Hugbúnaðarverkefni 2 / Software Project 2

5. Android Development Basics

HBV601G – Spring 2019

Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz 4 Prep

- Please prepare a small scrap of paper with the following information:

ID: _____@hi.is Date: _____

a) _____ e) _____
b) _____ f) _____
c) _____ g) _____
d) _____

- During class, I'll show you questions that you can answer very briefly
  - Just numbers or letters, no elaboration
- Hand in your scrap at the end of class

- All questions in a quiz weigh same
- All quizzes (ca. 10-12 throughout semester) have the same weight
  - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam

In-Class Quiz 3 Solution

- **Indicate which approach you would use in the following situations:**

a) Establishing a rough project scope
b) Establishing first release date (new project, new domain)
c) Establishing next release date (project you've been on for a year)
d) Getting an idea of whether you can still complete a user story before end of sprint
e) Sprint planning

- Best answers:

- (D) T-Shirt Sizing
- (B) Three-Point Estimates
- (C_1) Wideband Delphi
- (A) Projection from counts
- (C_2) Planning Poker

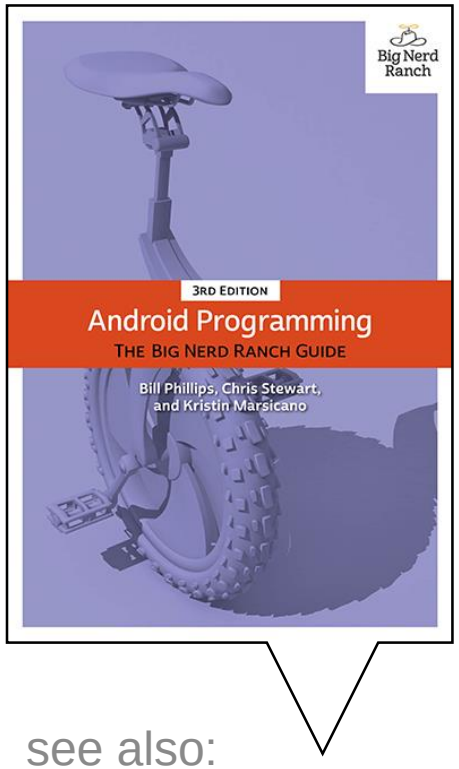
Update: Assignment 2 – Design Model

Note: Clarified from version shown in first week.

- By the deadline specified in your project plan, submit a **design model** in Ugla:
  - UML class diagram of your system (detail level reflecting state of implementation/planning)
    - Clearly distinguish server- and client-side components
    - For client (Android) side, show only Model and Controller classes, not View classes
  - Suitable UML behavioral diagrams to show:
    - User navigation in your app
    - Control flow between key components (within app, and between client and server)
- On the Monday after submission, present and **explain** your model to your tutor:
  - How will your system work? What influenced your design choices? What's still unknown?

- **Grading criteria** (25% of this assignment's grade each):
  - System structure is plausible, consistent with requirements and behavioral diagrams
  - User navigation is plausible and shown in a suitable diagram
  - Control flow is plausible and shown in a suitable diagram
  - UML diagrams are clean and syntactically correct

A Simple Android App



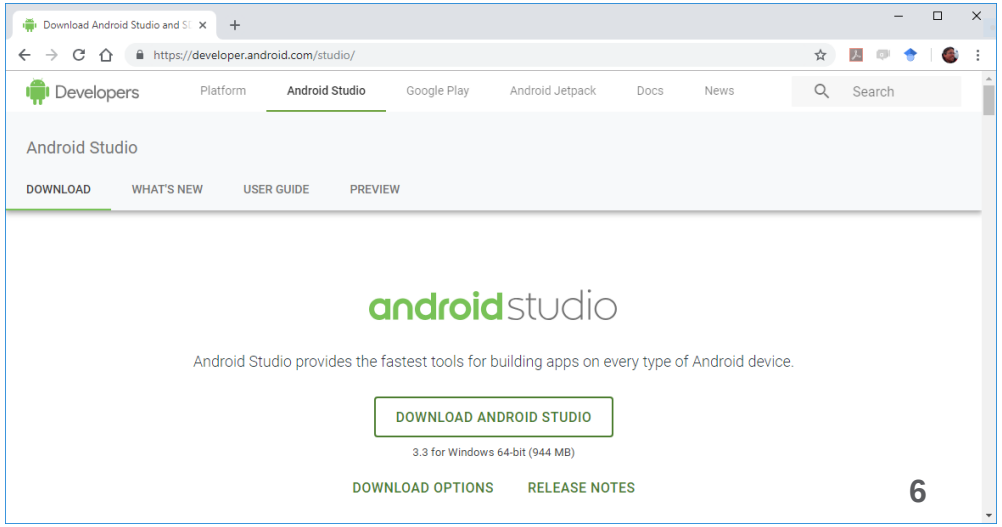
see also:

- Phillips et al.: Android Development, Ch. 1

Development Environment

- **Java Platform, Standard Edition SDK**
  - Download: https://www.oracle.com/technetwork/java/javase/downloads/index.html

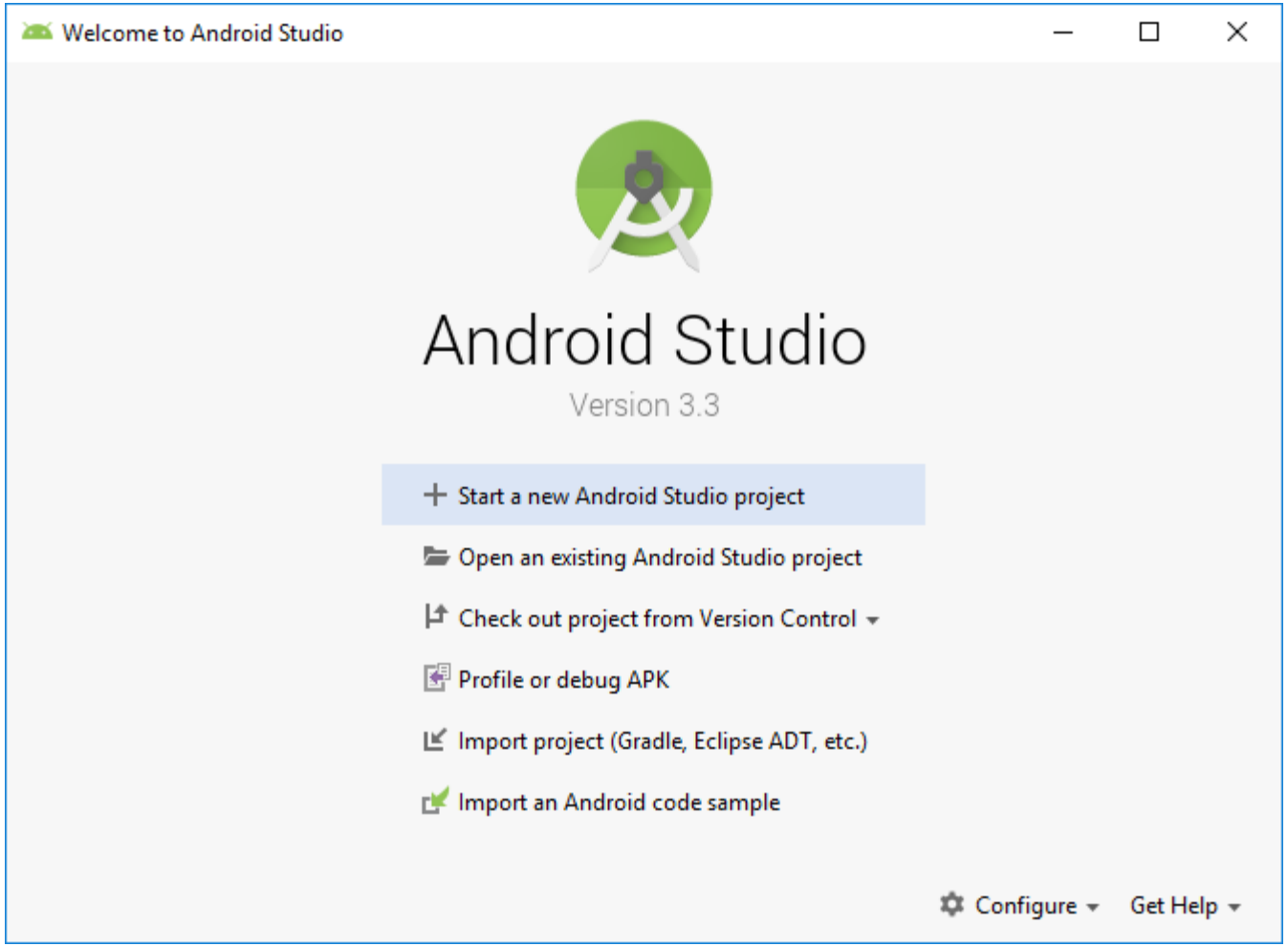
- **Android Studio**
  - IDE based on IntelliJ
  - Android SDK (class library)
  - Tools for debugging and testing apps
  - Android emulator
  - Download: https://developer.android.com/studio/
    - Caution: ~2 GB download, 3.6 GB installation



- A physical **Android device** for testing
  - Alternatively, use the included emulator

