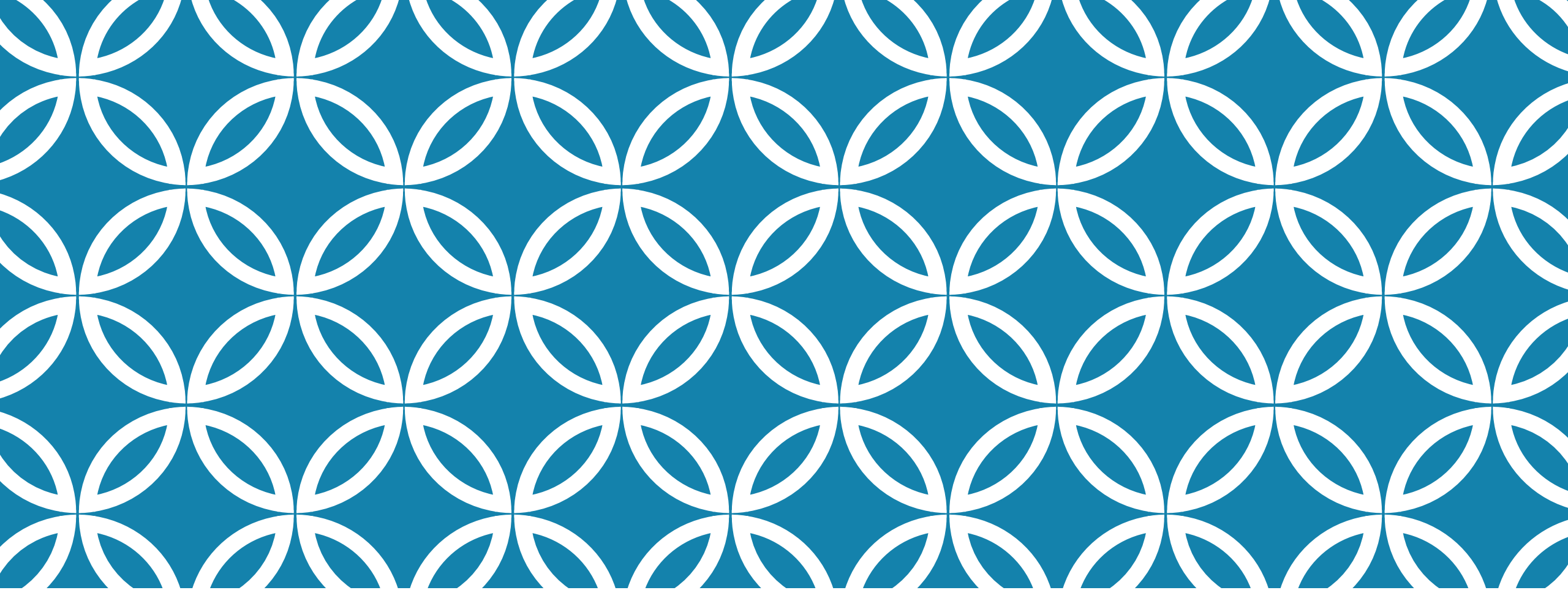


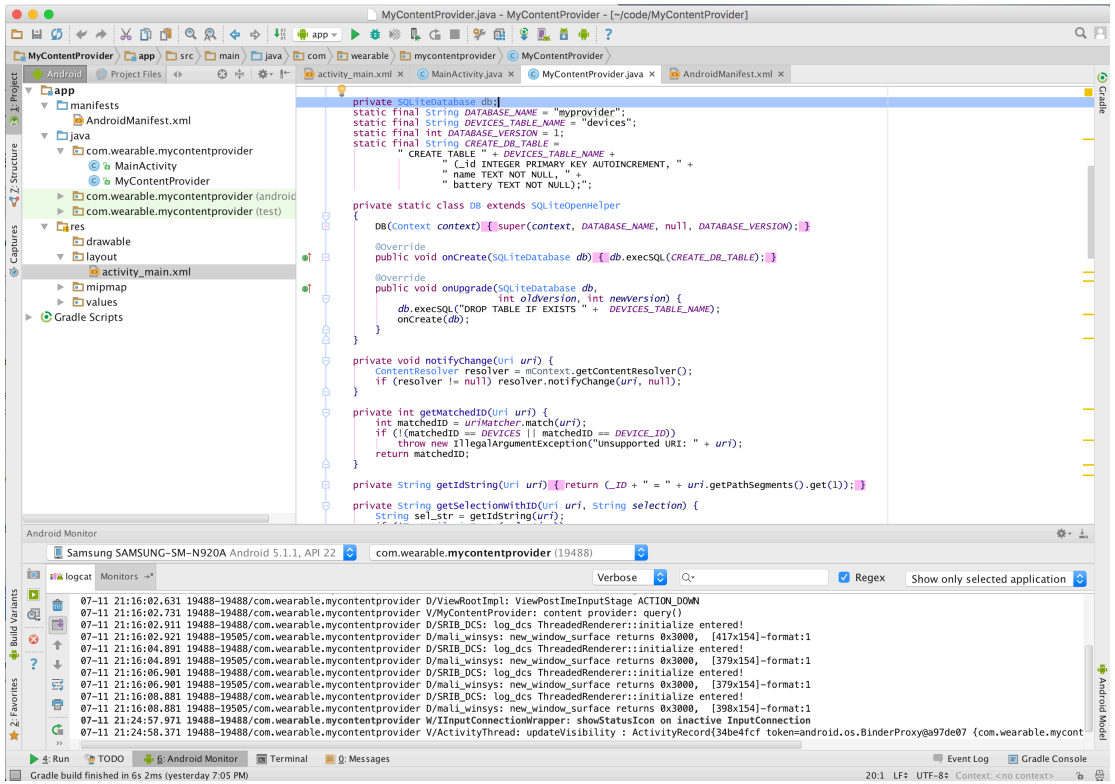
ANDROID HELLO WORLD

CS 175 Mobile Software Dev

by Dr. Angus Yeung

INSTALLING ANDROID STUDIO

The official IDE (Integrated Development Environment) for building apps on every type of Android device is Android Studio.



<1> DOWNLOAD THE SOFTWARE

You can download Android Studio from the Android official website:

http://developer.android.com/sdk/index.html

The tool supports Windows PC, Mac and Linux operating systems.

<2.1> INSTALL ANDROID STUDIO

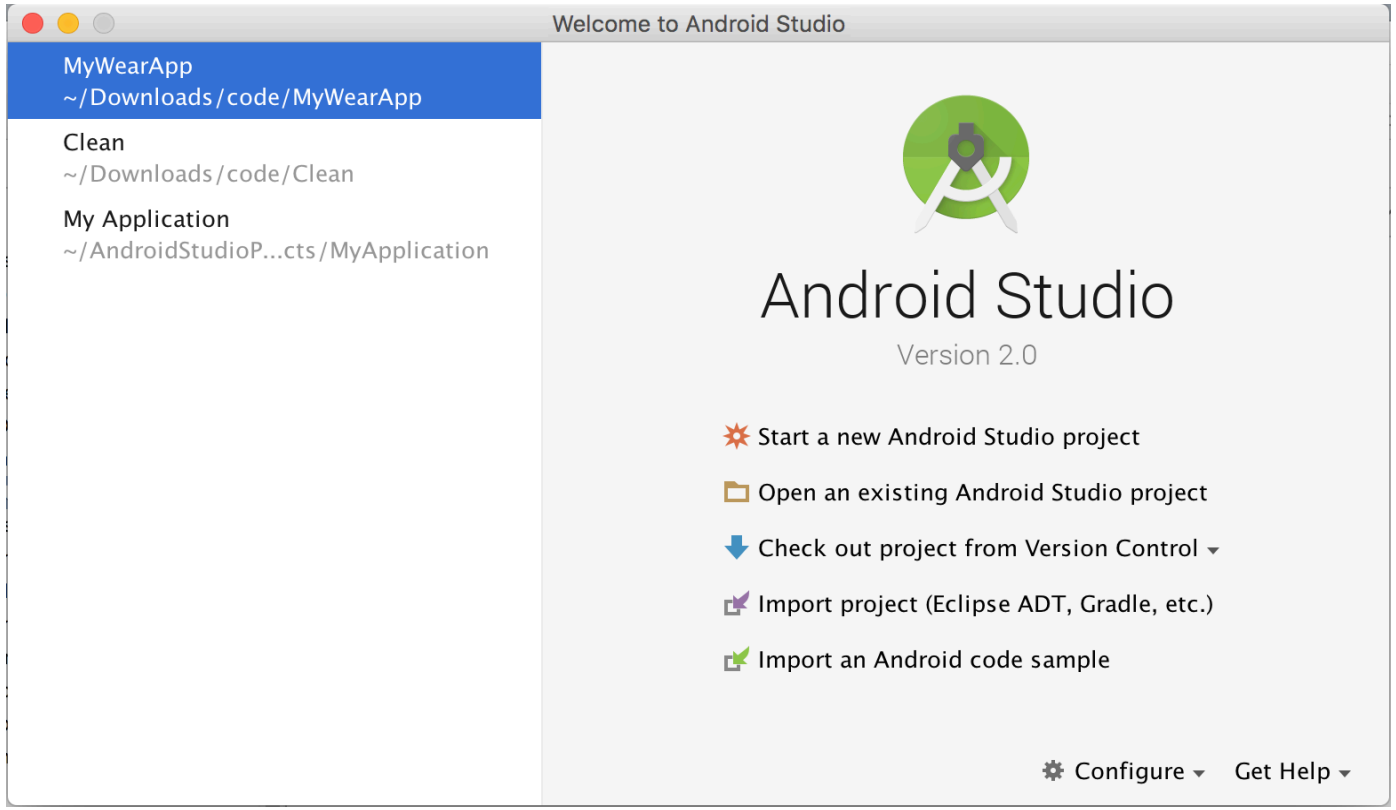
Follow the instruction provided in **Install Android Studio** web page to set up Android Studio properly on your system. Mostly the set up on your system takes just a few clicks.

If you encounter any problem during installation, refer to the Android support pages for help.

If you have older version installed on your computer, you can override the older Android Studio but import the configuration into the new version.

<2.2> LAUNCH THE APPLICATION

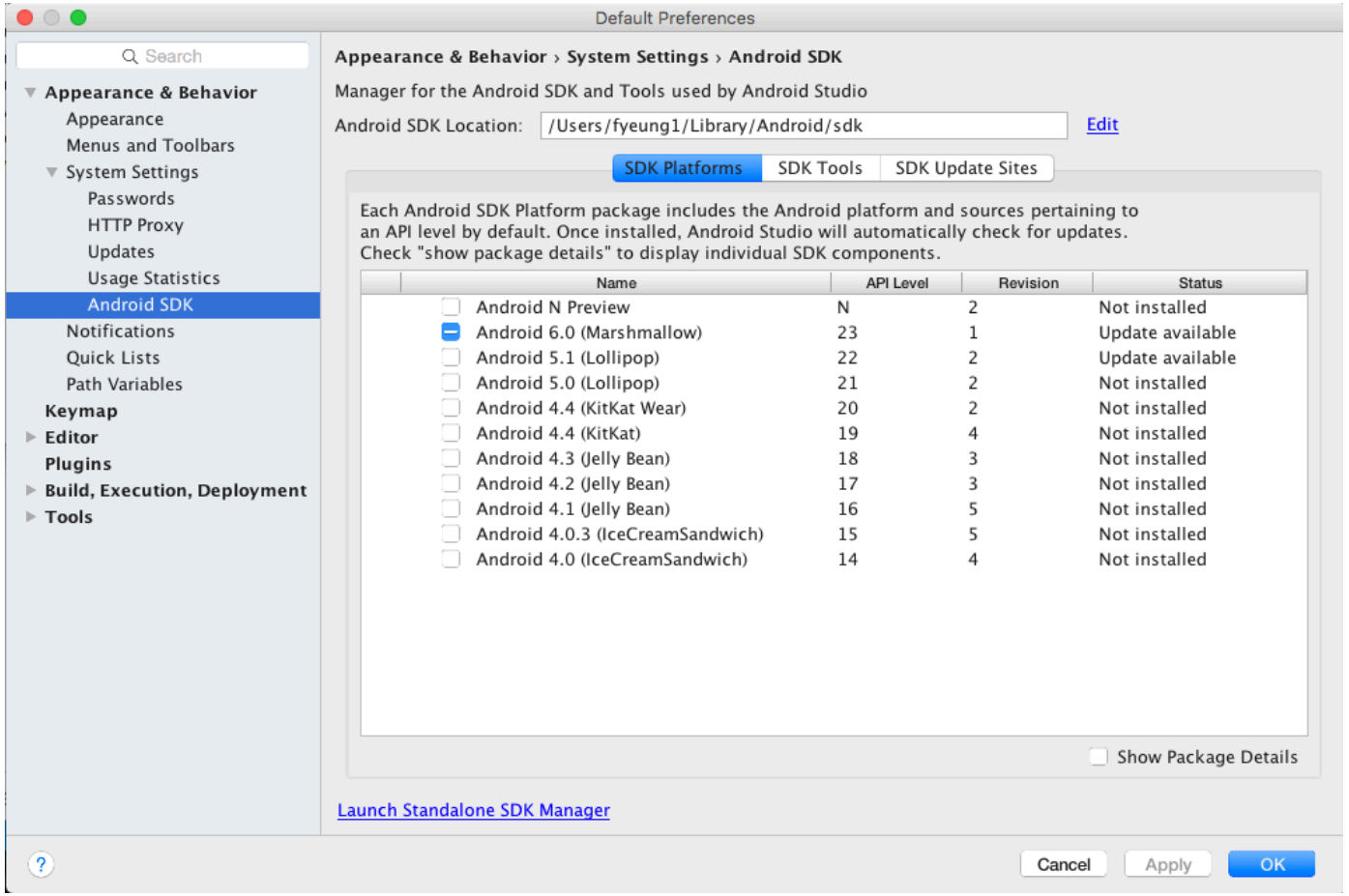
Double click the icon of Android Studio to launch the application. The **Welcome** screen will be shown.



