# Android 101

Maria Zlatkova

CS50 for MBAs

# Java

- Programming language used in Android Development
- Object Oriented
  - We define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.

# Objects

- Real-world objects share two characteristics
  - State
  - Behavior
- Dogs
  - States: Name, Color, Breed, Hungry
  - Behavior: Bark, Wag tail, Fetch ball
- Cars
  - States: Model, Year, Max speed, Current Speed
  - Behavior: Accelerate, Brake, Reverse
- Java Objects have
  - Fields (State)
  - Methods (Behavior)

# Java Classes

- A class is a template **that describes the behaviors/states of an object**

```
public class Dog {
    String breed;
    int age;

    public Dog (String breed, int age) {
        this.breed = breed;
        this.age = age;
    }
    public void growOld(int years) {
        age = age + years;
    }
    public double getAge() {
        return age;
    }
    public double getBreed() {
        return breed;
    }

}
```

# Classes

```
public class Main {
        public static void main(String[] args) {
                Dog dog = new Dog(”Retriever", 2);
                dog.growOld(5);
                System.out.println(”The dog is now " + dog.getAge() + “years
                old.);
        }
}
```

# Inheritance

- Different Objects may share common characteristics with each other.
- Object-Oriented Programming allows classes to inherit commonly used states and behaviors from other classes.

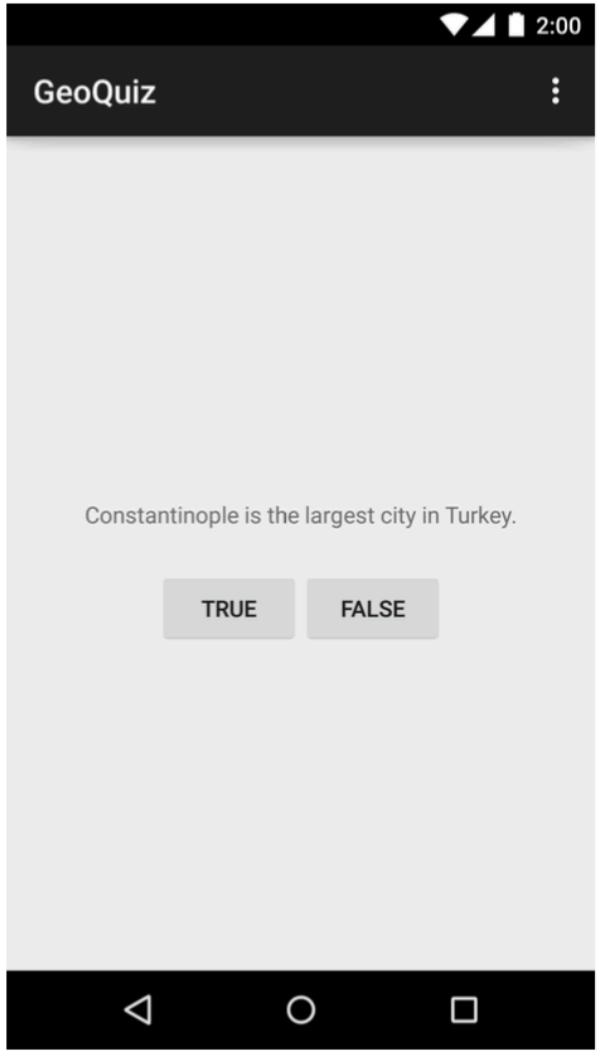
# What you'll need

- *Android SDK*
  - the latest version of the Android SDK
- *Android SDK tools and platform-tools*
  - tools for debugging and testing your apps
- *A system image for the Android emulator*
  - lets you create and test your apps on different virtual devices
- https://developer.android.com/sdk/.
- Java Development Kit (JDK7)
  - download from http://www.oracle.com