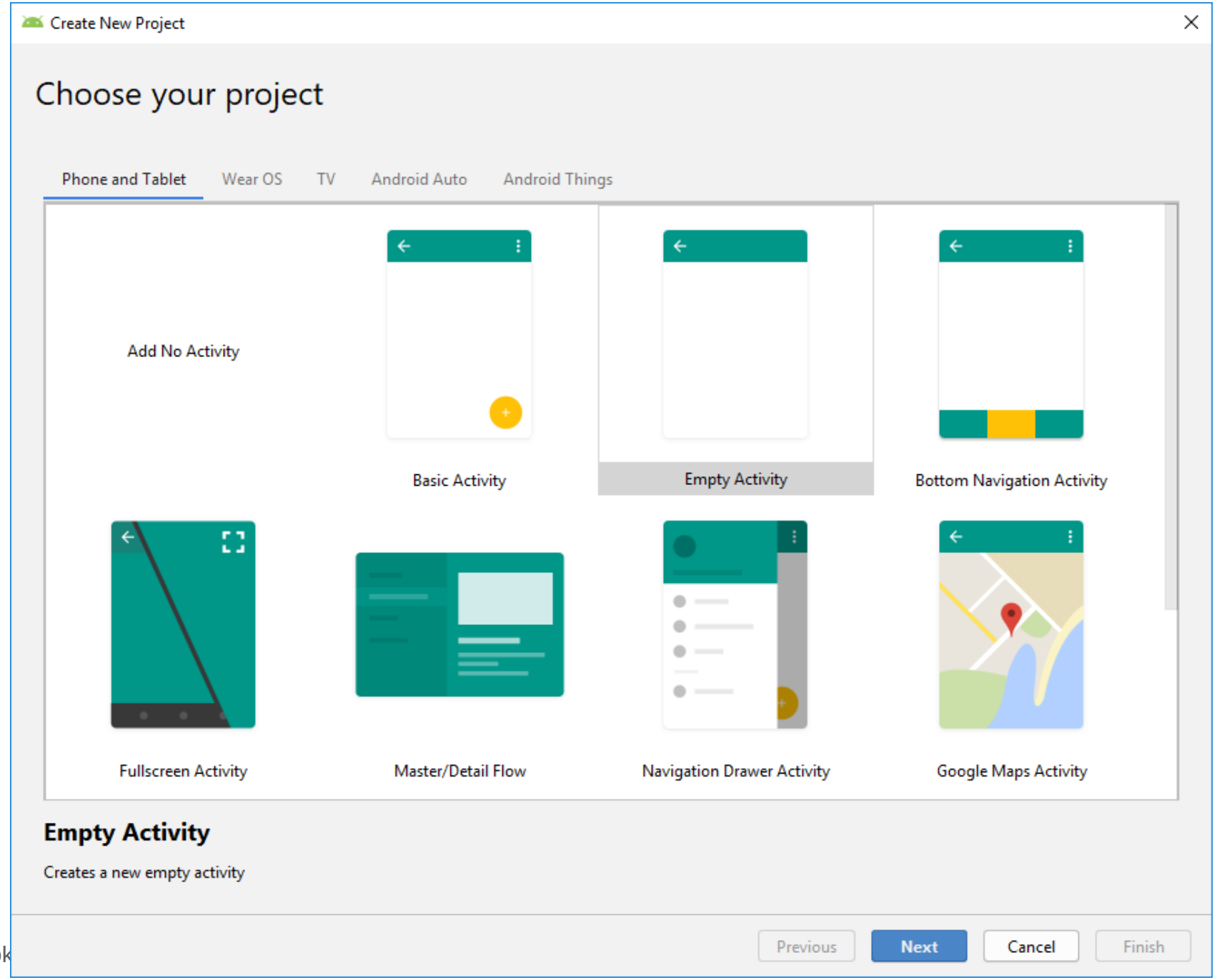
Creating a Project

- Start Android Studio
- Choose “Start a new Android Studio project”
  - If the welcome screen doesn’t appear, use “File > New > New Project…” menu



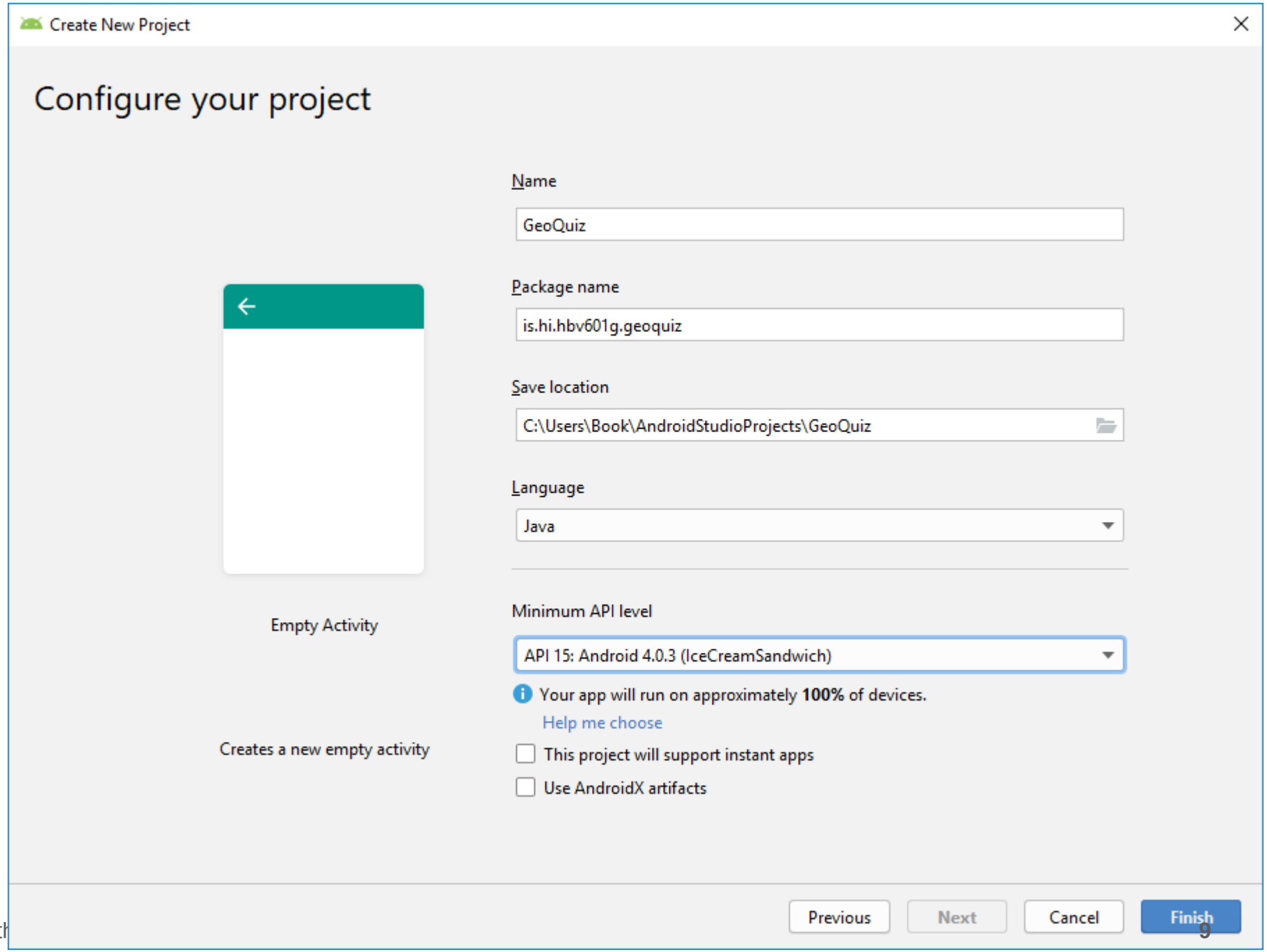
Choose a Project Type

- Use “Empty Activity” for starters
  - Other activities contain extra features
- Note that activities for many different device types are available



Configure Project

- You'll have to choose an API level
- Your app will not be available to devices running an Android version with a lower API level, so choose wisely



Choose API Level

- Pick a reasonable balance between required device functionality and desired market share

Android Platform/API Version Distribution

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1%
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1%

Ice Cream Sandwich

Contacts Provider

Social APIs
User profile
Invite intent
Large photos

Calendar Provider

Calendar APIs
Event intents

Voicemail Provider

Add voicemails to the device

Multimedia

Media effects for images and videos
Remote control client
Improved media player

Camera

Face detection
Focus and metering areas
Continuous auto focus
Camera broadcast intents

Connectivity

Android Beam for NDEF push with NFC
Wi-Fi P2P connections
Bluetooth health profile
Network usage and controls

Accessibility

Explore-by-touch mode
Accessibility for views
Accessibility services
Improved text-to-speech engine support

User Interface

Spell checker services
Improved action bar
Grid layout
Texture view
Switch widget
Improved popup menus
System themes
Controls for system UI visibility
Hover event support
Hardware acceleration for all windows

Enterprise

VPN services
Device policies
Certificate management

Device Sensors

Improved sensors
Temperature sensor
Humidity sensor

https://developer.android.com/about/versions/android-4.0.html

OK Cancel

Android Studio Overview

GeoQuiz [C:\Users\Book\AndroidStudioProjects\GeoQuiz] - ...\app\src\main\res\layout\activity_main.xml [app] - Android Studio

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

GeoQuiz > app > src > main > res > layout > activity_main.xml

app

1: Project | 7: Structure | 2: Favorites | Build Variants | Layout Captures

Android

- app
  - manifests
  - java
    - is.hi.hbv601g.geoquiz
      - MainActivity
    - is.hi.hbv601g.geoquiz (androidTest)
    - is.hi.hbv601g.geoquiz (test)
  - generatedJava
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
      - colors.xml
      - strings.xml
- Gradle Scripts

activity_main.xml | MainActivity.java

```
3    xmlns:android="http://schemas.android.com/apk/res/android"
4    xmlns:app="http://schemas.android.com/apk/res-auto"
5    xmlns:tools="http://schemas.android.com/tools"
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_constraintLeft_toLeftOf="parent"
16       app:layout_constraintRight_toRightOf="parent"
17       app:layout_constraintTop_toTopOf="parent" />
18
19 </android.support.constraint.ConstraintLayout>
20
```

Design | Text

Preview

Pixel | 10% | 8dp

Palette | Gradle | Preview

Build | Sync

- Build: completed successfully at 04-F...
  - Run build C:\Users\Book\Android...
    - Load build
    - Configure build
    - Calculate task graph
    - Run tasks

2 s 317 ms
1 s 862 ms
6 ms
908 ms
34 ms
911 ms

Device File Explorer

TODO | Terminal | Build | 6: Logcat | Event Log

Gradle build finished in 3 s 250 ms (4 minutes ago)

