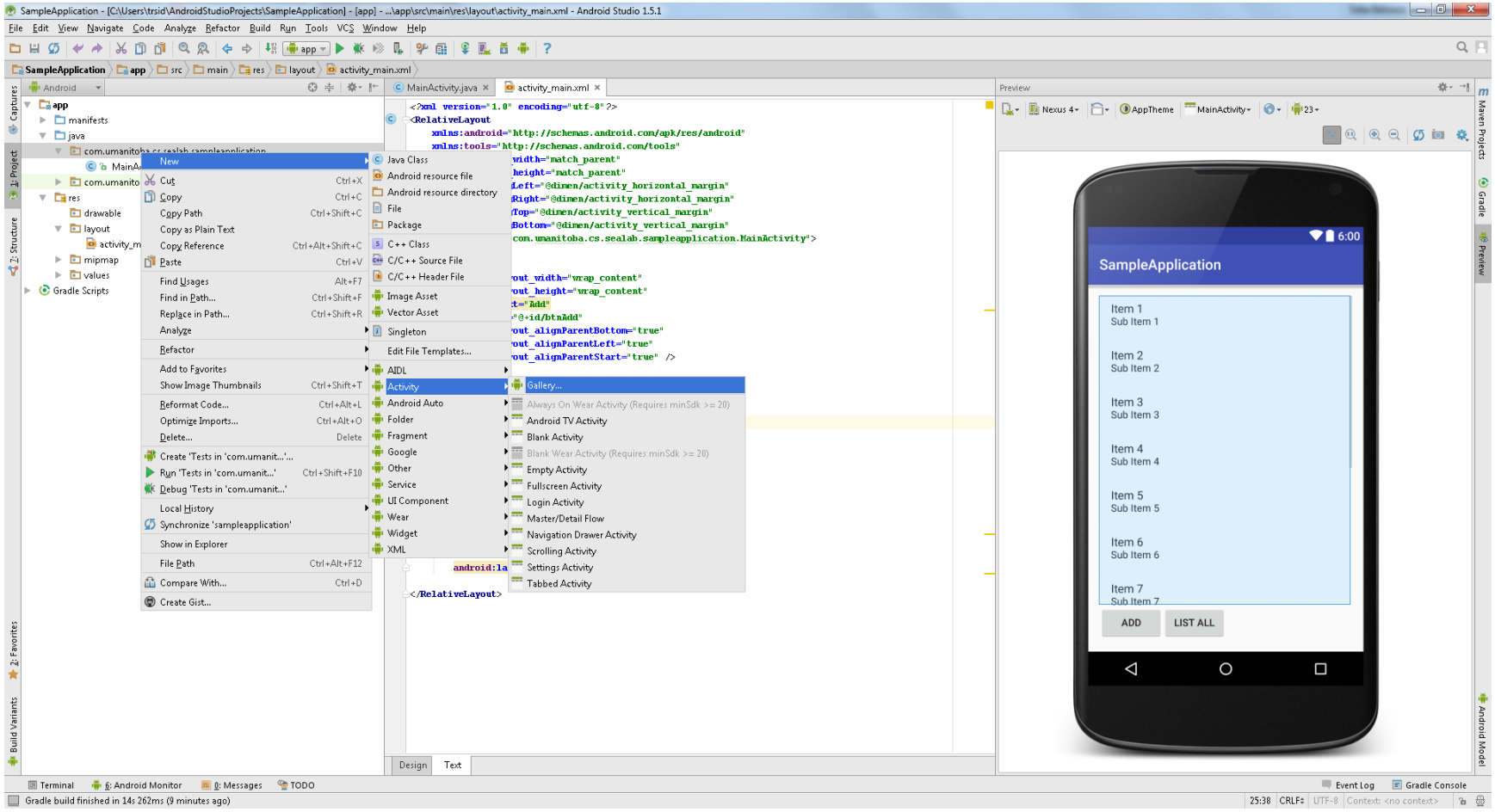


**Fig 22. Add a new Empty activity for showing the detail of each list item (cont.).**

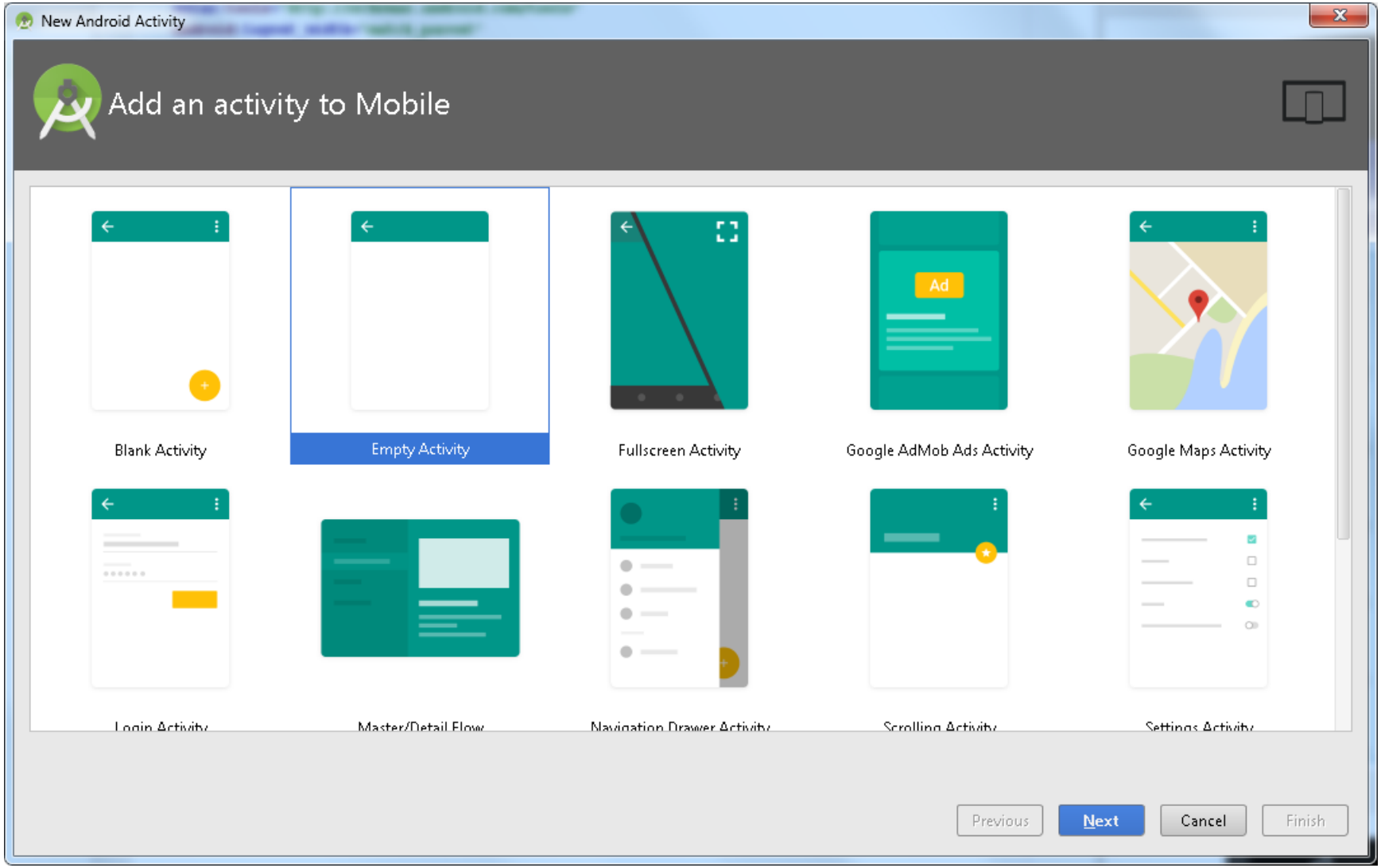


**Fig 23. Name the new activity StudentDetail.**

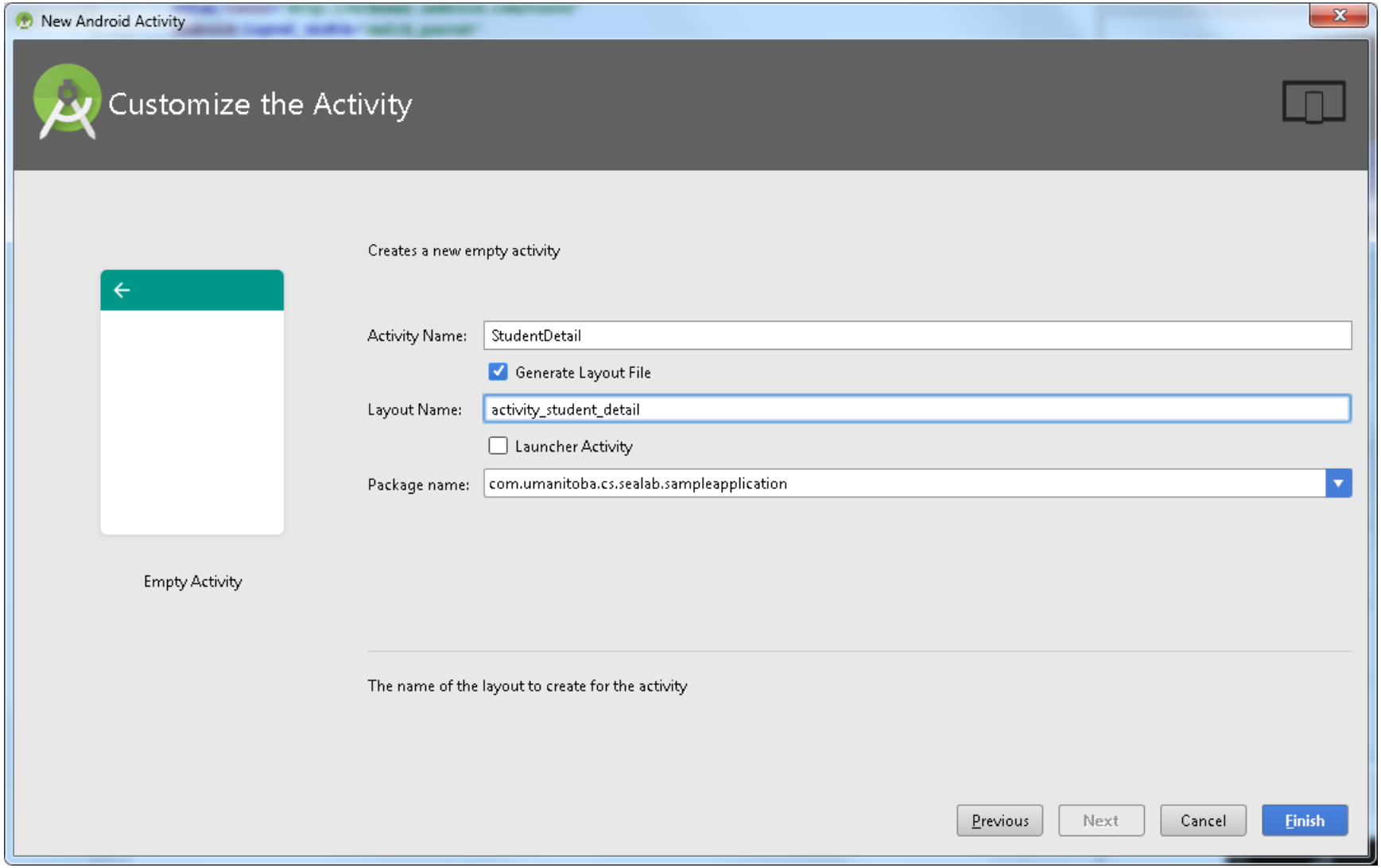
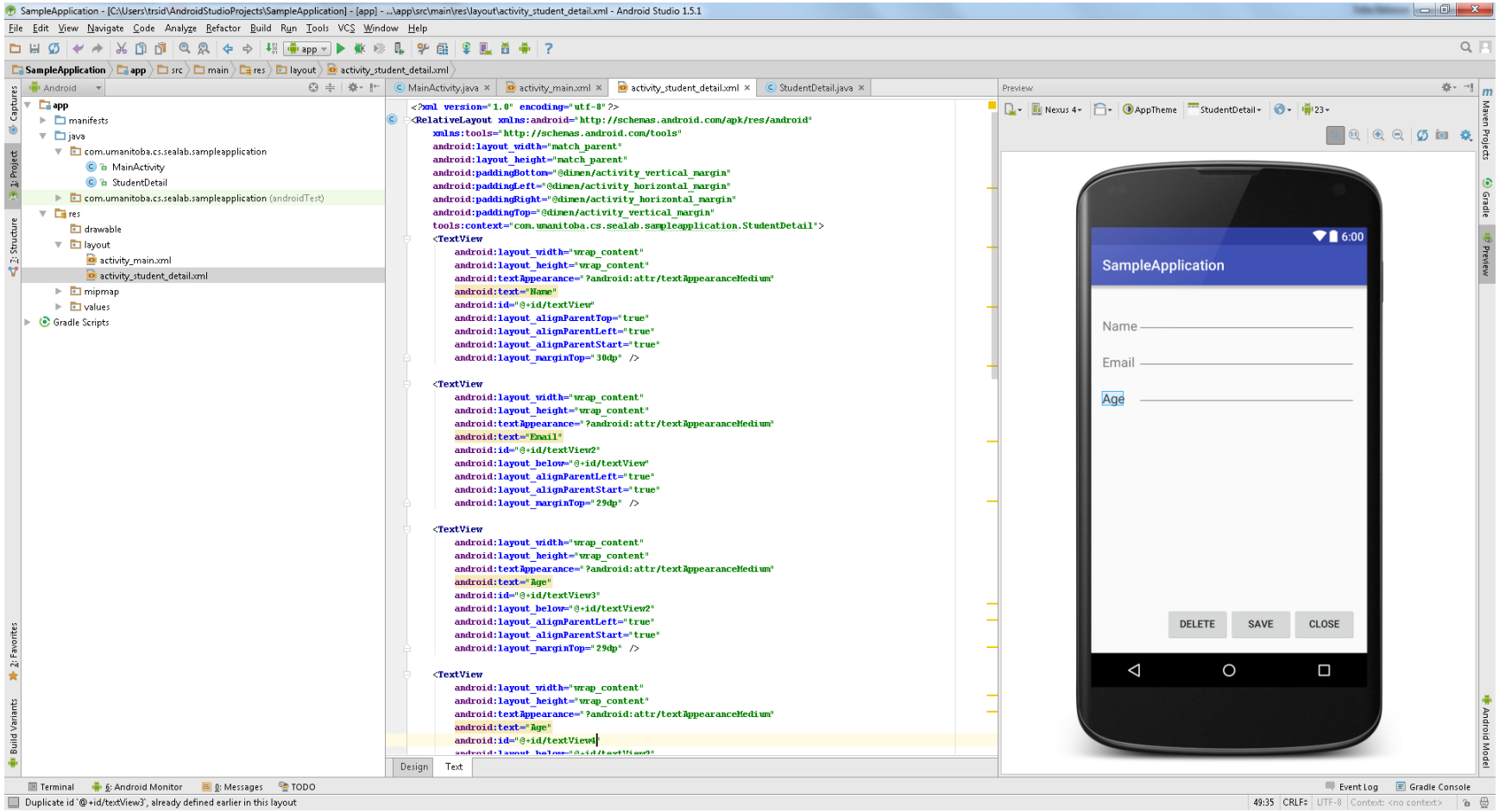


**Fig 24. Add Controls according to your fields in the activity_student_detail.xml just created above. Text views for Name, Email and Age and save button.**



## Code 2. activity_student_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.umanitoba.cs.sealab.sampleapplication.StudentDetail">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Name *"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="30dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Email *"
        android:id="@+id/textView2"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="29dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Age"
        android:id="@+id/textView3"
        android:layout_below="@+id/textView2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="29dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Age"
        android:id="@+id/textView4"
        android:layout_below="@+id/textView2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
```

```
        android:layout_marginTop="29dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/editTextName"
        android:layout_above="@+id/textView2"
        android:layout_toRightOf="@+id/textView"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"
        android:ems="10"
        android:id="@+id/editTextEmail"
        android:layout_above="@+id/textView3"
        android:layout_toRightOf="@+id/textView"
        android:layout_alignRight="@+id/editTextName"
        android:layout_alignEnd="@+id/editTextName" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:ems="10"
        android:id="@+id/editTextAge"
        android:layout_alignBottom="@+id/textView3"
        android:layout_alignLeft="@+id/editTextEmail"
        android:layout_alignStart="@+id/editTextEmail"
        android:layout_alignRight="@+id/editTextEmail"
        android:layout_alignEnd="@+id/editTextEmail" />
    <android.support.design.widget.FloatingActionButton
    android:id="@+id/btnSave"
    android:layout_width="180px"
    android:layout_height="180px"
    android:layout_gravity="bottom|end"
        android:src="@android:drawable/ic_menu_save"
        android:layout_alignParentBottom="true"
        android:layout_alignRight="@+id/editTextAge"
        android:layout_alignEnd="@+id/editTextAge" />

