Mobile Application Development

Introduction

In this Lecture, you will learn:

- ➢ Introduction
- ➢ Installing Tools
- ➢ Application Basics
- ➢ UI Design
- ➢ Resources
- ➢ Event Listeners
- ➢ Building and Running Application

Introduction

Android is a Linux kernel based mobile operating system. Android Applications are developed in Java or Kotlin or C++. We will use Java, so familiarity with the concepts including classes and objects, interfaces, event listeners, packages, anonymous inner classes, and generic classes is a pre-requisite.

The Standard IDE for Android Applications Development is Android Studio. Android studio is based on IntelliJ IDEA.
Android Studio installation should include:
Android SDK - the latest version of the Android SDK
Android SDK tools and platform tools - tools for debugging and testing apps
A system image for the Android emulator - lets you create and test your apps on virtual devices

Installing Tools:

Download and install Android Studio from
https://developer.android.com/studio

Installing Android SDK and platform tools:
After installing Android Studio, you need to install the Android SDK and platform tools. To install the SDK, open Android Studio, select Tools -> SDK Manager.

The SDK Manager is shown in Figure 1. Select and install each version of Android that you want to test with.

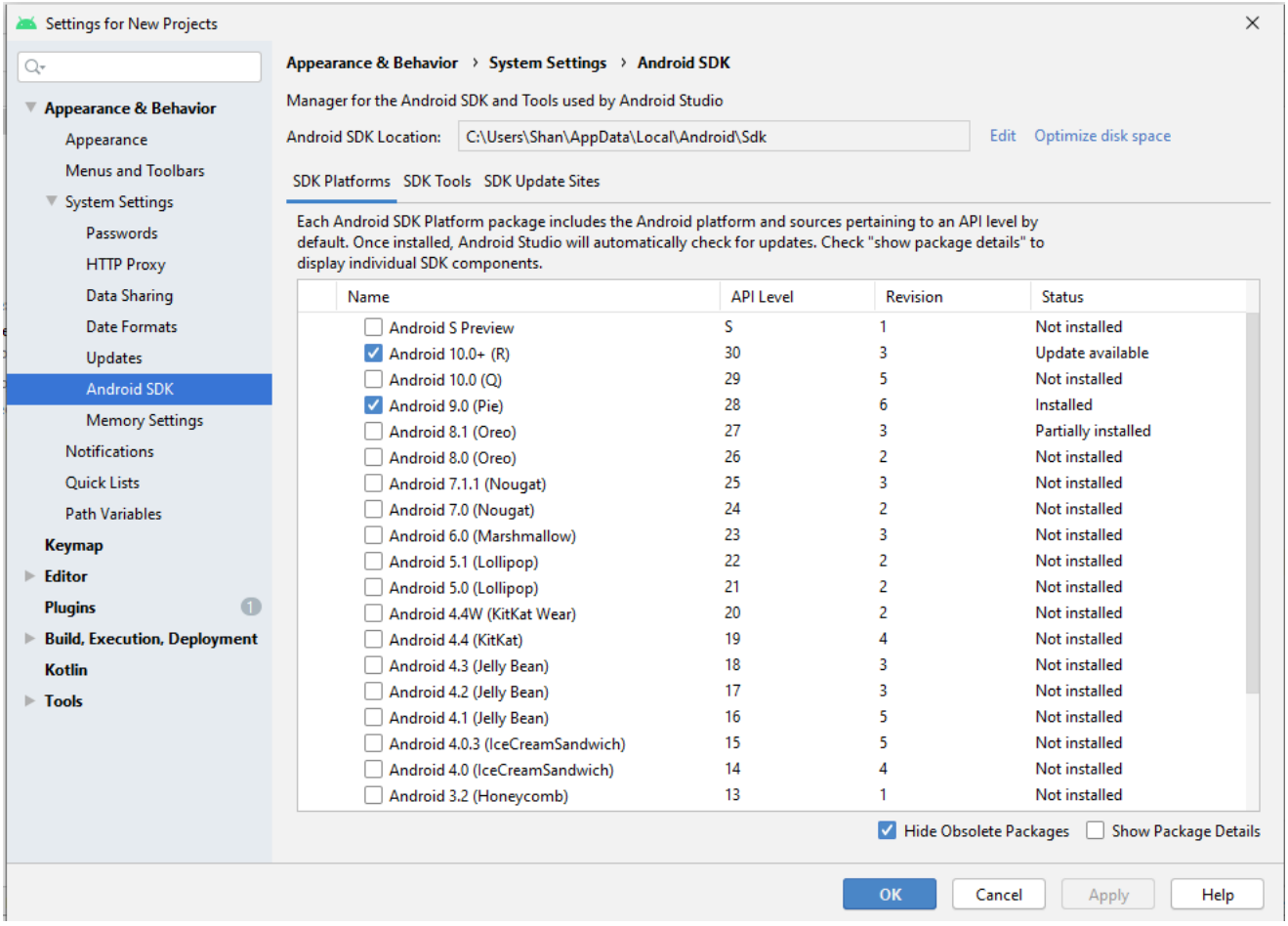

Figure 1: Android SDK Manager

Installing Android Emulator:

The emulator is a virtual device that is used for testing apps. However, it is no substitute for an actual Android device when measuring performance or experience on a real device.

To install an Emulator to test application, open the AVD manager from the menu options in Android Studio as Tools -> AVD Manager.

As shown in the Figure 2, AVD Manager lists the available virtual devices and shows a button to create a new virtual device. Click the Create Virtual Device button to create a new emulator. Follow the steps as discussed in the class lecture to setup your new emulator device.