20:1 CRLF UTF-8 Context: <no context>

Browse project files

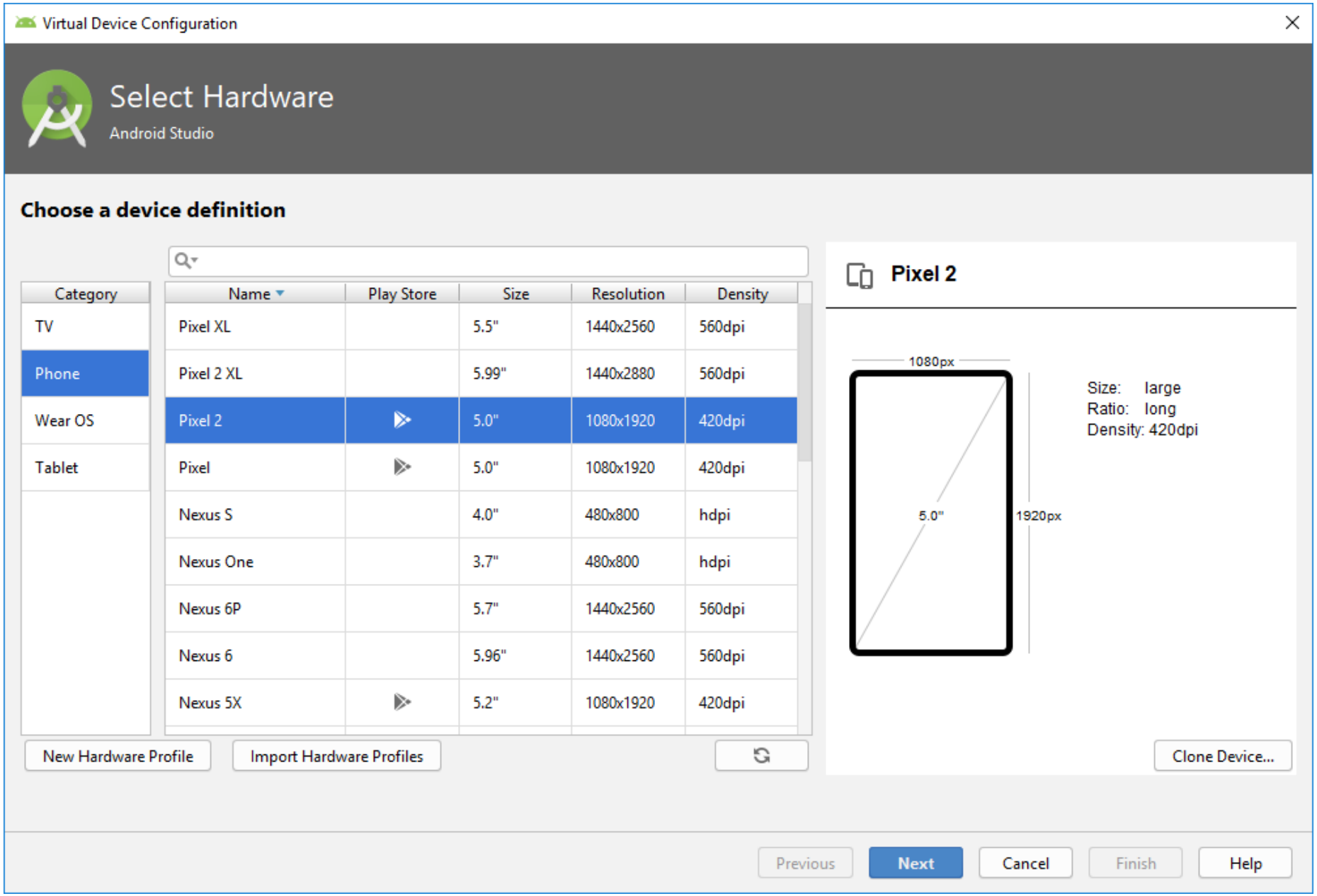
Switch between code and design view

Click on code to highlight corresponding widget (and vice versa)

Toggle tool windows

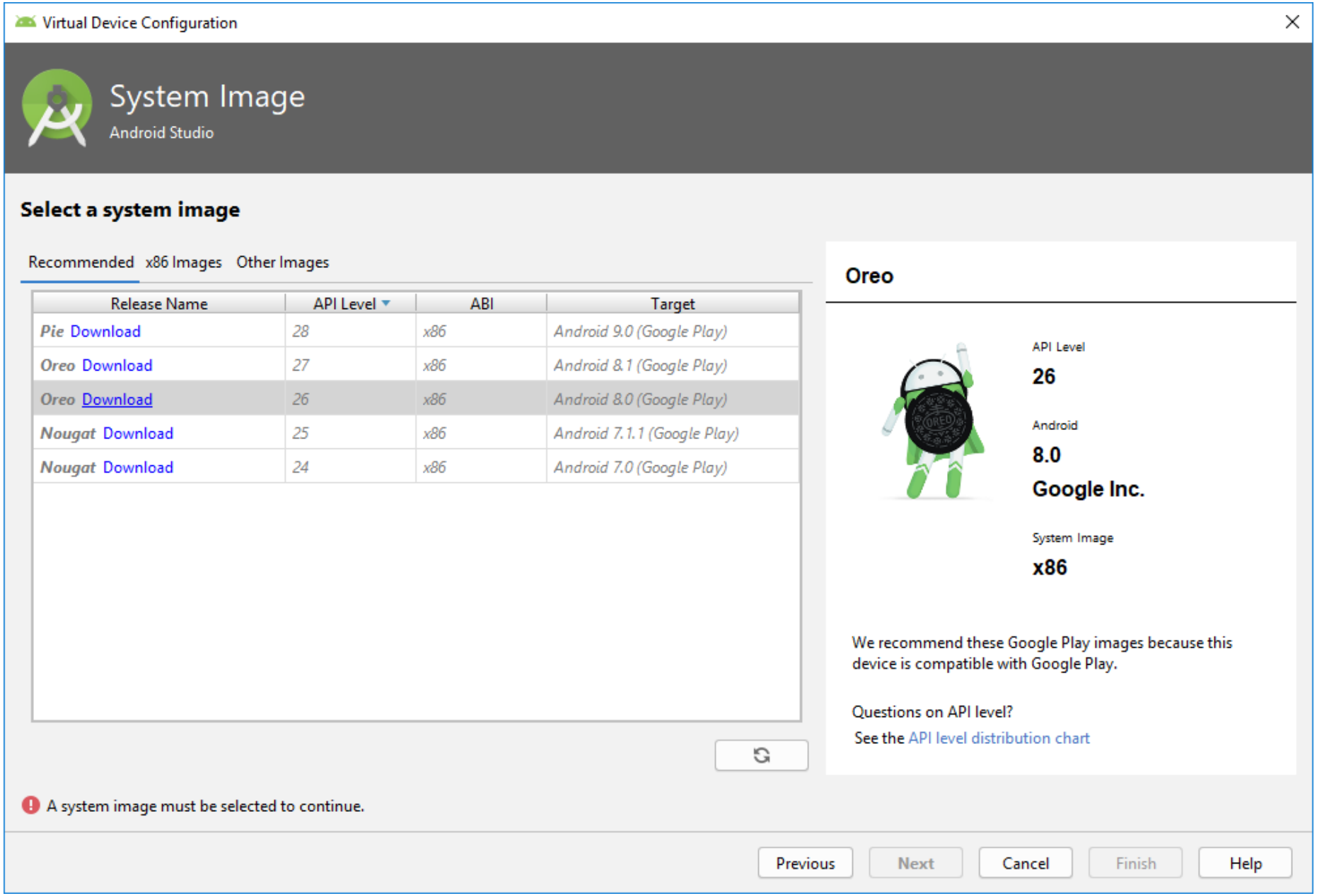
Prepare an Emulator

- Open menu “Tools > AVD Manager”
- Click “Create Virtual Device…” button
- Pick a device of your choice

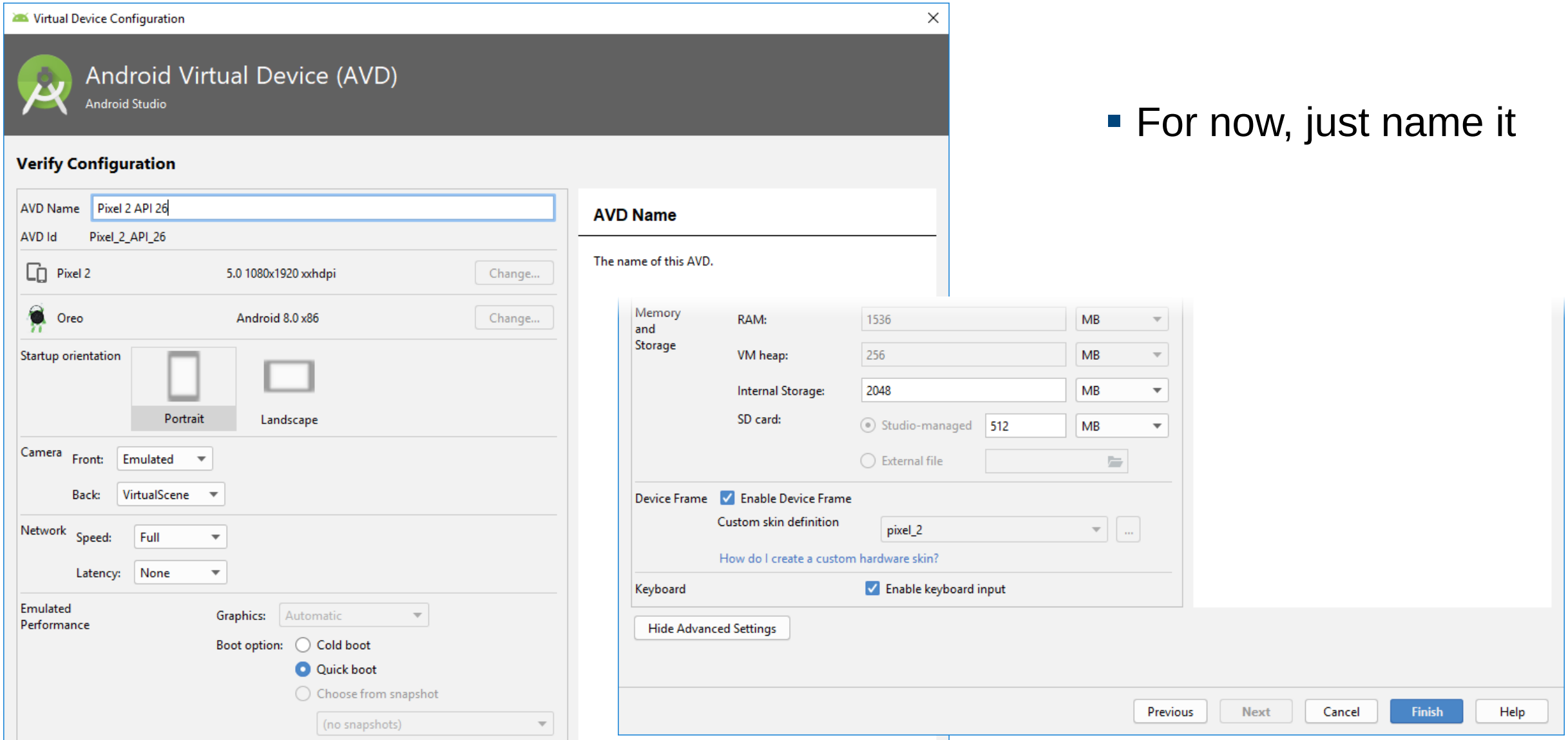


Choose a System Image

- Pick an operating system image of your choice
  - API level at least as high as your project's API level
- May need to download first