<3.1> CONFIGURE ANDROID SDK

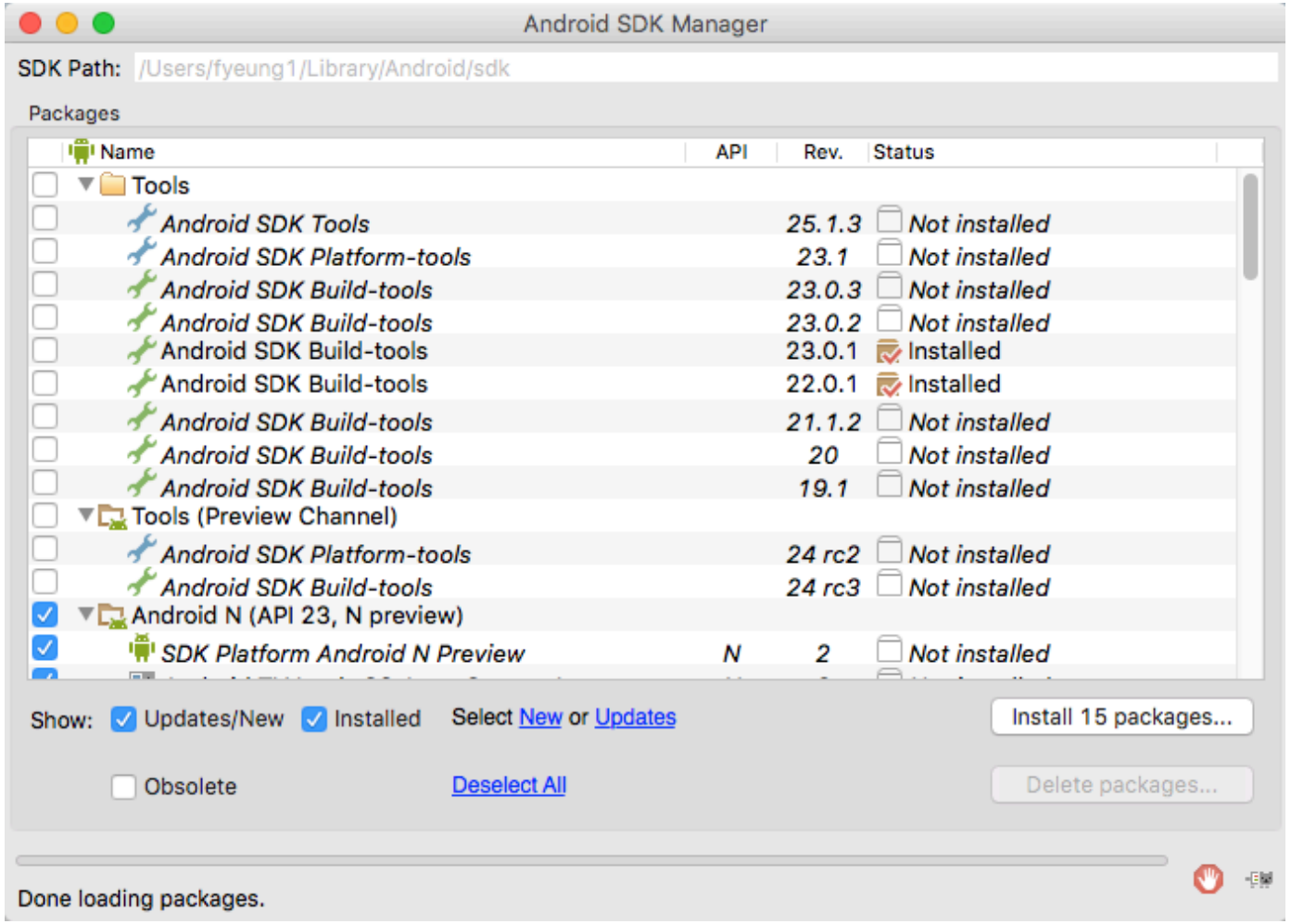
Click on the **Configure** link at the lower right of the **Welcome** screen.

You can see that the default package from Android Studio has included Android 6.0 (Marshmallow) SDK Platform at API level 23.



<3.2> LAUNCH STANDALONE SDK MANAGER

When you click on the **Launch Standalone SDK Manager** link, it will bring up a dialog for Android SDK Manager.

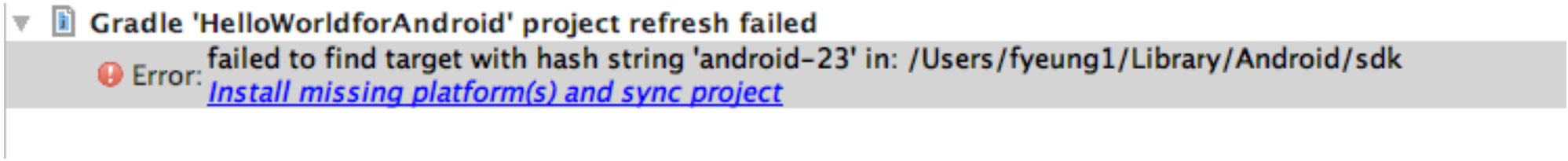


<4.1> UPDATE ANDROID SDK PACKAGE

If you there are any updates, the Android SDK Manager will prompt you to install the updates. Go ahead with the updates by following the provided instructions.

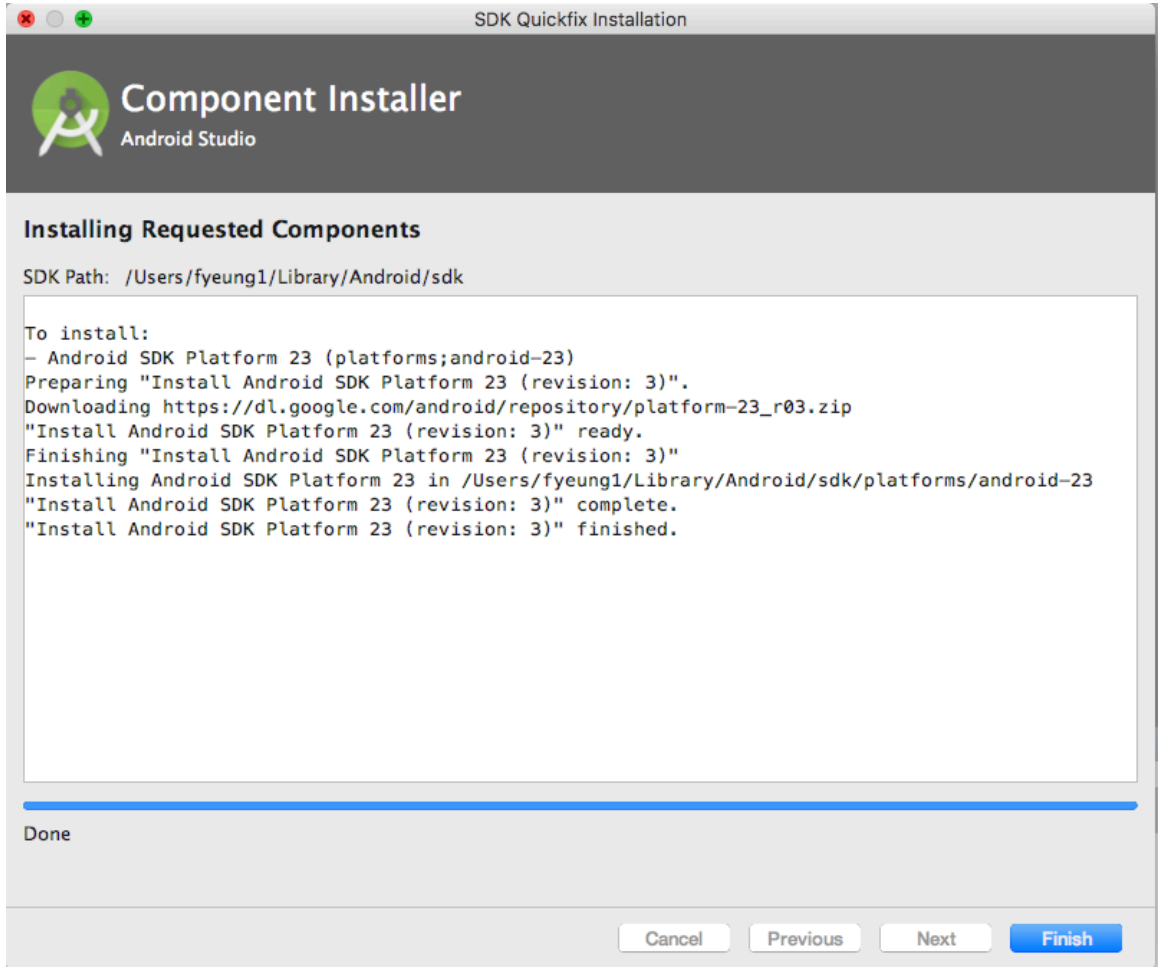
<4.2> MISSING PLATFORM COMPONENTS

Android Studio gives you very good tips when some required platform components or tools are missing in your system. Simply click on the provided link to install any required components.



<4.3> INSTALLATION OF COMPONENTS

When you are installing your Android SDK, take note of the location where the SDK is installed.



<4.4> INSTALLATION PATH

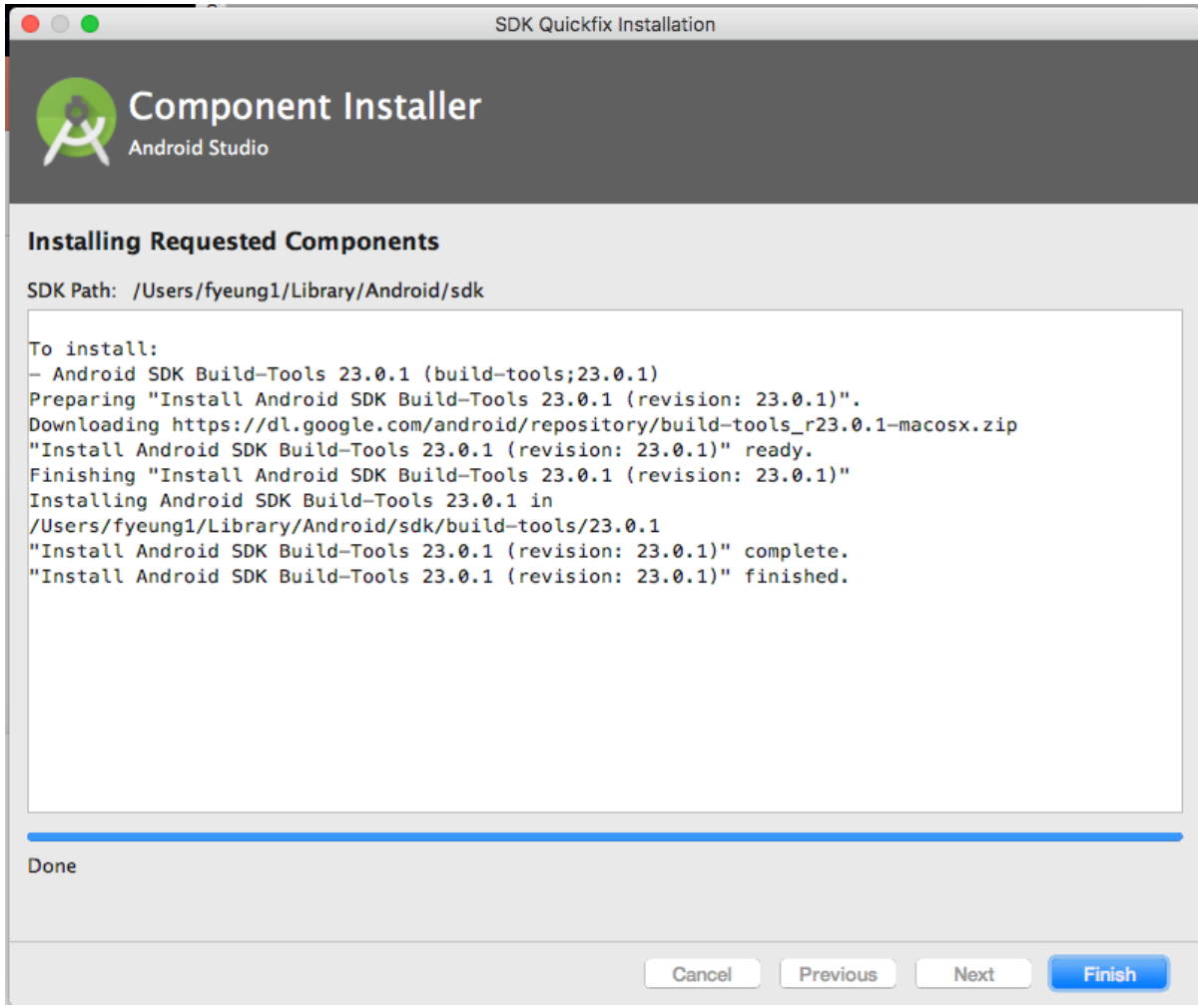
If we check out the installation path, we can see every version of Android SDK that is installed in your system.

For example, the console output below shows that we have Android SDK 23 and 26 installed.

```
fyeung1-mac08:~ fyeung1$ cd ~/Library/Android/sdk/platforms
fyeung1-mac08:platforms fyeung1$ ls
android-23 android-26
```

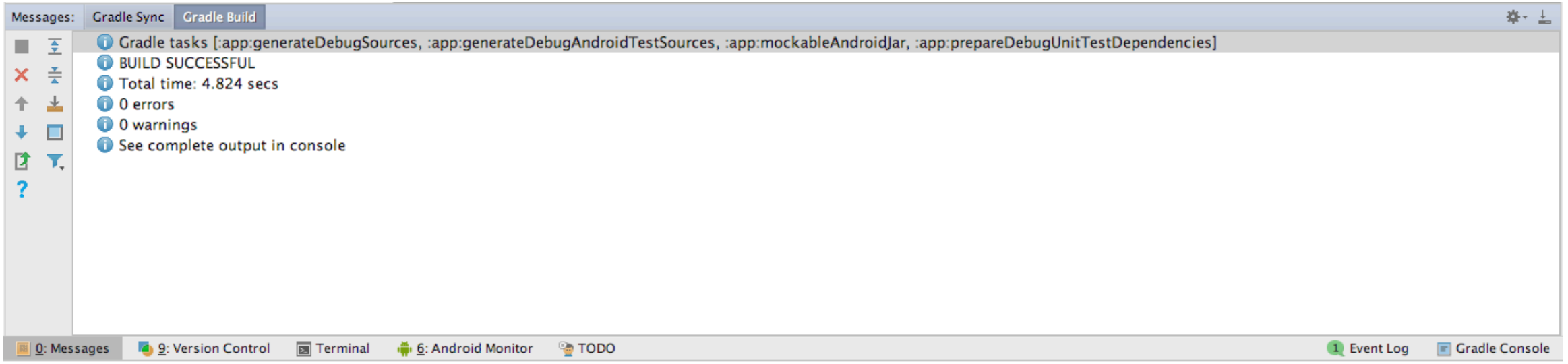
<4.5> ANDROID BUILD TOOLS

Similarly, you'll be asked to install Android Build Tools if your system doesn't have them installed yet.



<4.6> USING ANDROID STUDIO TO BUILD

If the required Android SDK and Build Tools are installed properly, you will be able to build your project within Android Studio.



