LE/EECS 4443: Mobile User Interfaces (LAB)

Week 1: Android App Anatomy

Shogo Toyonaga[1]

[1]Lassonde School of Engineering
York University

January 5, 2025

Table of Contents

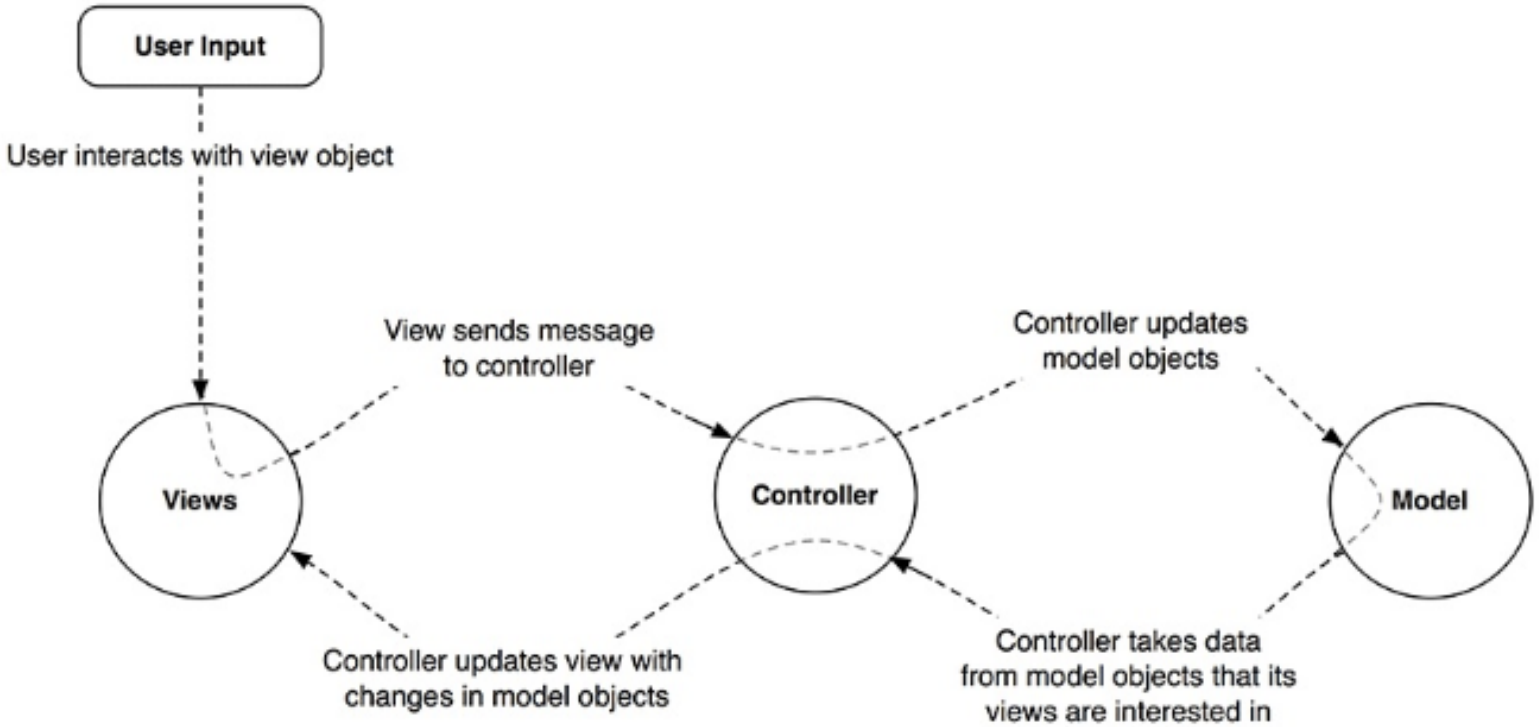
Learning Objectives

1. Apply the Model-View-Controller (MVC) software design pattern.
2. Understand Android Components (e.g., AndroidManifest)
3. Understand Layouts (View vs ViewGroup)
4. Mastering Event Listeners (e.g., onClick)

Model-View Controller (MVC)

- **Model:** Holds and manages data ("business logic").
- **View:** For simplicity, you can think of a View as anything you can see on the screen (e.g., Widgets). The sole purpose is to respond to user input.
- **Controller:** Used to connect the view and model together. The sole purpose is to respond to various events triggered by the View object and manage the flow of data to and from the Model(s).

Model-View Controller (MVC)



Model-View Controller (MVC)

1. **Advantages:**
   - Separation of Responsibilities
   - Components (e.g., Model) can be shared among different View(s) or Controllers
   - Easily integrates into existing frameworks as it is a commonly accepted software design pattern
2. **Disadvantages:**
   - Can be unnecessarily complex for simple applications
   - Different alternatives and definitions of MVC exist which can cause confusion

Case Study: Model-View Controller (MVC)

Example

Bob creates an Android Application that simulates a scantron quiz about himself.
Given the following files, identify the model, view, and controller.