</RelativeLayout>
```

**Fig 25. Add a new view (layout) for individual list item to be shown in the list in Fig 20. If you notice in the activity_main.xml file above, we added a ListView, which is like a repeater for a widget or set of widgets. Now we are creating a template of one entry that will be repeated by the ListView**

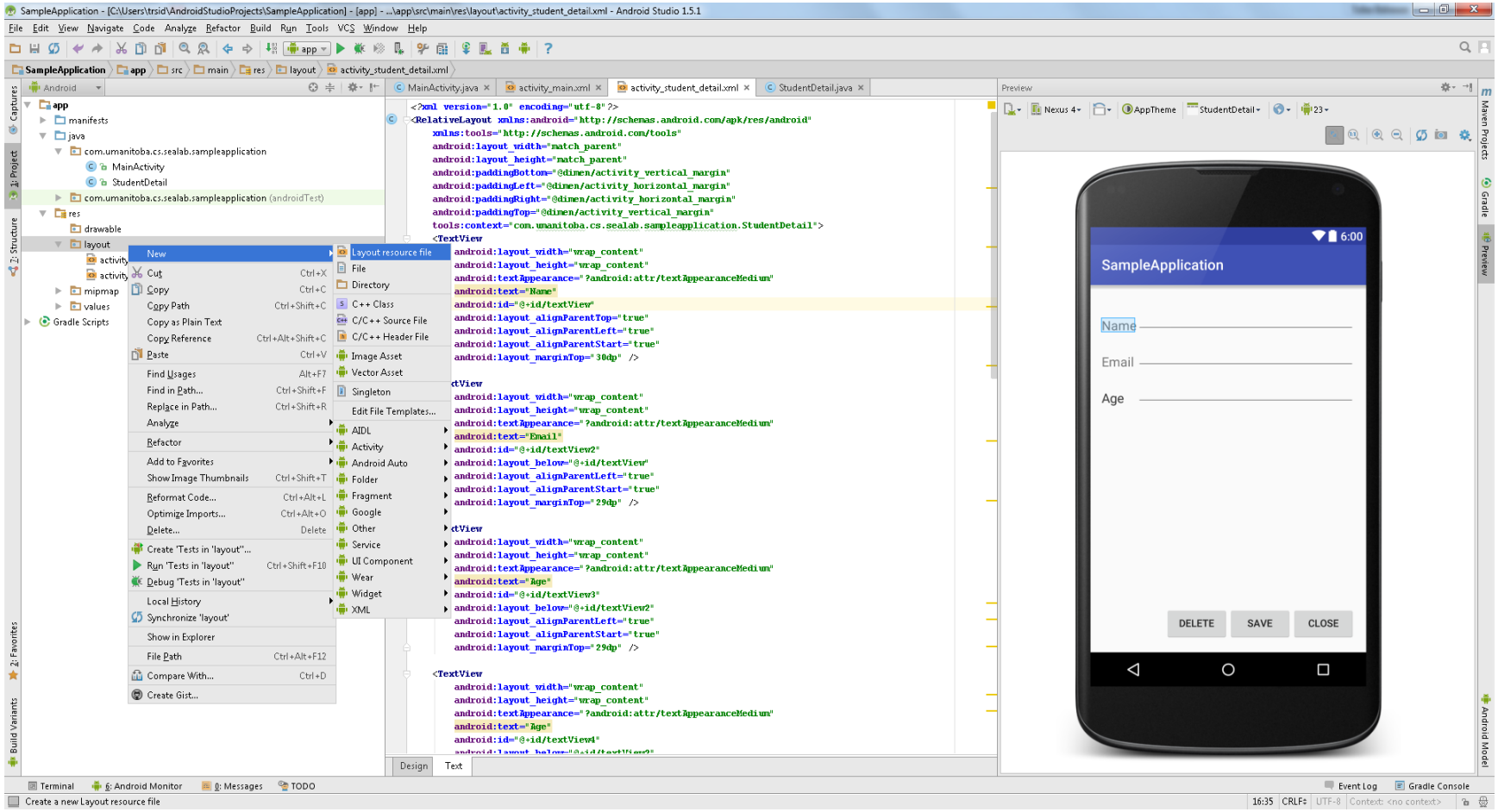
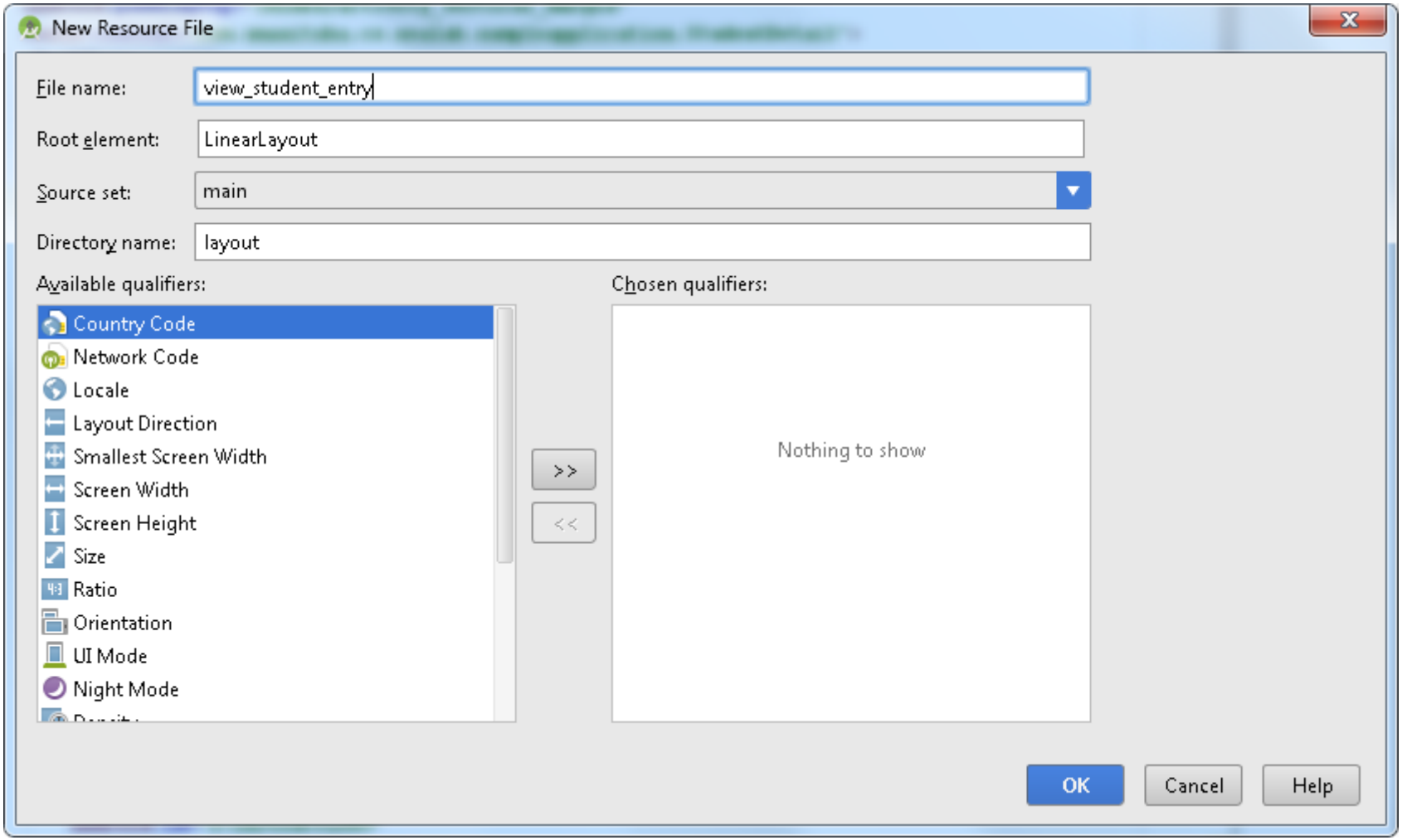


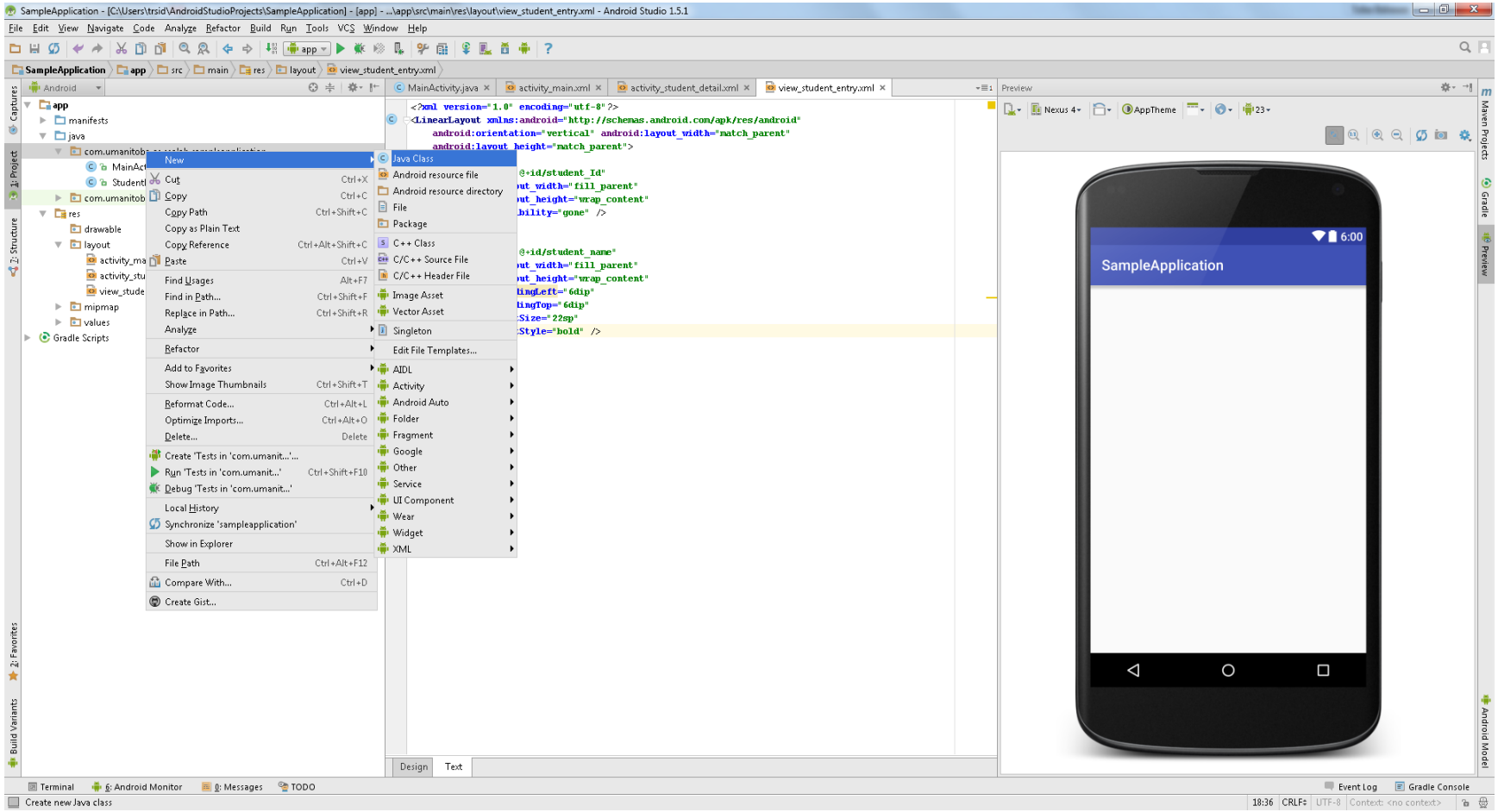
**Fig 26. Name the view view_student_entry and add the control in the Code 3 below**



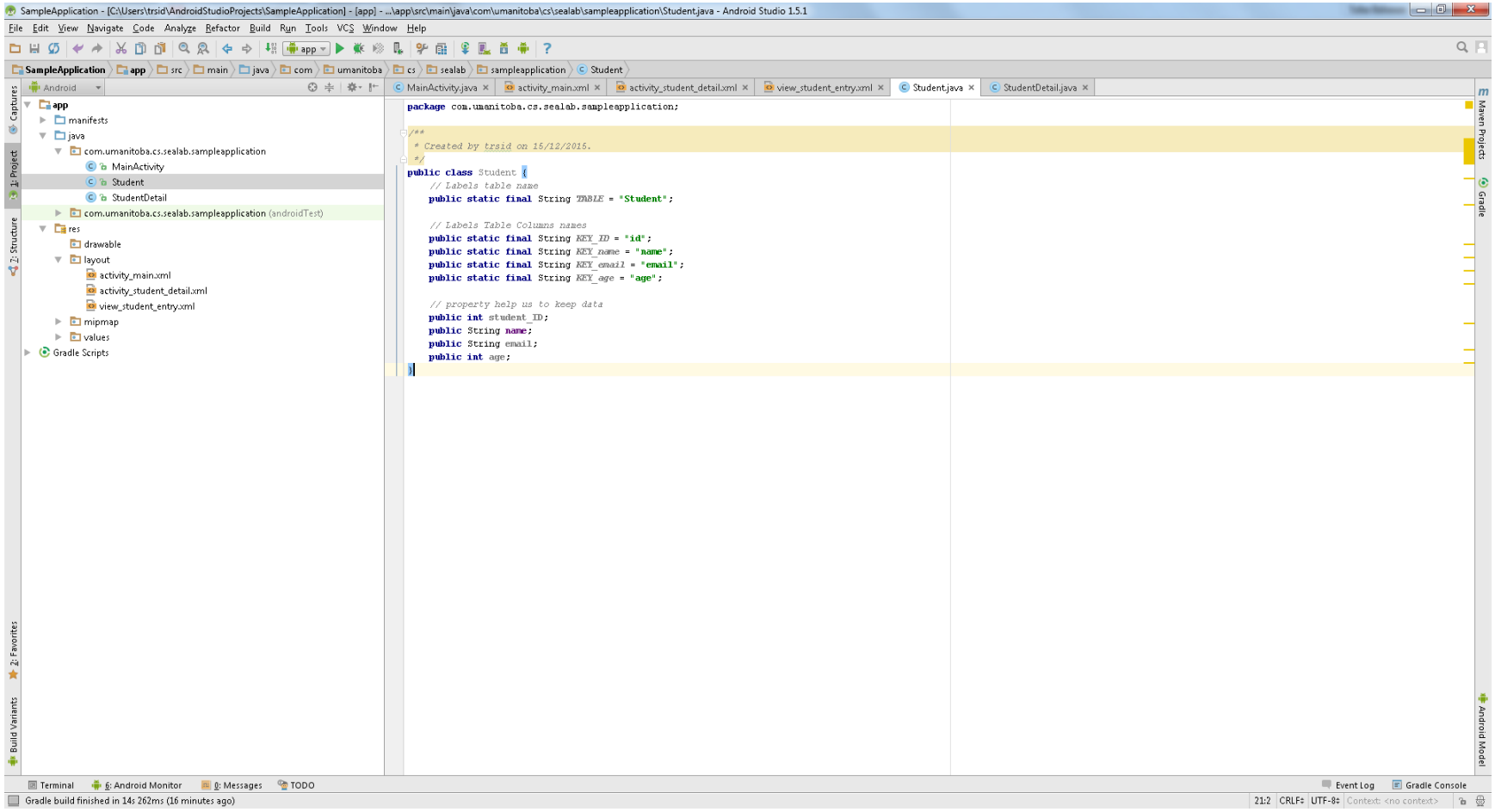
**Code 3. view_student_entry.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/student_Id"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:visibility="gone"/>

    <TextView
        android:id="@+id/student_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="6dip"
        android:textSize="26sp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/student_email"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="6dip"
        android:textSize="20sp" />
    <TextView
        android:id="@+id/student_age"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="6dip"
        android:textSize="20sp" />
</LinearLayout>
```

**Fig 27. Add a new class, Student**



## Fig 28. Add properties to your class



**Code 4. student.java**

```
public class Student {
    // Labels table name
    public static final String TABLE = "Student";

    // Labels Table Columns names
    public static final String KEY_ID = "id";
    public static final String KEY_name = "name";
    public static final String KEY_email = "email";
    public static final String KEY_age = "age";

    // property help us to keep data
    public int student_ID;
    public String name;
    public String email;
    public int age;
}
```

**Fig 31. This is the final structure of the project (on the left).**

**Notice the first folder manifests. Expand it and open the AndroidManifest.xml file to reveal the properties of your application. Basic configurations of the app is done here, eg. Setting image icon "android:icon="@mipmap/ic_launcher"", changing application name "android:label="Student Record Manager"" and most important; User Persmission. "<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />"**