# Android Jargon

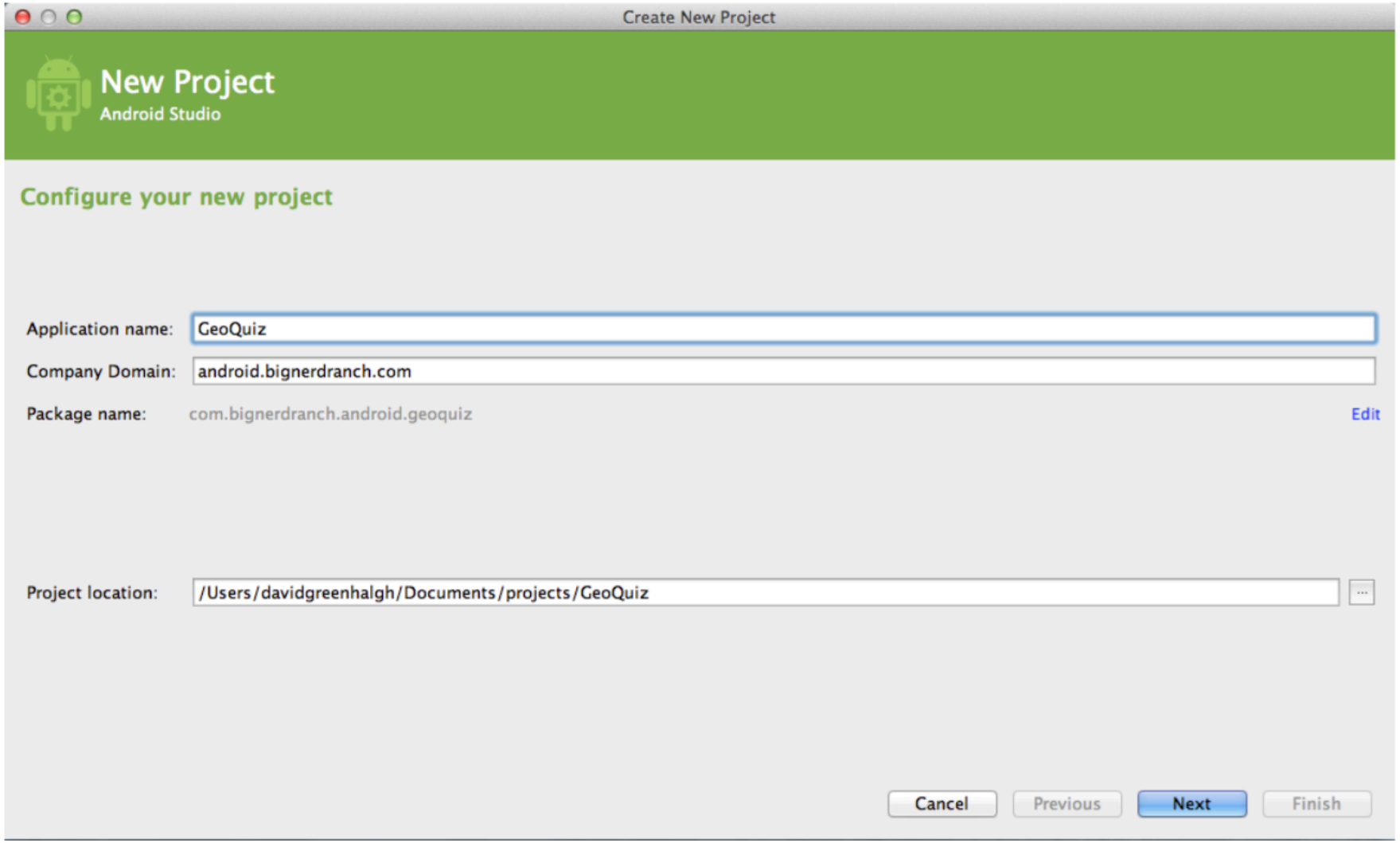
- Activity
  - A single screen within an application.
  - Usually limited to one single “activity” (e.g. Viewing, Adding, Editing)
- View
  - An object that draws to a rectangular area on the screen and handles clicks, keystrokes, and other interaction events.
- Intent
  - A “message” you can use to launch or communicate with other applications/activities.
- Manifest File
  - An XML file that each application must define that gives information about the application itself.
    - Version
    - Activities