1. Question.java
2. MainActivity.java
3. activity_main.xml

Code Demo: DemoGeoQuiz.zip

Android Components

Some of the most important components that you will learn about include:

1. **Activities**
2. **XML Layouts**
3. **String Constants**
4. **AndroidManifest.xml**
5. **Build Script**

Activities

- The **main activity** is defined as the first screen to appear when the user launches your application.
- Responsible for managing user interactions with a screen of information.
- Most apps leverage multiple screens comprising of multiple activities.

Note: Activities are located in app/java

Basic Activity

```
1  package com.example.demosandbox;
2  ...Imports...
3
4  public class MainActivity extends AppCompatActivity {
5
6      @Override
7      protected void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          EdgeToEdge.enable(this);
10         setContentView(R.layout.activity_main);
11         // TODO: Connect Widgets to Controller (with IDs)
12         ...
13         // TODO: Define Behaviours (Methods)
14         ...
15         // TODO: Set the Action Event Listeners
16         ...
17         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
               insets) -> {
18             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.
                   systemBars());
19             v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                   systemBars.bottom);
20             return insets;
21         });
22     }
23 }
```

XML (Layout)

- Given the layout, widgets are drawn on the screen in a pre-defined order
- Android supports a wealth of pre-defined and well-documented layouts and widgets that you can customize through attribute definitions.

Note: XML Layouts are located in app/res/layout

Basic Layout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas
       .android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintEnd_toEndOf="parent"
16         app:layout_constraintStart_toStartOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

String Resources (Constants)

- String resources are located in app/res/values, supporting all of the escape characters.
- We use string resources to define the name of our activities, button labels, text, etc.,

Main Benefit

By using string resources rather than defining hard-coded string constants, it allows us to re-use labels and change values that propagate across Activities efficiently.

Basic String Resource

```
1 <resources>
2     <string name="app_name">DemoSandbox</string>
3 </resources>
```

Note

You use the, “name” tag to reference the string in your XML or Activity. It acts similarly to an ID.

Android Manifest

Definition

The Android Manifest is an XML file containing metadata which describes your application to the Android OS. Every Activity in an application must be declared in the Manifest so that the OS can access it.

Note: The Manifesrt is located in app/manifests/AndroidManifest.xml

Basic AndroidManifest

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4
5      <application
6          android:allowBackup="true"
7          android:dataExtractionRules="@xml/data_extraction_rules"
8          android:fullBackupContent="@xml/backup_rules"
9          android:icon="@mipmap/ic_launcher"
10         android:label="@string/app_name"
11         android:roundIcon="@mipmap/ic_launcher_round"
12         android:supportsRtl="true"
13         android:theme="@style/Theme.DemoSandbox"
14         tools:targetApi="31">
15         <activity
16             android:name=".MainActivity"
17             android:exported="true">
18             <intent-filter>
19                 <action android:name="android.intent.action.MAIN" />
20
21                 <category android:name="android.intent.category.LAUNCHER" />
22             </intent-filter>
23         </activity>
24     </application>
25
26 </manifest>
```

Build Script (build.gradle)

- **build.gradle**, located in Gradle Scripts/build.gradle is used to store vital information about your application to build and deploy it across multiple Android systems.
  1. **minSdk:** A hard floor for which the OS should refuse to install the app.
  2. **targetSdk:** The API level that your application is designed to run on.
  3. **compileSdk:** The API level to use when building your code. The best choice is to use the latest API level.
  4. **Dependencies:** The external binaries or other library modules that are necessary to make your application work properly.

Basic Build.Gradle

```
plugins {
    alias(libs.plugins.android.application)
}
android {
    namespace 'com.example.demosandbox'
    compileSdk 34
    defaultConfig {
        applicationId "com.example.demosandbox"
        minSdk 24
        targetSdk 34
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'
                ), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_11
        targetCompatibility JavaVersion.VERSION_11
    }
}
dependencies {....}
```

Build.Gradle

Note

It is generally a good practice to add code into the Controller to manage **compatibility issues** between your build version and older API levels that don't support certain methods.

Introduction

1. **View:** An object (screen, widget, pop-up, etc .,) that draws someing on the screen that the user can interact with
   - EditText
   - Button
   - TextView
   - ScrollView
2. **ViewGroup:** A widget that contains and arranges other widgets
   - LinearLayout
   - RelativeLayout
   - TableLayout

Layout Definition

Warning

In most cases, you should only declare UI elements in the XML file. In this way, your UI definitions are separate from your application code (Controller).

Note: Views and ViewGroups have many different types of attributes and associated values/behaviours. Play around with Android Studio and read the documentation to get hands-on practice!

Introduction

- **Definition:** An event listener is an interface that defines methods used to support user interactions with a View (e.g., View.OnClickListener)
- **Implementation:** An event listener can be instantiated and connected to a View using one of the following strategies:
  1. Implementing the proper interface and overriding the default method(s)
  2. Defining a method and connecting it as the event via XML Layout
  3. Implementing the interface and connecting it to the View by defining an Anonymous Inner Class

PauseChamp

Comment

Do not panic!
This may seem quite overwhelming, however, take a quick look through my code demos. Everything will be fine!

Method #1: Implement the EventListener Interface