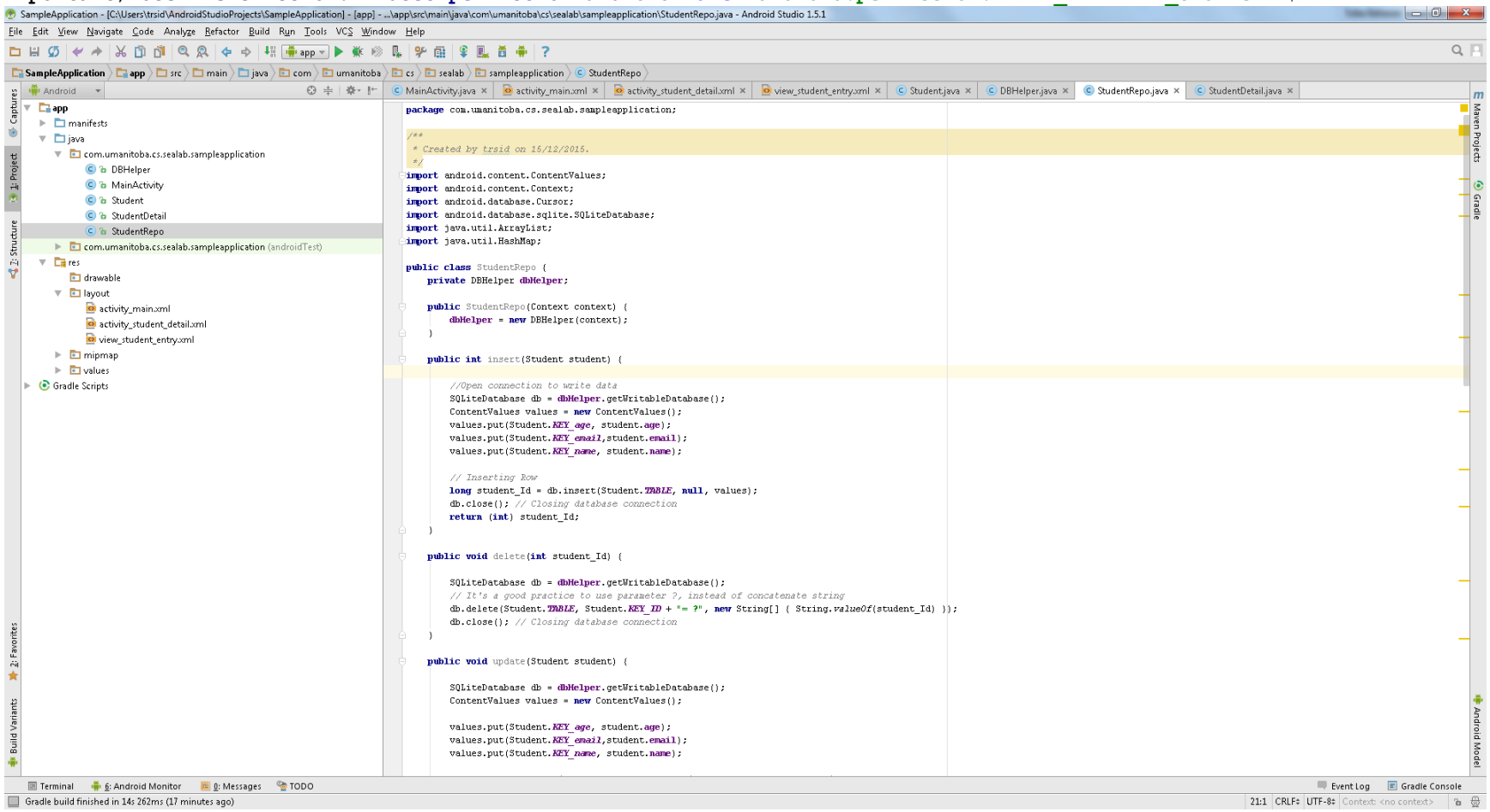
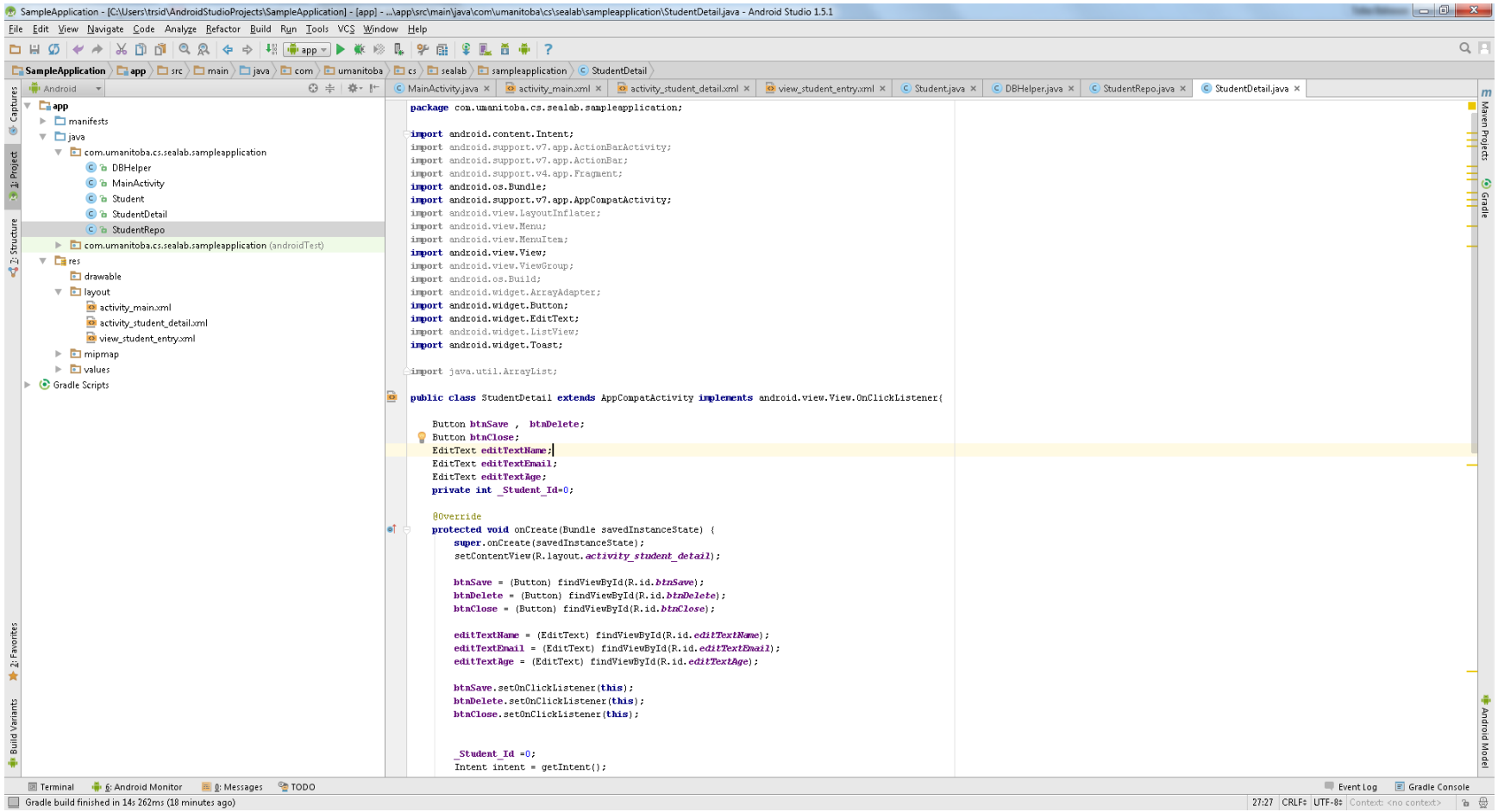


**Fig 32. Copy Code 7 to the already created StudentDetail class**

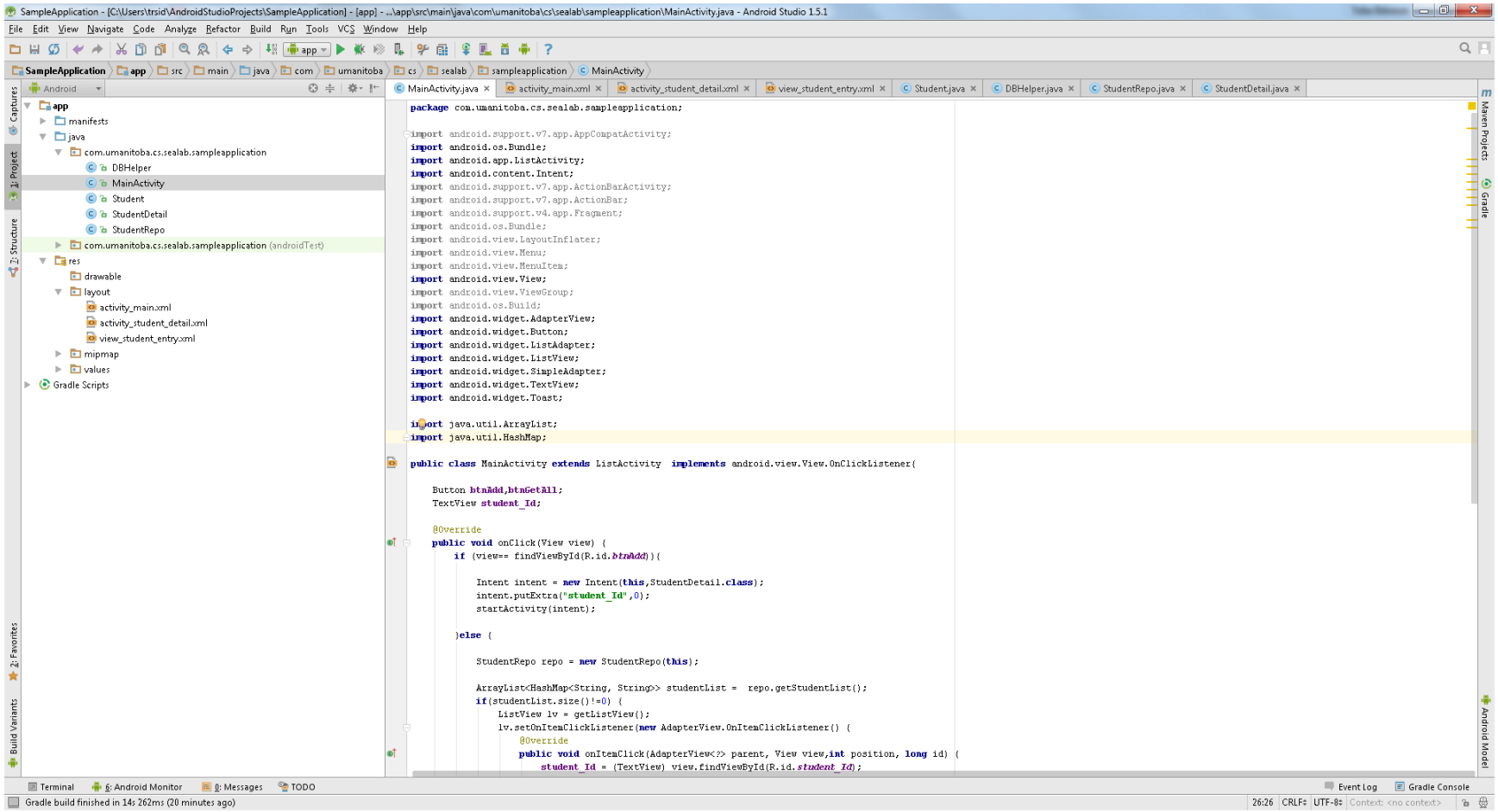


## Code 7. StudentDetail.java

```
package com.umanitoba.cs.sealab.sampleapplication;
import android.content.Context;
import android.content.Intent;
import android.support.design.widget.FloatingActionButton;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class StudentDetail extends AppCompatActivity implements android.view.View.OnClickListener{
    FloatingActionButton btnSave;
    EditText editTextName;
    EditText editTextEmail;
    EditText editTextAge;
    private int _Student_Id=0;
    String filename = "studentData";

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_student_detail);
        //attaching click listener to button
        btnSave = (FloatingActionButton) findViewById(R.id.btnSave);
        btnSave.setOnClickListener(this);
        //disabling the button for validation, this will be enabled if all the fields on the page are validated
        btnSave.setEnabled(false);
        //findViewById gets reference to the widget by ID
        editTextName = (EditText) findViewById(R.id.editTextName);
        editTextEmail = (EditText) findViewById(R.id.editTextEmail);
        editTextAge = (EditText) findViewById(R.id.editTextAge);
        //since we are going to get control to this Activity through another Activity, we can receive the data passed by the previous activity by intent.getInExtra("key",
defaultValue)
        _Student_Id =0;
        Intent intent = getIntent();
        _Student_Id =intent.getIntExtra("student_Id", 0);
        //setting default texts
        editTextAge.setText("");
        editTextName.setText("");
        editTextEmail.setText("");
        //attaching text changed listener, so every time you write something in the textbox, validation will occur and if passed button will be enable. Currently the validation is
only checking if the fields are empty and the email is in correct format. You can write custom validations here.
        TextWatcher listener = new TextWatcher(){
            @Override public void beforeTextChanged(CharSequence s, int start, int count, int after) {}
            @Override public void onTextChanged(CharSequence s, int start, int before, int count) {}
            @Override public void afterTextChanged(Editable s){
                btnSave.setEnabled(!editTextAge.getText().toString().isEmpty() && !editTextName.getText().toString().isEmpty() &&
isEmailValid(editTextEmail.getText().toString()));
            }
        };
        editTextAge.addTextChangedListener(listener);
        editTextName.addTextChangedListener(listener);
```

```
        editTextEmail.addTextChangedListener(listener);
    }

    public void onClick(View view) {
        if (view == findViewById(R.id.btnSave)) {
            //reading data to file. Since we have saved our data to file, we need to know the id of the new item to be inserted. So we will fetch the last item, and increment its
id by 1 as the new items id.
            StringBuffer fileContent = new StringBuffer("");
            try {
                FileInputStream f = openFileInput(filename);

                byte[] buffer = new byte[1024];
                int n = 0;
                while ((n = f.read(buffer)) != -1) {
                    fileContent.append(new String(buffer, 0, n));
                }
            } catch (java.io.IOException e) {
                e.printStackTrace();
                Toast.makeText(this, "Adding first record.", Toast.LENGTH_SHORT).show();
            }
            String[] data = fileContent.toString().split("~");
            int lastId = 0;
            if (!fileContent.toString().isEmpty()) {
                lastId = Integer.parseInt(data[data.length - 1].split(",")[0]);
            }
            Student student = new Student();
            student.student_ID = lastId + 1;
            student.age = Integer.parseInt(editTextAge.getText().toString());
            student.email = editTextEmail.getText().toString();
            student.name = editTextName.getText().toString();
            //preparing the string to be appended to the string for the new record
            String s = "";
            s = "~" + student.student_ID + "," + student.name + "," + student.age + "," + student.email + "~";
            //writing the final data string to file
            FileOutputStream outputStream;
            try {
                outputStream = openFileOutput(filename, Context.MODE_APPEND);
                outputStream.write(s.getBytes());
                outputStream.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            Toast.makeText(this, "First Record Added, Tada", Toast.LENGTH_SHORT).show();
            //finish will exit and destroy this activity and will go back to the one it is called from
            finish();
        }
    }
    @Override public void finish(){
        super.finish();
        //if you want to do some tasks like garbage removal or need to close any open adapters or connection, you should do that here.
    }
    //utility method
    public boolean isEmailValid(String email) {
        boolean isValid = false;
        String expression = "^[\\w\\.-]+@([\\w\\-]+\\.)+[A-Z]{2,4}$";
        CharSequence inputStr = email;
        Pattern pattern = Pattern.compile(expression, Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(inputStr);
        if (matcher.matches()) {
            isValid = true;
        }
        return isValid;
    }
}
```

## Fig 33. Copy Code 8 to MainActivity.java



## Code 8. MainActivity.java

```
package com.umanitoba.cs.sealab.sampleapplication;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.support.design.widget.FloatingActionButton;
import android.os.Bundle;
import android.app.ListActivity;
import android.content.Intent;
import android.text.TextUtils;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
import android.widget.Toast;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.HashMap;

public class MainActivity extends ListActivity  implements android.view.View.OnClickListener{

    String filename = "studentData";
    FloatingActionButton btnAdd;
    Button btnShowFileAction;//btnGetAll,btnReadFromFile,btnWriteToFile;
    TextView student_Id;

    //We binded all the click actions with onClick function, inside which we can differentiate among sender widgets using the id property of view
    @Override  public void onClick(View view) {
        FileOutputStream outputStream;
        if (view== findViewById(R.id.btnAdd)){
            //Moving to another activity
            Intent intent = new Intent(this,StudentDetail.class);
            intent.putExtra("student_Id", 0);
            startActivity(intent);
            //For adding animation to activity transition
            overridePendingTransition(R.anim.abc_fade_in, 100000);
        }
        else if(view == findViewById(R.id.btnShowFileAction)) {
            Intent intent = new Intent(MainActivity.this, FileWriting.class);
            startActivity(intent);
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //attaching listeners to widgets
        btnAdd = (FloatingActionButton) findViewById(R.id.btnAdd);
        btnAdd.setOnClickListener(this);
        btnShowFileAction= (Button) findViewById(R.id.btnShowFileAction);
        btnShowFileAction.setOnClickListener(this);
    }
```

```
    private void ListStudents()
    {
        File file = new File(filename);
        if(file != null) {
            //reading from file
            StringBuffer fileContent = new StringBuffer("");
            try {
                FileInputStream f = openFileInput(filename);
                byte[] buffer = new byte[1024];
                int n = 0;
                while ((n = f.read(buffer)) != -1)
                {
                    fileContent.append(new String(buffer, 0, n));
                }
            } catch (java.io.IOException e) {
                e.printStackTrace();
                //showing toast notification
                Toast.makeText(this, "No Data file currently existing in mobile.",Toast.LENGTH_SHORT).show();
                //showing yes no confirmation alert
                new AlertDialog.Builder(this)
                        .setMessage("Do you want to create a new record?")
                        .setCancelable(false)
                        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                Intent intent = new Intent(MainActivity.this,StudentDetail.class);
                                intent.putExtra("student_Id", 0);
                                startActivity(intent);
                                overridePendingTransition(R.anim.abc_fade_in, 100000);
                             }
                        })
                        .setNegativeButton("No", null)
                        .show();
                return;
            }
            //since we store our results in a file in csv format, we are going to just read the file and perform split operations to see the records
            String[] data = fileContent.toString().split("~");
            ArrayList<HashMap<String, String>> studentList = new ArrayList<HashMap<String, String>>();
            for(int a = 0 ; a < data.length; a++) {
                HashMap<String, String> student = new HashMap<String, String>();
                if(data[a].split(",").length > 1){
                    String[] thisData = data[a].split(",");
                    student.put("id", thisData[0]);
                    student.put("name", thisData[1]);
                    student.put("age", thisData[2]);
                    student.put("email", thisData[3]);
                    studentList.add(student);
                }
            }
            if(studentList.size()!=0) {
                //since this Activity extends ListActivity, getListView will give the reference to List on the page
                ListView lv = getListView();
                //here we are attaching a listener (for deletion) on list item click action
                lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> parent, View view,int position, long id) {
                        //since we have a textview in view_student_entry.xml as student_Id, we will fetch that and confirm if user wants to delete the entry.
                        int student_Id = Integer.parseInt(((TextView)view.findViewById(R.id.student_Id)).getText().toString());
                        confirm(student_Id);
                    }
                });
                //we are now binding the actual data with the List
                ListAdapter adapter = new SimpleAdapter( MainActivity.this,studentList, R.layout.view_student_entry, new String[]{"id","name","age","email"}, new int[]
{R.id.student_Id, R.id.student_name, R.id.student_age,R.id.student_email});
                setListAdapter(adapter);
            }else{
                //in case of no data found in file
                ListView lv = getListView();
```

```
                ListAdapter adapter = new SimpleAdapter( MainActivity.this,studentList, R.layout.view_student_entry, new String[]{"id","name","age","email"}, new int[]
{R.id.student_Id, R.id.student_name, R.id.student_age,R.id.student_email});
                setListAdapter(adapter);
                Toast.makeText(this,"No student!",Toast.LENGTH_SHORT).show();
            }
        }
    }

    private void confirm(final int student_ID) {
        new AlertDialog.Builder(this)
                .setMessage("Are you sure you want to delete this record?")
                .setCancelable(false)
                .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        StringBuffer fileContent = new StringBuffer("");
                        try {
                            FileInputStream f = openFileInput(filename);

                            byte[] buffer = new byte[1024];
                            int n = 0;
                            while ((n = f.read(buffer)) != -1) {
                                fileContent.append(new String(buffer, 0, n));
                            }
                        } catch (java.io.IOException e) {
                            e.printStackTrace();
                        }
                        String[] data = fileContent.toString().split("~");
                        String dataString = "";
                        for (int i = 0; i < data.length; i++) {
                            if (!data[i].isEmpty()) {
                                if (Integer.parseInt(data[i].split(",")[0]) != student_ID) {
                                    dataString += TextUtils.join(",", data[i].split(",")) + "~";
                                }
                            }
                        }
                        FileOutputStream outputStream;
                        try {
                            outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
                            outputStream.write(dataString.getBytes());
                            outputStream.close();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                        ListStudents();
                    }
                })
                .setNegativeButton("No", null)
                .show();
    }
    //calls in the start too, or if you change orientation
    protected void onResume(){
        super.onResume();
        ListStudents();
    }
}
```

Another activity is added in the sample project as a reference, to give an idea of how to navigate between different activities. The activity is called FileWriting and performs the basic functions of reading from and writing to a file.