# Let’s make an app!

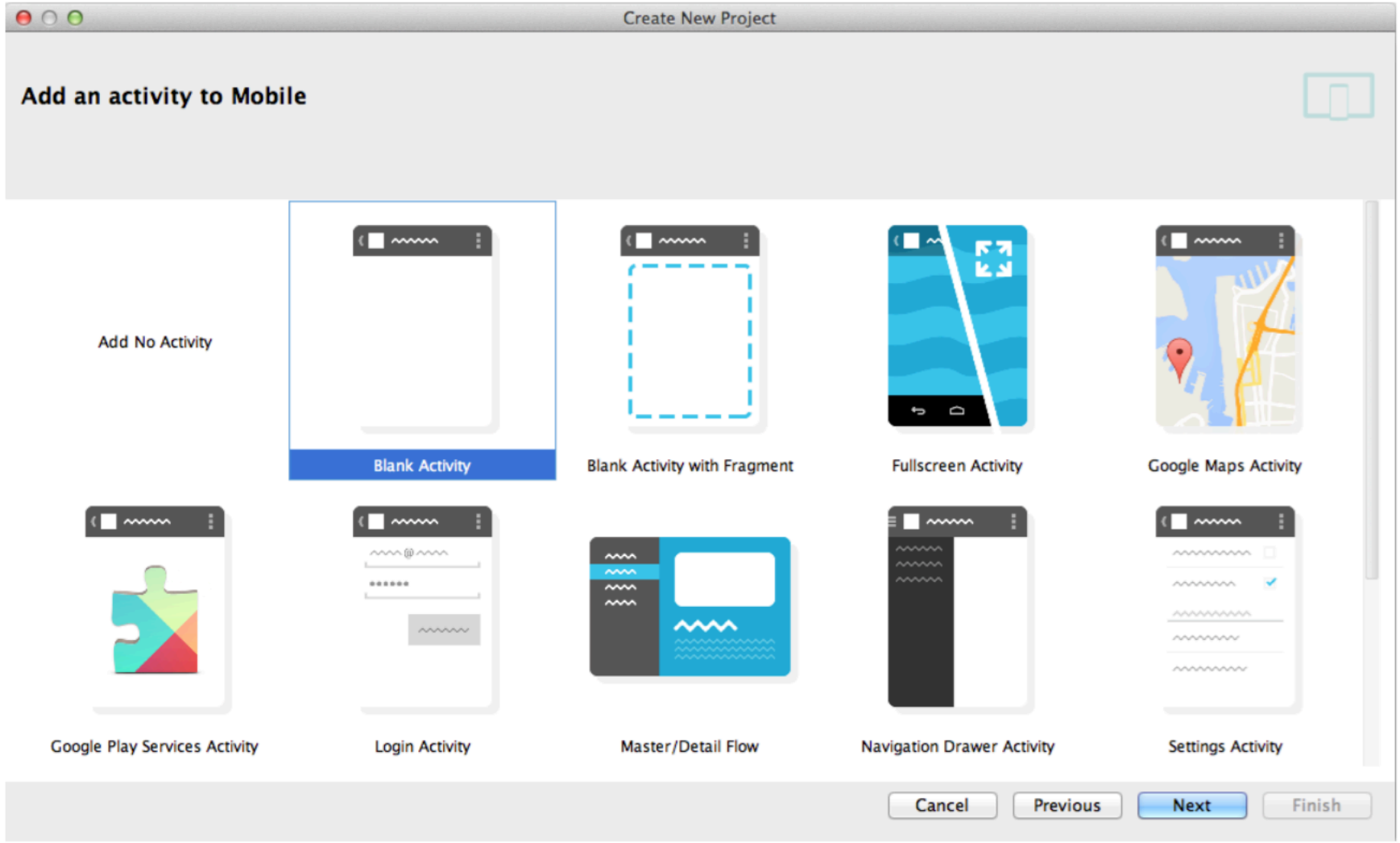



Material taken from

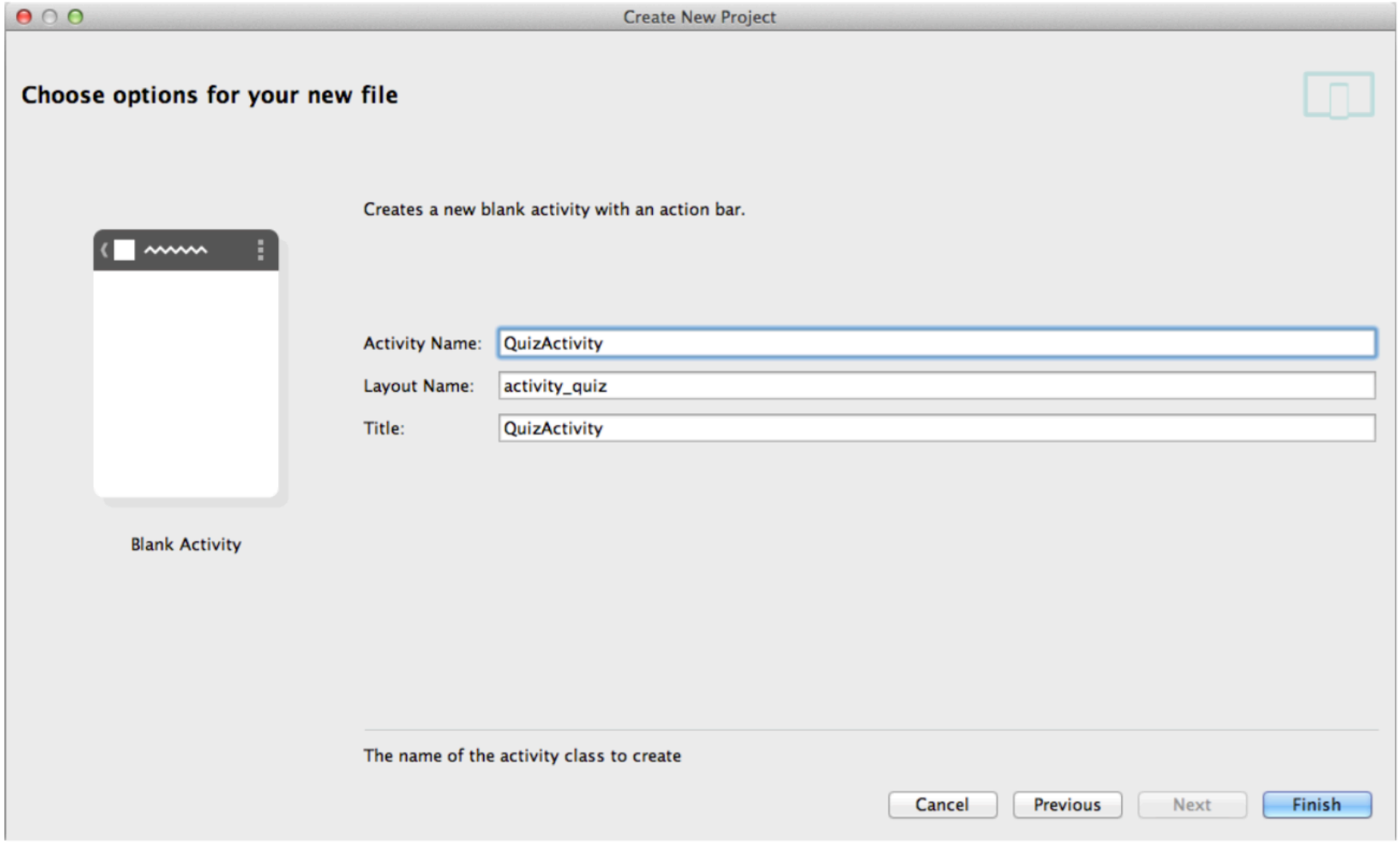
# Creating a project



# Selecting a new Activity



# Activity Details



# Android Studio

activity_quiz.xml – [app] – GeoQuiz – [~/Documents/projects/GeoQuiz] – Android Studio (Beta) 0.8.2

GeoQuiz (~/Documents/projects/GeoQuiz)
- .idea
- app
  - build
  - libs
  - src
    - androidTest
    - main
      - java
      - res
        - drawable-hdpi
        - drawable-mdpi
        - drawable-xhdpi
        - drawable-xxhdpi
        - layout
          - activity_quiz.xml
        - menu
        - values
        - values-w820dp
      - AndroidManifest.xml
  - .gitignore
  - build.gradle
  - proguard-rules.pro
- build

```
<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:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".QuizActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Messages: Gradle Build | Gradle Sync