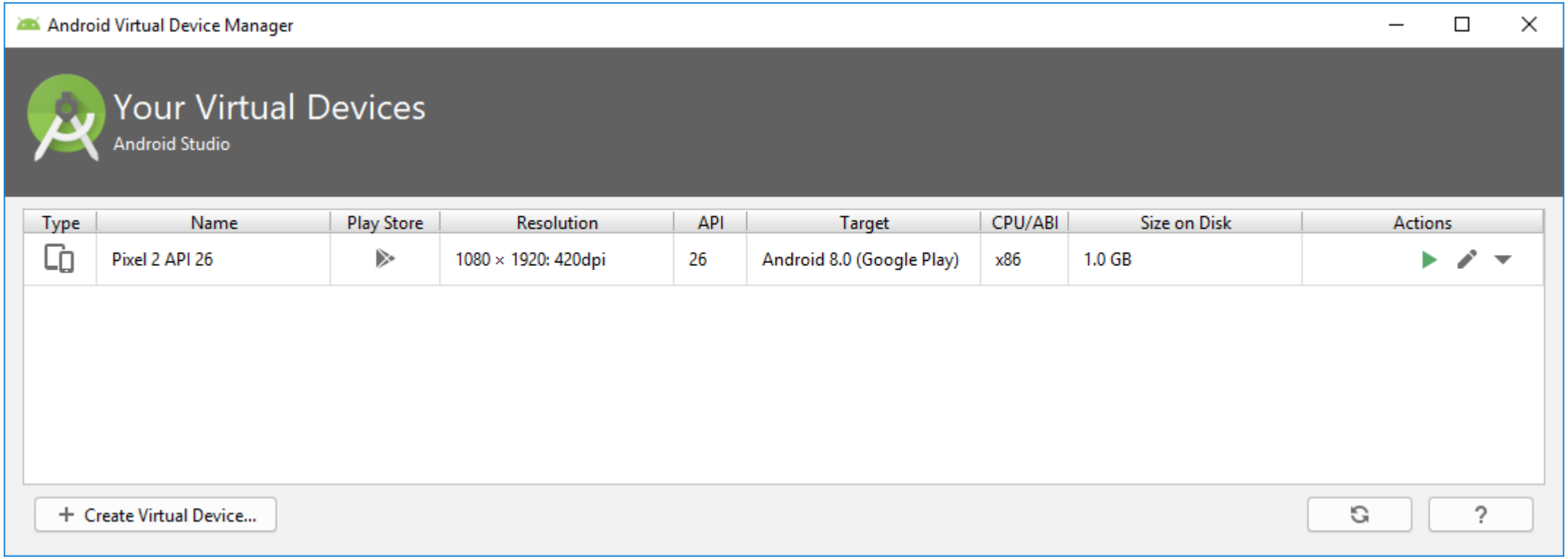
Configure the Virtual Device



- For now, just name it

Managing Virtual Devices

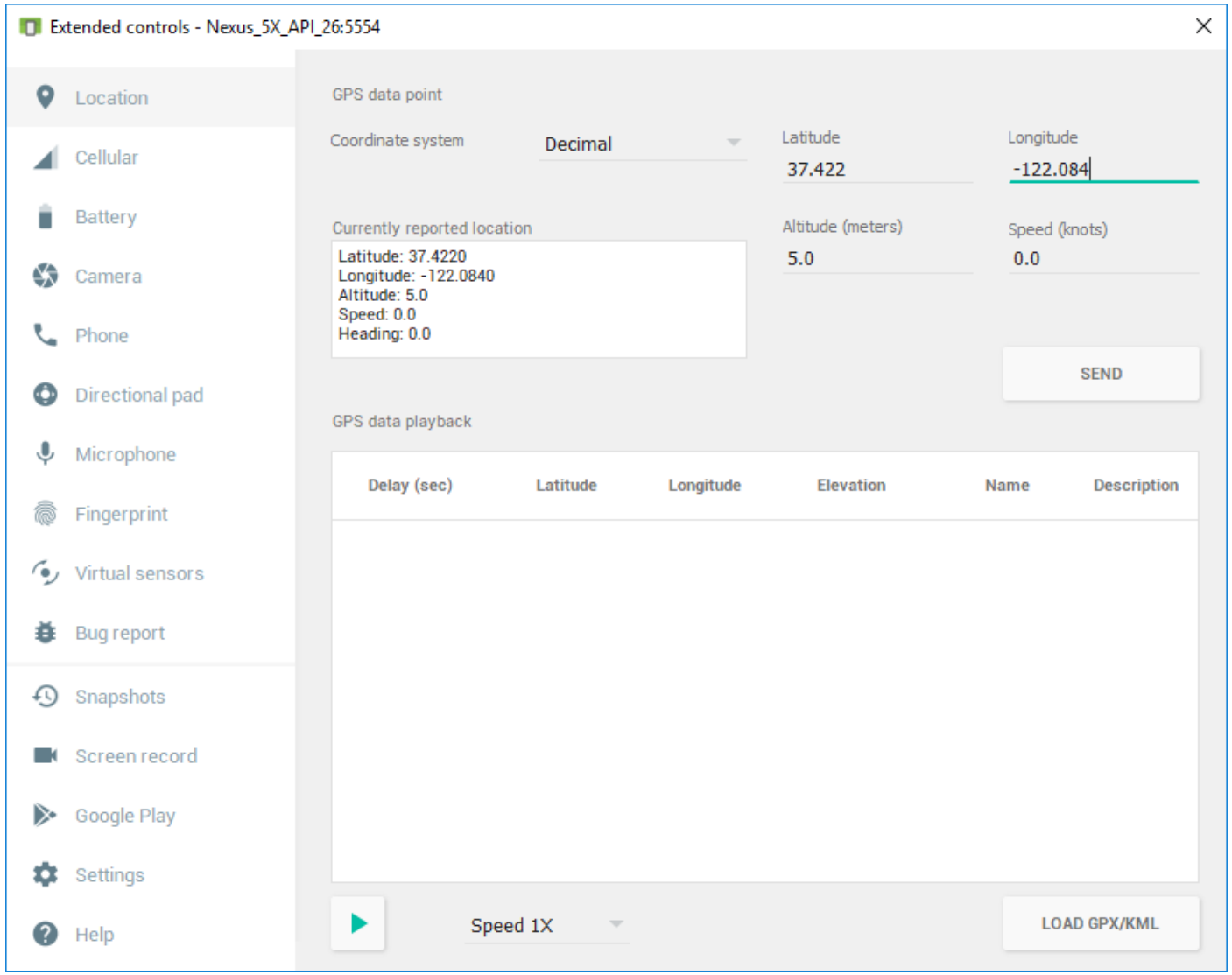
- Your new device shows up in the AVD Manager
  - Use the dropdown menu on the right to manage your virtual devices
- Start virtual device by clicking the ▶ icon



Working With the Virtual Device

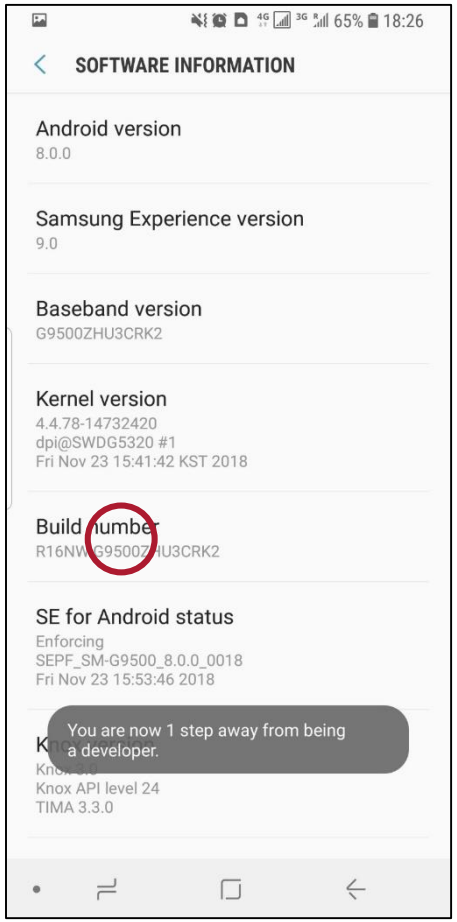
- Boot-up takes several minutes
  - Keep running for later tests

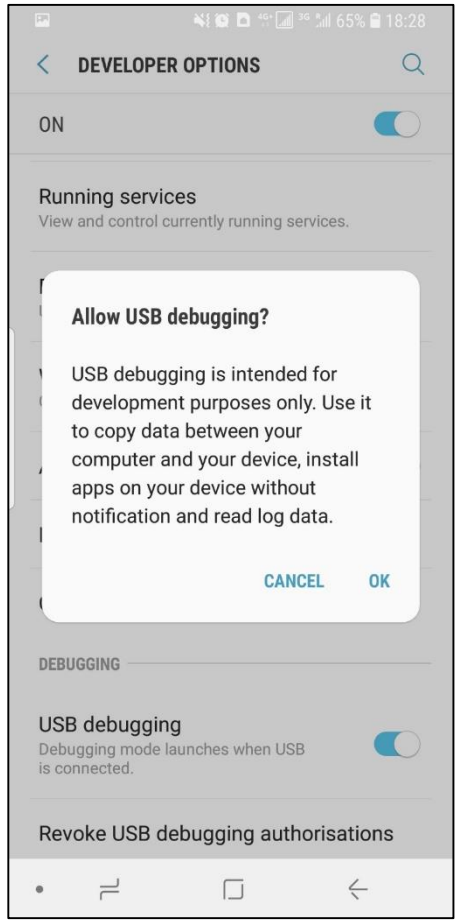
- Emulate sensor inputs etc. by opening "…" menu

