Lab Report for EI447 - Mobile Internet

Lab 1: Introduction to Android Programming

Yiwen Song

2020-4-5

Preliminaries

Development Environment

- Operating system: Windows 10, function version 1909.
- IDE: Android Studio 3.6.2 (latest version).
- Build: Gradle 5.6.4.
- JDK: Java 1.8.0_212 for AS Runtime Environment. Java > 8 not suggested for the lab project.
- Android Simulator: Pixel 2 API Q (1080*1920; Android 10.0/Pure Android; x86).
- Android Tester: OXF-AN00/Honor V30 (1080*2400; Android 10.0/MagicUI 3.0.1; ARM).

Software Information

- Support Android > 5.0 devices.
- Open-source under GNU/GPL License v3.0. Any modifications/redistributions of the project software must be restricted by the GPL license. For more information, check the *open source* of the program.
- Copyright information: During the development of the project, the following external references have been used:
  - Template for the report is *eisvogel*.

Hello World

The “Hello World” Program can be completed by two ways: (1) use the xml file to demonstrate the “Hello World” string; or (2) use a java program to generate a “Hello World” string and demonstrate it on the app.

XML Realization

It is very easy to realize a “Hello World” app by modifying the main xml file. After creating the android project in Android Studio, we just need to add the following sentences in the *main_activity.xml* file.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

And we can see the “Hello Android” string at the center of the screen, just as follows:

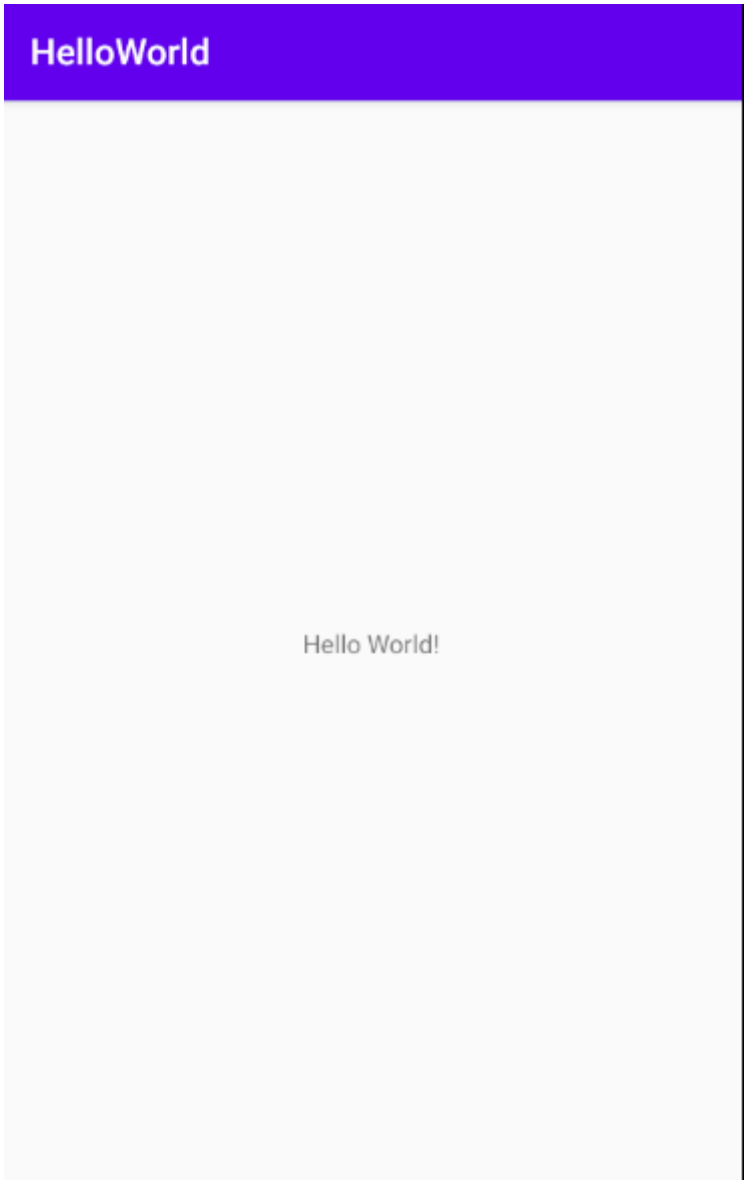

Figure 1: Hello World Demo 1

The xml realization is commented in the lab code. The java realization is demonstrated instead.