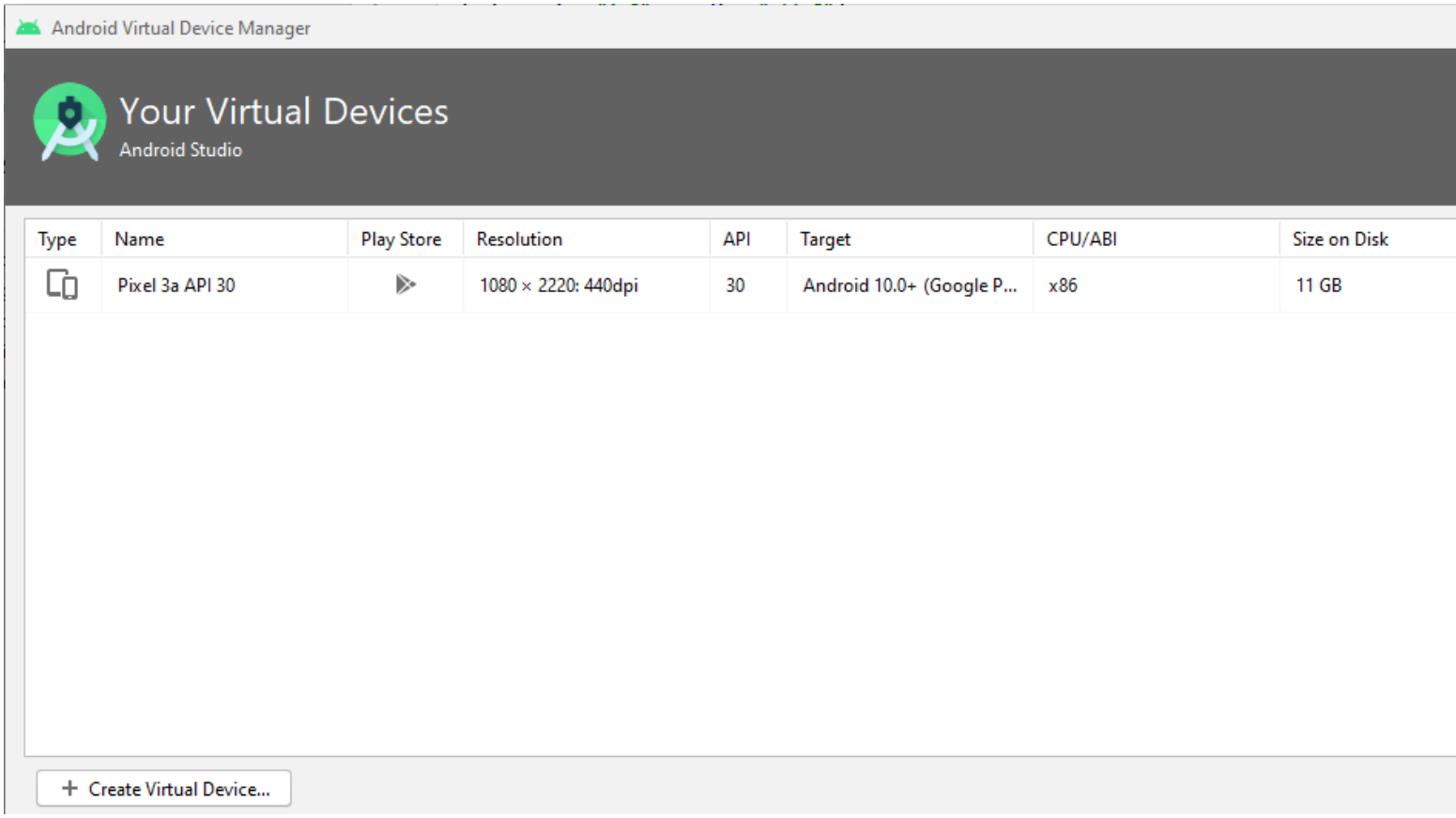

Figure 2: AVD Manager showing the virtual devices

First Application

After installing the Android SDK tools and creating emulator, you are ready to create your first Android Application. The application we are going to create is named GeoQuiz which tests user's knowledge of geography. The user presses TRUE or FALSE to answer a question and the app provides instant feedback.

Creating Project

To create a new project, open Android Studio and choose *File* → *New* → *New Project*....

Your will see a dialog to select template for your project. Select Empty Activity template for your new project and press Next button. On the next dialog as shown in Figure 3, type your app name as GeoQuiz and leave other fields as default and press Finish button to create your project.