<5.1> RUNNING ADB IN A TERMINAL

```
fyeung1-mac08:sdk fyeung1$ pwd
/Users/fyeung1/Library/Android/sdk
fyeung1-mac08:sdk fyeung1$ ls
build-tools    emulator       extras         licenses       patcher        platform-tools platforms      tools
fyeung1-mac08:sdk fyeung1$ cd platform-tools/
fyeung1-mac08:platform-tools fyeung1$ ls
NOTICE.txt       api              etc1tool         hprof-conv       package.xml      sqlite3
adb              dmtracedump      fastboot         lib              source.properties systrace
fyeung1-mac08:platform-tools fyeung1$ ./adb
Android Debug Bridge version 1.0.39
Revision 3db08f2c6889-android
Installed as /Users/fyeung1/Library/Android/sdk/platform-tools/./adb

global options:
 -a         listen on all network interfaces, not just localhost
 -d         use USB device (error if multiple devices connected)
 -e         use TCP/IP device (error if multiple TCP/IP devices available)
 -s SERIAL
     use device with given serial number (overrides $ANDROID_SERIAL)
 -p PRODUCT
     name or path ('angler'/'out/target/product/angler');
     default $ANDROID_PRODUCT_OUT
 -H         name of adb server host [default=localhost]
 -P         port of adb server [default=5037]
 -L SOCKET  listen on given socket for adb server [default=tcp:localhost:5037]

general commands:
 devices [-l]             list connected devices (-l for long output)
 help                     show this help message
 version                  show version num
```

Locate adb in your installation folder and run it.

The executable adb is in the platform-tools folder.

<5.2> APPENDING TO $PATH

```
fyeung1-mac08:~ fyeung1$ cat ~/.bash_profile


# Append all the directories terminal will be looking for
export PATH=/bin:/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:/Users/fyeung1/bin:/Users/fyeung1/Voke/bento4/bin:
/Users/fyeung1/Library/Android/sdk/platform-tools:$PATH

# Use Sublime Text as the defaul text editor
# '-w' wait for the files to be closed before returning
# I created a symbolic link so I can use subl directly on command line:
# ln -s /Applications/Sublime\ Text.app/Contents/SharedSupport/bin/subl /Users/fyeung1/bin/.
export EDITOR='subl -w'
alias edit='subl -w'

# This alias edits the profile itself
alias eprofile='edit ~/.bash_profile &'

# This alias reloads the profile
alias xprofile='source ~/.bash_profile'

# You may uncomment the following lines if you want `ls' to be colorized:
export CLICOLOR=1
export LSCOLORS=GxFxCxDxBxegedabagaced

# Environment variables for running make
#export LIBTOOL=glibtool
#export LIBTOOLIZE=glibtoolize make
```

Append the path where adb is to the environmental variable $PATH in .bash_profile (for Mac OS X).

Remember to run "source .bash_profile" for the change to take place.

<5.3> MORE ON ADB COMMANDS

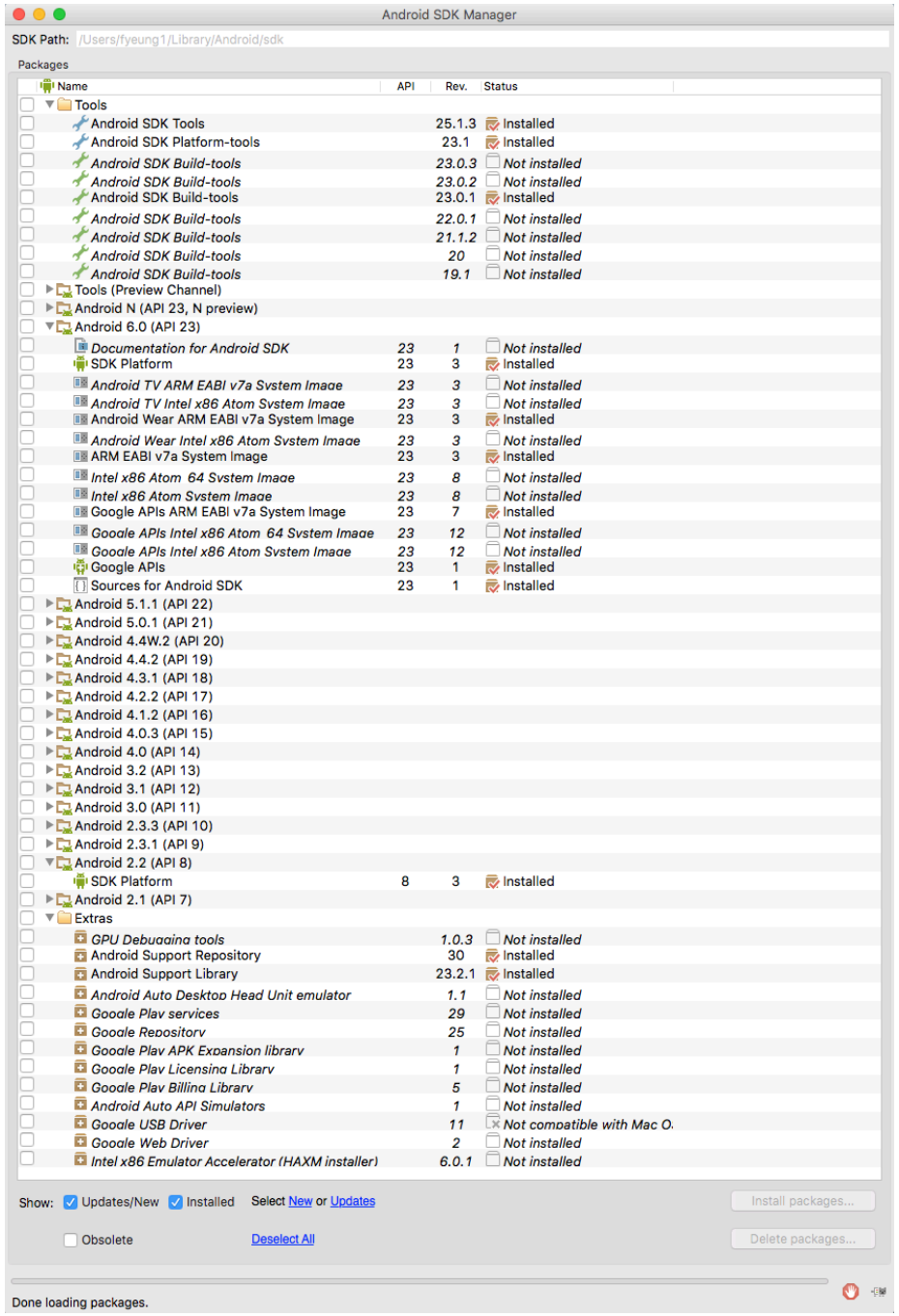
Adb bridges between your computer and the connected device whether it is a real or a virtual device. Here is a list of useful commands:

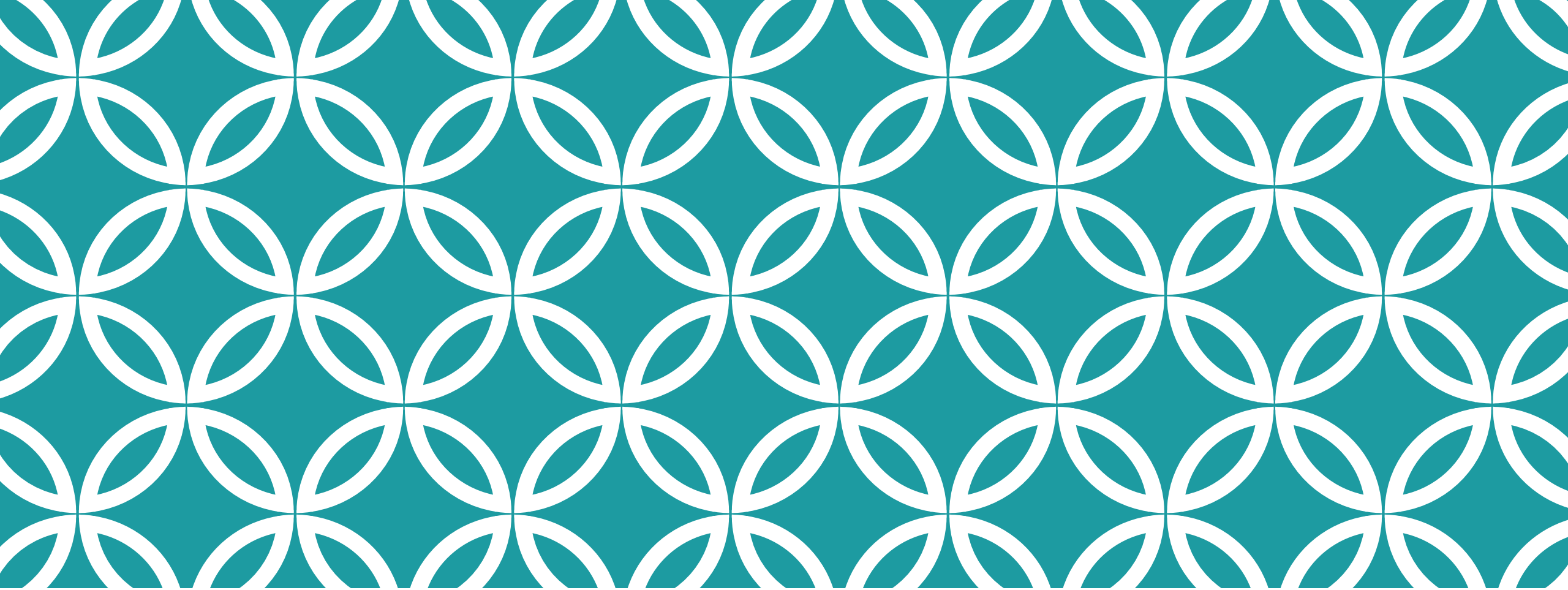
- See a list of the devices available over ADB: `adb devices`
- Install a packaged apk to the connected device: `adb install <apk path>`
- Upload a file to the device: `adb push <local> <remote>`
- Download a file from the device : `adb pull <remote> <local>`

<6> VERIFY THE INSTALLED SDK PACKAGES AND TOOLS

For your reference, here is a list of SDK versions and tools that you may want to install.

Now, you are ready to create your first Android application!





CREATING THE FIRST APP

CS 175 Mobile Software Dev

by Dr. Angus Yeung

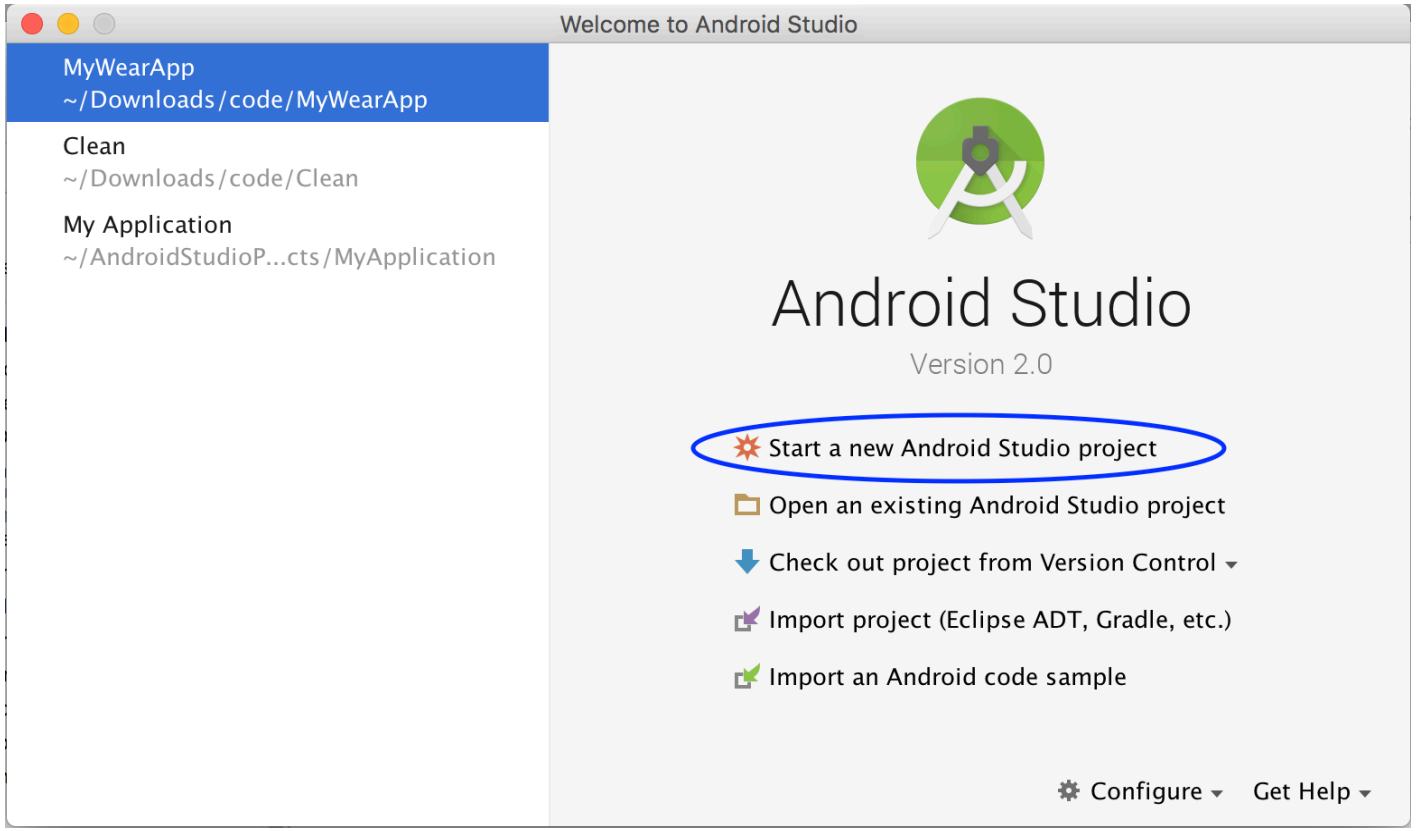
CREATING AN ANDROID PROJECT

With Android Studio, Android SDK packages and tools properly installed on your computer, you are now ready to try creating a new Android project.

<1> START A NEW ANDROID STUDIO PROJECT

In Android Studio, create a new project by clicking **Start a new Android Studio project** in the **Welcome** screen.

If you have a project opened already, you can select **New Project** from the **File** menu.