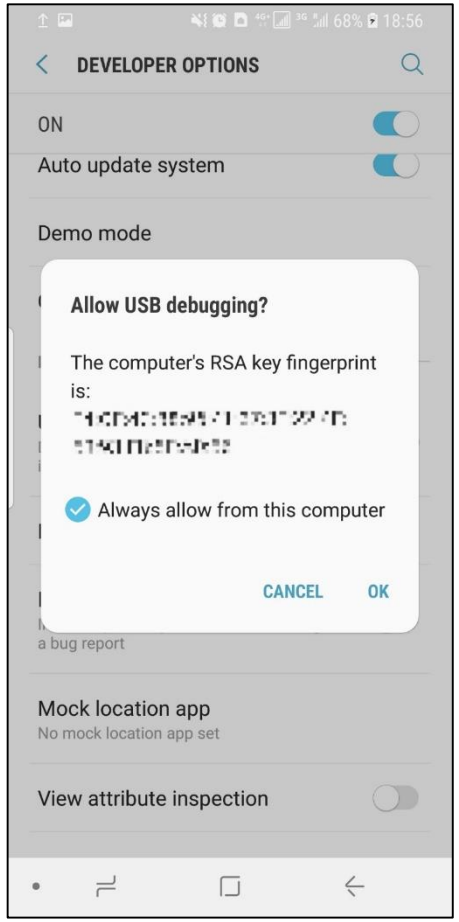


Preparing a Physical Device

- To deploy apps on a physical device, enable USB debugging on that device:
  - Go to “Settings > About Phone > Software Information” (or similar)
  - Press “Build Number” seven times
  - Go to “Settings > Developer options” (was invisible before)
  - Enable “USB debugging”
    - Confirm that USB debugging shall be allowed in general
- Then connect device to computer using USB cable
  - Authorize computer for USB debugging



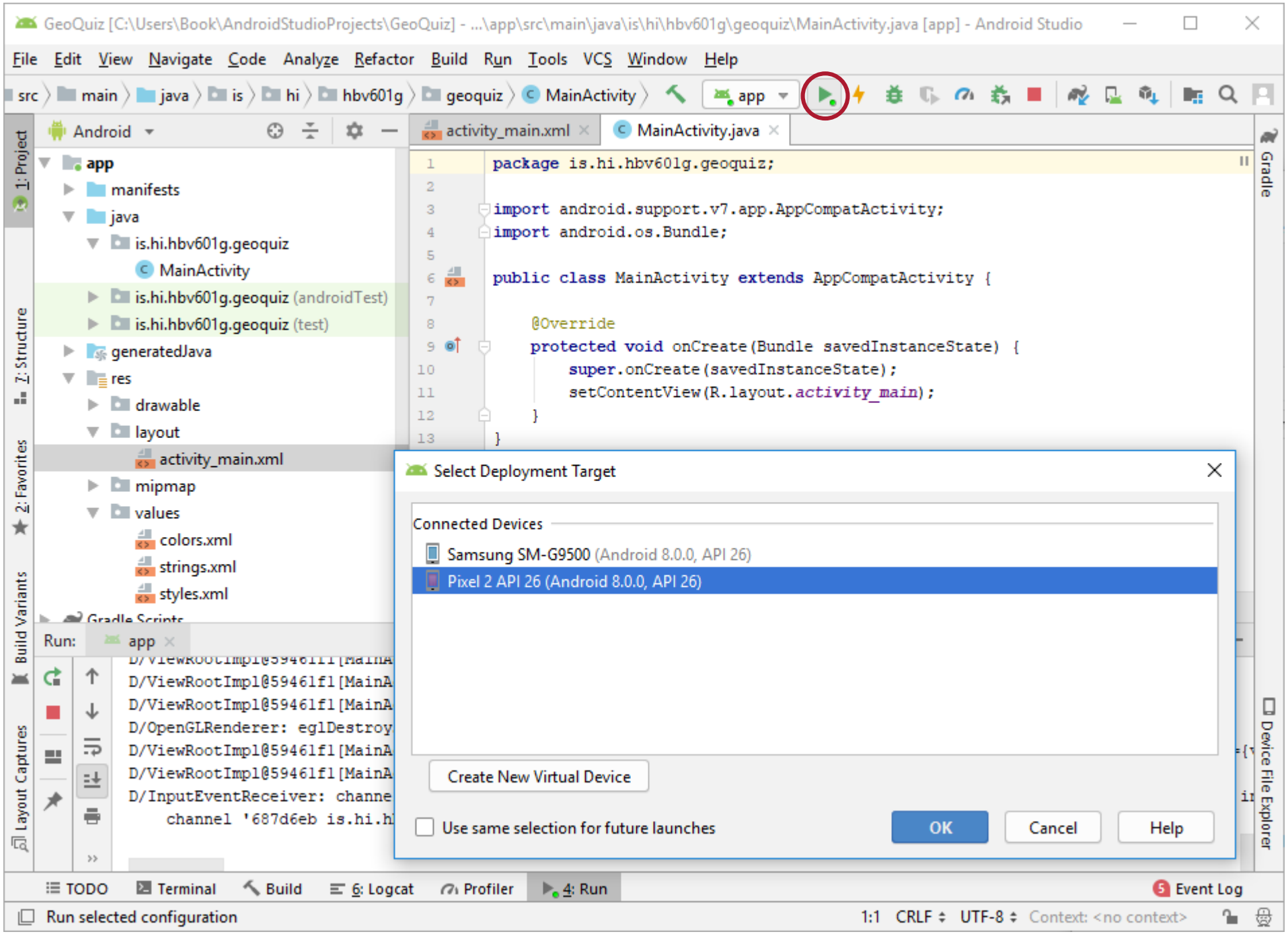




(Example screenshots from Android 8.0 on Samsung Galaxy S8)

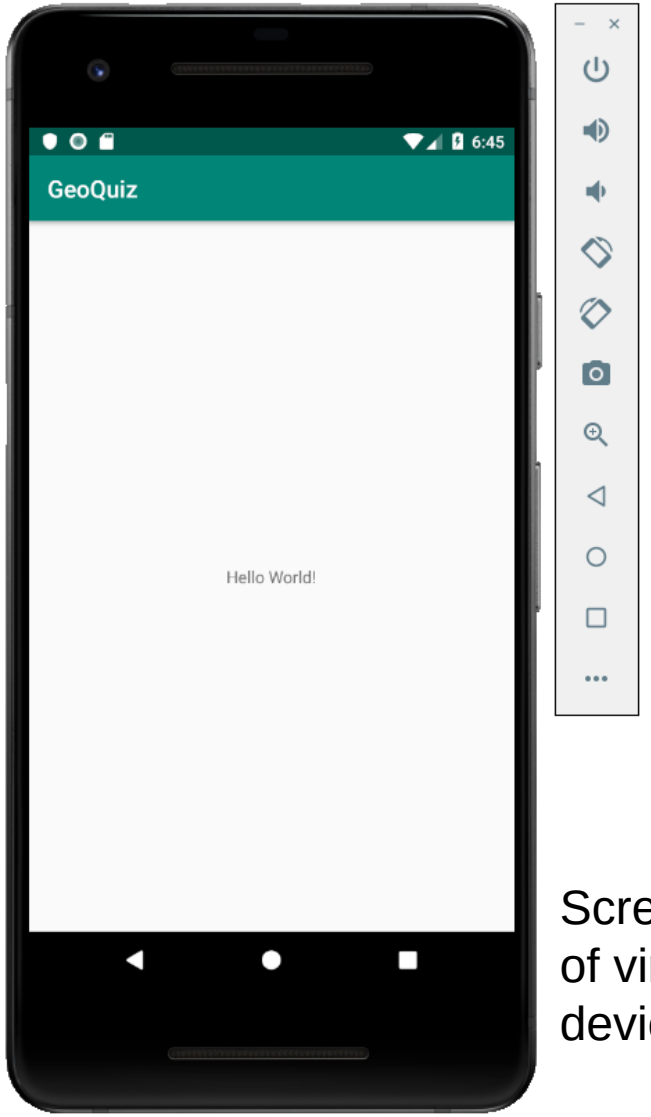
Running the App

- Click on green ▶ icon
  - or press Ctrl+R
- Pick desired device (virtual or physical, if connected)
- App will be deployed and started on device

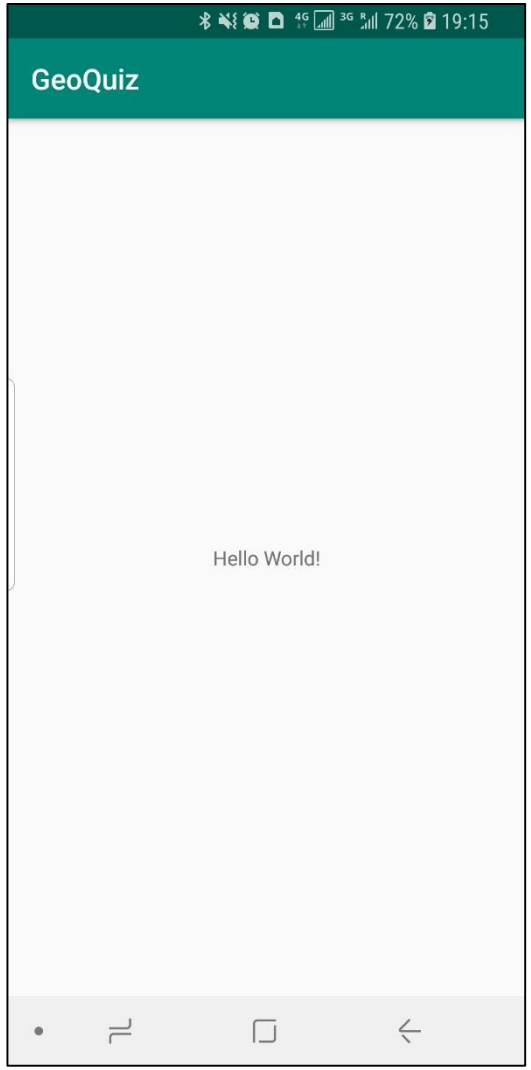


Testing the App

- App will start on device
- Interact as desired
  - Use virtual device sidebar for emulation of physical sensor input (e.g. rotation, camera, location…)
- Back out of app using ◄ icon to stop execution