```
Gradle tasks [:app:generateDebugSources]
:app:preBuild
:app:preDebugBuild
:app:checkDebugManifest
:app:preReleaseBuild
:app:prepareComAndroidSupportAppcompatV71910Library
:app:prepareDebugDependencies
:app:compileDebugAidl
:app:compileDebugRenderscript
:app:generateDebugBuildConfig
:app:generateDebugAssets UP-TO-DATE
:app:mergeDebugAssets
```

Gradle build finished in 4 sec (moments ago)

# The User Interface & default activity layout

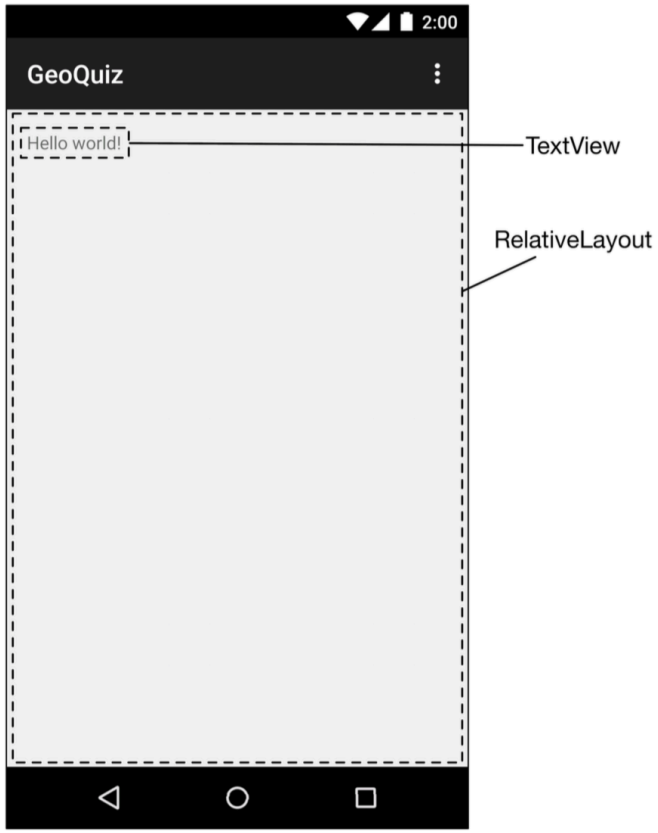
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".QuizActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

# What constitutes our app at first?

- The default activity layout defines two *widgets*: a **RelativeLayout** and a **TextView**.
- *Widgets* are the building blocks you use to compose a user interface. A widget can show text or graphics, interact with the user, or arrange other widgets on the screen. Buttons, text input controls, and checkboxes are all types of widgets.