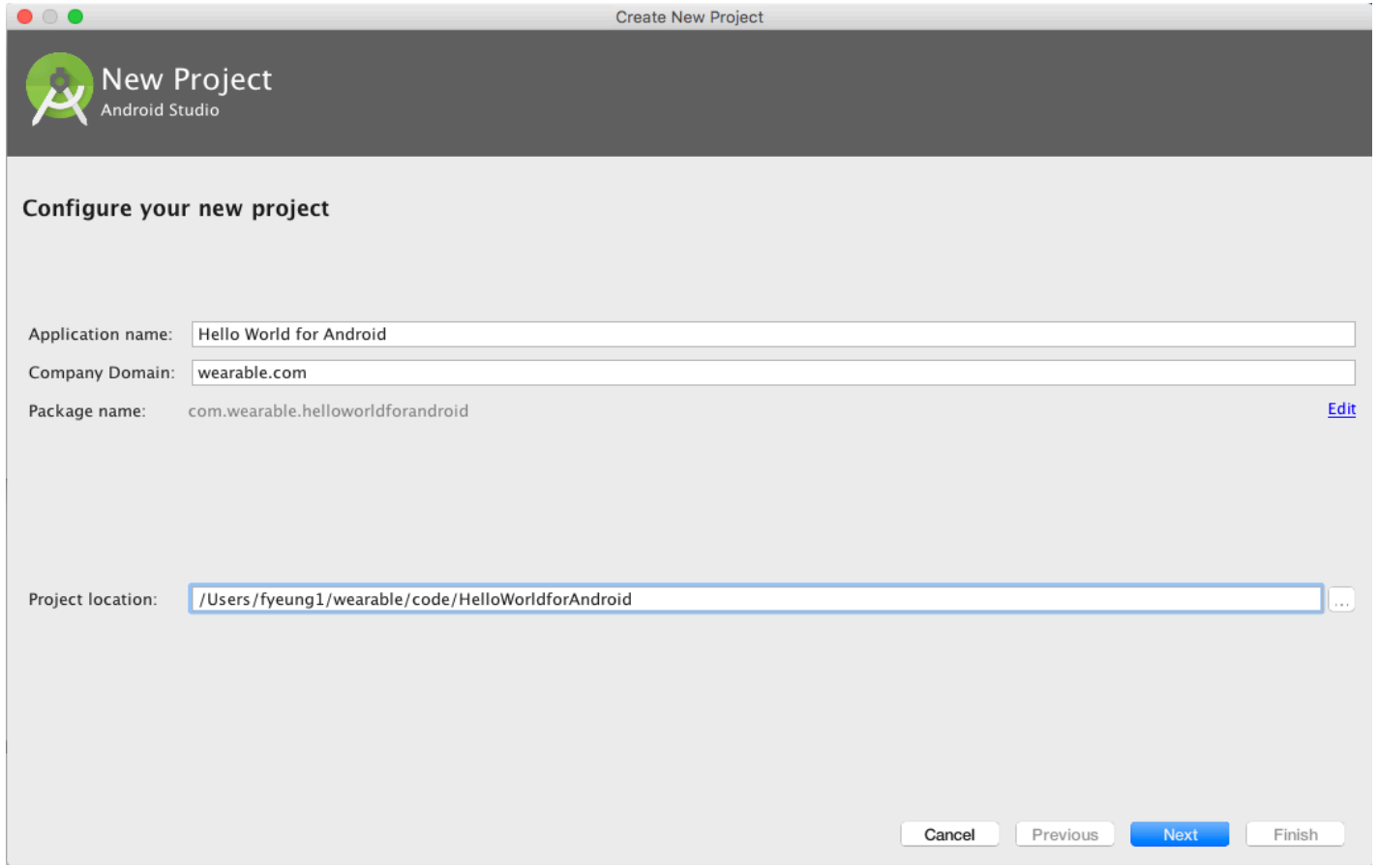
<2> CONFIGURE YOUR NEW PROJECT

Now you are ready to fill out more information about your app.

Enter **Application Name** as “Hello World for Android” and **Company Domain** as “wearable.com”.

Specify **Project Location** as “wearable/code/HelloWorldforAndroid” in your user directory,.e.g., /Users/LoginName.

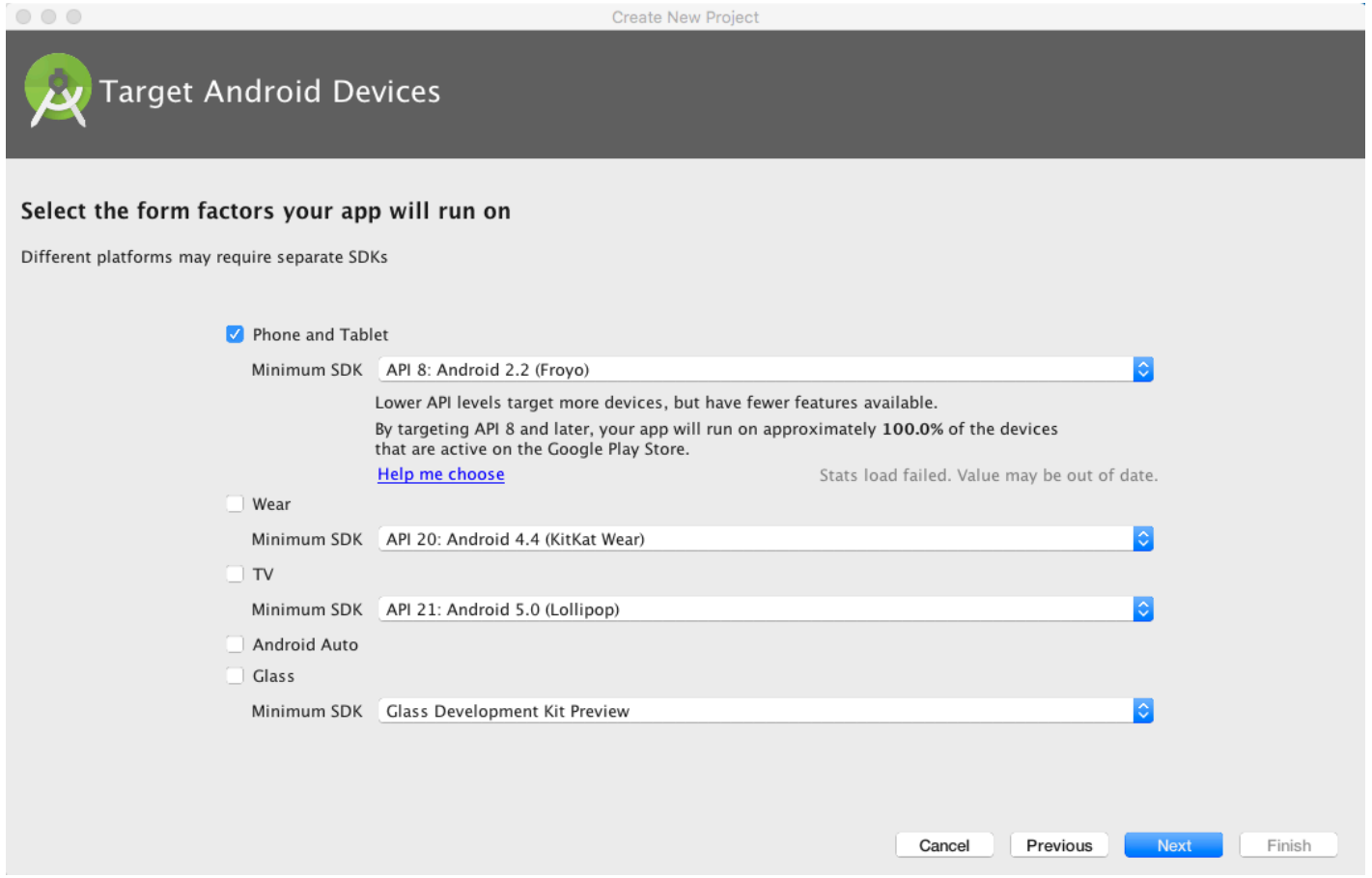
Click **Next** to proceed.



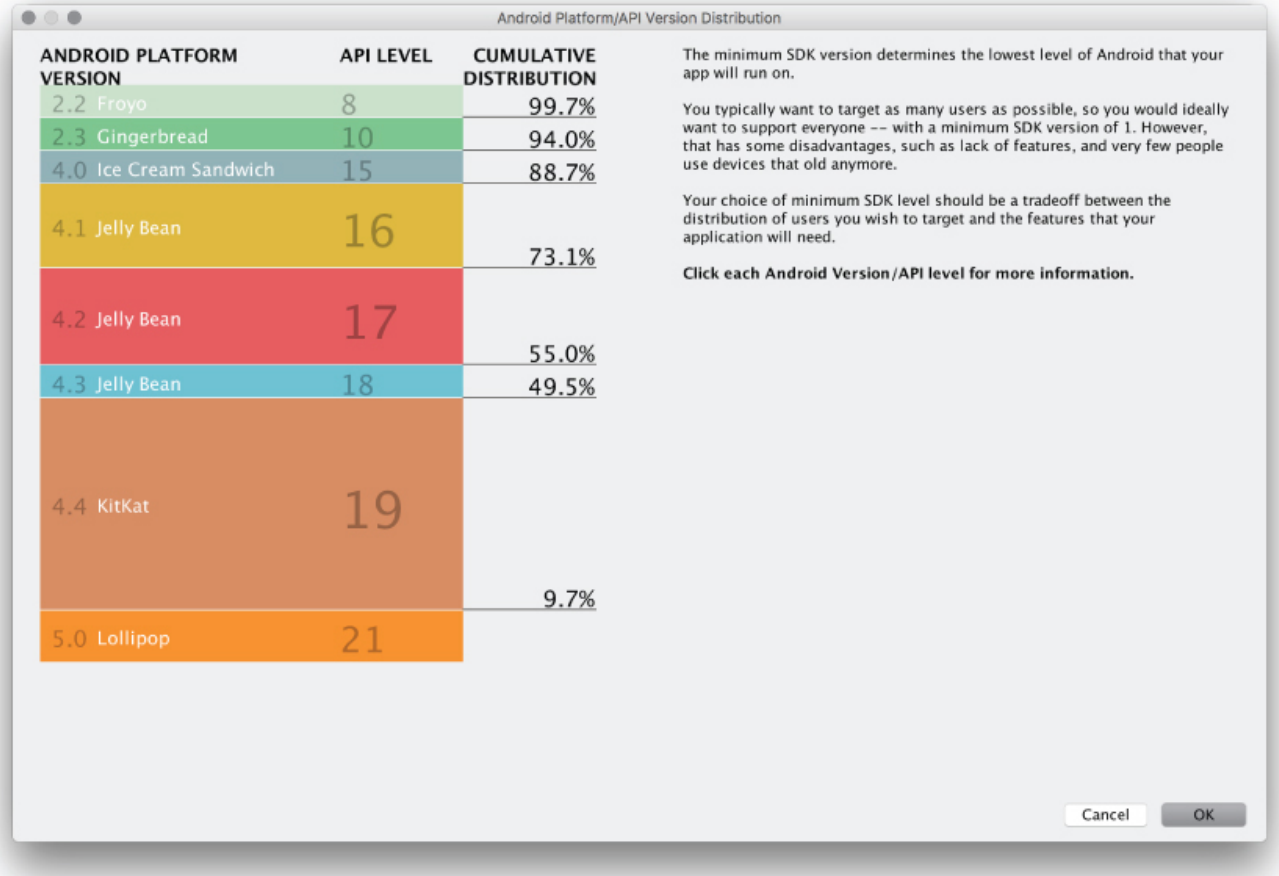
<3.1> CHOOSE THE FORM FACTORS

For **Target Android Devices** settings, choose **Phone and Tablet**, and **API 8: Android 2.2 (Froyo)** as the **Minimum SDK**.

That **Minimum SDK** is the earliest version of Android that your app supports. You want to set this to the lowest version available that allows your app to provide its core feature set to support as many Android devices as possible.



<3.2> ANDROID PLATFORM/API DISTRIBUTION

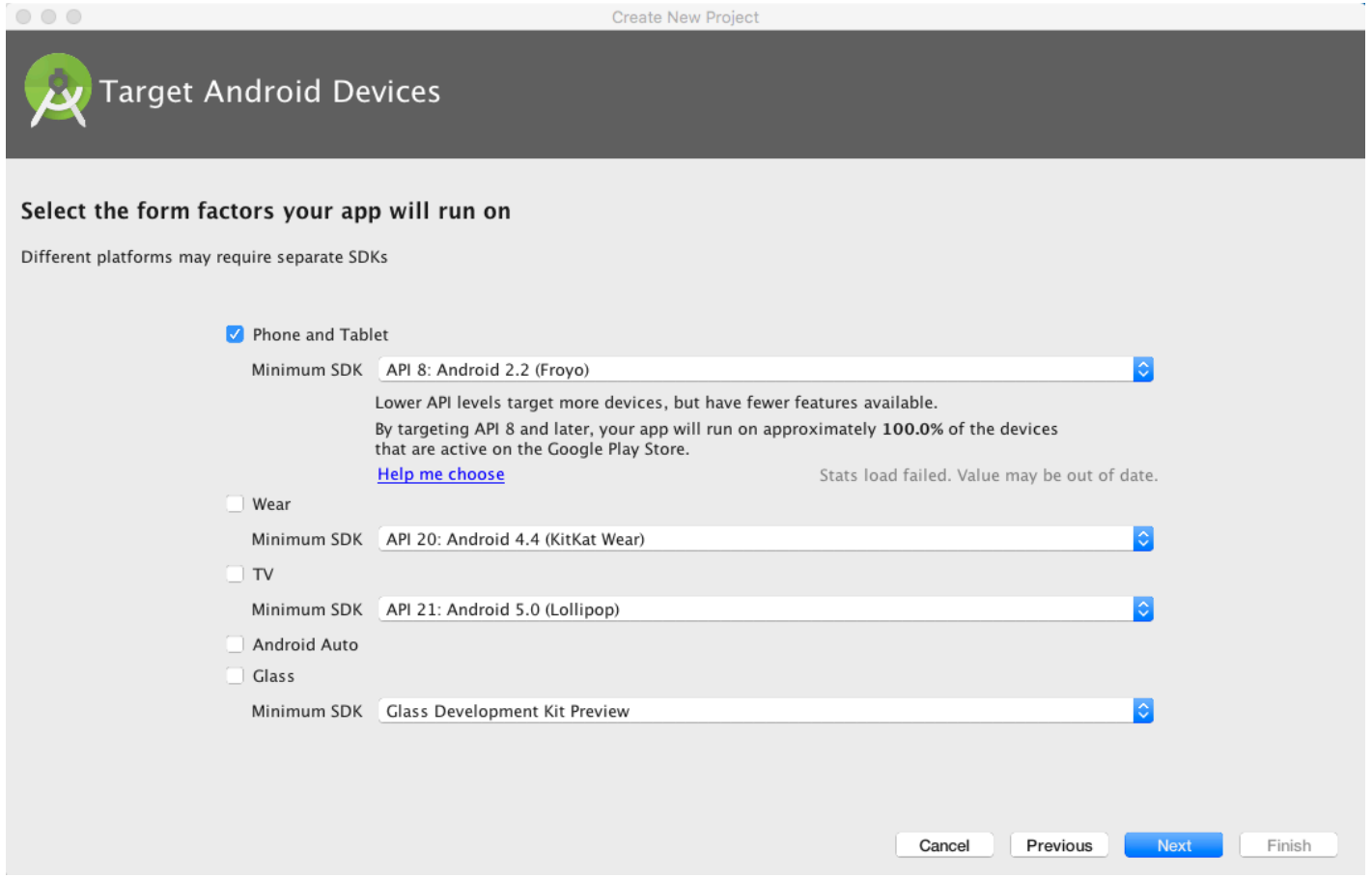


The distribution list helps you to decide about your application's target audience.

<3.3> CHOOSE THE FORM FACTORS

By targeting API 8, your app will run on almost all devices that are active on the Google Play Store.