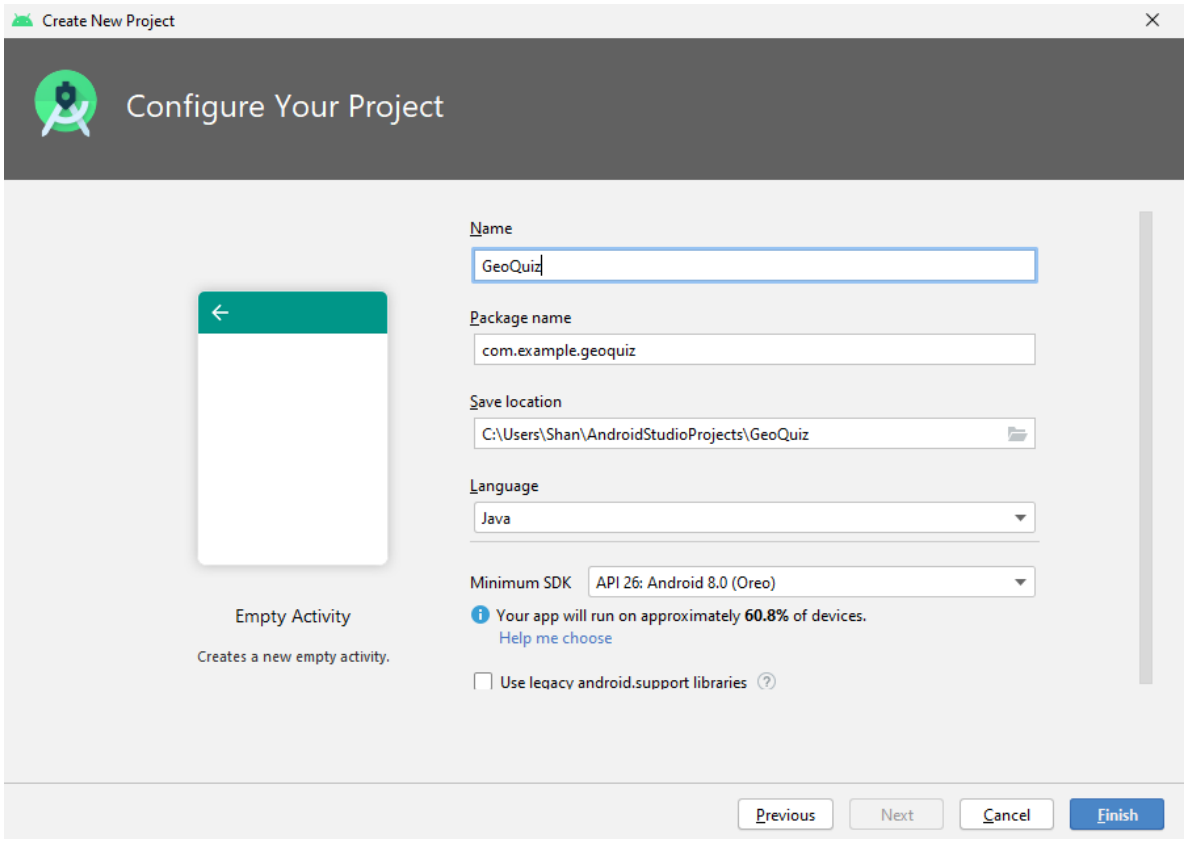

Figure 3: Configuring the project

Android Studio opens your project in a window, as shown in Figure 4. The different panes of the project window are called tool windows. The left view is the project tool window. From here, you can view and manage the files associated with your project.

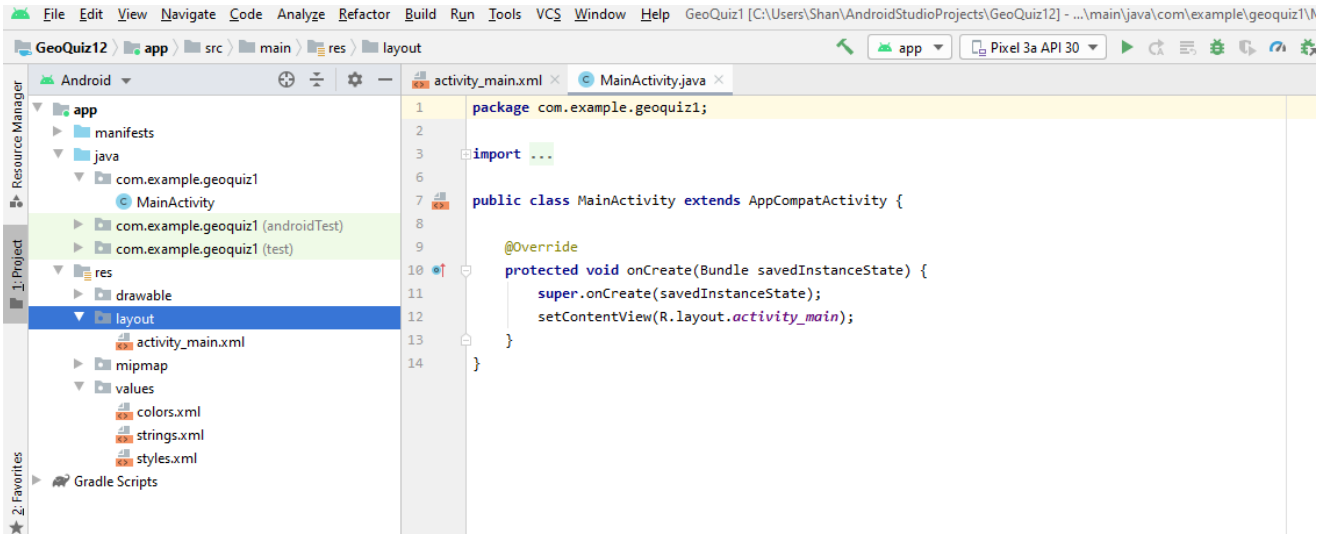

Figure 4: New project window

Laying Out the UI

Open the app/res/layout/activity_main.xml file. If you see a graphical preview of the file, select Text tab or Code tab to see the XML. activity_main.xml defines the default activity layout. The contents may change, but the XML should look something like Listing 1.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <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" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 1: Contents of a newly created activity UI

The default activity layout defines two widgets: a ConstraintLayout and a TextView.

Widgets are the building blocks to compose a UI. A widget can show text or graphics, interact with the user, or contain other widgets. Buttons, text input controls, and checkboxes are all types of widgets. Every widget is an instance of the View class or one of its subclasses (such as TextView or Button).

You need to create new widgets for the Quiz screen. An outer layout will hold the widgets in it. A TextView will hold the quiz to ask. Two Button widgets will allow user to answer the quiz as True or False.

The listing 2 shows updated XML layout showing the quiz and the buttons.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Canberra is the Capital of Australia."
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="True"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.371"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.594" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="False"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.708"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.594" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 2: Updated XML layout showing the quiz and the buttons

Figure 5: Application UI corresponding to the XML in listing 2

View hierarchy

The widgets exist in a hierarchy of View objects called the view hierarchy. If you get a closer look at the listing 2, the two Buttons and TextView widget exist within the root element ConstraintLayout.