Section 3

# Setting up Emulator and running the app

**Fig 34. We are done with the application. Now downloading a faster emulator for testing our app. Google “Download Genymotion”**

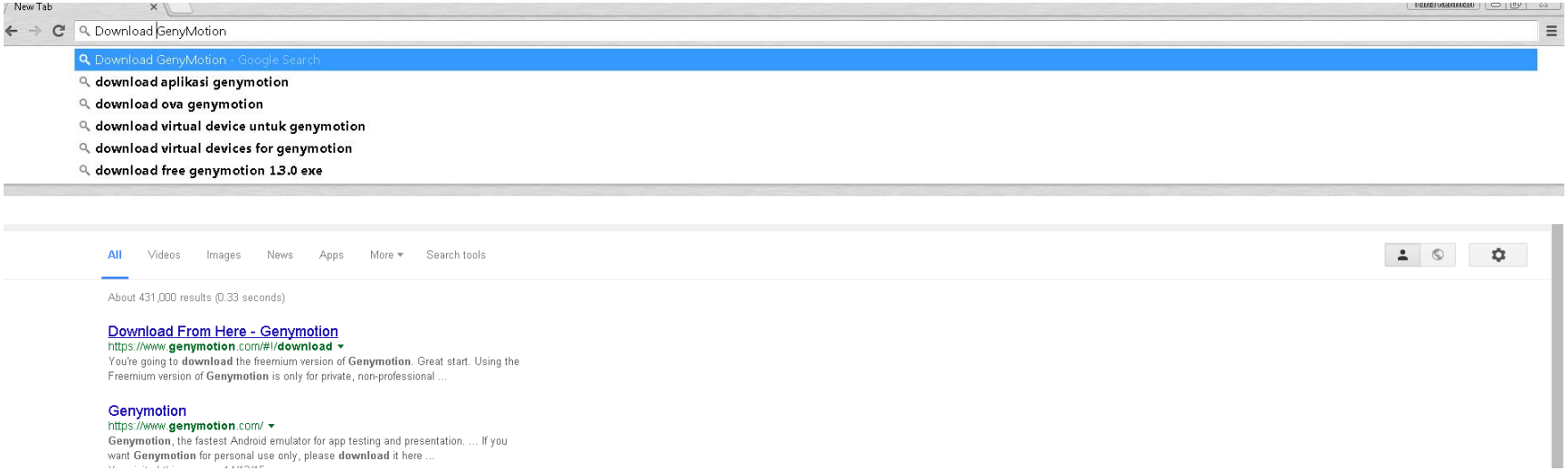


**Fig 35. Download Genymotion for free (Personal Use)**

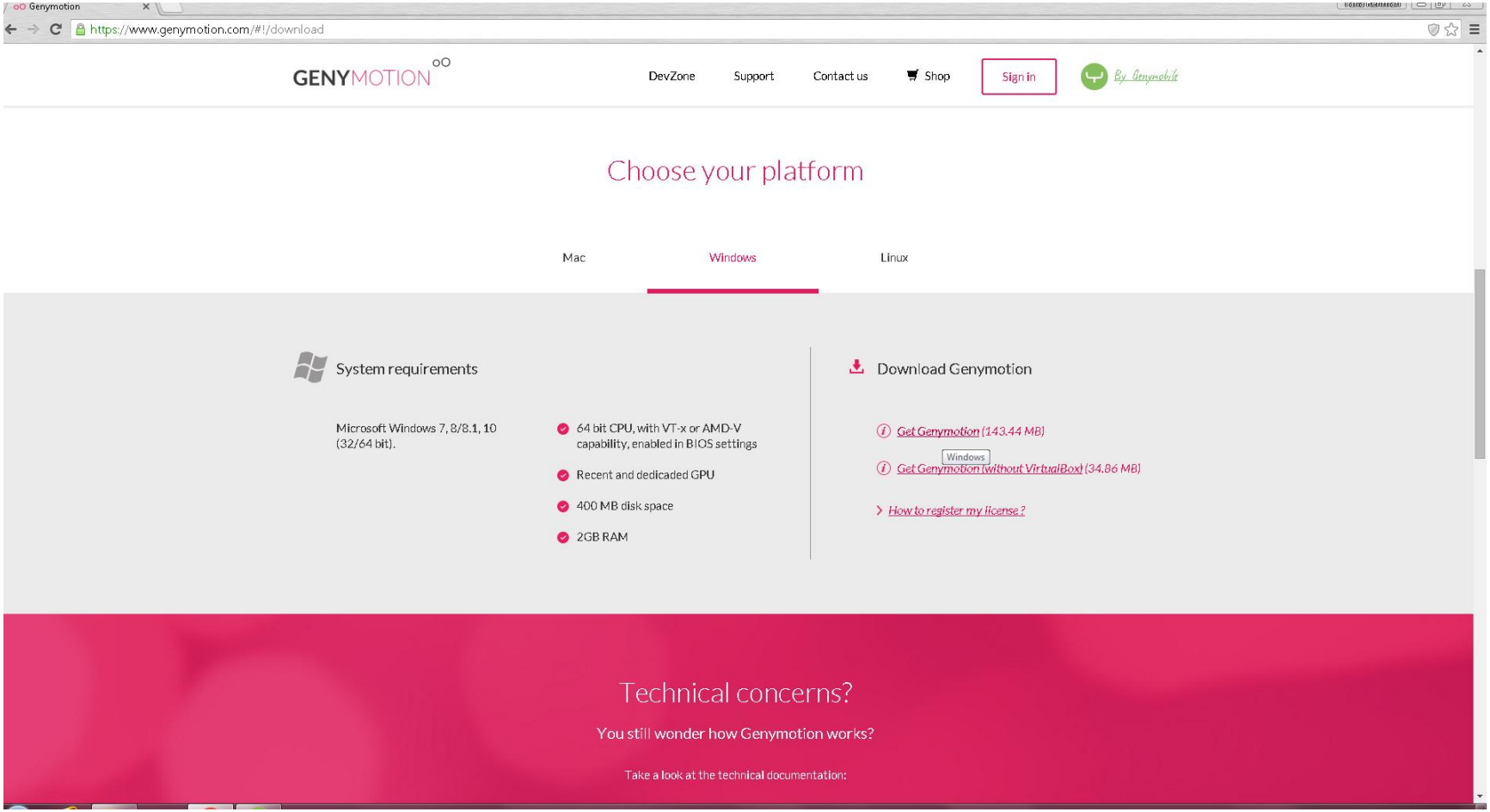
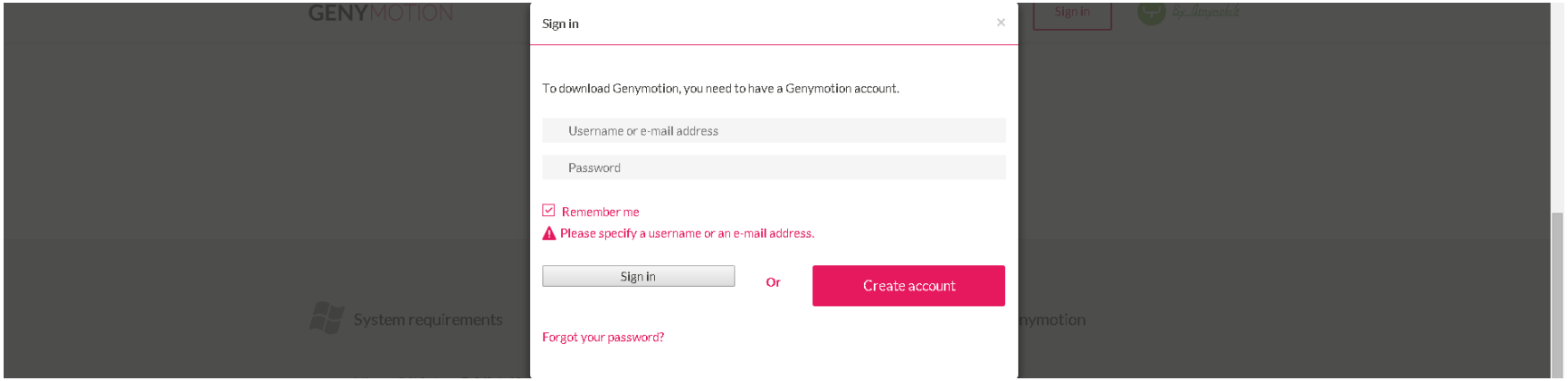


**Fig 36. On login screen, login or create a new account. Remember the credentials as we are going to need them later**



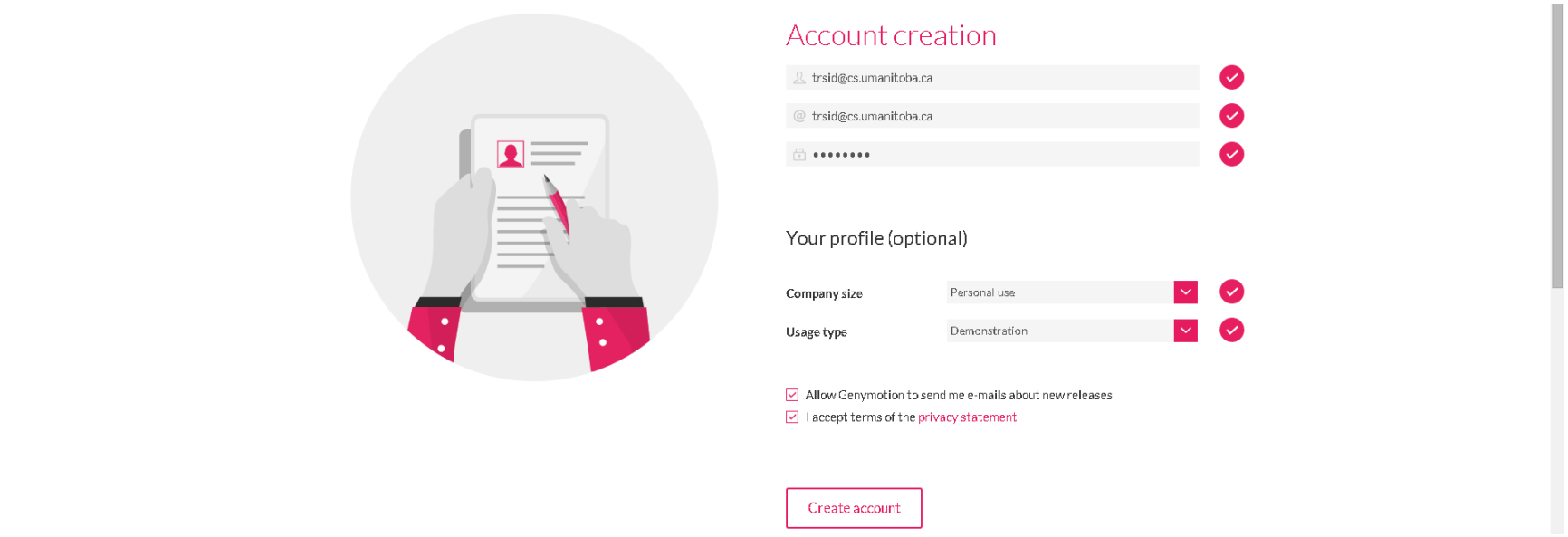



**Fig 37. Download after logging in**

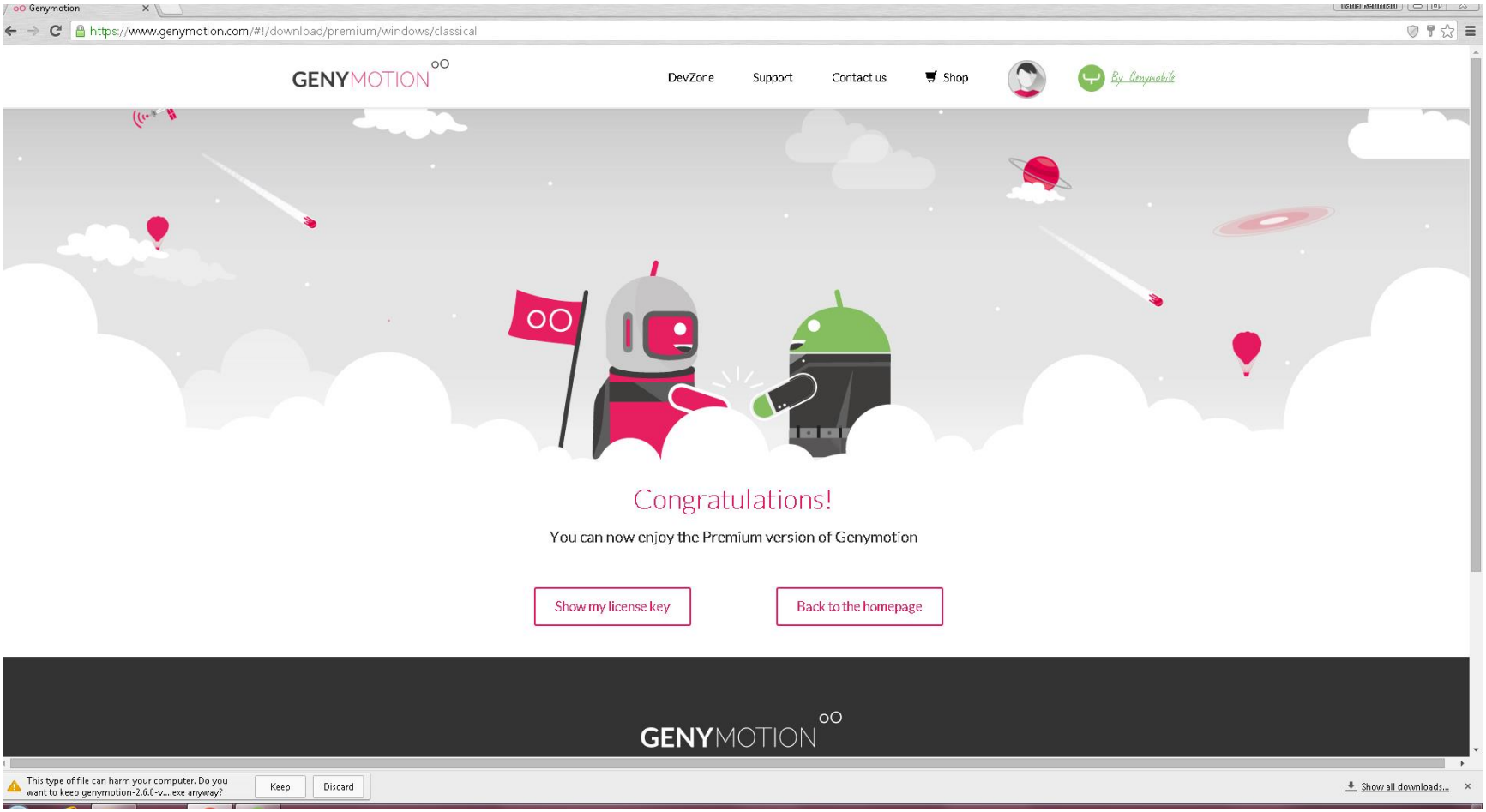


**Fig 38. Run the setup. It needs VirtualBox® to run, so it will install that as well. Keep pressing Next or Install as shown below.**

**Fig 39. Done with the installation. Press finish to launch Genymotion only. After accepting the usage notice add a new virtual device (auto popup) using the same credentials that we used to download Genymotion.**

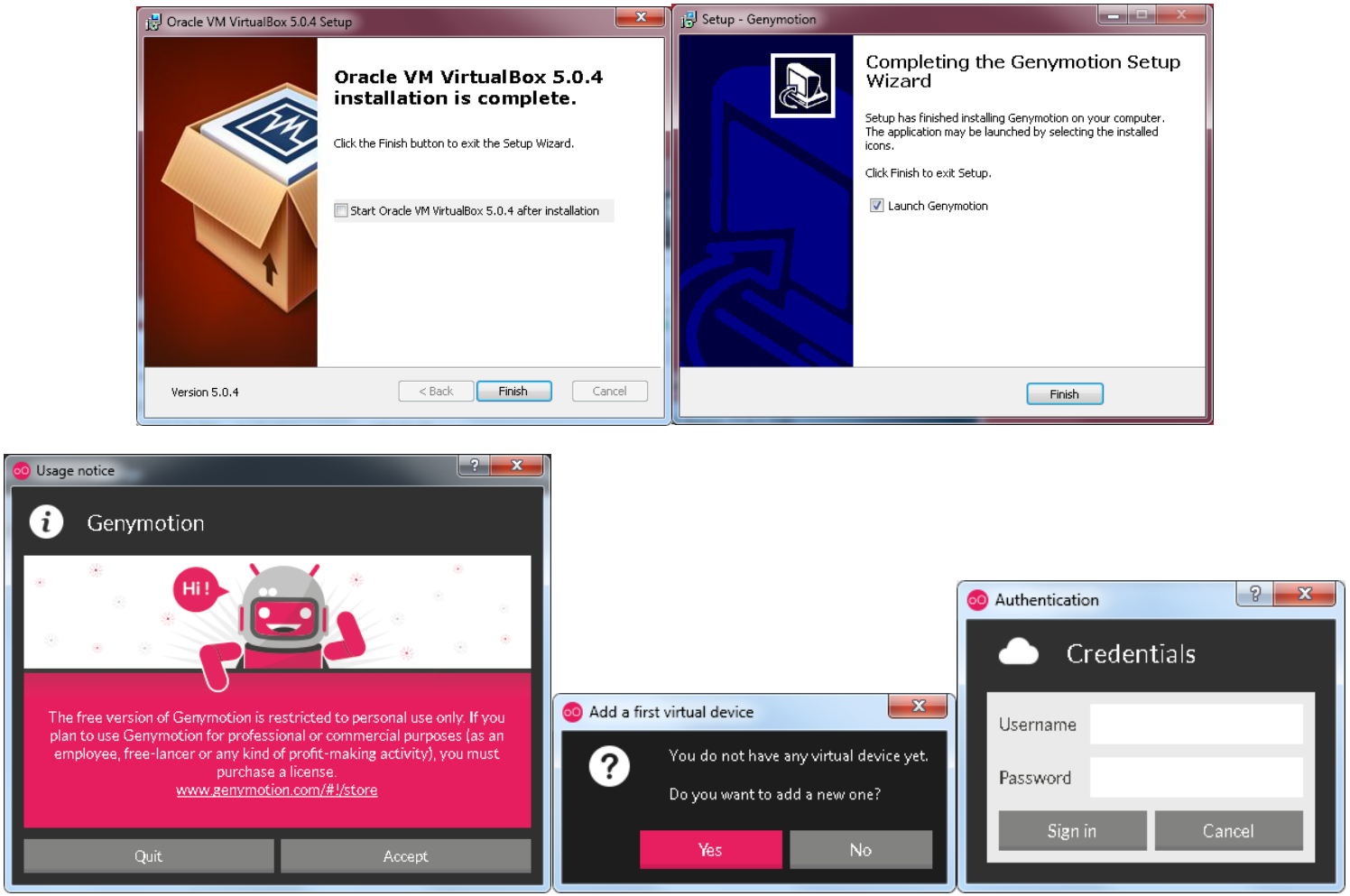


**Fig 40. Add a new virtual device; Nexus 7. Lower the resolution of the device, (slightly) better the performance. After finishing, start the device from the application menu.**