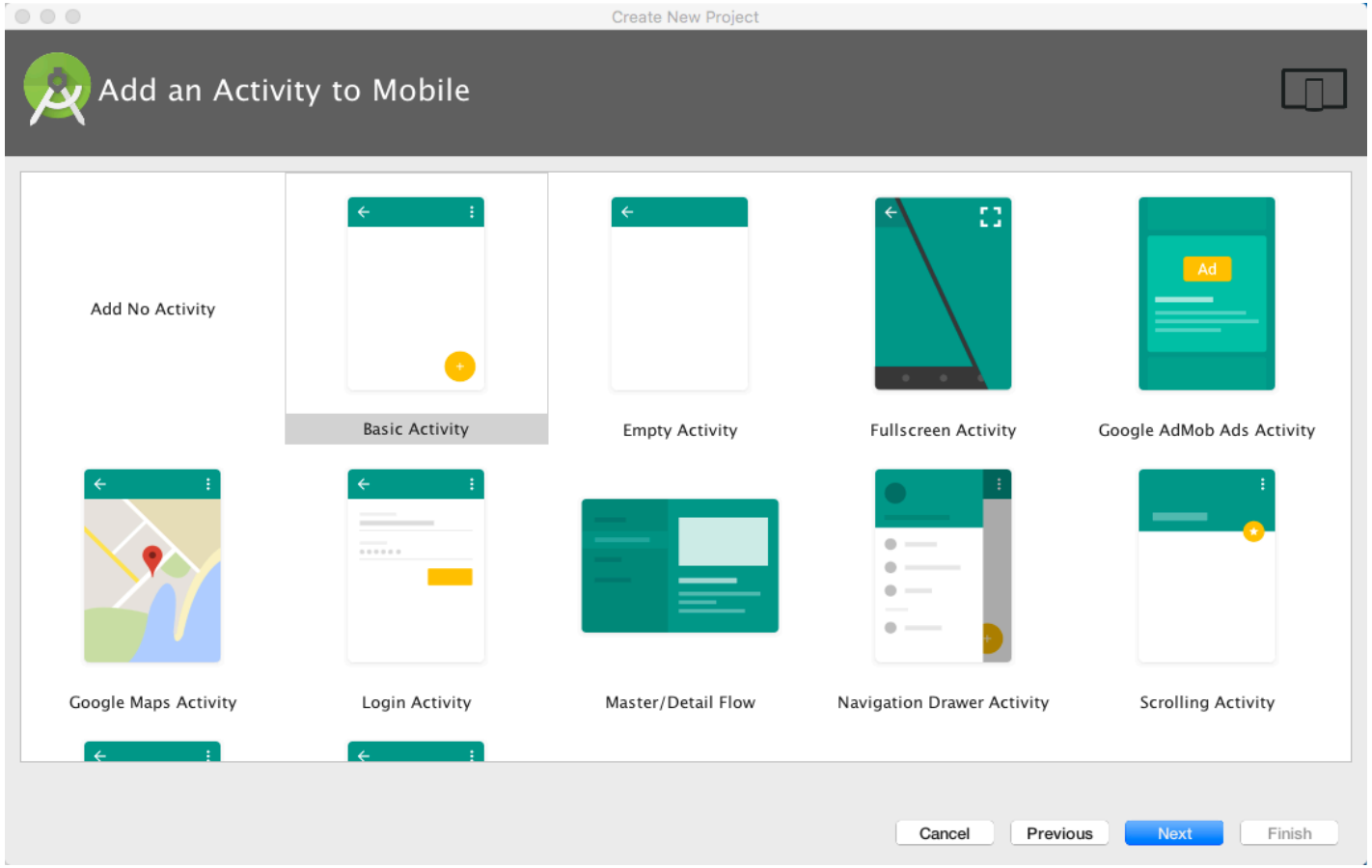
Click **Next** to proceed to the **Add an Activity to Mobile** screen.



<4> ADD A BASIC ACTIVITY TO YOUR PROJECT

Click the **Basic Activity** option.

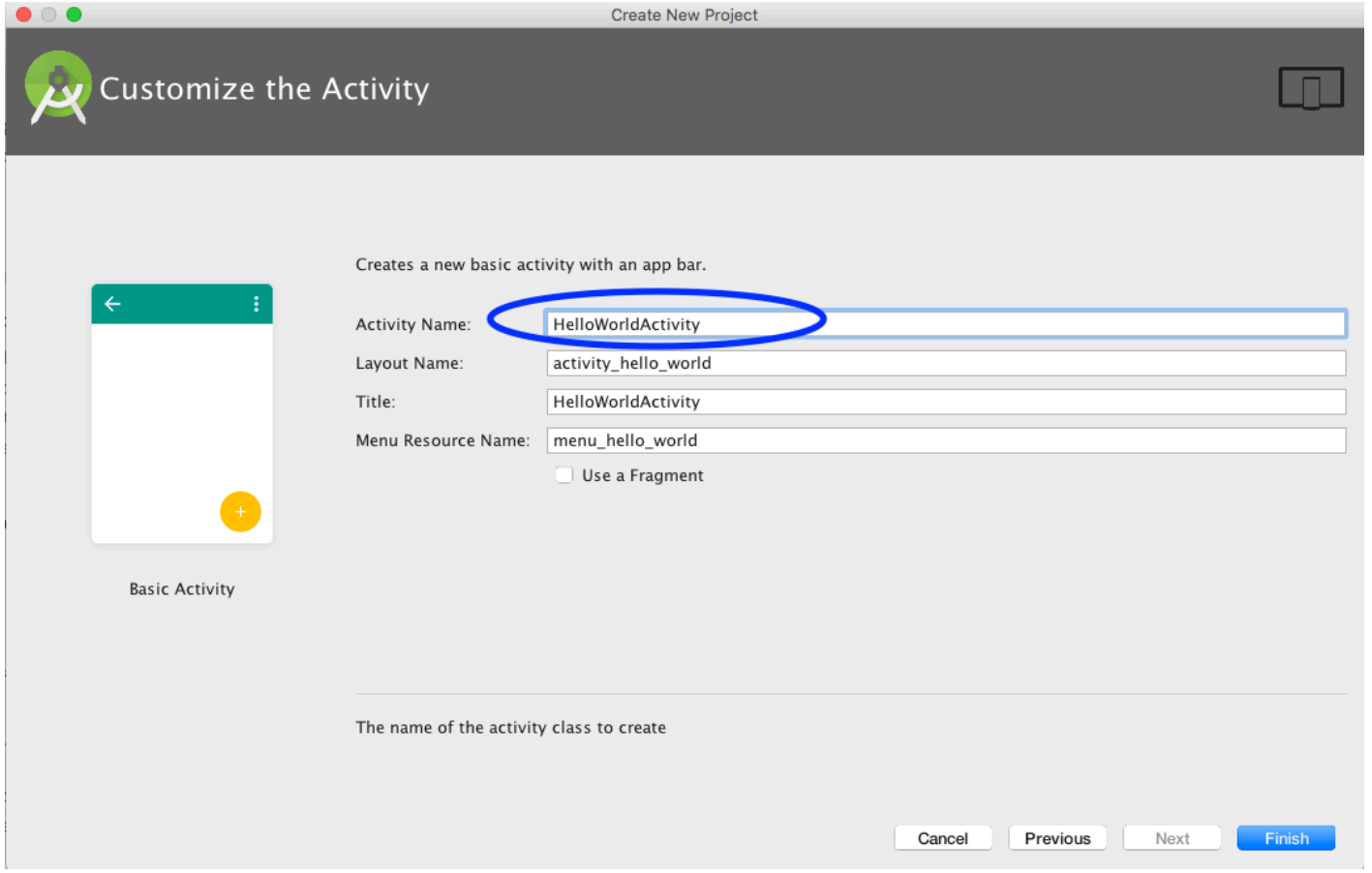
Click **Next** to proceed.



<5> CUSTOMIZE THE ACTIVITY

Change the **Activity Name** to `HelloWorldActivity`. Other fields such as Layout Name, Title, and Menu Resource Name will be updated automatically.

Click **Finish**.



UNDERSTANDING THE HELLOWORLD APP

Your first Android project now contains a **working** basic app that contains some default files.

Android Studio takes care of a lot of configuration and programming details so that you have **a skeleton application** to work with.

You need to have a basic understanding of the code in order to further expand the skeleton app.

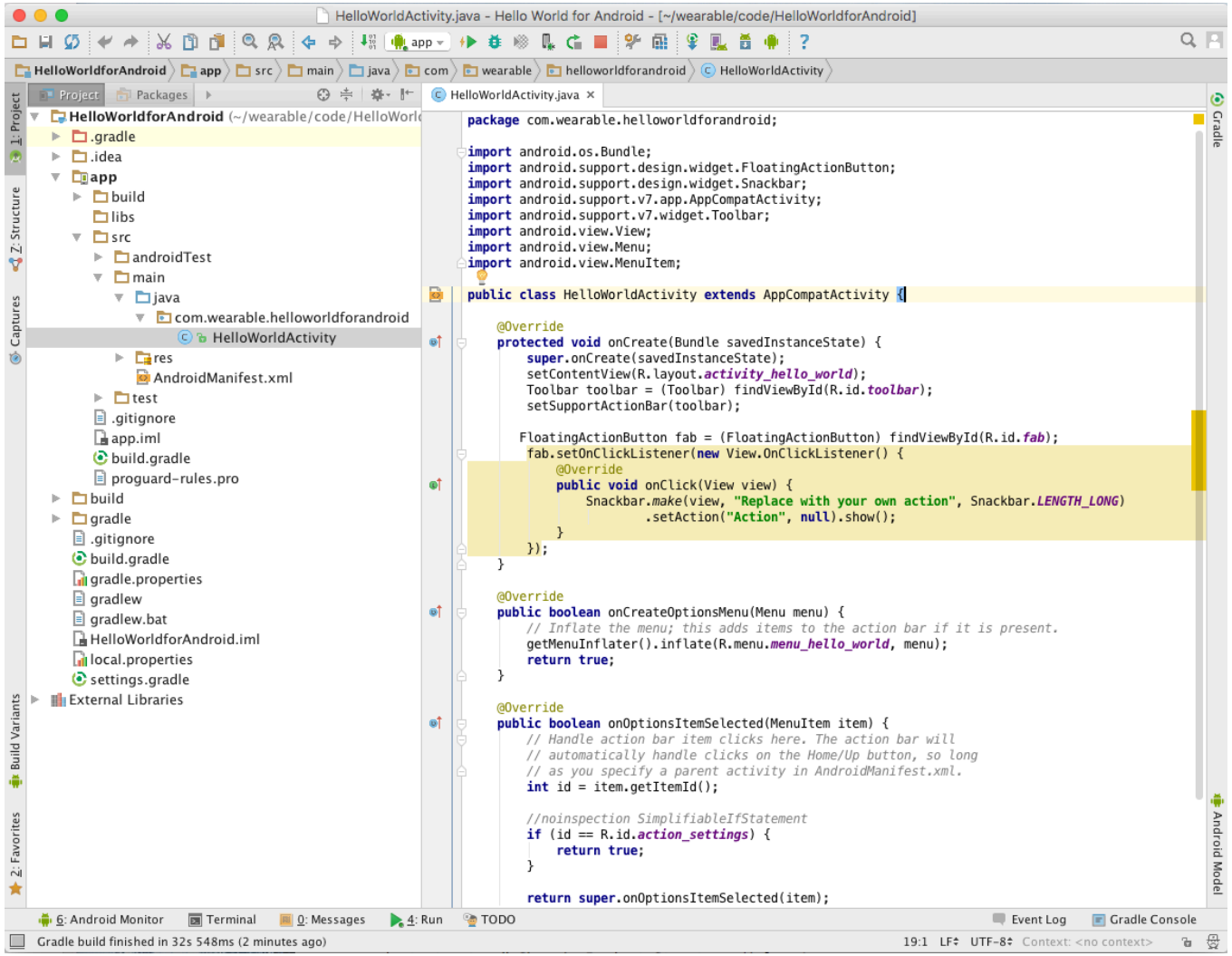
<1> LOCATE THE MAIN SOURCE CODE

The core source code file is **HelloWorldActivity.java**.

It contains your own custom class **HelloWorldActivity** which serves as the main UI screen for your app.

You can review this file by selecting Project view tab in the left panel in Android Studio and expanding the tree:

HelloWorldforAndroid > app > src > main > java > com.wearable.helloworldforandroid > HelloWorldActivity.



<2> REVIEW THE ANDROID MANIFEST FILE

Also check out the manifest file: HelloWorldforAndroid > app > src > main > AndroidManifest.xml.

It is an important file that describes the fundamental characteristics of your app and defines each of its components.

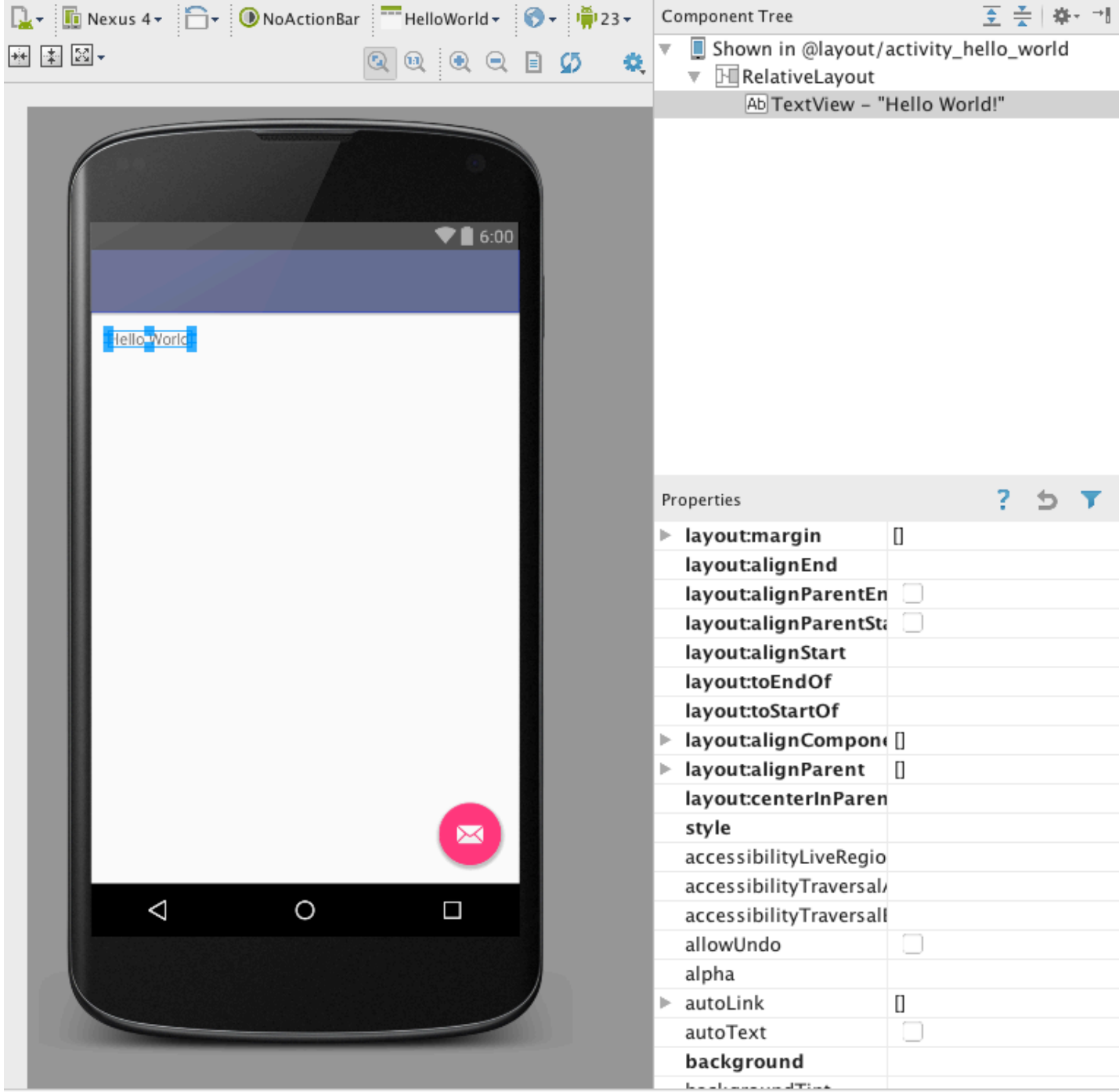
The file contains the attributes for your app, such as **icon, label, theme,** as well as the specific activity to launch.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.wearable.helloworldforandroid" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello World for Android"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HelloWorldActivity"
            android:label="Hello World for Android"
            android:theme="@style/AppTheme.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

<3> CHECK OUT THE UI RESOURCES

The UI files are found in this folder: HelloWorldforAndroid > app > src > main > res > layout. The layout file is activity_hello_world.xml. If you double-click the file, it shows you a preview of the screen UI.