Java Realization

We can also realize the “Hello World” program by writing a Java program. By creating a *TextView* object and setting the text to be *Hello World!*, we have a “Hello World” string. Then by putting the code in the *onCreate* function we can display the string when the program starts.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView tv = new TextView(this);
    tv.setText("Hello World!");
    tv.setTextSize(20);
    setContentView(R.layout.activity_main);
    setContentView(tv);
}
```

The result is shown as follows.

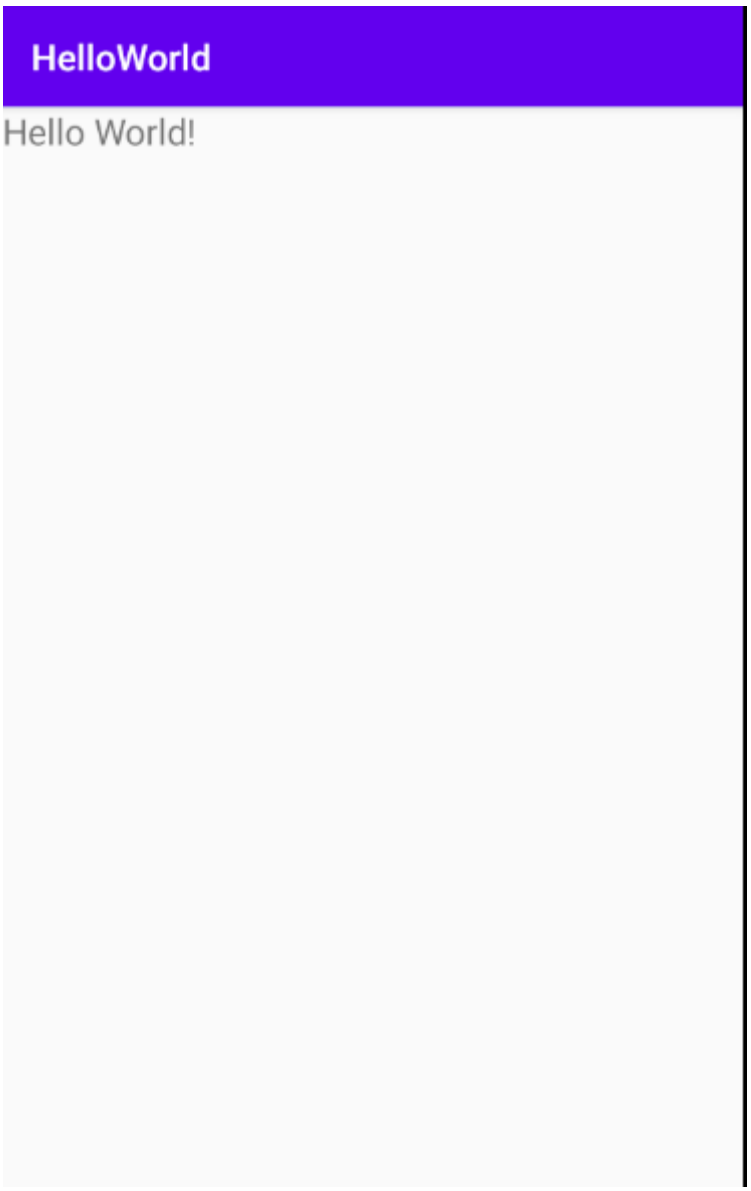

Figure 2: Hello World Demo 2

Paint Board

Based on the given example code, we can complete a more advanced paint board app with more functions shown as follows.

- Change the color of lines;
- Change the width of lines;

- Clear all;
- Eraser;
- Save the painting.

We will demonstrate in the following sections how each functions are realized.

Front Panel

We used a menu toolbar as the main component for the front panel. An xml file supports this toolbar. We put the different items for selection in the menu toolbar, just as shown below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/back" android:title="@string/back" />
    <item android:title="@string/color">
        <menu>
            <group android:checkableBehavior="single">
                <item android:id="@+id/red" android:title="@string/color_red"/>
                <item android:id="@+id/blue" android:title="@string/color_blue"/>
                <item android:id="@+id/green" android:title="@string/color_green"/>
            </group>
        </menu>
    </item>
    <item android:title="@string/width">
        <menu>
            <group android:checkableBehavior="single">
                <item android:id="@+id/width_1" android:title="@string/width_1"/>
                <item android:id="@+id/width_2" android:title="@string/width_2"/>
                <item android:id="@+id/width_3" android:title="@string/width_3"/>
            </group>
        </menu>
    </item>
    <item android:id="@+id/clearAll" android:title="@string/clearAll"/>
    <item android:id="@+id/eraser" android:title="@string/eraser"/>
    <item android:id="@+id/save" android:title="@string/save"/>

</menu>
```

Then we can check that the front panel appears like the figure shown below. Finally we just need to link each button in our java main program to realize their functions, which will be talked about that later.