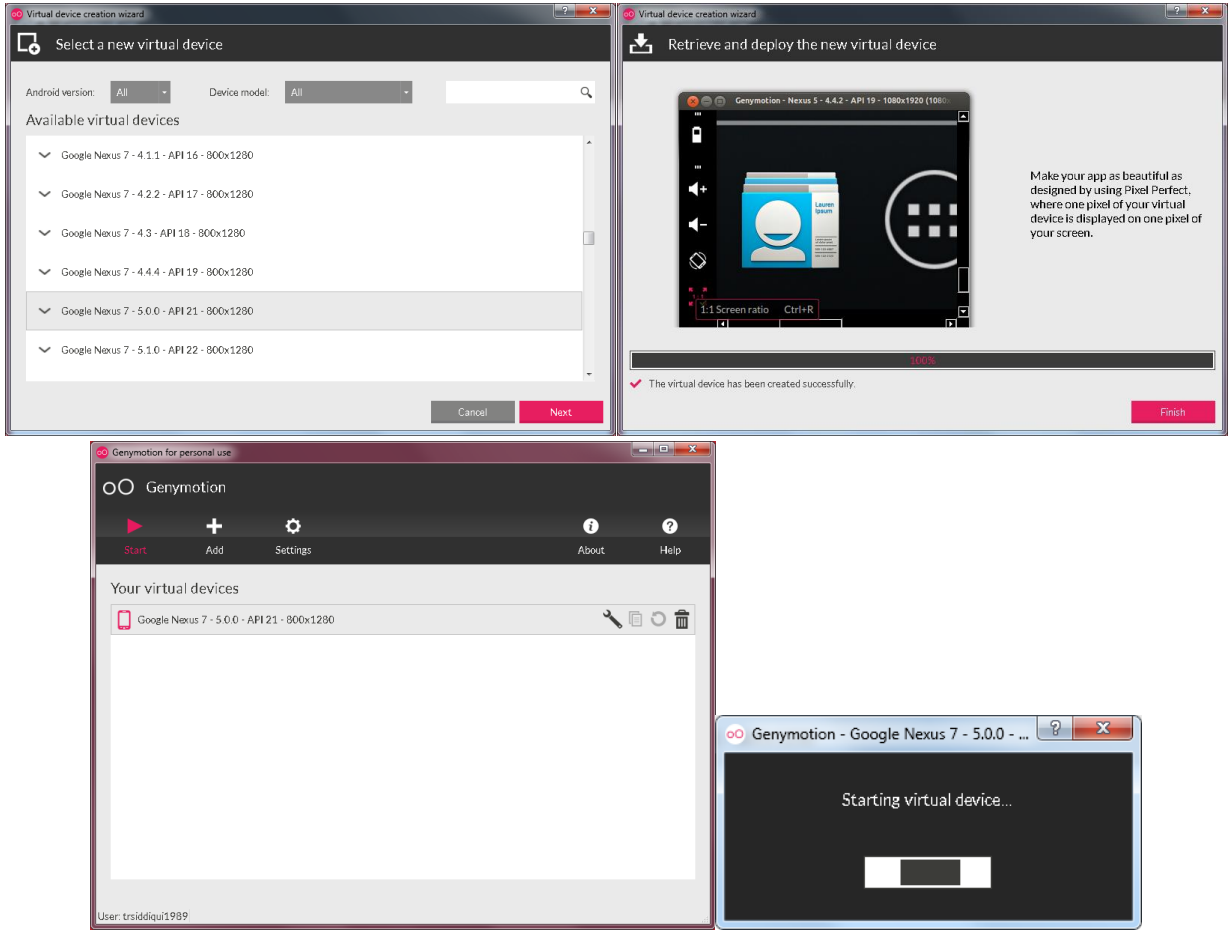


**Fig 41. This is what our new device looks like. Once it finishes loading, it will work like any other android phone/tablet.**

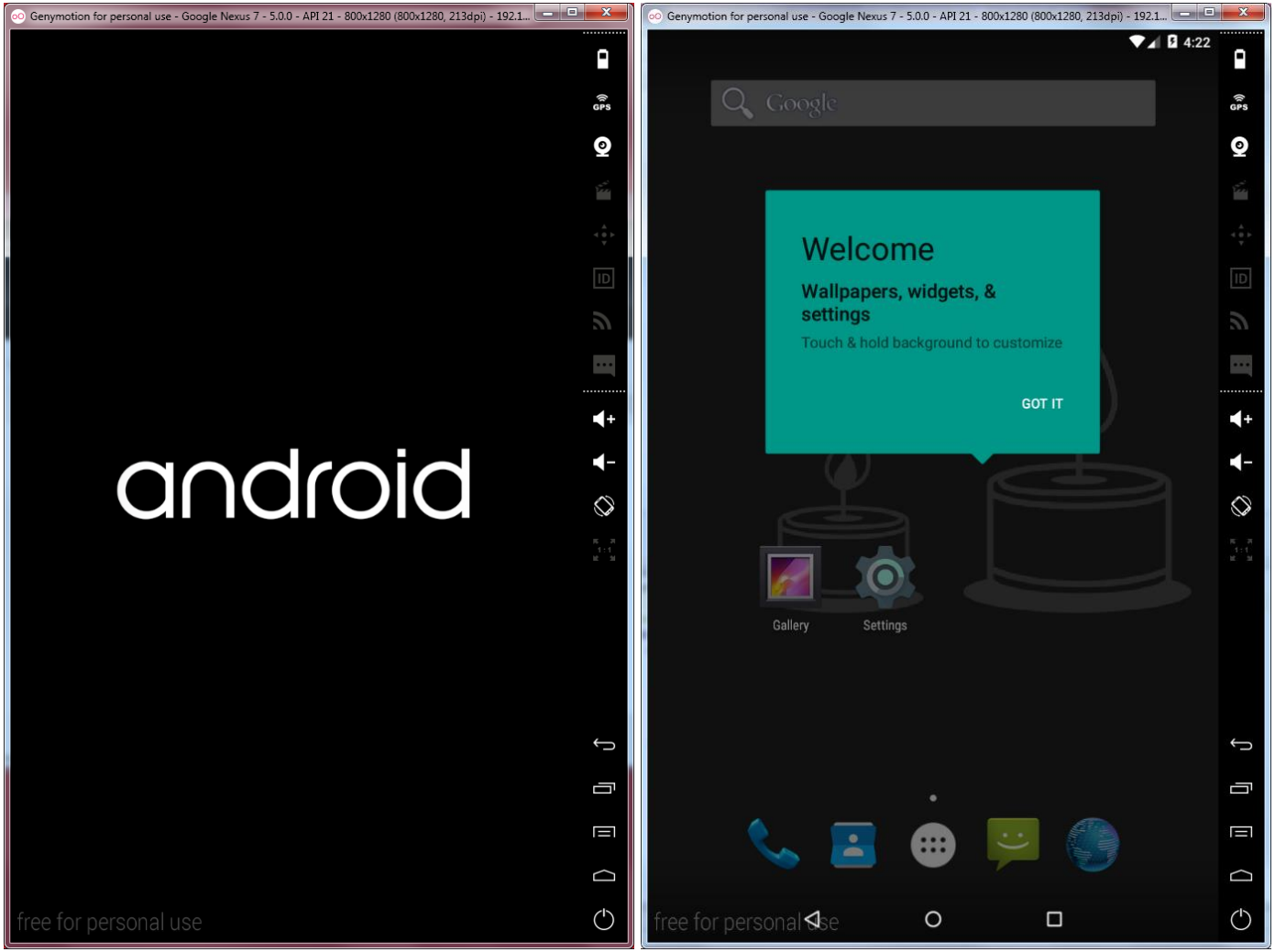


**Fig 42. Coming back to Android Studio®. Go to menu Build and click on Build APK.**

**You can also attach your own android device/phone(not recommended) to your computer and run the application by the toolbar Run → 'Run app'. Before you connect your mobile, do not forget to activate USB_Debugging on your phone (which allows the IDE to run adb<an interface to talk to android OS through USB> commands on your phone). To do that, go to Settings on your android device. In the last section 'System' you will find Developer Options (if not, go to About Phone and click on the item Build number repeatedly until it says you are a developer). Go to Developer options and enable it. Also enable USB debugging and go back to settings. Now open Security and under the section of Device administration, enable Unknown sources and ignore the warnings. This option keeps a check on the source of the application if it is coming from an authenticated source like Google Play Store ®. Since we are not a known source, to install our developed app on the device, we need to check this option.**

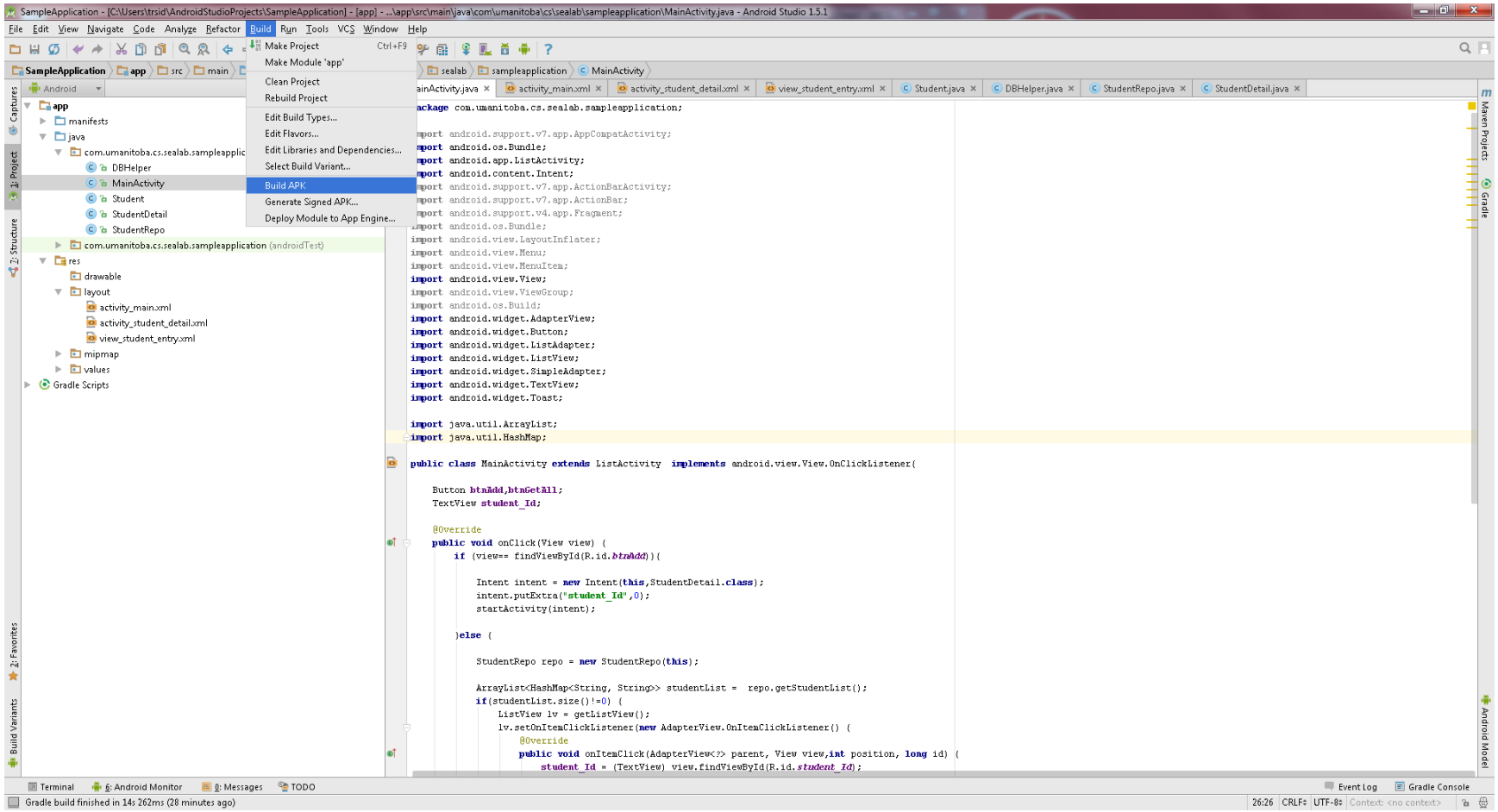
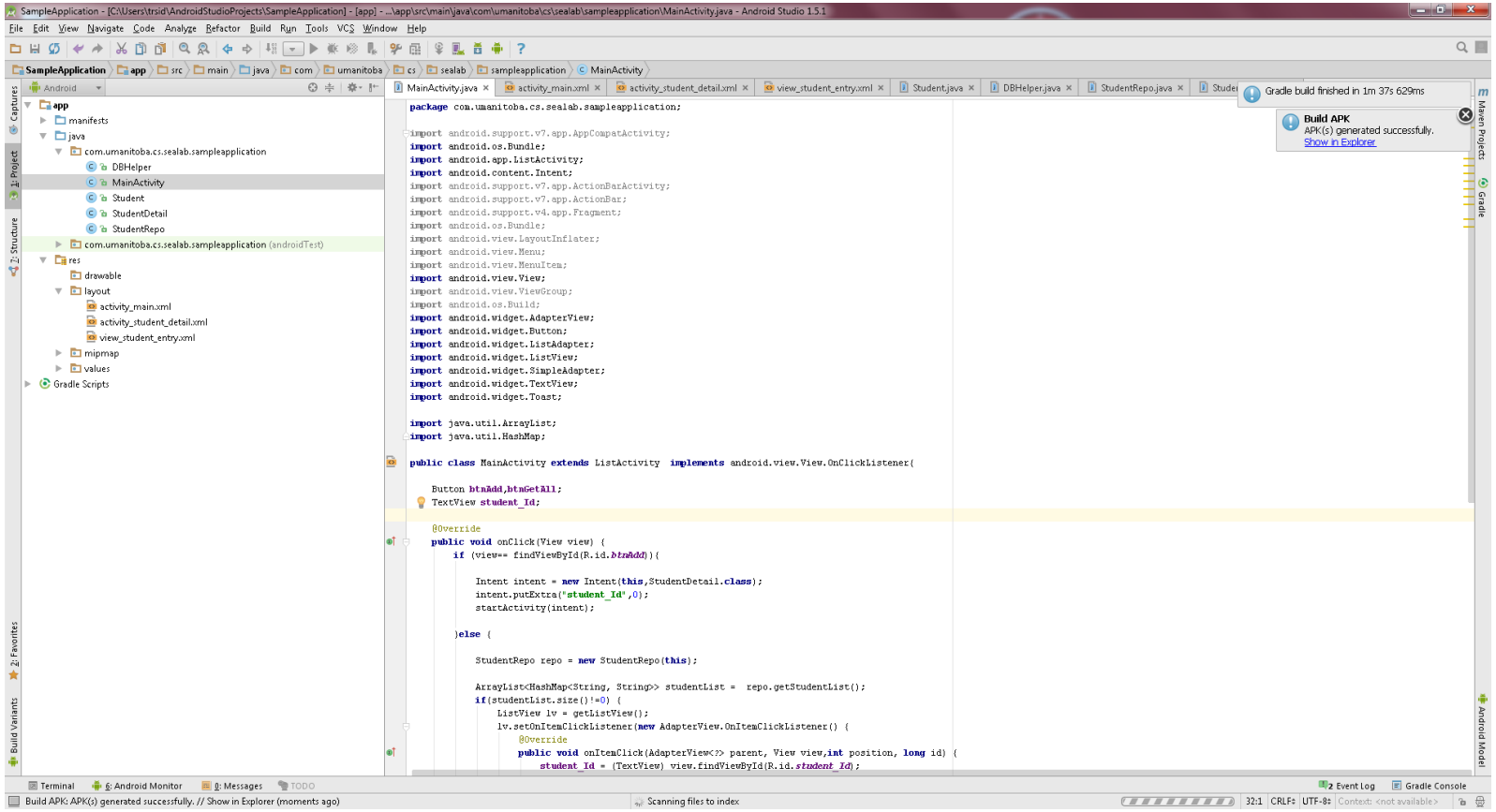


**Fig 43. After building, it will show a link to the APK file generated. Click on the link on the top right of the screen to open the folder.**



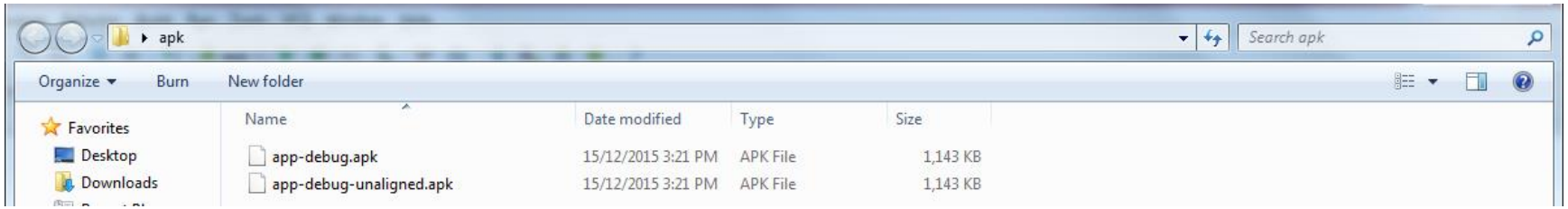



**Fig 44. Now drag the APK file and drop it to our emulator Genymotion®. It will show a “File Transfer” window and open the application.**