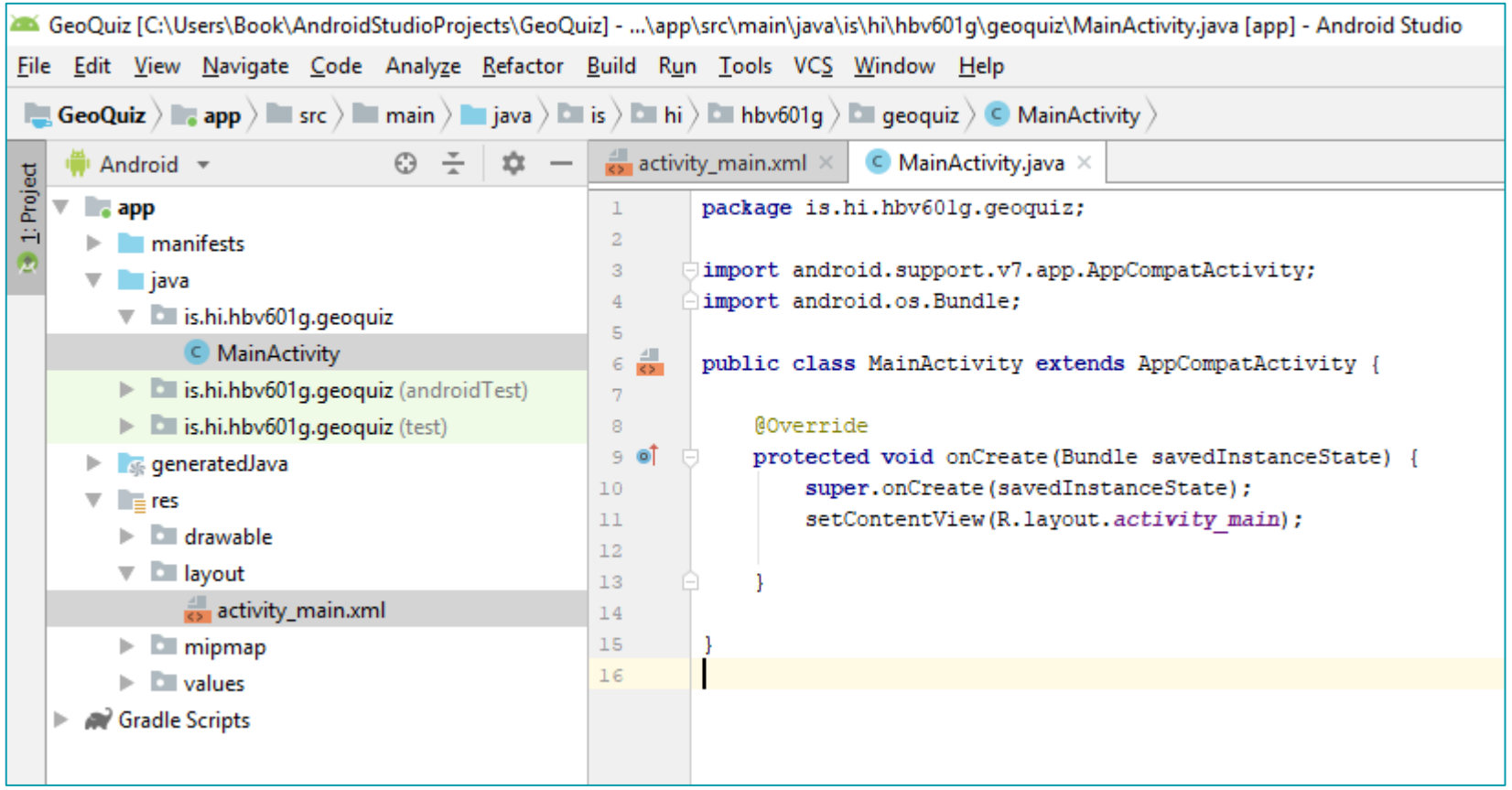
Screenshot of virtual device



Screenshot from physical device

Basic Project Structure

- **Java classes** go into \app\src\main\java
  - Most importantly: Activity classes
- **Resource files** go into \app\src\main\res
  - Most importantly: Layout files in …\layout

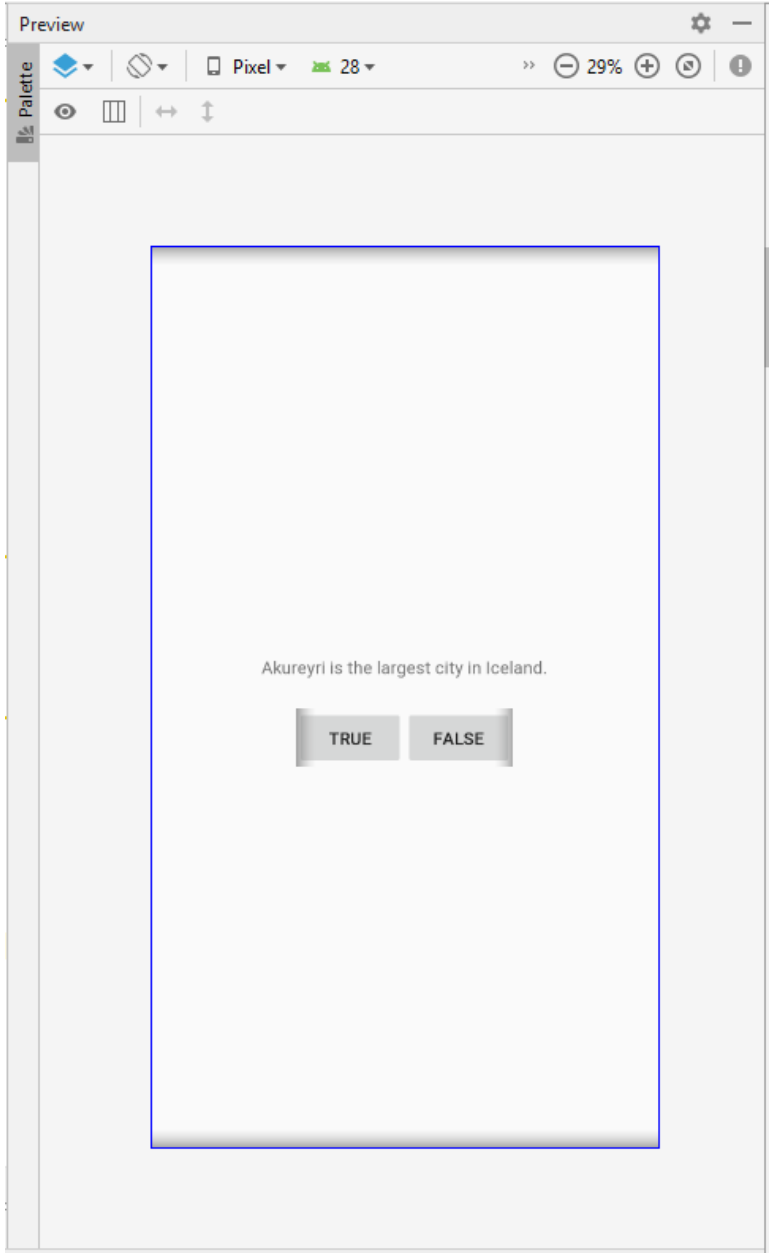


- Naming convention:
  - Activities end with **`Activity`**
  - Corresponding layouts start with **`activity_`**

Activities and Layouts

- An **activity** is responsible for managing user interaction with a screen of information, i.e. to implement a part of the functionality of your app.
  - Activities are implemented in subclasses of **`Activity`**
- A **layout** defines a set of user interface (UI) objects and their screen positions.
  - Layouts are defined in XML files stored in **`app/src/main/res/layout`**
  - Each XML element in the file corresponds to one UI widget.

➢ The activity **`MainActivity`** manages the UI that the layout **`activity_main.xml`** defines.
  - Example:
    - **`activity_main.xml`** defines where the “TRUE” and “FALSE” buttons are placed and how they are labeled.
    - **`MainActivity`** implements what happens when one of the buttons is clicked.



Defining a Layout

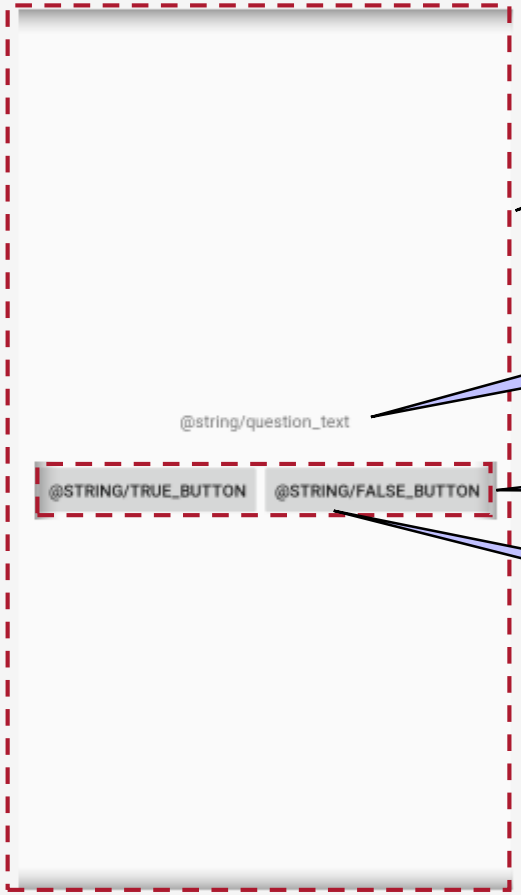
activity_main.xml | MainActivity.java

```
<LinearLayout xmlns:android="http://schemas.android.com/apk
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:text="@string/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button"/>

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button"/>

    </LinearLayout>

</LinearLayout>
```

Design | Text

Preview

Pixel 2 | 28 | 22%



LinearLayout (vertical orientation)

TextView

LinearLayout (horizontal orientation)

Button

Widgets in the View Hierarchy

- A **widget** is a user interface component, i.e. a building block of a layout.
  - Widgets can show text or graphics, interact with the user, or arrange other widgets on screen
- The widgets of each layout form a hierarchy of **`View`** objects (“view hierarchy”)
- Elements of the view hierarchy can be
  - visible UI elements
    - e.g. **`TextView`**s, **`Button`**s etc.
  - **`ViewGroup`**s (i.e. widgets containing and arranging other widgets in particular ways)
    - e.g. **`LinearLayout`**, **`FrameLayout`**, **`TableLayout`**, **`RelativeLayout`**
      - e.g. **`LinearLayout`** places the contained widgets vertically or horizontally next to each other
- Attributes of the XML elements indicate how the widgets shall be configured.