Figure 3: Front Panel

Basic Paint Board

The basic paint board program is similar to the example code. We just state some changes here. Firstly, we set the width and height of the paint board to match the screen width and height, using the code below.

```
screen_width = context.getResources().getDisplayMetrics().widthPixels;
screen_height = context.getResources().getDisplayMetrics().heightPixels
    ;
bitmap = Bitmap.createBitmap(screen_width, screen_height, Bitmap.Config
    .ARGB_8888);
```

We use a *Path* object to record each stroke path, and after the program detects the *ACTION_UP* action, we actually move the drawn path onto the bitmap. This is a more robust way so that *the paint board supports multi-touch mode*.

```
@SuppressLint("ClickableViewAccessibility")
@Override
public boolean onTouchEvent(MotionEvent event) {
    float currentX = event.getX(), currentY = event.getY();
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            path.moveTo(currentX, currentY);
            x = currentX; y = currentY;
            break;
        case MotionEvent.ACTION_MOVE:
            float dx = Math.abs(currentX-x), dy = Math.abs(currentY-y);
```

```
12                 if (dx+dy >= 3) {
13                     path.lineTo(currentX, currentY);
14                 }
15                 break;
16             case MotionEvent.ACTION_UP:
17                 canvas.drawPath(path, p);
18                 path.reset();
19                 break;
20         }
21         invalidate();
22         return true;
23     }
```

Line Color

The function *Paint.setColor()* controls the color for the painted lines. For example, if we need to set color of the lines to be drawn to be RED (0xFFFF0000), we just need to write the code:

```
1  paint = new Paint(Paint.DITHER_FLAG);
2  // SEVERAL CODES
3  paint.setColor(Color.RED);
4  // OTHER CODES
```

These codes dynamically calls the libraries in *android.graphics*. Then, as we have created the Paint Board object, named DrawView (dv), with a public Paint object *p* inside, we just need to write the following codes in the main program to realize the function for setting colors.

```
1  switch (item.getItemId()) {
2      case R.id.red:
3          dv.p.setColor(Color.RED);
4          break;
5      case R.id.blue:
6          dv.p.setColor(Color.BLUE);
7          break;
8      case R.id.green:
9          dv.p.setColor(Color.GREEN);
10         break;
11 }
```

Line Width

Similar as line color, the line width is controlled by the *Paint.setStrokeWidth()* function. Therefore, we just need to write the following codes in the main program, together with the *switch (item.getItemId())*:

```
1  case R.id.width_1:
```

```
        dv.p.setStrokeWidth(5);
        break;
    case R.id.width_2:
        dv.p.setStrokeWidth(10);
        break;
    case R.id.width_3:
        dv.p.setStrokeWidth(15);
        break;
```

Clear All

In order to realize the “clear all” function, we just need to create another bitmap, and let the bitmap cover the original one. The detailed code is shown below.

```
public void clearAll() {
    bitmap = null;
    bitmap = Bitmap.createBitmap(screen_width, screen_height, Bitmap.
        Config.ARGB_8888);
    canvas = new Canvas();
    canvas.setBitmap(bitmap);
    invalidate();
}
```

Eraser

The android *Paint* object provides the function for an eraser using the code shown below:

```
@SuppressLint("ResourceAsColor")
public void eraser() {
    p.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.CLEAR));
    p.setColor(R.color.defaultBackground);
    p.setStrokeWidth(50);
}
```

By setting the *Xfermode* being CLEAR and setting the color of lines to be background color, we can surely set the current lines to act as an “eraser”. We set the stroke width to be 50 to make the eraser be more convenient.

Save

In order to save the painted picture into the internal or external storage of the android device, we need to request permissions from the device firstly. We have not realized the function in our program to check if the permission has been provided by the user. Therefore, if the permission authority is denied,

the save function will not perform normally. To request permission, we added the following codes in the manifest xml file:

```
<uses-permission android:name="android.permission.
    MOUNT_UNMOUNT_FILESYSTEMS"
     tools:ignore="ProtectedPermissions" />
<uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE"/>
```

Then, we need to access the storage to confirm the place we are going to save the file. This is implemented by a *getFilePath()* function, which automatically searches if there is an external storage (e.g., SD card), and returns path which locates the app info at the external storage if there exists. Otherwise, it returns the same directory at the internal storage.

```
public static String getFilePath(Context context, String fileSize,
    String fileName, int j) {
     String directoryPath = "";
     if (j == 0) {
         if (Environment.MEDIA_MOUNTED.equals(Environment.
             getExternalStorageState())) {
              directoryPath = Objects.requireNonNull(context.
                 getExternalFilesDir(fileSize)).getAbsolutePath()+
                      File.separator+fileName+".png";
         }
         else {
              directoryPath = context.getFilesDir().getAbsolutePath()+
                 File.separator+fileName+".png";
         }
     }
     else {
         directoryPath = Environment.getExternalStorageDirectory()+File.
             separator+
                  Environment.DIRECTORY_DCIM+File.separator+fileName+".
                     png";
     }
     return directoryPath;
}
```