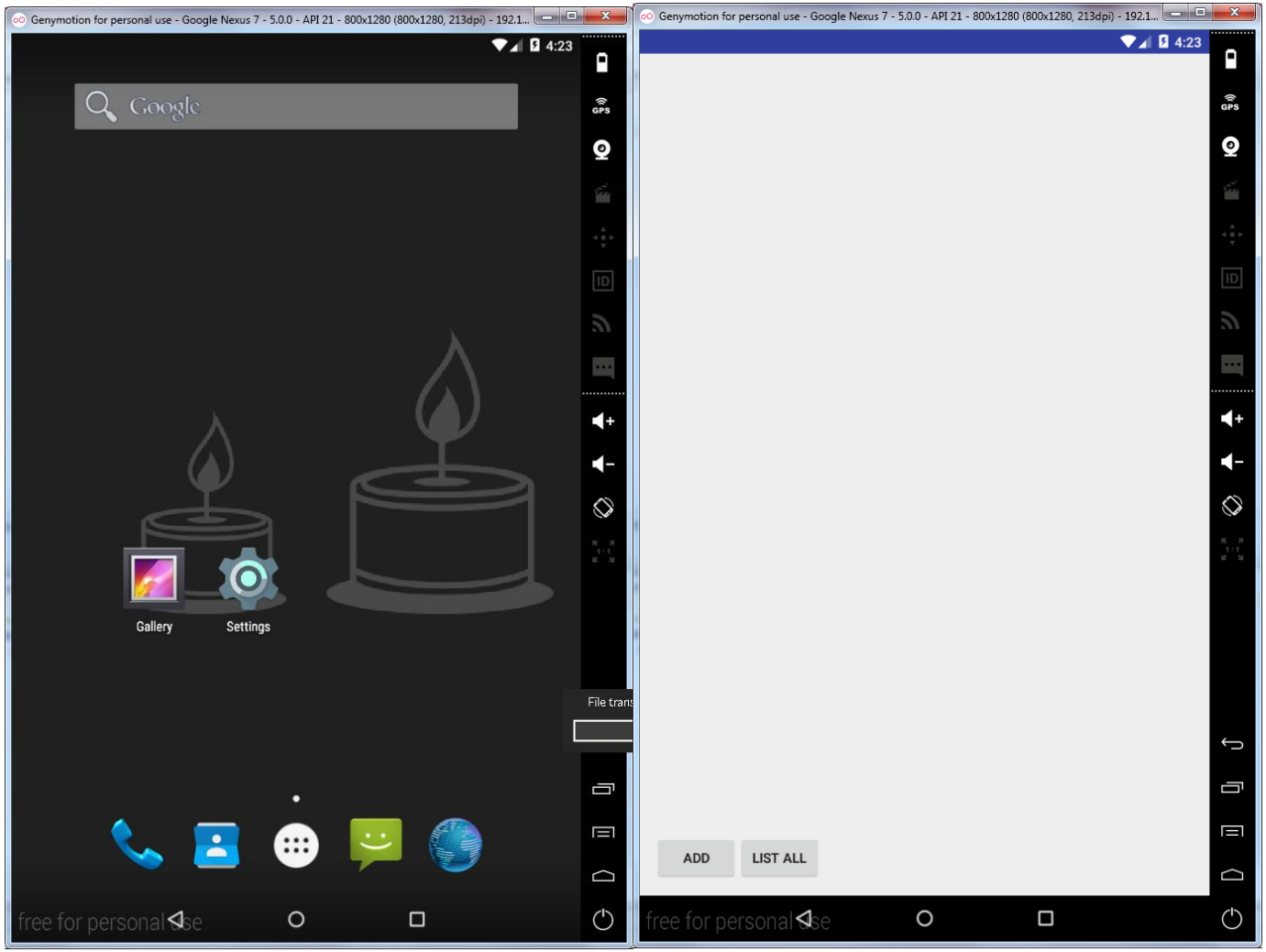
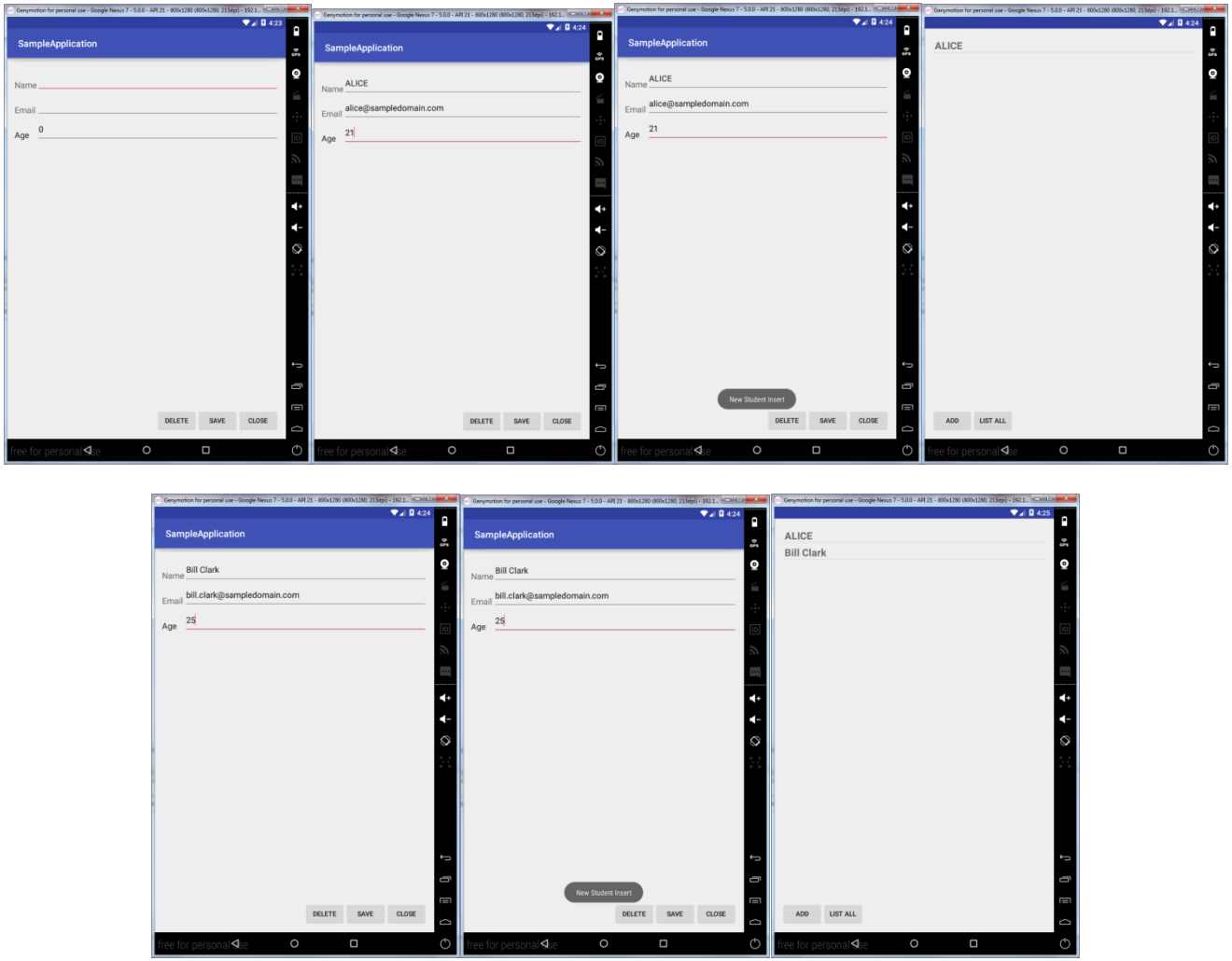


**Fig 45. Test our application (the application developed with the pasted code will look and perform differently)**



Section 4

# Validating the changes to the DB on device

**Fig 46. Download SQLite Browser for Eclipse from the link [Android SQLite Browser for Eclipse]**

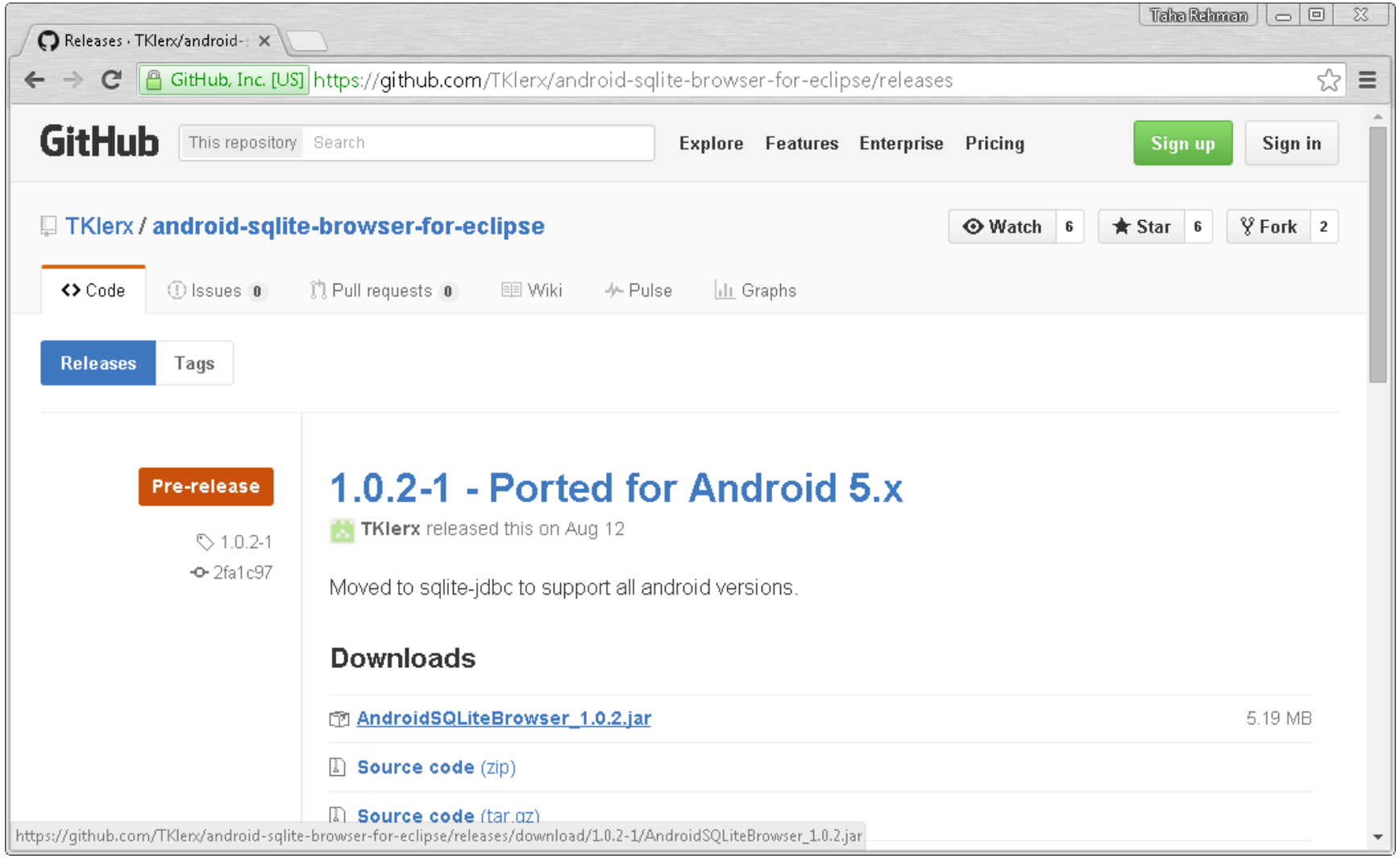


**Fig 47. Find the path of Eclipse plugin directory by Opening the below Section in Settings (File → Settings) and copy by selecting it manually and pressing Ctrl + C. Now open the folder manually using File Explorer and browse to <PATH_YOU_JUST_COPIED> → tools → lib → monitor-x86 → plugins**

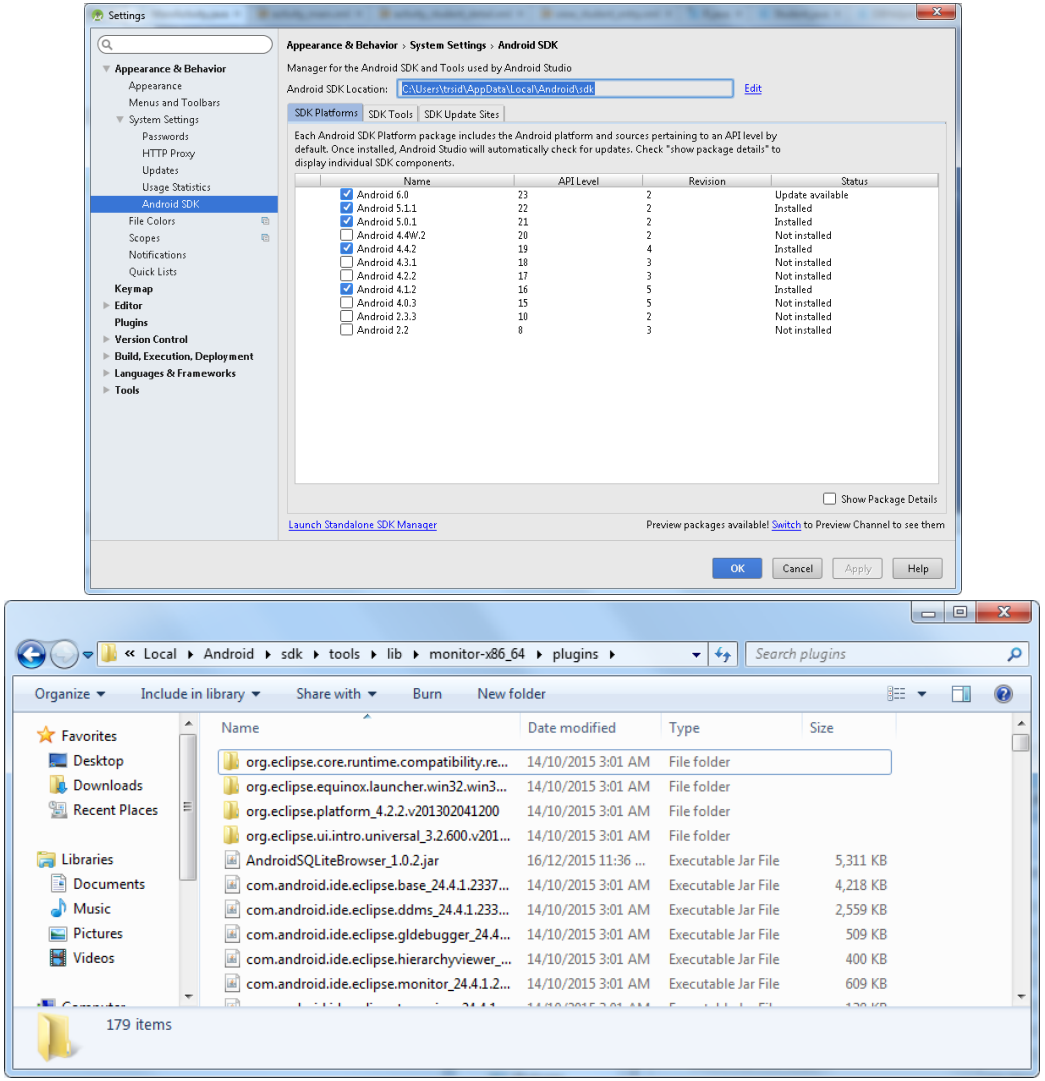


**Fig 48. Copy the downloaded file and paste to above folder.**

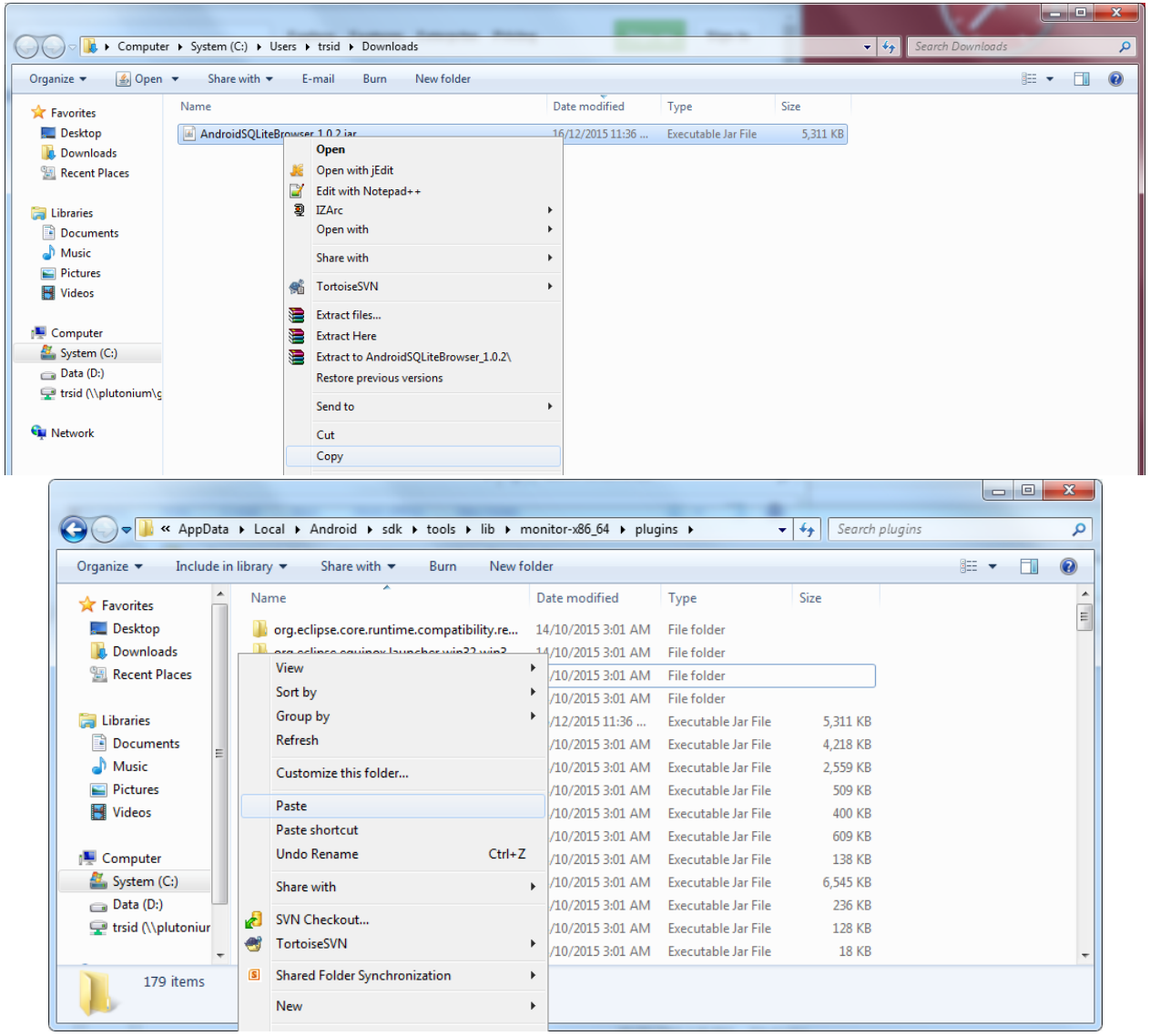


**Fig 49. Keep the app running in Genymotion® and Open Android Device Monitor by clicking on the following button in the toolbar.**