- The Android SDK includes many widgets that you can configure to get the appearance and behavior you want. Every widget is an instance of the **View** class or one of its subclasses (such as **TextView** or **Button**).

# Default Widgets

Figure 1.9 Default widgets as seen on screen

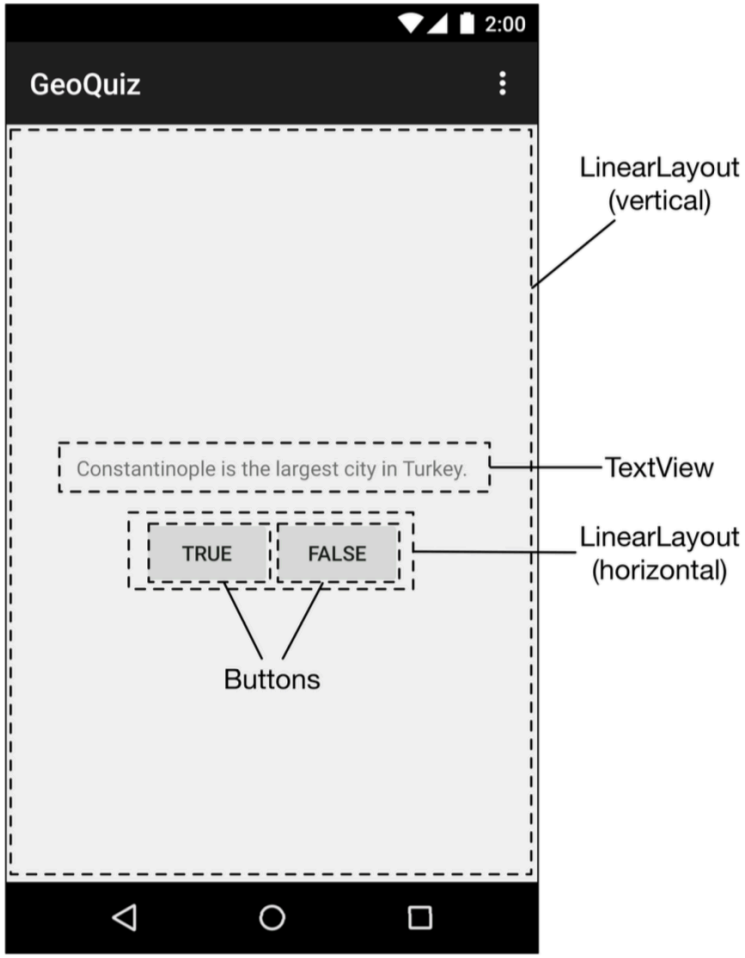


# What widgets do we need for QuizActivity?

- a vertical **LinearLayout**
- a **TextView**
- a horizontal **LinearLayout**
- two **Buttons**

# QuizActivity Widgets

Planned widgets as seen on screen



# Let's define these widgets in activity_quiz.xml

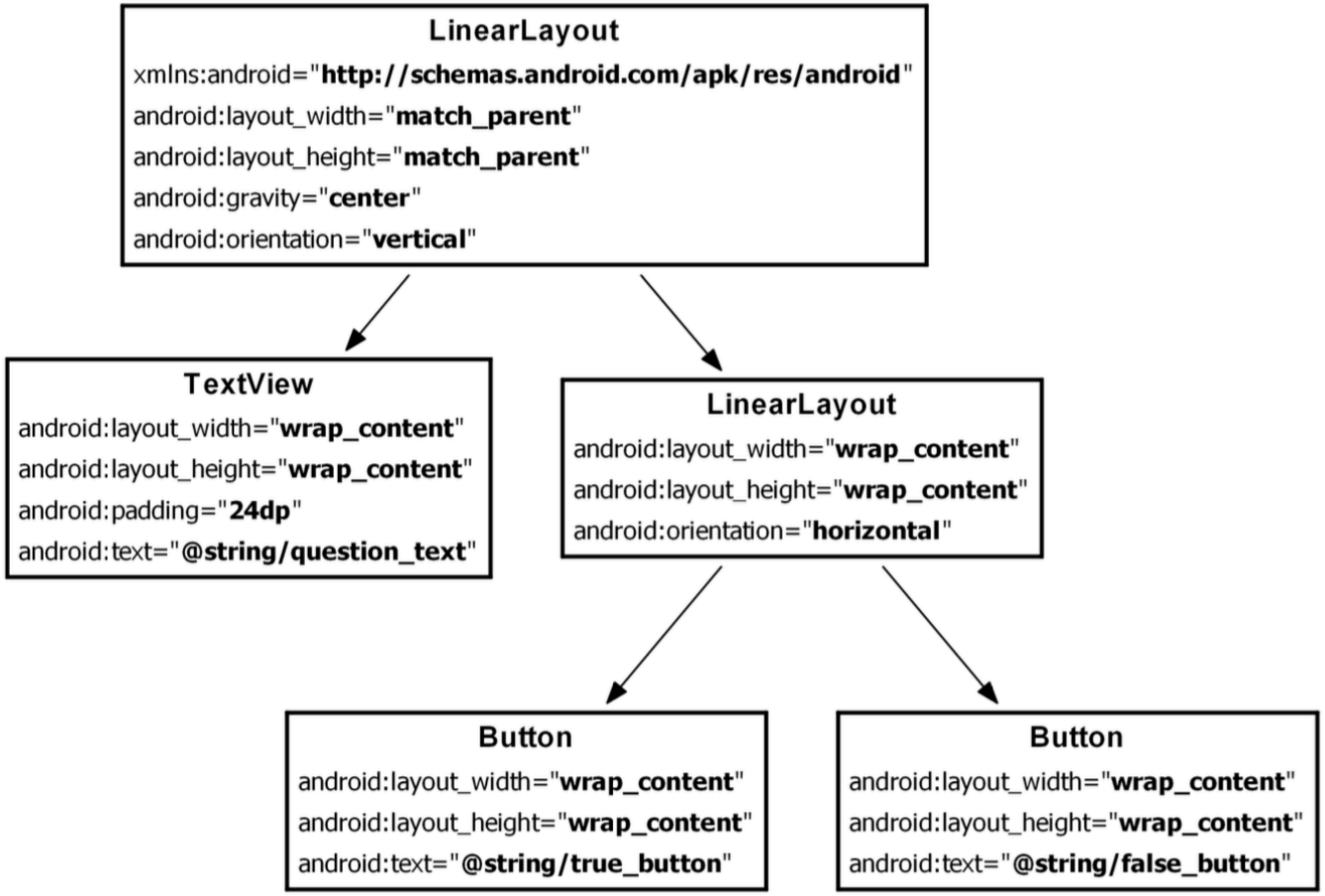
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical" >

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

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

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

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

  </LinearLayout>

</LinearLayout>
```