A **ViewGroup** is a widget that contains and arranges other widgets. When a widget is contained by a ViewGroup, that widget is said to be a child of the ViewGroup. The root ConstraintLayout has three children: a TextView and two Button widgets.

Widget attributes

The attributes are used to configure the widgets. Some of the attributes used in the listing 2 include:

android:layout_width and *android:layout_height* attributes are required for almost every type of widget to set widget's width and height. They are typically set to either match_parent or wrap_content:

match_parent view will be as big as its parent
wrap_content view will be as big as its contents require

The ConstraintLayout is the root element, but it still has a parent – the view that Android provides for your app's view hierarchy to live in.
The other widgets in the layout have their widths and heights set to wrap_content.

android:padding attribute This attribute tells the widget to add the specified amount of space to its contents when determining its size.

TextView and *Button* widgets have android:text attributes. This attribute tells the widget what text to display.
Notice that the values of these attributes are not literal strings. They are references to string resources.
A string resource is a string that lives in a separate XML file called a strings file. You can give a widget a hardcoded string, like android:text="True", but it is discouraged. Placing strings into a separate file and then referencing them is better because it makes localization easy.

Creating string resources

Every project includes a default strings file named strings.xml. Open res/values/strings.xml.
Add the three new strings in this file as shown in listing 3 that your layout requires.

```
<resources>
   <string name="app_name">GeoQuiz1</string>
   <string name="question_text">Canberra is the capital of Australia.</string>
   <string name="true_button">True</string>
   <string name="false_button">False</string>
</resources>
```

Listing 3: Adding string resources in strings.xml

Now referring @string/false_button in any XML file in the project, you will get the literal string “False” at runtime.

The default strings file is named strings.xml, you can name a strings file anything you want. You can also have multiple strings files in a project.

View Objects

The class file for MainActivity is in the app/java directory of your project. The java directory is where the Java code for your project lives. The AppCompatActivity is a subclass of Android’s Activity class that provides compatibility support for older versions of Android.

The onCreate(Bundle) method is called when an instance of the activity subclass is created. To bind the activity with UI, the following method is invoked:
public void setContentView(int layoutResID)
This method inflates a layout and loads on screen. When a layout is inflated, each widget in the layout file is instantiated as defined by its attributes. You specify which layout to inflate by passing in the layout’s resource ID.

Resources and resource IDs

A layout is a resource. A resource is a piece of your application that is not code – things like image files, audio files, and XML files.

Resources for your project live in a subdirectory of the app/res directory. In the project tool window, you can see that activity_quiz.xml lives in res/layout/. Your strings file, which contains string resources, lives in res/values/.
To access a resource in code, you use its resource ID. The resource ID for your layout is R.layout.activity_main.

To see the current resource IDs for the project, load project view and open R.java file. This file is generated for your app just before it is installed on a device or emulator.
By default, Android Studio uses the Android project view which hides the actual directory structure of the project so that you can focus on the files and folders that you need most often.

To generate a resource ID for a widget, define android:id attribute in the widget's definition. In activity_main.xml, add an android:id attribute to each button as shown in listing 4.

```
<Button
    android:id="@+id/buttonTrue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="True"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.371"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.594" />
```

Listing 4: Adding id attribute

Wiring Up Widgets

After defining resource ids in the XML, you can access them in the code and wire up the event handlers. Define member variables for the Buttons as shown in listing 5.

```
public class QuizActivity extends AppCompatActivity {
    private Button mTrueButton;
    private Button mFalseButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Listing 5: Adding member variables (MainActivity.java)

To add click event listener to a widget, get references to the inflated View objects set listeners on those objects to respond to user actions.

To **get reference** to an inflated widget call:

public View findViewById(int id)

This method accepts a resource ID of a widget and returns a View object.

```
public class MainActivity extends AppCompatActivity {
    private Button mTrueButton;
    private Button mFalseButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

Listing 6 Getting references to widgets (MainActivity.java)

Setting listeners

Android applications are event driven which wait for an event, such as Button press. When your application is waiting for a specific event, it is "listening for" that event. The object that responds to an event is called a listener, and the listener implements a listener interface for that event.