Finally, we can save the bitmap as a png file in the android device. This includes a call for a *saveBitmap()* function, which creates a new file, compresses the bitmap to png format, and saves tha png at the new file.

```
public void save(int i, int j) {
     try {
         saveBitmap("DrawingPicture", "DrawingPicture"+1, j);
     } catch (IOException e) {
         e.printStackTrace();
     }
}
```

```
8
9  public void saveBitmap(String fileSize, String fileName, int j) throws
       IOException {
10     String directoryPath;
11     directoryPath = getFilePath(getContext(), fileSize, fileName, j);
12     File file = new File(directoryPath);
13     try {
14         FileOutputStream fileOS = new FileOutputStream(file);
15         bitmap.compress(Bitmap.CompressFormat.PNG, 100, fileOS);
16         fileOS.flush();
17         fileOS.close();
18         Toast.makeText(getContext(), "Saved successfully to"+
               directoryPath, Toast.LENGTH_SHORT).show();
19     } catch (Exception e) {
20         Toast.makeText(getContext(), e.toString(), Toast.LENGTH_LONG).
               show();
21     }
22     Intent intent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
23     Uri uri = Uri.fromFile(file);
24     intent.setData(uri);
25     getContext().sendBroadcast(intent);
26 }
```

Demonstration

The following figure demonstrates all the functions for the paint board app except “clear all” (which can’t be demonstrated by figures). We can see in the figure that all functions are perfectly performed.

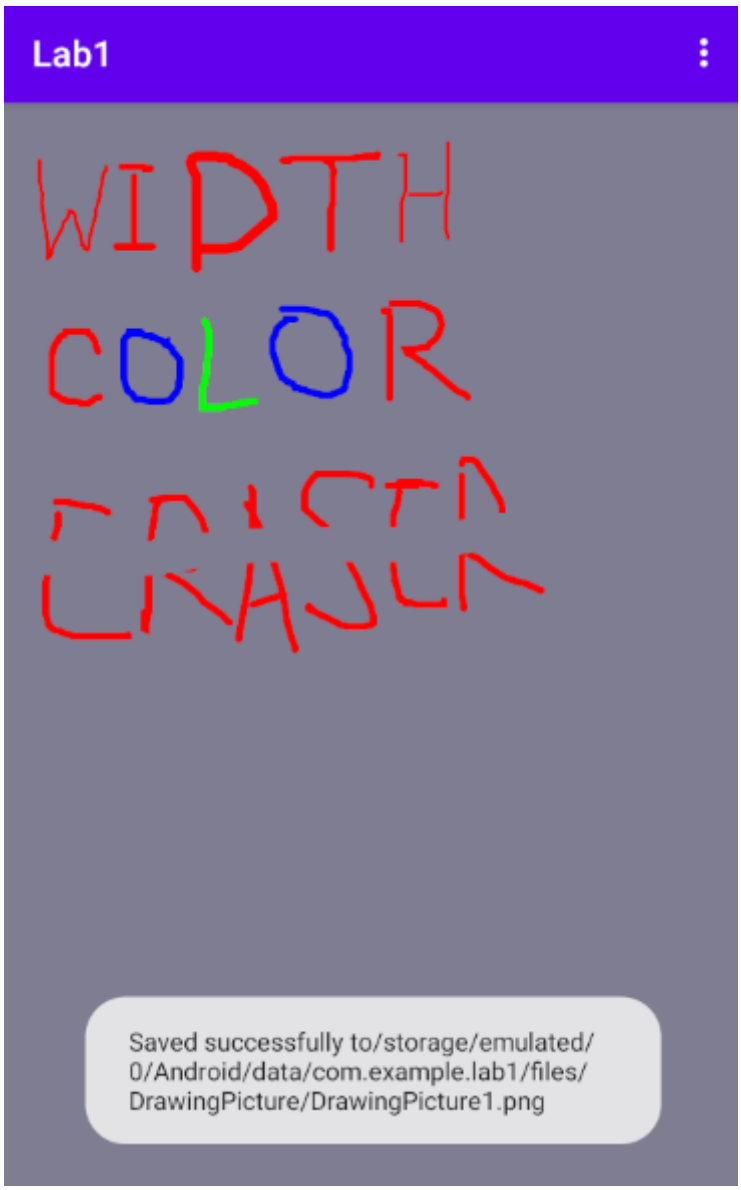

Figure 4: Demo for the Paint Board App