# What does this all mean?

- Every widget has a corresponding XML element. The name of the element is the type of the widget.
- Each element has a set of XML *attributes*.
  - Each *attribute* is an instruction about how the widget should be configured.

# The View Hierarchy

- Your widgets exist in a hierarchy of **View** objects called the *view hierarchy*.



# The View Hierarchy

- The root element of this layout's view hierarchy is a **LinearLayout**. As the root element, the **LinearLayout** must specify the Android resource XML namespace at http://schemas.android.com/apk/res/android.
- **LinearLayout** inherits from a subclass of **View** named **ViewGroup**. A **ViewGroup** is a widget that contains and arranges other widgets. You use a **LinearLayout** when you want widgets arranged in a single column or row.
  - Other **ViewGroup** subclasses are **FrameLayout**, **TableLayout**, and **RelativeLayout**.
- When a widget is contained by a **ViewGroup**, that widget is said to be a *child* of the **ViewGroup**. The root **LinearLayout** has two children: a **TextView** and another **LinearLayout**. The child **LinearLayout** has two **Button** children of its own.

# Widget Attributes

- **android:layout_width and android:layout_height**
  - The android:layout_width and android:layout_height attributes are required for almost every type of widget. 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

# Widget Attributes

- For the root **LinearLayout**, the value of both the height and width attributes is match_parent. The **LinearLayout** 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 your layout have their widths and heights set to wrap_content.
- The **TextView** is slightly larger than the text it contains due to its android:padding="24dp" attribute. This attribute tells the widget to add the specified amount of space to its contents when determining its size. You are using it to get a little breathing room between the question and the buttons.

# Widget Attributes

- **android:orientation**
  - The android:orientation attribute on the two **LinearLayout** widgets determines whether their children will appear vertically or horizontally. The root **LinearLayout** is vertical; its child **LinearLayout** is horizontal.
  - The order in which children are defined determines the order in which they appear on screen. In a vertical **LinearLayout**, the first child defined will appear topmost. In a horizontal **LinearLayout**, the first child defined will be leftmost. (Unless the language of the device is a language that runs right-to-left, such as Arabic or Hebrew. In that case, the first child will be rightmost.)

# Widget Attributes

- **android:text**
  - The **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 hard-coded string, like android:text="True", but it is usually not a good idea. Placing strings into a separate file and then referencing them is better because it makes localization easy.
  - The string resources you are referencing in activity_quiz.xml do not exist yet. Let's fix that.

# Creating string resources

- Every project includes a default strings file named strings.xml.
  In the Project tool window, find the app/res/values directory, reveal its contents, and open strings.xml.
- The template has already added a few string resources for you. We remove the unused string named hello_world and add the three new strings that your layout requires.
- Now, whenever you refer to @string/false_button in any XML file in the GeoQuiz project, you will get the literal string "False" at runtime.

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

    <string name="hello_world">Hello world!</string>
    <string name="question_text">
        Constantinople is the largest city in Turkey.
    </string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="action_settings">Settings</string>
</resources>
```

# String Resources

- Although 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. As long as the file is located in res/values/, has a resources root element, and contains child string elements, your strings will be found and used appropriately.

# From Layout XML to View Objects

- How do XML elements in activity_quiz.xml become **View** objects? The answer starts in the **QuizActivity** class.
- When you created the GeoQuiz project, a subclass of **Activity** named **QuizActivity** was created for you. The class file for **QuizActivity** is in the app/java directory of your project. The java directory is where the Java code for your project lives.

```
package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;


public class QuizActivity extends AppCompatActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.quiz, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

- This file has three **Activity** methods: **onCreate(Bundle)**, **onCreateOptionsMenu(Menu)**, and **onOptionsItemSelected(MenuItem)**.
- The **onCreate(Bundle)** method is called when an instance of the activity subclass is created. When an activity is created, it needs a user interface to manage. To get the activity its user interface, you call the following **Activity** method:
  - **public void setContentView(int layoutResID)**
  - 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.
  - To access a resource in code, you use its *resource ID*. The resource ID for your layout is **R.layout.activity_quiz.**

# Using Widgets

- In an activity, you can get a reference to an inflated widget by calling the following **Activity** method: public View findViewById(int id)
  - This method accepts a resource ID of a widget and returns a **View** object.

- In QuizActivity.java, use the resource IDs of your buttons to retrieve the inflated objects and assign them to your member variables. Note that you must cast the returned **View** to **Button** before assigning it.

```
public class QuizActivity 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);
    }

...
}
```

# Setting listeners

- When your application is waiting for a specific event, we say that it is “listening for” that event. The object that you create to respond 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, so you do not have to write your own. In this case, the event you want to listen for is a button being pressed (or “clicked”), so your listener will implement the **View.OnClickListener** interface.