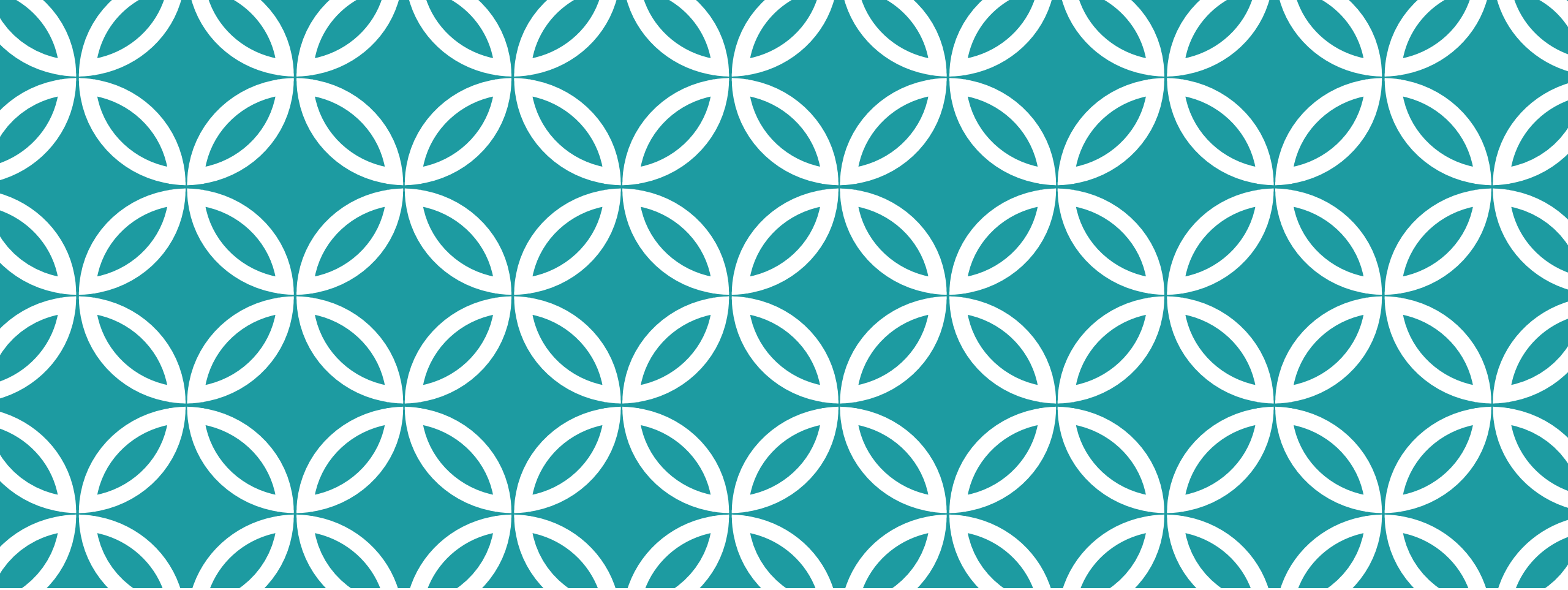
If you double-click the content_hello_world.xml file, the user interface specified by this XML file will be shown. Click on the textview element as shown below will show you more properties about this element in the panels on your right. You may modify the "Hello World!" string using the **text** field in the **Properties** panel.



<4> EXAMINE THE GRADLE FILE

The build.gradle file specifies each module in your app, lists the dependencies, and sets the SDK versions for your project. This file is equivalent to Makefile in other programming languages such as C and C++.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.1"

    defaultConfig {
        applicationId "com.wearable.helloworldforandroid"
        minSdkVersion 8
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.+'
    compile 'com.android.support:design:23.+'
}
```



RUNNING YOUR FIRST APP

CS 175 Mobile Software Dev

by Dr. Angus Yeung

RUNNING YOUR FIRST APP

You can run your first app directly from Android Studio. There are two ways of running your app: using a **real Android device** or running an **emulator**.

USING A REAL ANDROID DEVICE

To set up a real Android device to work with your app, you plug in your Android device to your host computer using a USB cable.

If you are developing on Windows, you might need install USB driver specific for your device. Refer to Android documentation on OEM USB Drivers for instructions.

ENABLE THE DEVELOPER OPTION FOR THE PHONE

For some devices, you can go to Settings > Developer options to turn on the debugging feature. The developer options is hidden on some newer Android devices.

Go to Settings > About phone and tap Build number **seven times** to make the developer options available.

Also enable the USB debugging option.

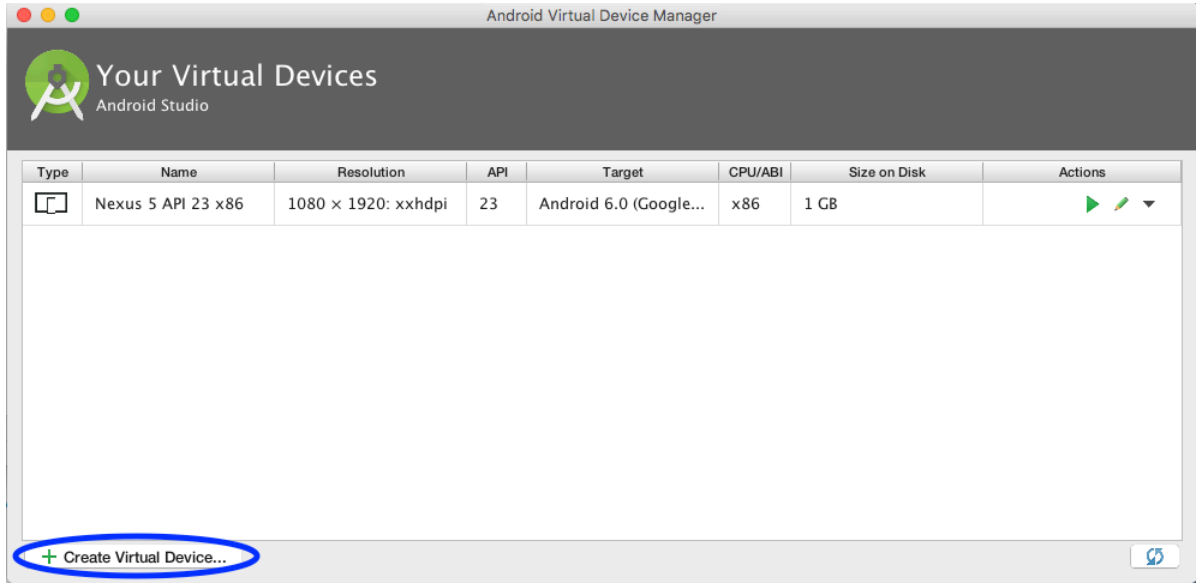
USING AN EMULATOR

To run your app on the emulator, you need to first create an AVD (Android Virtual Device).

Launch the **AVD Manager** by selecting Tools > Android > AVD Manager and click Create Virtual Device.

Choose Phone > Nexus 5 hardware and follow the rest of instructions to complete the creation of a new AVD profile.

Each AVD profile is created for once only. You can continue to use one of the created AVD profiles available in your Android Studio.

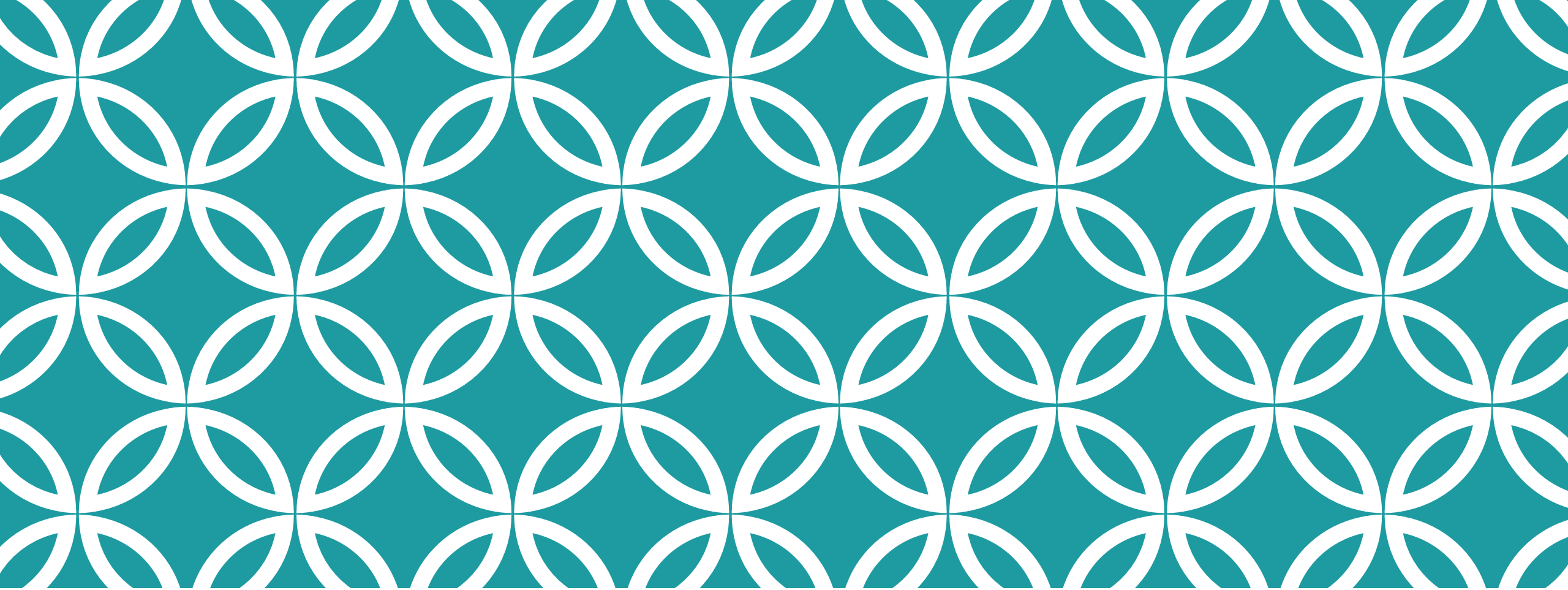


RUN THE APP

Once you have set up an AVD profile or connected to a real Android device, you are ready to run the app.

Click on the **Run App** icon from the toolbar.

In the **Choose Device** window, click the **Launch emulator** radio button. From the **Android virtual device** pull-down menu, select the emulator you have just created and click **OK** to continue.



ADDING UI ELEMENTS TO ACTIVITY

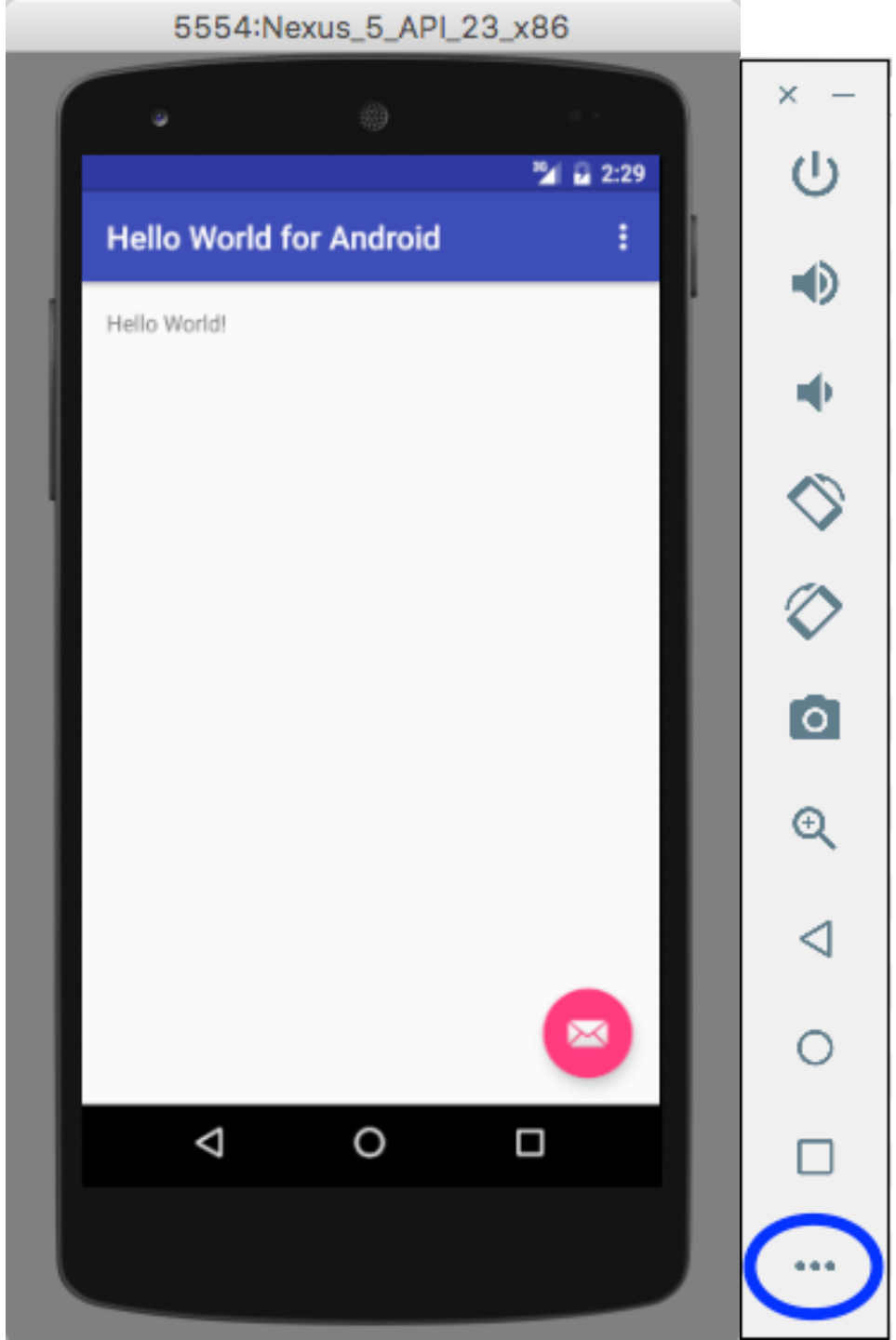
CS 175 Mobile Software Dev
by Dr. Angus Yeung

MINI CONTROL PANEL

The emulator usually takes a few minutes to load itself. After the emulator is properly initialized, your Hello World app appears on the emulator screen as shown in the figure below.

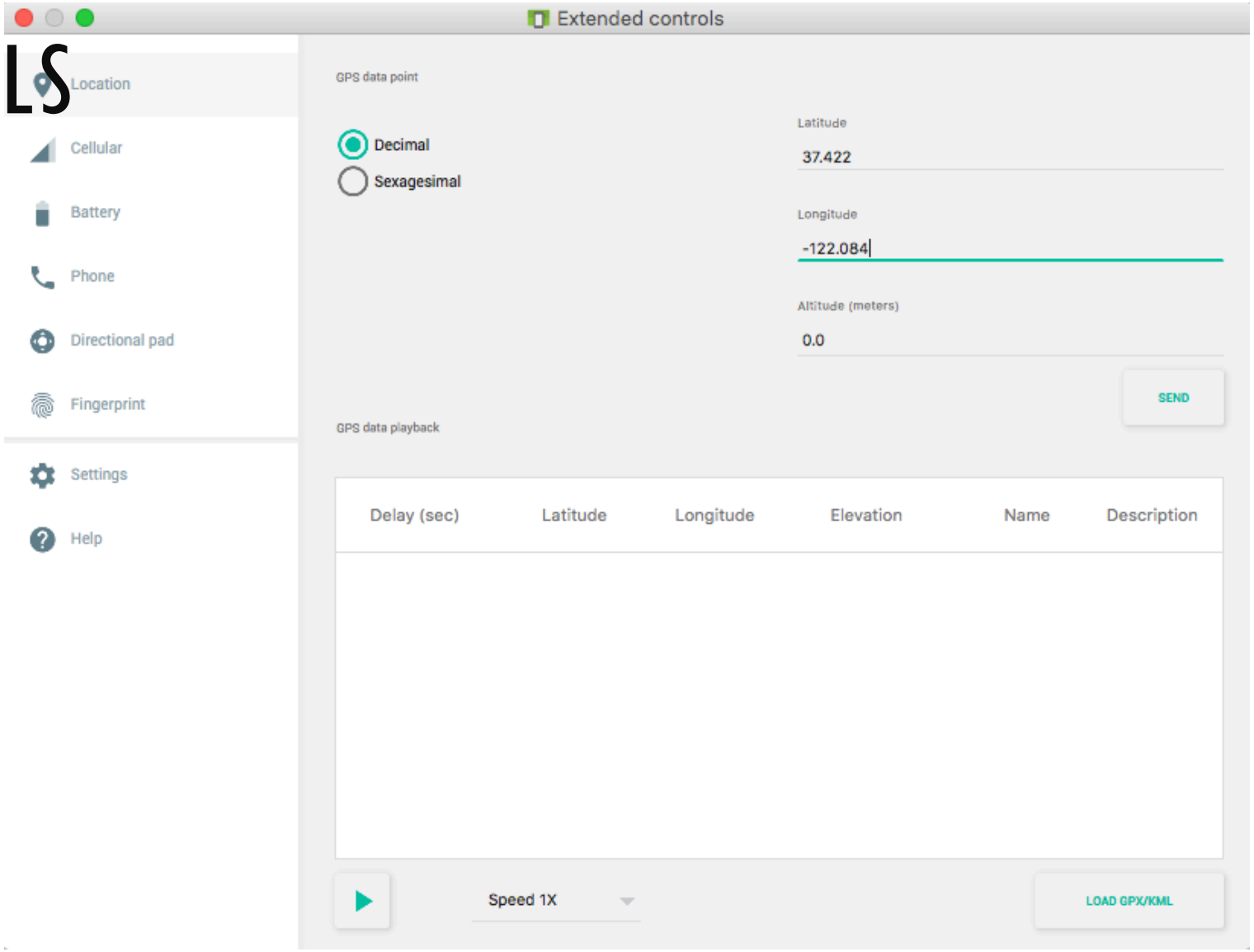
You can see a **mini control panel** next to the phone emulator. It gives you many features such as simulating vibration, power button, audio control, etc.

Click on the last button in the mini control panel will give you an **extended control panel**.



EXTENDED CONTROLS

You can access more configuration features for the emulator using the **Extended controls** window.



ADDING A TEXT FIELD

Let's revisit the layout file activity_hello_world.xml and the content UI file content_hello_world.xml in the Hello World App project.

You are going to add a text field to the skeleton application and get more familiar with the technique to work with UI elements.

<1> WORK WITH THE LAYOUT FILE

Double click the activity_hello_world.xml file and click on the **Text** tab as shown in below (circled in blue color).

The CoordinatorLayout element contains three elements: **AppBarLayout,** included layout content_hello_world, and **FloatingActionButton**.

The **AppBarLayout** contains a Toolbar element. The layout content_hello_world references to another XML file content_hello_world.xml.

<2> LOCATE THE TEXTVIEW UI ELEMENT

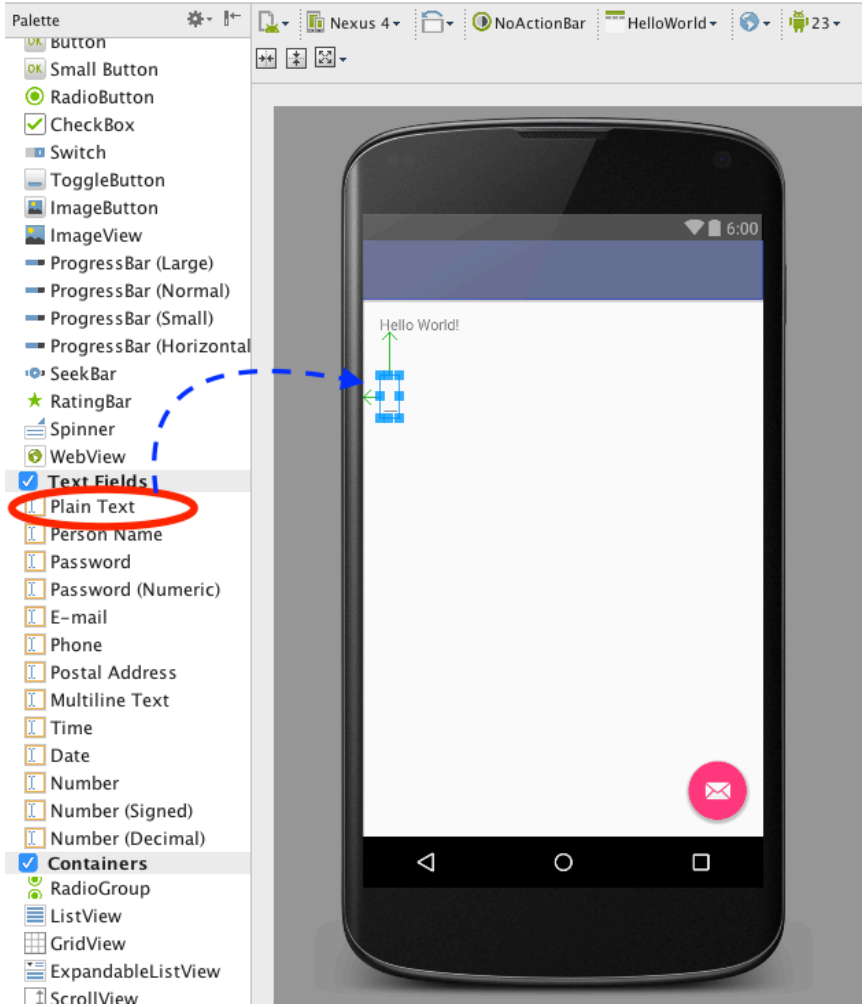
The content of content_hello_world.xml is displayed in the figure below. A **RelativeLayout** element encloses a **TextView** element with string "Hello World!".

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..."
    tools:showIn="@layout/activity_hello_world"
    tools:context="com.wearable.helloworldforandroid.HelloWorldActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

<3> ADD A NEW EDITTEXT UI ELEMENT

To add a new **TextField** to the UI, click on the **Design** tab when content_hello_world.xml is displayed.

Now, drag the **Plain Text** item from the **Palette** to the phone screen.



<4> EXAMINE THE ADDED EDITTEXT

Click on the Text tab to return to the code view.

You should see that a new element **EditText** has been appended to the XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..."
    tools:showIn="@layout/activity_hello_world"
    tools:context="com.wearable.helloworldforandroid.HelloWorldActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="40dp" />
</RelativeLayout>
```