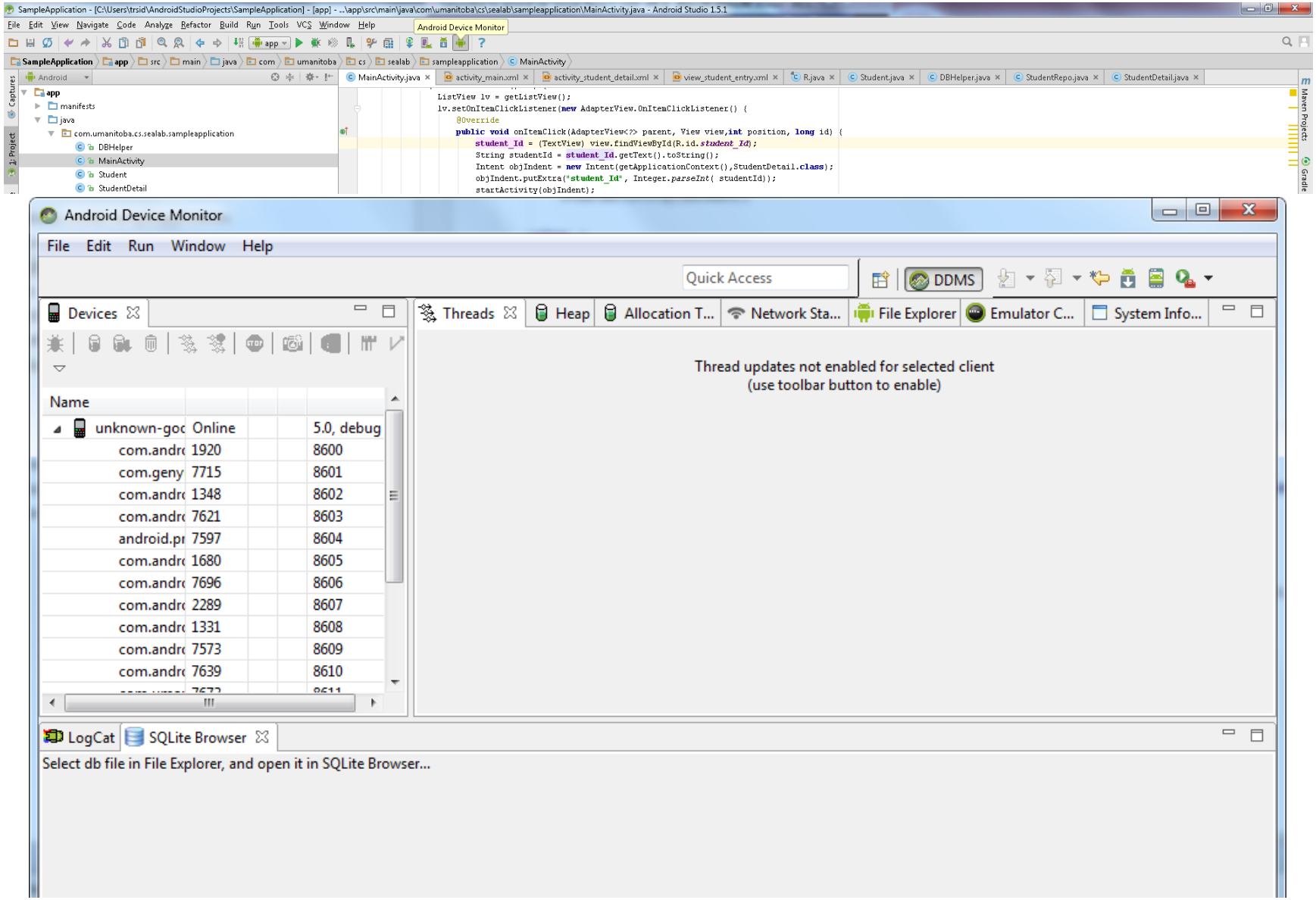


**Fig 50. Open File Explorer tab, navigate to root → data → data → <applicationName>(sealab.cs.umanitoba.com.studentrecordkeeper in our case) → databases and single left click on the database file (crud.db)**

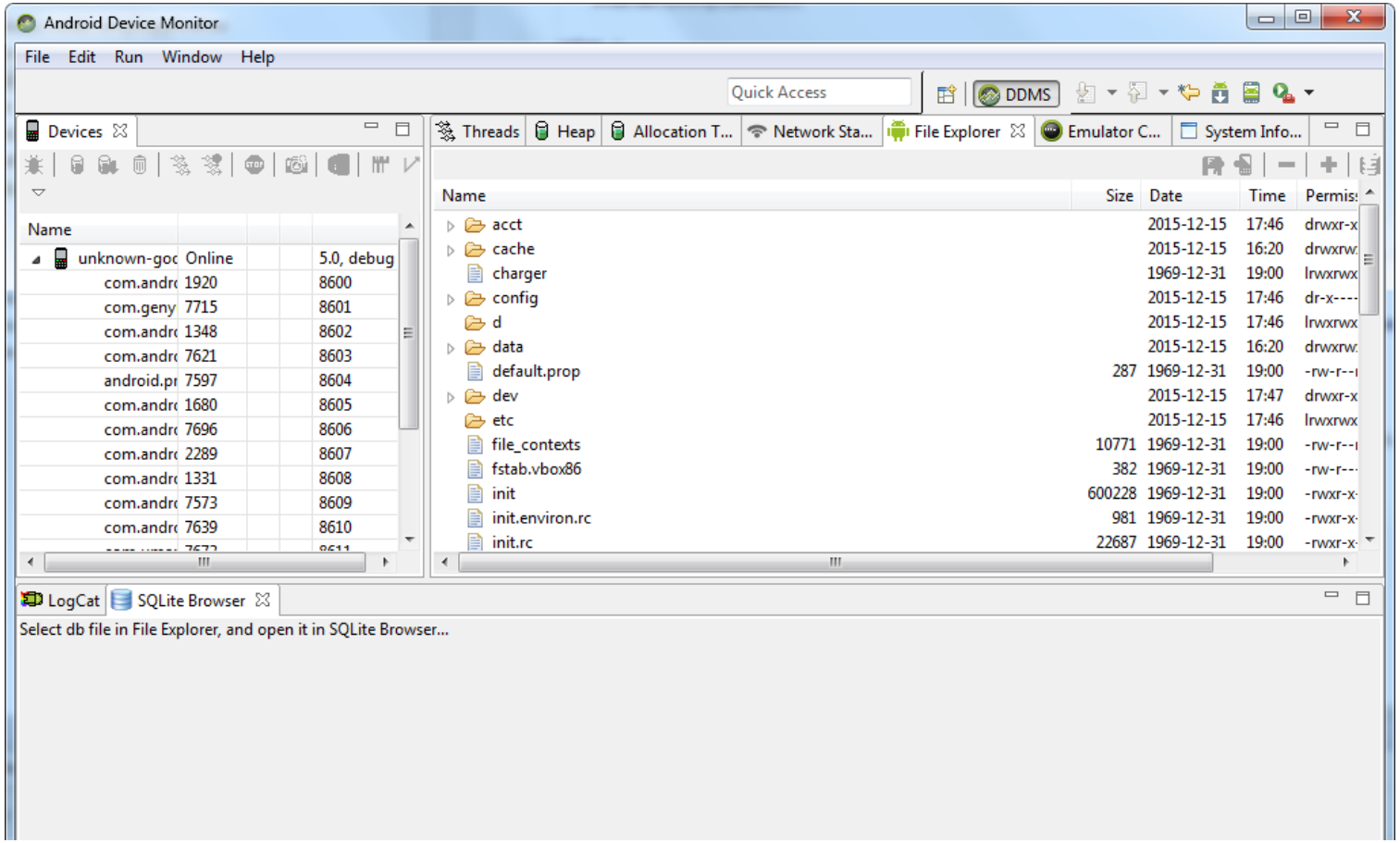


**Fig 52. When you click on the database file, SQLite Browser plugin button will enable on the top right section of the Device Monitor. Click on the button to show the content of the database in the SQLite Browser tab in the bottom section of the window.**

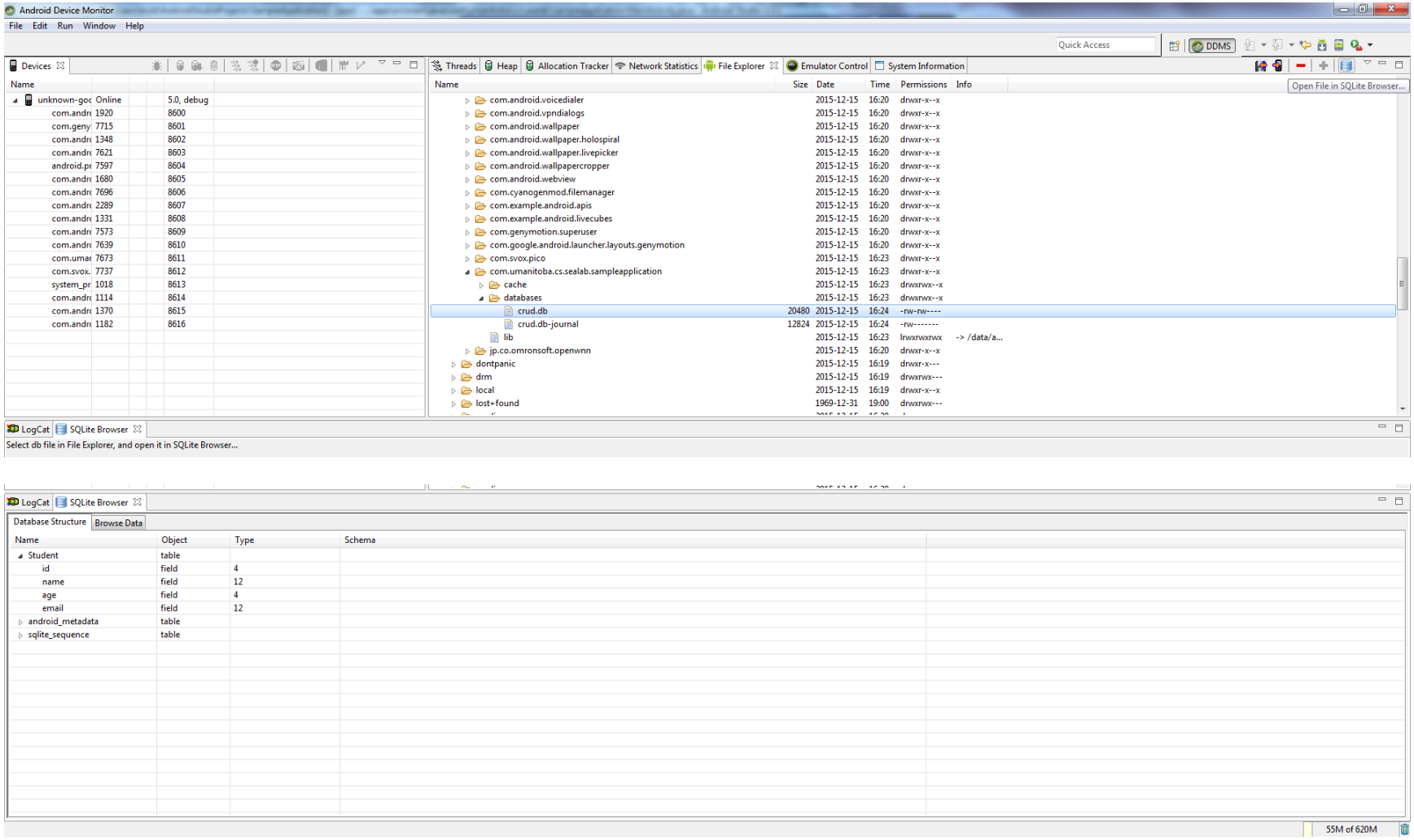
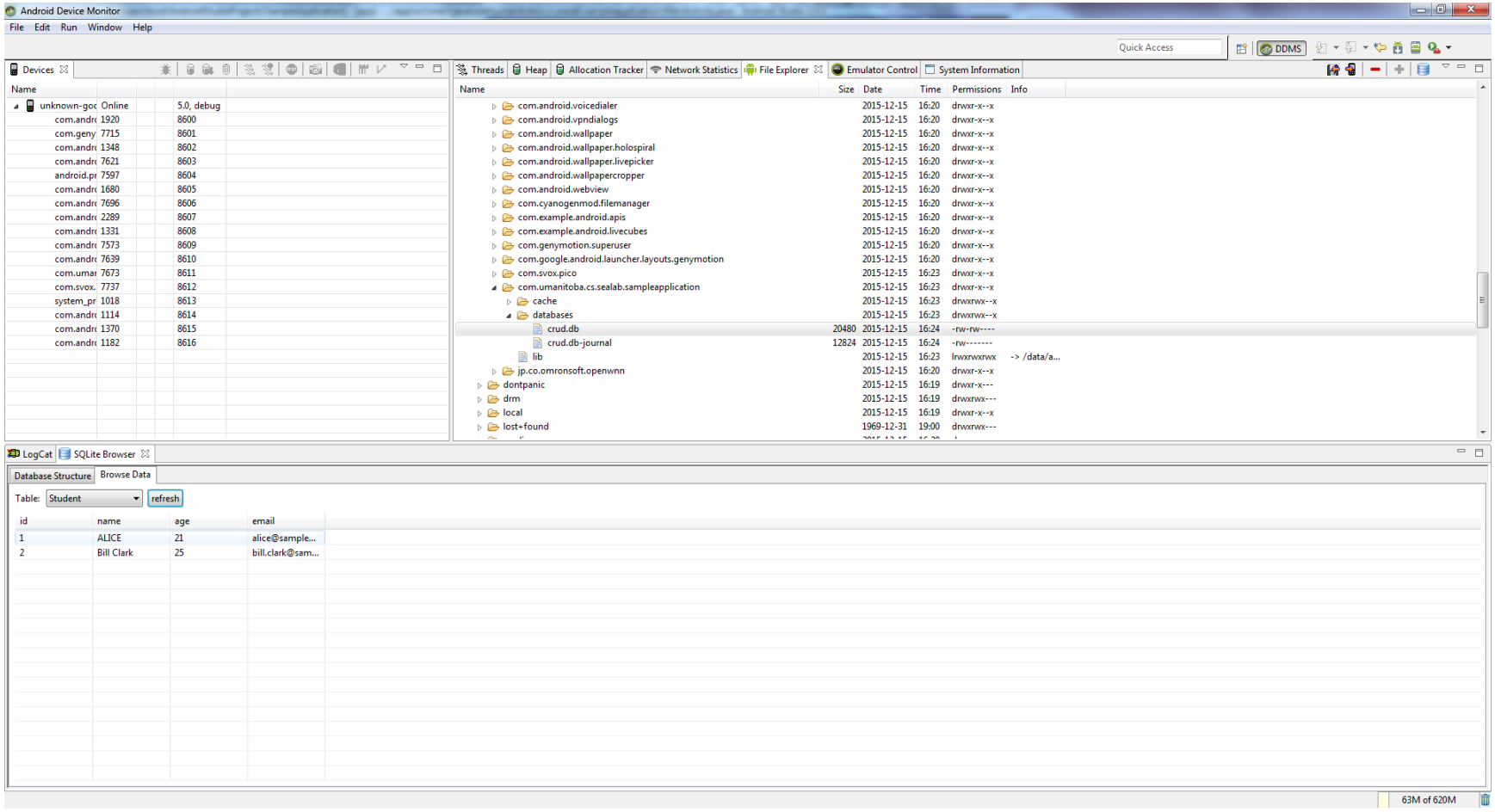
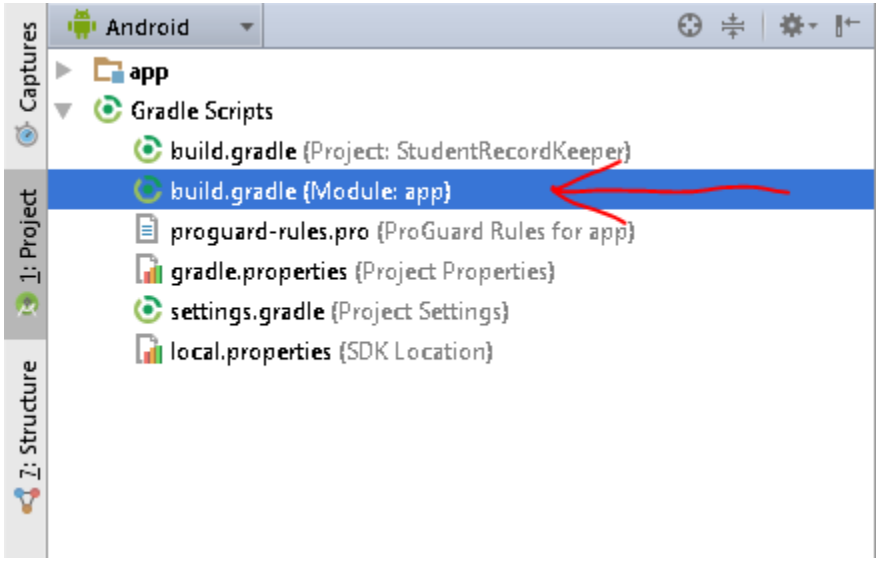