Widget Attributes

- **`layout_width`, `layout_height`**: Widget size
  - **`match_parent`**: as big as parent widget
  - **`wrap_content`**: as big as required by child widgets
- **`padding`**: Space added around widget
  - unit **`dp`**: density-independent pixels

activity_main.xml | MainActivity.java

```
<LinearLayout xmlns:android="http://schemas.android.com/a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:text="@string/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button"/>

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button"/>

    </LinearLayout>

</LinearLayout>
```

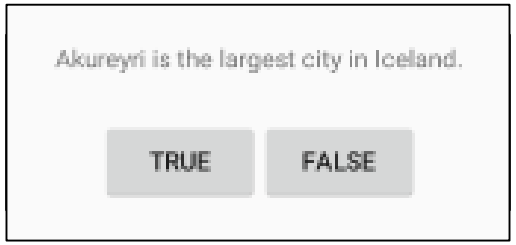
Design | Text

Preview

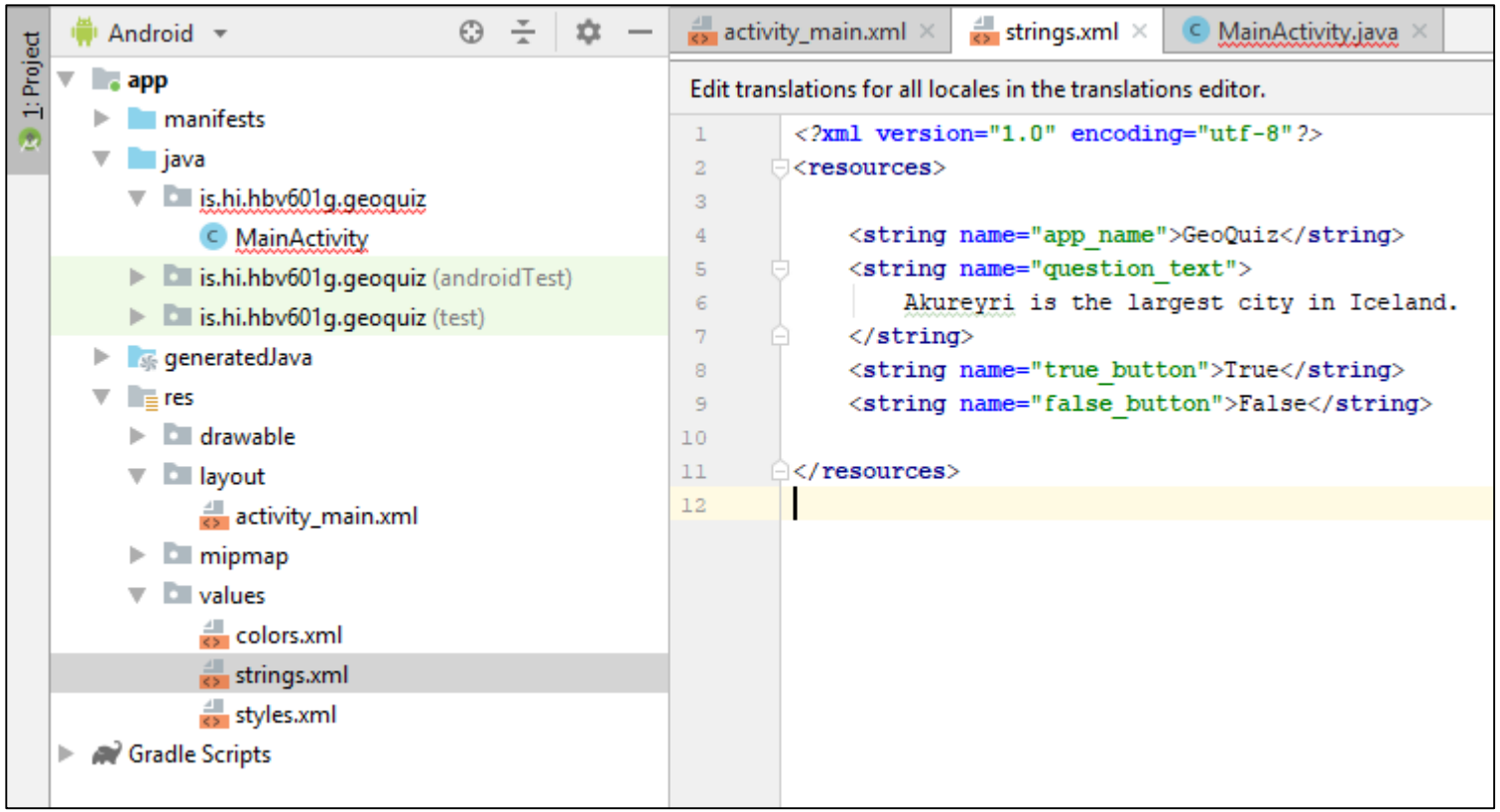
Palette

Pixel 2 | 28 | 22%

Defining String Resources



- **`text`** attributes of widgets can contain
  - literal text, e.g. **`"Text"`**
  - better: References to string resources
    - Format: **`@string/`*`name`***
- Strings are defined in a strings file
  - Default location: **`app/src/main/res/values/strings.xml`**
  - Each string is identified by **`name`** in a **`string`** element

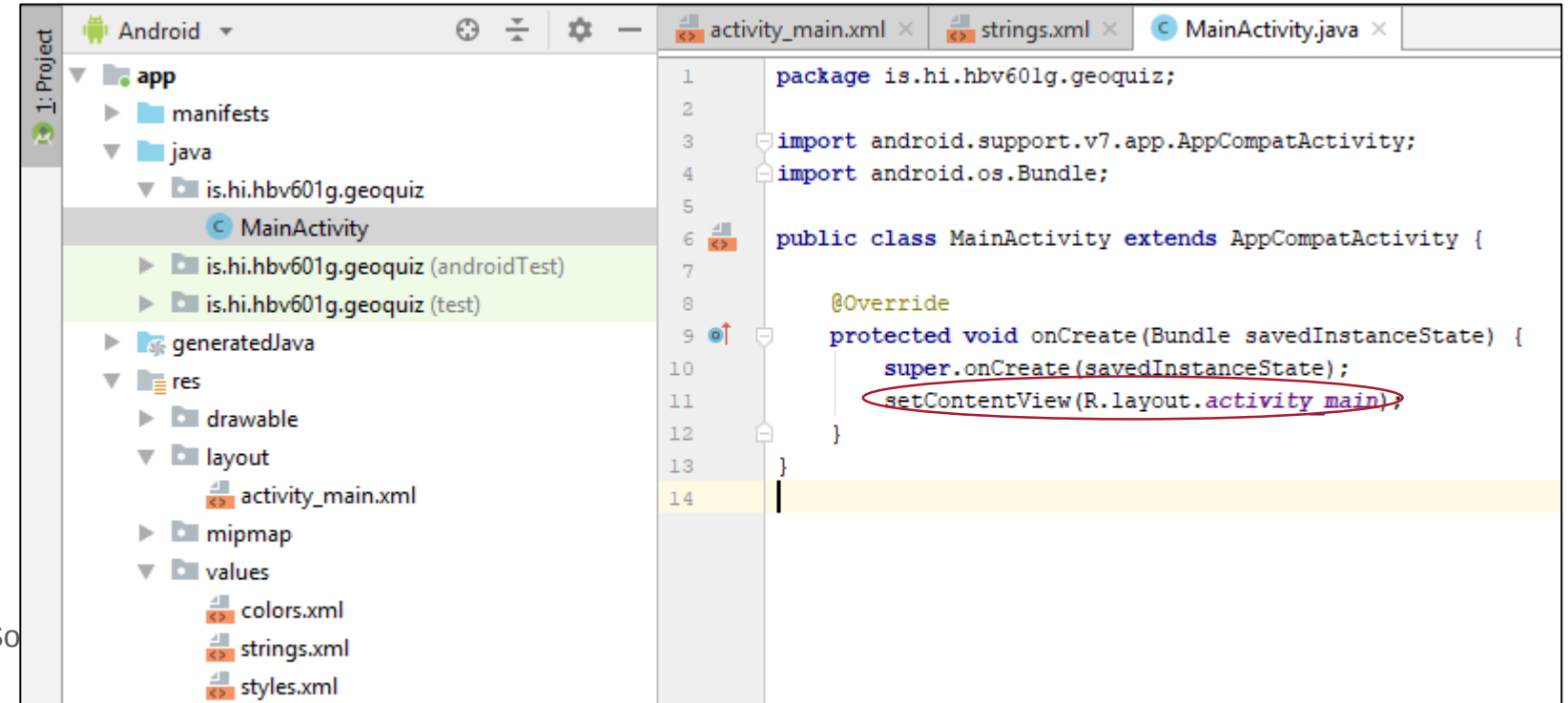


Referring to a Layout From an Activity

- When an instance of an activity is created, its **`onCreate`** method is called

- To indicate which user interface the activity shall manage, we pass the ID of the desired layout to the **`setContentView`** method
  - here: **`setContentView(R.layout.activity_main)`**

- This “inflates” the layout
  - i.e. parses the XML layout file and creates according Java objects for all widgets



Resources

- A resource is any piece of an app that is not code
  - e.g. images, XML files etc.
- Resources are stored in subdirectories of app/src/main/res, e.g.
  - …/layout/activity_main.xml
  - …/values/strings.xml
- In Java code, resources are referenced using constants defined in the automatically-generated Java class **R**

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY. */

package is.hi.hbv601g.geoquiz;

public final class R {

  public static final class layout {
    public static final int activity_main=0x7f040019;
    // ...
  }

  public static final class string {
    public static final int false_button=0x7f0a0012;
    public static final int true_button=0x7f0a0015;
    // ...
  }

  public static final class id { ... }

  // ...