<5.1> CHECK THE ATTRIBUTES

The attributes of **EditText** indicate its alignment, string ID, top margin, and the element used for anchoring

android:layout_width and **android:layout_height**

- Specify the view should be only as big as needed to fit the contents of the view when the value is "wrap_content".

android:id

- Provide a unique identifier for the view, which you can use to reference the object from your app code, such as to read and manipulate the object.

  The at sign (@) id is required when you refer to a resource object from XML. It is followed by the resource type (id in this case) and the resource name. The plus sign (+) before the resource type is needed only when you are defining a resource ID for the first time.

<5.2> CHECK THE ATTRIBUTES

android:layout_below

- Position the top edge of this view below the given anchor view ID “textView”.

android:layout_alignParentLeft and
android:layout_alignParentStart

- Make the left edge of this view match the left edge of the parent and the start edge of this view match with the start edge of the parent.

android:layout_marginTop

- Add extra spaces on the top side of this view.

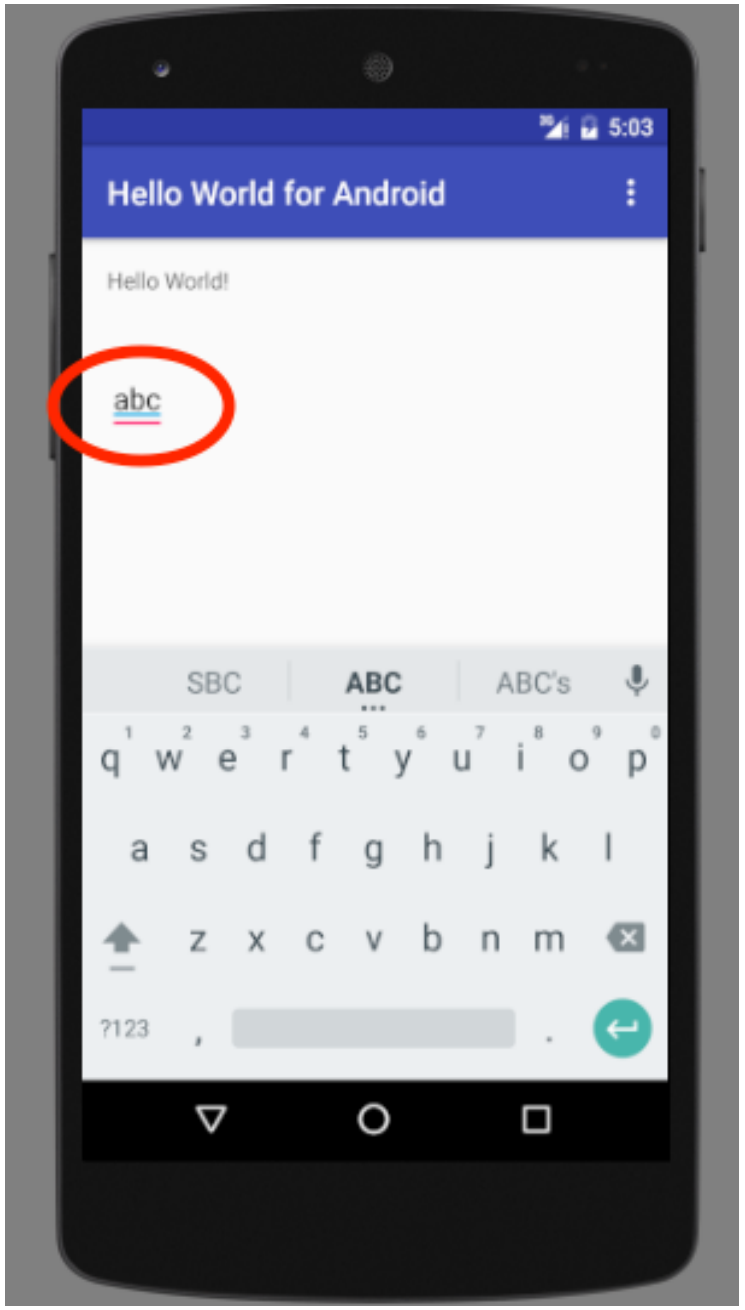
<6> RUN THE APP

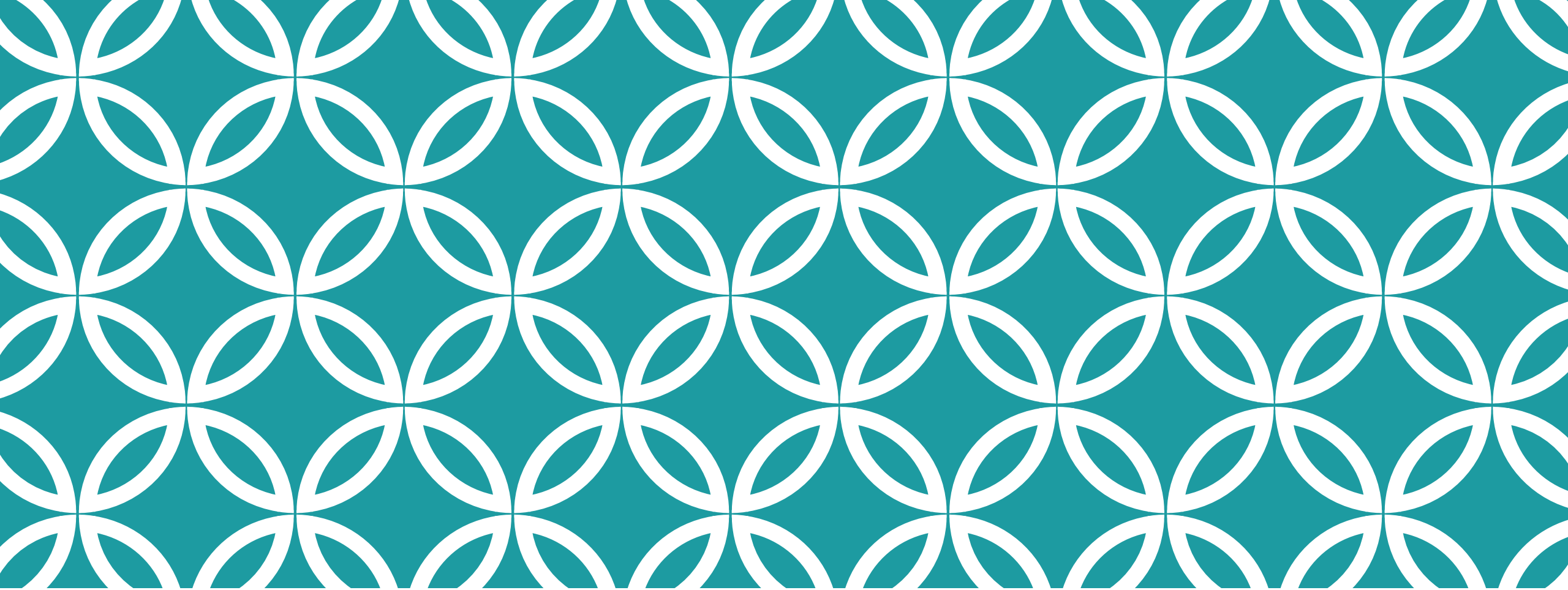
Click on the **Run 'App'** button to run the app.