**Fig 53. Click on the Browse Data tab and see the data you inserted in the application.**



Section 5

# Writing Automated UI Tests using Robotium®

1. Open ***build.gradle (Module: app)*** under **Gradle Scripts** (Project View)
   a. 

2. <u>If you already have a dependency section at root level of this file,</u>
   a. <u>append the following line below existing dependencies</u>
      i. androidTestCompile 'com.jayway.android.robotium:robotium-solo:5.5.4'
   b. else, append the following section “dependencies” to the end of the file
      i.
```
dependencies {
      androidTestCompile 'com.jayway.android.robotium:robotium-solo:5.5.4'
}
```

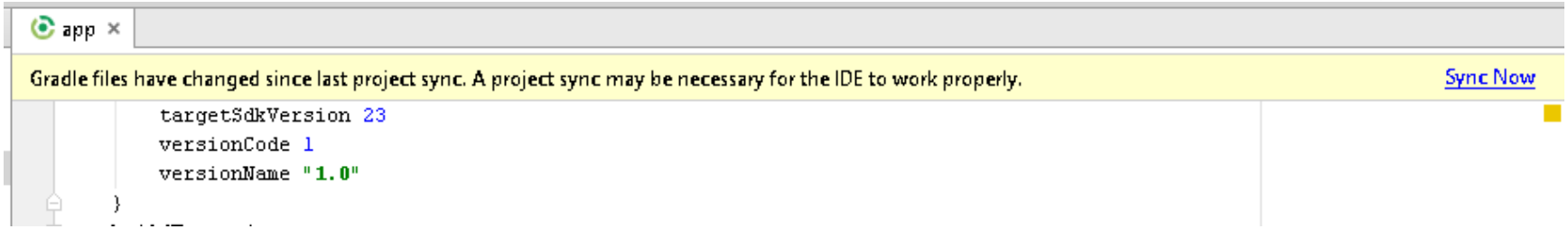
app

```
//filename: build.gradle
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "sealab.cs.umanitoba.com.studentrecordkeeper"
        minSdkVersion 14
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
    androidTestCompile 'com.jayway.android.robotium:robotium-solo:5.5.4'
}
```

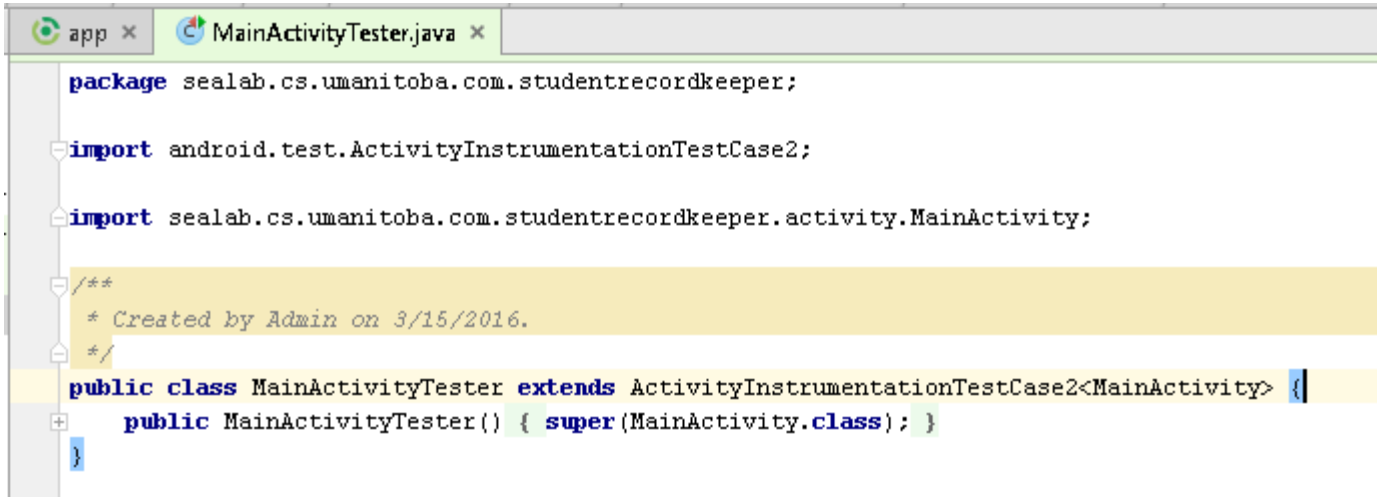
*In the above step you specified that while building the 'androidTest' module of your app, above library should be used.*

Note: This is the latest version of the library “5.5.4” used at the time of documenting this tutorial(3/14/2015). Please confirm the latest version from GITHUB and include that.

3. Since the build file is updated, Gradle will ask to sync the project according to the latest config. changes




4. Press on Sync Now button. Your repository now contain a reference to Robotium library.
5. Now go to your “androidTest” package under app→java →com…..something (androidTest). (If you don’t see this, try changing the Build Variant)
6. Add a new java class to test an existing activity. Let’s call it MainActivityTester, and extend it with “***ActivityInstrumentationTestCase2<MainActivity>***”, where MainActivity is the name of the activity you want to test. Also, add a constructor for the class as shown below.




7. Now override the setUp and tearDown method of the extended class. These are the methods which are called when a test case execution is started and finished respectively. Also, declare a variable of object Solo

in Robotium Library which is to be used as reference when writing code for actions. Initialize solo object in setUp and call solo.finishOpenedActivities() in teardown(). If you don't call this method in tearDown(), at the end of a test case your application will keep running the same activity and the system will consider your tests as failed.

app × MainActivityTester.java ×

```
package sealab.cs.umanitoba.com.studentrecordkeeper;

import android.test.ActivityInstrumentationTestCase2;
import com.robotium.solo.Solo;
import sealab.cs.umanitoba.com.studentrecordkeeper.activity.MainActivity;

public class MainActivityTester extends ActivityInstrumentationTestCase2<MainActivity> {

    private Solo solo;

    public MainActivityTester() {
        super(MainActivity.class);
    }
    @Override
    public void setUp() throws Exception {
        solo = new Solo(getInstrumentation());
        getActivity();
    }
    @Override
    public void tearDown() throws Exception {
        solo.finishOpenedActivities();
    }
}
```

8. Now start adding your test method, taking care of following.
    a. Declare your test functions as public, as private methods are not executed when running a test suite as they are not accessible.
    b. Prepend "test" before your method name, also include "throws Exception".

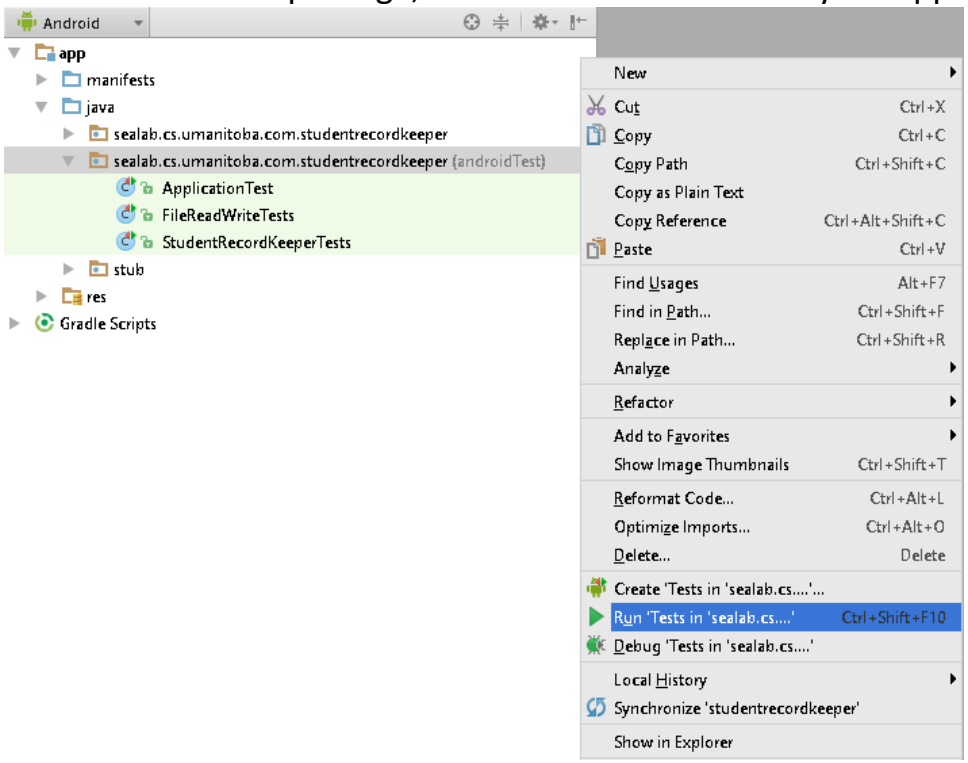
c. Public test methods are executed by the order of their name. So if you have 4 test methods as testA, testB, testC and testD, their order of execution would be testA→testB→testC→testD
d. Same goes with the classes. If you have more than one class in your test suite. Their order of execution will be by their names. For instance, let's suppose we have another class named "FileWritingTester" with "MainActivityTester" in our test suite. When we run our test suite, tests from "FileWritingTester" will run before tests from "MainActivityTester".

**9.** Adding a new test method to test our AddStudent functionality. Pasted below is the lines of code of the method and used variables only. **(Comments are provided above each line of code in the repository available on [GITHUB](#))**

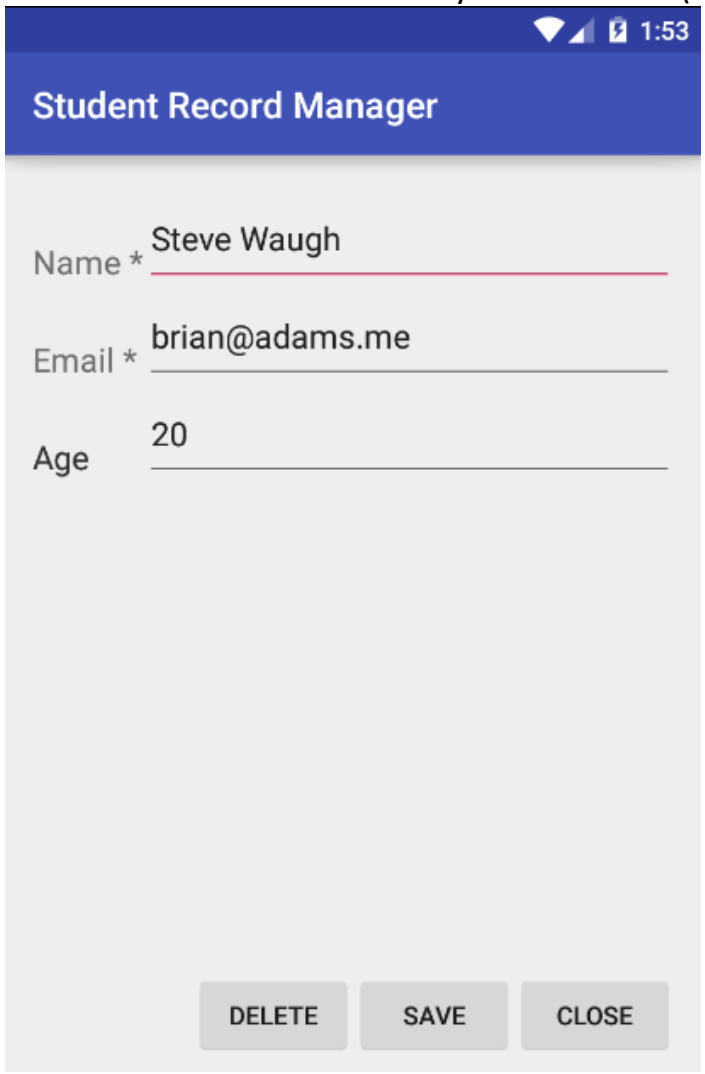
```
private static final String NAME1 = "Brian Adams";
private static final String EMAIL1 = "brian@adams.me";
private static final String AGE1 = "20";
private static final String NAME2 = "David Flemming";
private static final String EMAIL2 = "david@flemming.me";
private static final String AGE2 = "20";
private static final String EDITTED_NAME1 = "Steve Waugh";
public void testAAddStudent() throws Exception {
    solo.unlockScreen();
    solo.clickOnView(solo.getView(R.id.btnAdd));
    solo.assertCurrentActivity("Expected StudentDetail Activity", StudentDetail.class);
    solo.sleep(500);
    solo.enterText(0, NAME1);
    solo.sleep(500);
    solo.enterText(1, EMAIL1);
    solo.sleep(500);
    solo.clearEditText(2);
    solo.enterText(2, AGE1);
    solo.sleep(500);
    solo.clickOnView(solo.getView(R.id.btnSave));
    solo.sleep(500);
    solo.clickOnView(solo.getView(R.id.btnClose));
    solo.assertCurrentActivity("Expected MainActivity Activity", MainActivity.class);
    solo.clickOnView(solo.getView(R.id.btnAdd));
    solo.assertCurrentActivity("Expected StudentDetail Activity", StudentDetail.class);
    solo.enterText(0, NAME2);
    solo.enterText(1, EMAIL2);
    solo.clearEditText(2);
    solo.enterText(2, AGE2);
```

```
    solo.clickOnView(solo.getView(R.id.btnSave));
    solo.clickOnView(solo.getView(R.id.btnClose));
    solo.takeScreenshot();
    boolean studentsFound = solo.searchText(NAME1) && solo.searchText(NAME2);
    assertTrue("Student 1 and Student 2 are found", studentsFound);
}
```

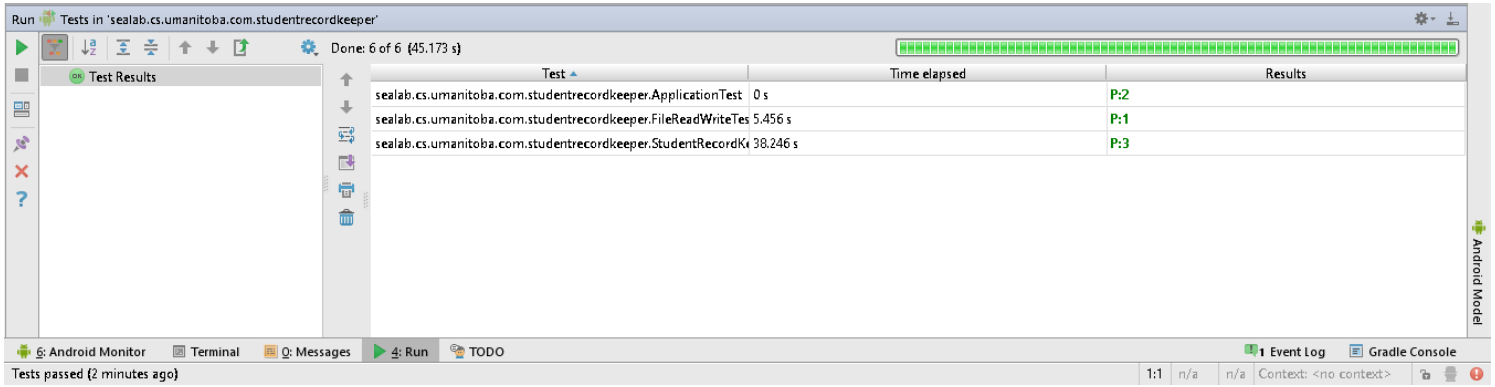
10. Looking at the above code you can see that the methods like unlockScreen, clickOnView, enterText are telling the system to interact with the UI. Robotium executes the above methods, interacts with the UI and assert the output with the methods like assertCurrentActivity, assertTrue.
    a. Description(reference) of all the methods used above can be seen at https://robotium.googlecode.com/svn/doc/com/robotium/solo/Solo.html
11. After you are done writing your scenarios you will have a bunch of test cases (in one or more files under androidTest.). Right click on androidTest package, and select “Run Tests in <your app>” .



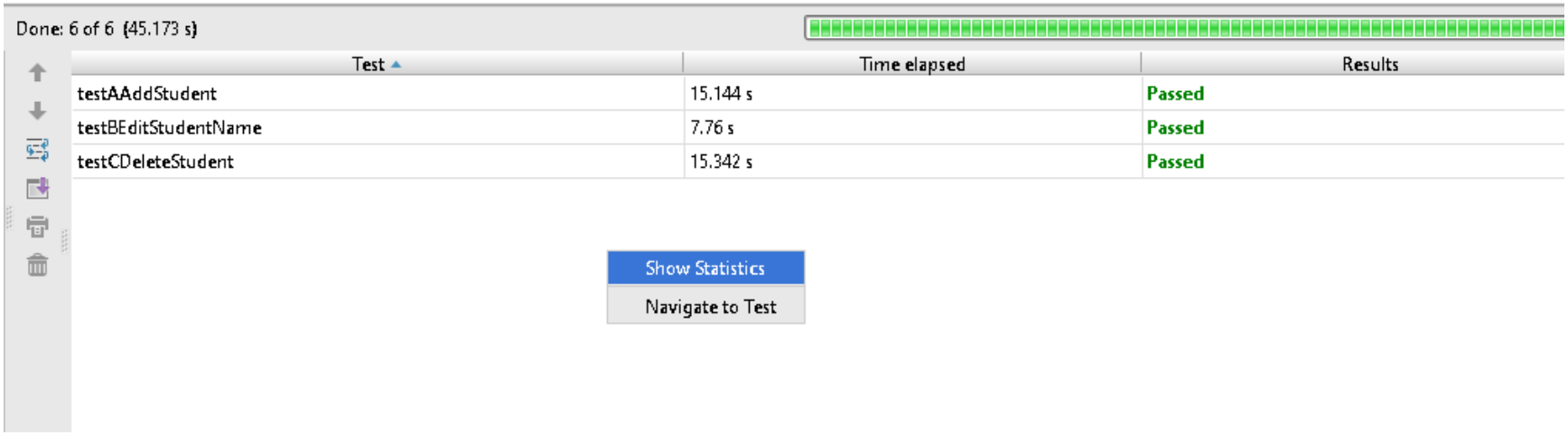
12. Select your emulator (or a physical android device connected to your machine – usb-debugging enabled) and you will see automated GUI test execution on your emulator (or device).



13. At the end of execution, results of the test(s) will be shown in the Test Result window, which if not visible can be accessed by clicking on the ***Run*** tab in the bottom.




14. You can see the statistics of the tests executed by clicking on the menu items in the right window above. To go back, right click in the same window and select Show Statistics.



**References:**


1) http://www.tutorialspoint.com/android/android_acitivities.htm
2) Android Development by John Braico, 2015.
3) http://developer.android.com/guide/topics/ui/menus.html
4) http://developer.android.com/training/basics/fragments/index.html
5) http://code.tutsplus.com/tutorials/android-user-interface-design-linear-layouts--mobile-4047