The Android SDK comes with listener interfaces for various events. The event you want to listen for is a button being pressed (or "clicked"), so your listener will implement the View.OnClickListener interface.

(MainActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);
    mTrueButton = (Button) findViewById(R.id.true_button);

    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //handle click event here
        }
    });
    mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

Listing 7 Setting a listener for the TRUE button

The Listing 7 sets a listener to when mTrueButton has been pressed. The setOnClickListener(OnClickListener) method takes a listener as its argument. It takes an object that implements OnClickListener.

Anonymous inner classes

The *setOnClickListener's* listener argument is implemented as an anonymous inner class. The syntax is a little tricky, remember that everything within the outermost set of parentheses is passed into *setOnClickListener(OnClickListener)*
Within these parentheses a new nameless class is created and its entire Implementation is passed.

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        }
});
```

This anonymous inner implementations of the listeners' methods are used right where they are needed. Furthermore there is no need to create a named class because the class will only be used once.

Since the anonymous class implements OnClickListener, it must implement that interface's sole method, onClick(View). The method definition will be completed later.

Now implement the *setOnClickListener* for the *mFalseButton* in the same way.

Toasts

To make the buttons useful we are going to show a pop-up message called a toast. A toast is a short message that informs the user of something but does not require any input or action. Toasts will show whether the user answered the quiz correctly or incorrectly.

When you type a period after the Toast class, a pop-up window will appear with a list of suggested methods and constants from the Toast class. This is called code completion.
From the list of suggestions, select
makeText(Context context, int resID, int duration)
Code completion will add the complete method call for you.

```
mTrueButton.setOnClickListener(newView.OnClickListener() {
        @Override
        public void onClick(View v) {
                Toast.makeText(MainActivity.this, R.string.correct_toast,
                Toast.LENGTH_SHORT).show();
        }
        });

mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                Toast.makeText(MainActivity.this, R.string.incorrect_toast,
                Toast.LENGTH_SHORT).show();
        }
});
```

Listing 8: Making toast message

In makeText(…), you pass the instance of MainActivity as the Context argument. At this point in the code, you are defining the anonymous class where this refers to the View.OnClickListener. So you need to pass this argument as *MainActivity.this* to get the correct context.

Since you used code completion, the Toast class is automatically imported. When you accept a code completion suggestion, the necessary classes are imported automatically.

Running on the Emulator

A device is needed to run the application – either a hardware device or a virtual device. Virtual devices are powered by the Android emulator which we will be using to test our applications.
To create new Android virtual device (AVD), choose *Tools → AVD Manager*. When the AVD Manager appears, click the +Create Virtual Device... button in the lower-left corner of the window. This will pop up a new window to create a new virtual device. Follow the instructions given to create the virtual device.

Once you have created an AVD, you can run GeoQuiz on it. From the Android Studio toolbar, click the run. Alternatively, from the main menu, you can choose Run → Run 'app' to run the application.
Android Studio will find the virtual device you created, start it, install the application package on it, and run the app. Keep the emulator running – while updating and testing the application since loading emulator takes time. Testing on real devices gives more accurate results.

If your app crashes when launching or when you press a button, useful information will appear in the Logcat view in the Android DDMS tool window. (you can open Logcat by clicking the Android Monitor button at the bottom of the Android Studio window.) From this Logcat, try to find the cause of the error.

Android Build Process

In the build process, the Android tools take the app resources, code, and the AndroidManifest.xml file (which contains metadata about the application) and turn them into an .apk file. This file is then signed with a debug key, which allows it to run on the emulator. To distribute your .apk file, you need to sign it with a release key.

As part of the build process, AAPT (Android Asset Packaging Tool) compiles layout file resources into a more compact format to turn activity_main.xml contents into View objects in the application.

These compiled resources are packaged into the .apk file. When setContentView(…) is called in the onCreate(Bundle) method, the LayoutInflater class instantiates each of the View objects.

The build tools such as AAPT are integrated into Android IDE. But if you want to build and debug app outside of Android Studio, you can use a command-line build tool called Gradle. To use Gradle from the command line, navigate to your project's directory and run the following command:
> *gradlew.bat tasks*
This will show you a list of available tasks you can execute.
On Windows:
> gradlew.bat installDebug
Command will install your app on whatever device is connected. However, it will not run the app so you need to run it manually on the device.

You can explore Gradle further in online documentation.

QUESTIONS?