}
```

Note: inner class

In-Class Quiz #4: Basic Android Components

- **Fill the blanks in the following sentences with the words (A)ctivity, (L)ayout, (R)esource and (W)idget:**

a) A _____ manages the user's interaction with the UI defined by a _____.

b) A _____ defines a set of _____ and their screen positions.

c) A view hierarchy contains the _____ of one _____.

d) The **`setContentView`** method of a _____ inflates the given _____.

e) Inflating a _____ means creating Java objects for all _____ declared in it.

f) A _____ is any piece of an app that is not executable code.

g) The auto-generated **`R`** class contains references to all _____ and _____ defined in XML files under **`app/src/main/res/`**.

Creating IDs for Widgets

- To add behavior to our buttons, we need to refer to them in Java
  - Define IDs for them by adding **`id`** attributes to the XML layout file

activity_main.xml | strings.xml | MainActivity.java

```xml
<LinearLayout xmlns:android="http://schemas.android.com/
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:text="@string/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button"/>

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button"/>

    </LinearLayout>

</LinearLayout>
```

Design | Text

Preview

Pixel 2 | 28 | 22%

Akureyri is the largest city in Iceland.

TRUE FALSE

The **+** sign in **`@+id`** indicates we are *defining* the ID

`@string` (without **+**) indicates we are *referencing* the ID

Referring to Widgets from an Activity

- Recap:
  - Defining the Button widgets with the attributes **`android:id="@+id/name"`** in the layout file created IDs for the buttons in the **`R.id`** class
  - **`setContentView`** inflates the layout, i.e. parses the XML layout file and creates according Java objects for all widgets
  - Both processes (generating the **`R`** class and generating the widget classes) are automatic!

```java
package is.hi.hbv601g.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

  private Button mTrueButton;
  private Button mFalseButton;

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

    mTrueButton = (Button) findViewById(R.id.true_button);
    mFalseButton = (Button) findViewById(R.id.false_button);
  }
}
```

- To obtain the generated widget classes, pass ID of desired widget to the **`findViewById`** method and cast the returned **`View`** to expected type
  - e.g. `mTrueButton = (Button) findViewById(R.id.true_button);`
    - Note: Android naming convention: Class attributes (“member variables”) are prefixed with **`m`**

Setting Listeners

- Android apps are event-driven
  - i.e. after starting, they wait ("listen") for user- or system-initiated events
- Android provides listener interfaces for various events, e.g.
  - **`View.OnClickListener`** for short clicks/touches on widgets
  - **`View.OnLongClickListener`** for long clicks/touches on widgets
- You provide the listener implementation
  - i.e. code saying what happens upon the event

```java
package is.hi.hbv601g.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

  private Button mTrueButton;
  private Button mFalseButton;

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

    mTrueButton = (Button) findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // to be implemented
      }
    });

    mFalseButton = (Button) findViewById(R.id.false_butt
  }
}
```

Note: Anonymous inner class

Anonymous Inner Classes

- Example: **`setOnClickListener`** expects an **`OnClickListener`** instance
  - We could write an **`OnClickListener`** implementation in a regular Java class (e.g. **`TrueButtonOnClickListener.java`**) as a separate file somewhere in our project…
    - …store an instance of it in a local variable, such as **`trueButtonOnClickListener = new TrueButtonOnClickListener()`**
    - …and then pass this instance to **`setOnClickListener`**, i.e. **`mTrueButton.setOnClickListener(trueButtonOnClickListener)`**
  - But it's more compact to do all that (implementation, instantiation, param passing) at once:

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

It's OK for the class to be anonymous since we are referring to it only here

- Advantages
  - Avoiding lots of small listener class files cluttering up our project
  - Maintaining the listener implementation exactly where it is needed
    - "when this event happens to this widget, this code will be executed"

Making Toasts

- A **toast** is a short message display that does not require any user interaction.
- To create a toast, call the following method from the Toast class:
  `public static Toast makeText(Context context, int resId, int duration)`
  - **`Context`** parameter usually provides reference to current activity
  - **`resId`** refers to the string that shall be displayed (needs to be defined in **`strings.xml`**)
  - **`duration`** indicates how long to show toast (**`Toast.LENGTH_SHORT`** or **`_LONG`**)
- To show the toast, call its **`show`** method

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

We cannot use just **`this`**, since that would refer to the anonymous class!

`makeText` is a static factory method returning a **`Toast`** instance that we now show

Complete Activity with Event Listeners

```java
public class MainActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.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();
            }
        });

    }
}
```

/java/is/hi/hbv601g/geoquiz/MainActivity.java

Recap: Corresponding Layout and String Resource File

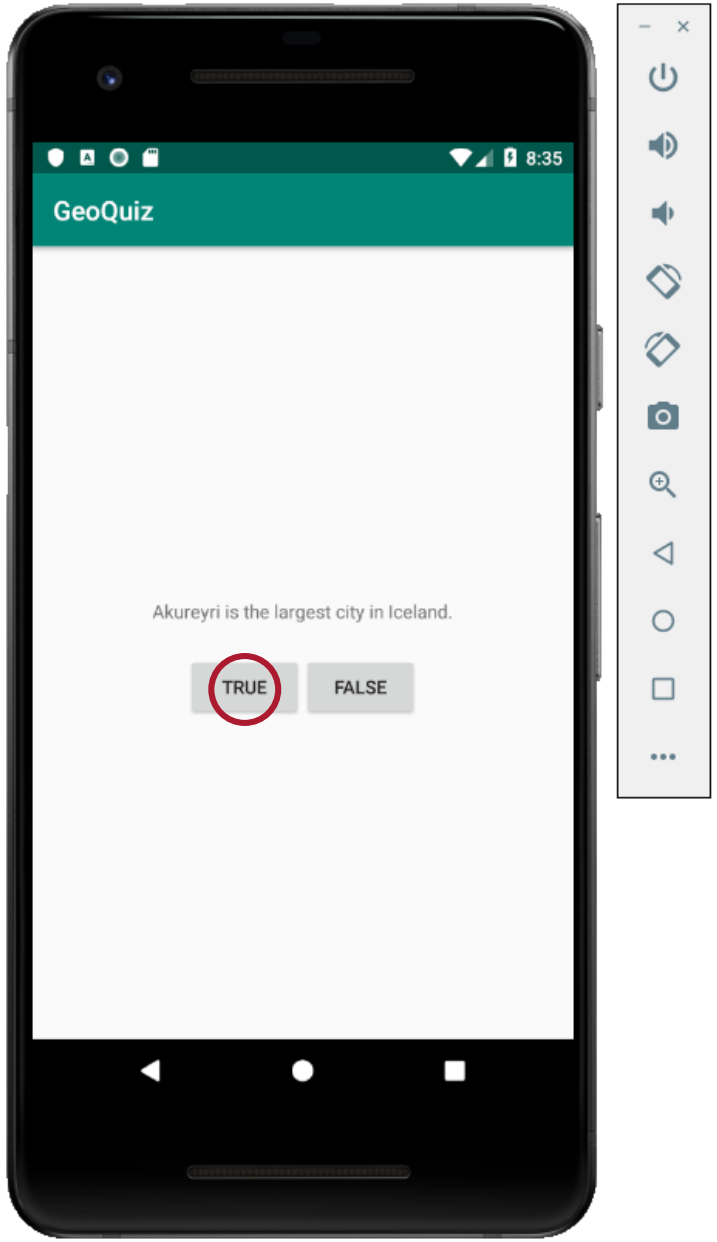
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:text="@string/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button"/>

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button"/>

    </LinearLayout>

</LinearLayout>
```

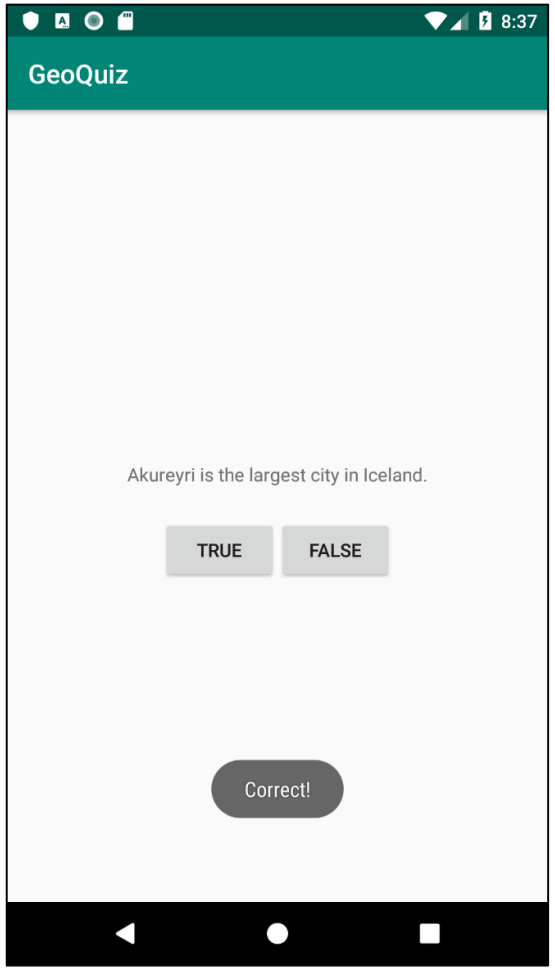
/res/layout/activity_main.xml

/res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">GeoQuiz</string>
    <string name="question_text">
        Akureyri is the largest city in Iceland.
    </string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>

</resources>
```

Resulting Application





- …obviously requires some more logic ☺