# On Click Listener

```
    ...

    @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) {
                // Does nothing yet, but soon!
            }
        });

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

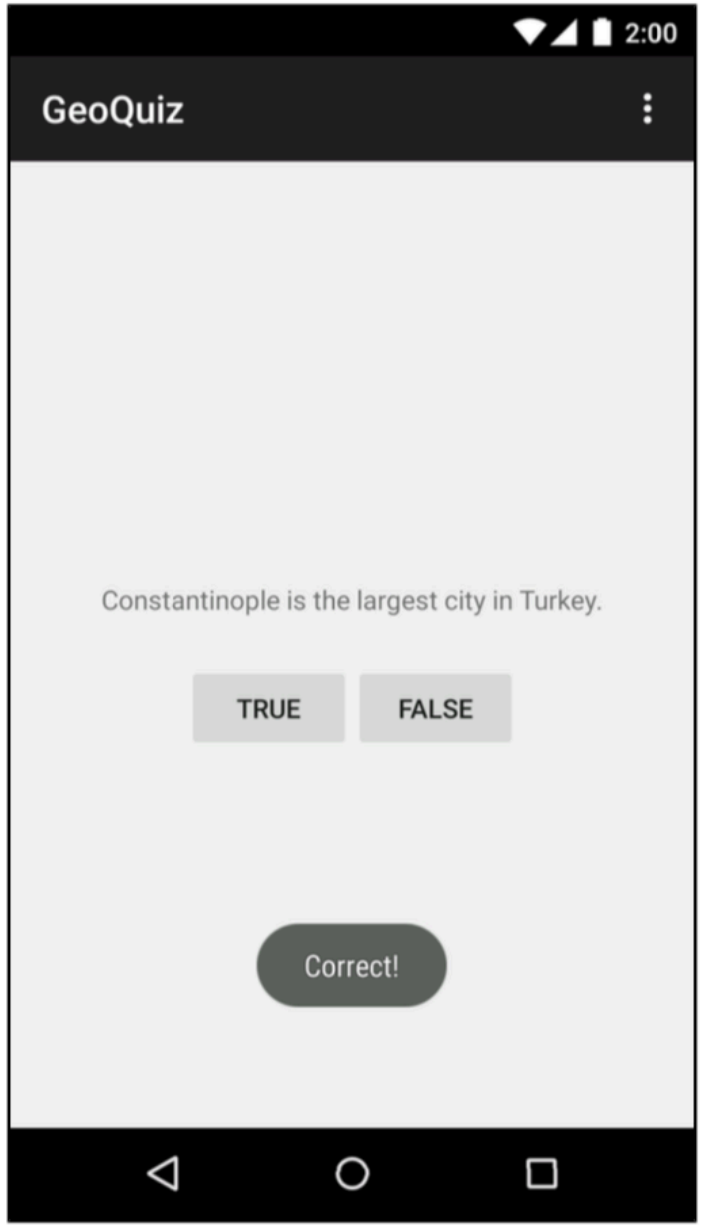
# On Click Listener

```
...

    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Does nothing yet, but soon!
        }
    });

    mFalseButton = (Button) findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Does nothing yet, but soon!
        }
    });
}
```

# Toasts!

- You are going to have a press of each button trigger 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. You are going to make toasts that announce whether the user answered correctly or incorrectly

# Toasts


2:00
GeoQuiz
Constantinople is the largest city in Turkey.
TRUE
FALSE
Correct!

# Toast Strings

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

  <string name="question_text">Constantinople is the largest city in Turkey.</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>
  <string name="action_settings">Settings</string>
</resources>
```

# Toasts

- To create a toast, you call the following method from the **Toast** class:
- public static Toast makeText(Context context, int resId, int duration)
- The **Context** parameter is typically an instance of **Activity** (**Activity** is a subclass of **Context**). The second parameter is the resource ID of the string that the toast should display. The **Context** is needed by the **Toast** class to be able to find and use the string's resource ID. The third parameter is one of two **Toast** constants that specify how long the toast should be visible.
- After you have created a toast, you call **Toast.show()** on it to get it on screen.

# Toasts

```
...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
            // Does nothing yet, but soon!
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
            // Does nothing yet, but soon!
    }
});
```

# Running on the Emulator

- To run an Android application, you need a device – either a hardware device or a *virtual device*. Virtual devices are powered by the Android emulator, which ships with the developer tools.
- Let's try it out!