```
1  public class QuizActivity extends AppCompatActivity implements View.
       OnClickListener {
2
3      // Step 1: Generate Variables for Inflated View Objects
4      private Button mTrueButton;
5      private Button mFalseButton;
6      private Button mNextButton;
7      private Button mCheatButton;
8      private Button mPrevButton;
9      private TextView mQuestionTextView;
10     // Application Logic
11     private int mCurrentIndex = 0;
12     private int question;
13     private Question[] mQuestionBank;
14     private static final String KEY_INDEX = "index";
15     private static final int REQUEST_CODE_CHEAT = 0;
16     private boolean mIsCheater;
17     ...
```

Method #1: Implement the EventListener Interface

```
1   private void initialize() {
2       mTrueButton = findViewById(R.id.true_button);
3       mFalseButton = findViewById(R.id.false_button);
4       mNextButton = findViewById(R.id.next_button);
5       mCheatButton = findViewById(R.id.cheat_button);
6       mPrevButton = findViewById(R.id.prev_button);
7       mQuestionTextView = findViewById(R.id.question_text_view);
8       mQuestionBank = new Question[]{
9               new Question(R.string.question_australia, true),
10              new Question(R.string.question_oceans, true),
11              new Question(R.string.question_mideast, false),
12              new Question(R.string.question_africa, false),
13              new Question(R.string.question_americas, true),
14              new Question(R.string.question_asia, true)
15      };
16      question = mQuestionBank[mCurrentIndex].getTextResId();
17  }
```

Method #1: Implement the EventListener Interface

```
1
2   @Override
3     public void onClick(View v) {
4         if (v == mTrueButton) {
5             checkAnswer(true);
6         } else if (v == mFalseButton) {
7             checkAnswer(false);
8         } else if (v == mNextButton || v == mQuestionTextView) {
9             mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
10            mIsCheater = false;
11            updateQuestion();
12        } else if (v == mPrevButton) {
13            int new_index = (mCurrentIndex - 1) % mQuestionBank.length;
14            if (new_index < 0) {
15                Toast.makeText(getApplicationContext(), "We have reached the
                      last question.", Toast.LENGTH_SHORT).show();
16            } else {
17                mCurrentIndex = new_index;
18                updateQuestion();
19            }
20        } else if (v == mCheatButton) {
21            boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
22            Intent intent = CheatActivity.newIntent(QuizActivity.this,
                    answerIsTrue);
23            startActivityForResult(intent, REQUEST_CODE_CHEAT);
24        }
25    }
```

Method #1: Implement the EventListener Interface

```
1  @Override
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          EdgeToEdge.enable(this);
5          // Generate Activity UI
6          setContentView(R.layout.activity_main);
7          // Step 2: Get References to the Inflated View Objects
8          initialize();
9          // Step 3: Connect the Listener to the Views
10         mTrueButton.setOnClickListener(this);
11         mFalseButton.setOnClickListener(this);
12         mNextButton.setOnClickListener(this);
13         mQuestionTextView.setOnClickListener(this);
14         mPrevButton.setOnClickListener(this);
15         mCheatButton.setOnClickListener(this);
16         // Step 4: Draw Question to the TextView
17         updateQuestion();
18         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
               insets) -> {
19             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.
                   systemBars());
20             v.setPadding(systemBars.left, systemBars.top, systemBars.right,
                   systemBars.bottom);
21             return insets;
22         });
23     }
```

Method #3: Using Anonymous Inner Classes

```
1  public class MainActivity extends AppCompatActivity {
2      LinearLayout parent;
3      TextView raw_score;
4      SeekBar seeker;
5      Button party_rock;
6      int[] colors = {Color.WHITE, Color.LTGRAY, Color.GRAY, Color.BLUE, Color.
           BLACK, Color.YELLOW, Color.CYAN, Color.GREEN, Color.RED, Color.MAGENTA
           };
```

Method #3: Using Anonymous Inner Classes

```
1     private void initialize() {
2         parent = findViewById(R.id.main);
3         raw_score = findViewById(R.id.updater);
4         seeker = findViewById(R.id.seek);
5         party_rock = findViewById(R.id.randomizer);
6     }
```

Method #3: Using Anonymous Inner Classes

```
1  @Override
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          EdgeToEdge.enable(this);
5          setContentView(R.layout.activity_main);
6          initialize();
7          seeker.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener()
               {
8              @SuppressLint("DefaultLocale")
9              @Override
10             public void onProgressChanged(SeekBar seekBar, int progress, boolean
                   fromUser) {
11                 raw_score.setText(String.format("%d", progress));
12                 parent.setBackgroundColor(colors[progress - 1]);
13             }
14
15             @Override
16             public void onStartTrackingTouch(SeekBar seekBar) {
17
18             }
19
20             @Override
21             public void onStopTrackingTouch(SeekBar seekBar) {
22
23             }
24         });
25         ....
```

Code References

1. **DemoGeoQuiz.zip**
2. **DemoSliders.zip**

Conclusion

Closing Remarks

Thank You For Your Time!
Questions?