A new text edit field appears below of the **Hello World!** string.

If you click on the text edit field, a keyboard appears.

Use the keyboard to enter some words in the text edit field.



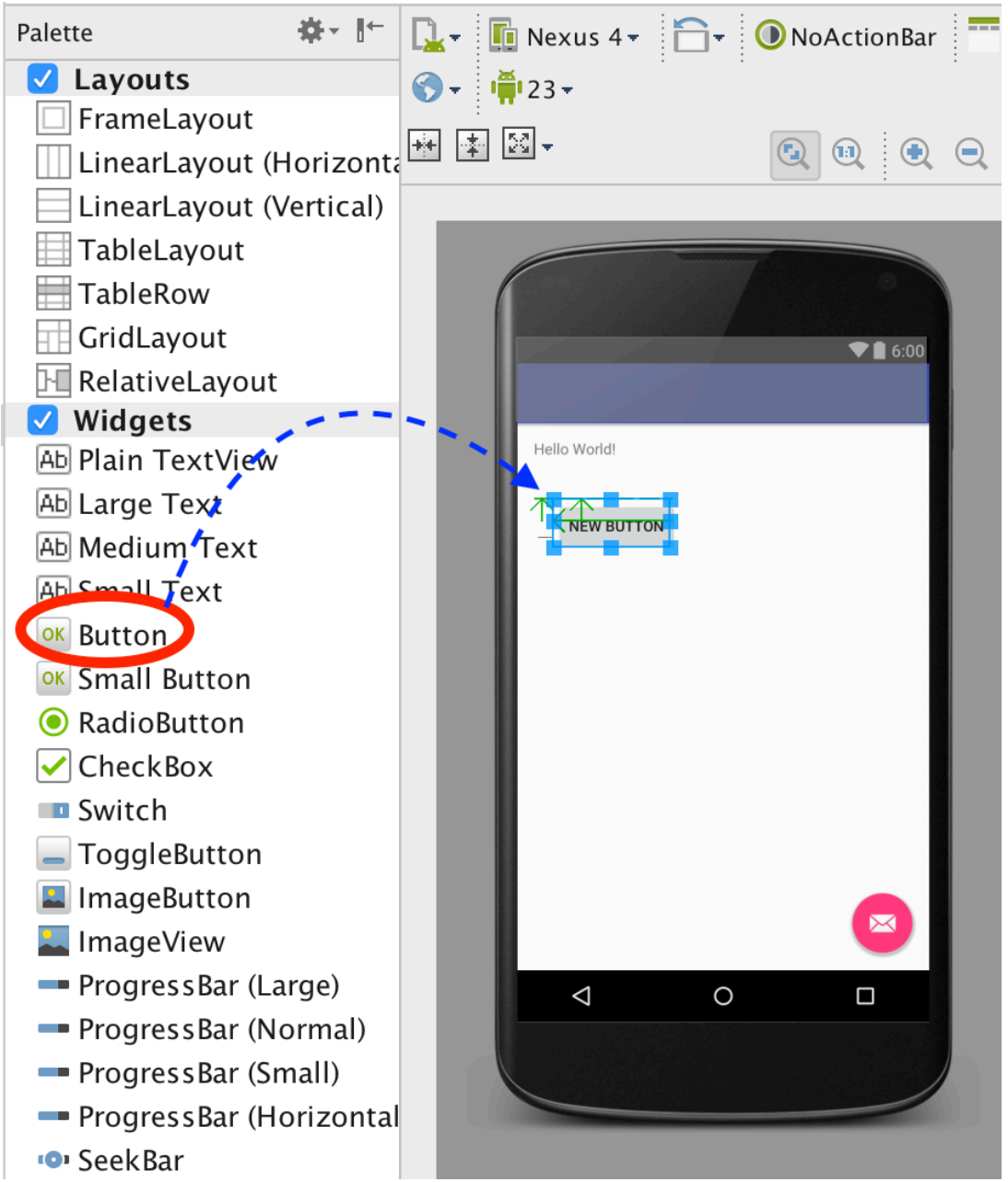


INVOKING A SECOND ACTIVITY

CS 175 Mobile Software Dev
by Dr. Angus Yeung

<7> ADD A NEW BUTTON

Add a new button next to the text edit field by dragging the **Button** item from **Palette**. Make sure the alignment is to the right of the text edit field.



<8> CHECK THE BUTTON'S ATTRIBUTES

The attributes for this new button are appended to the XML file.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_toRightOf="@+id/editText"
    android:layout_toEndOf="@+id/editText" />
```

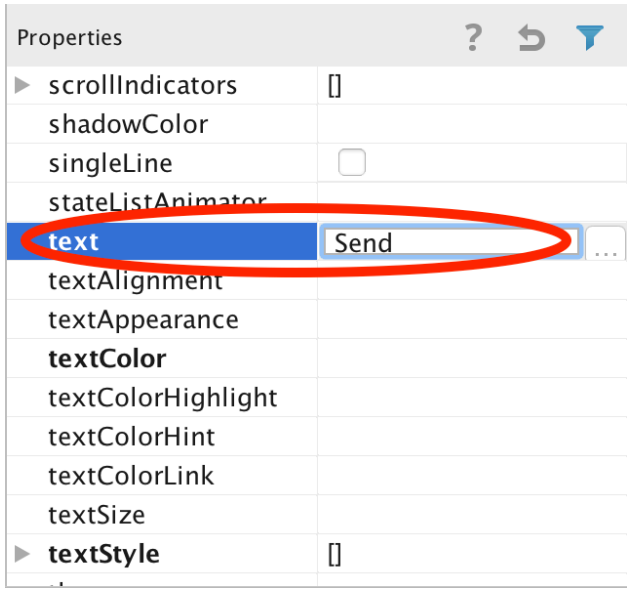
The last three attributes describe the layout of **Button** relative to the **editText** item.

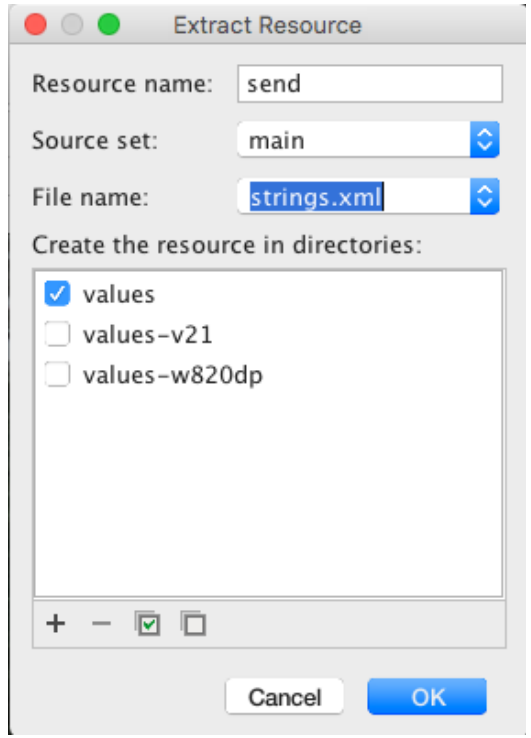
<9> MODIFY THE BUTTON'S PROPERTIES

Find the **text** item in the **Properties** window and change the value to "Send".

After you finish modification, Android Studio will ask if you want to add the string "**Send**" to the strings.xml file. Putting all string resource is usually a good idea, making it very convenient for your app's internalization.

Go ahead to add the resource to the XML file.





STARTING ANOTHER ACTIVITY

You are going to continue to add more capabilities to your skeleton app. Here you will start another activity from your main activity in several simple steps:

- Add a button in the main activity and attach code that will send out an Intent with string message.
- Create a new activity as the second screen.
- Add a textview UI element to the second screen.
- Receive the Intent in the second screen and display the Intent's string onto the textview.

<1.1> ADD THE BUTTON'S ONCLICK OPTION

The first step is to start a new activity when the user clicks the **Send** button.

Add the **android:onClick** attribute to the <Button> element in your content_hello_world.xml file.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/send"

    android:onClick="sendMessage"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_toRightOf="@+id/editText"
    android:layout_toEndOf="@+id/editText" />
```

The android:onClick attribute specifies that sendMessage is the method in your activity that the system calls when the user clicks the **Send** button.

<1.2> IMPLEMENT THE CODE FOR SEND BUTTON

Double-click the **HelloWorldActivity** java file in the **Project** window and add the sendMessage() method stub shown below.

```
/** Called when the  user clicks on the Send button */
public void sendMessage(View view) {
      // Do something in response to button
}
```

The method must be public to allow other classes to call it. Inside of the sendMessage() method, create an **Intent** to start an activity called **SecondScreen**.

<1.3> IMPLEMENT THE CODE FOR SEND BUTTON

Add the following implementation code to the sendMessage() method:

```
/** Called when the  user clicks on the Send button */
public void sendMessage(View view) {
      Intent intent = new Intent(this, SecondScreen.class);
      EditText editText = (EditText) findViewById(R.id.editText);
      String msg = editText.getText().toString();
      intent.putExtra(EXTRA_MESSAGE, msg);
      startActivity(intent);
}
```

In the above implementation, you first get a reference to the **EditText** object by using the ID R.id.editText that you specified in the content_hello_world.html file, then put the retrieved text from the **EditText** object in the **Intent**.

The last step is to fire up the **SecondScreen** activity by supplying the **Intent** as argument to the StartActivity() method.

<1.4> ADD THE EXTRA MESSAGE DEFINITION

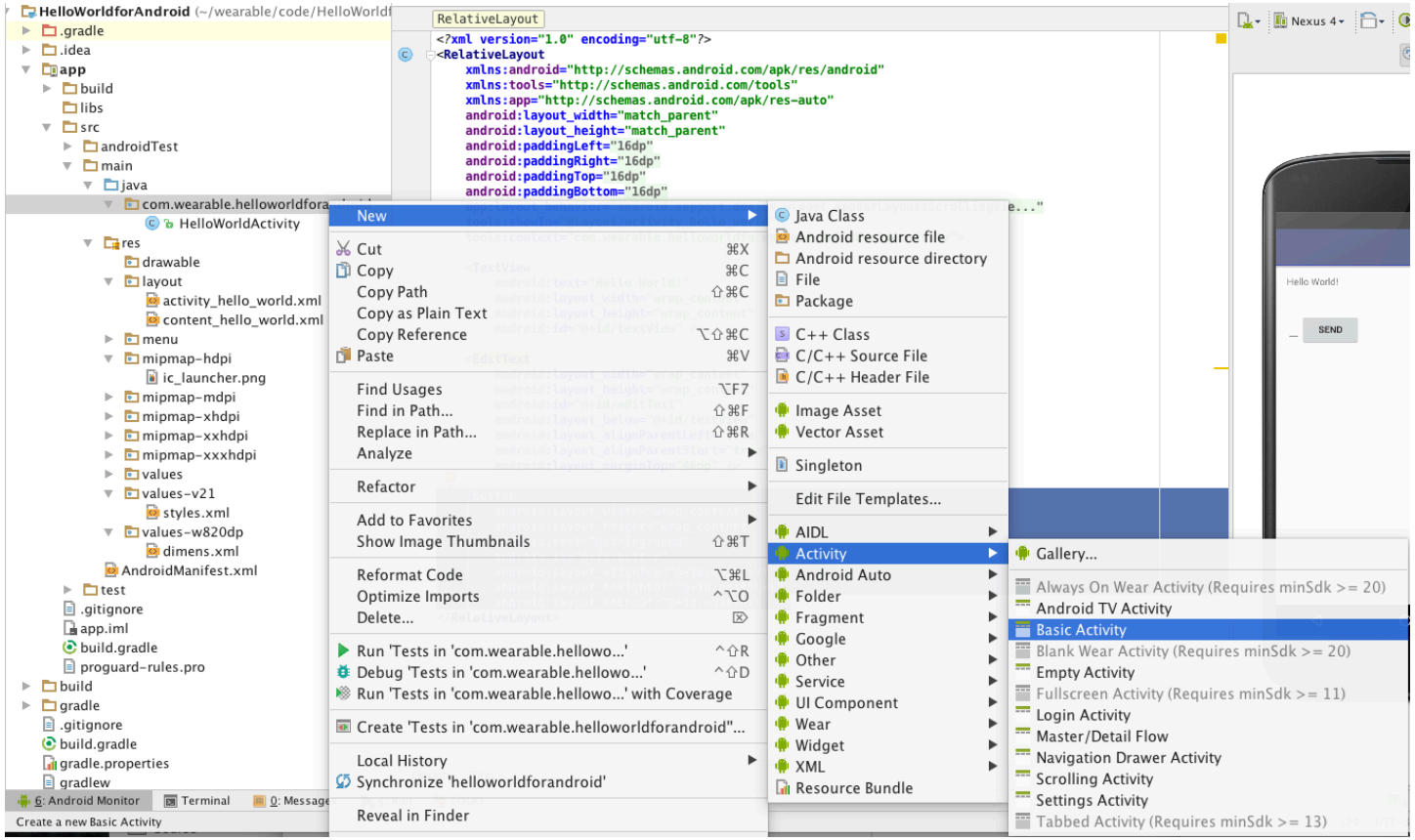
At the top of the HelloWorldActivity class, add the EXTRA_MESSAGE definition.

java/com.wearable.helloworldforandroid/HelloWorldActivity.java

```
public class HelloWorldActivity extends AppCompatActivity {

    public static final String EXTRA_MESSAGE = "com.wearable.helloworldforandroid.MESSAGE";
    . . .
}
```

<2.1> CREATE A NEW ACTIVITY

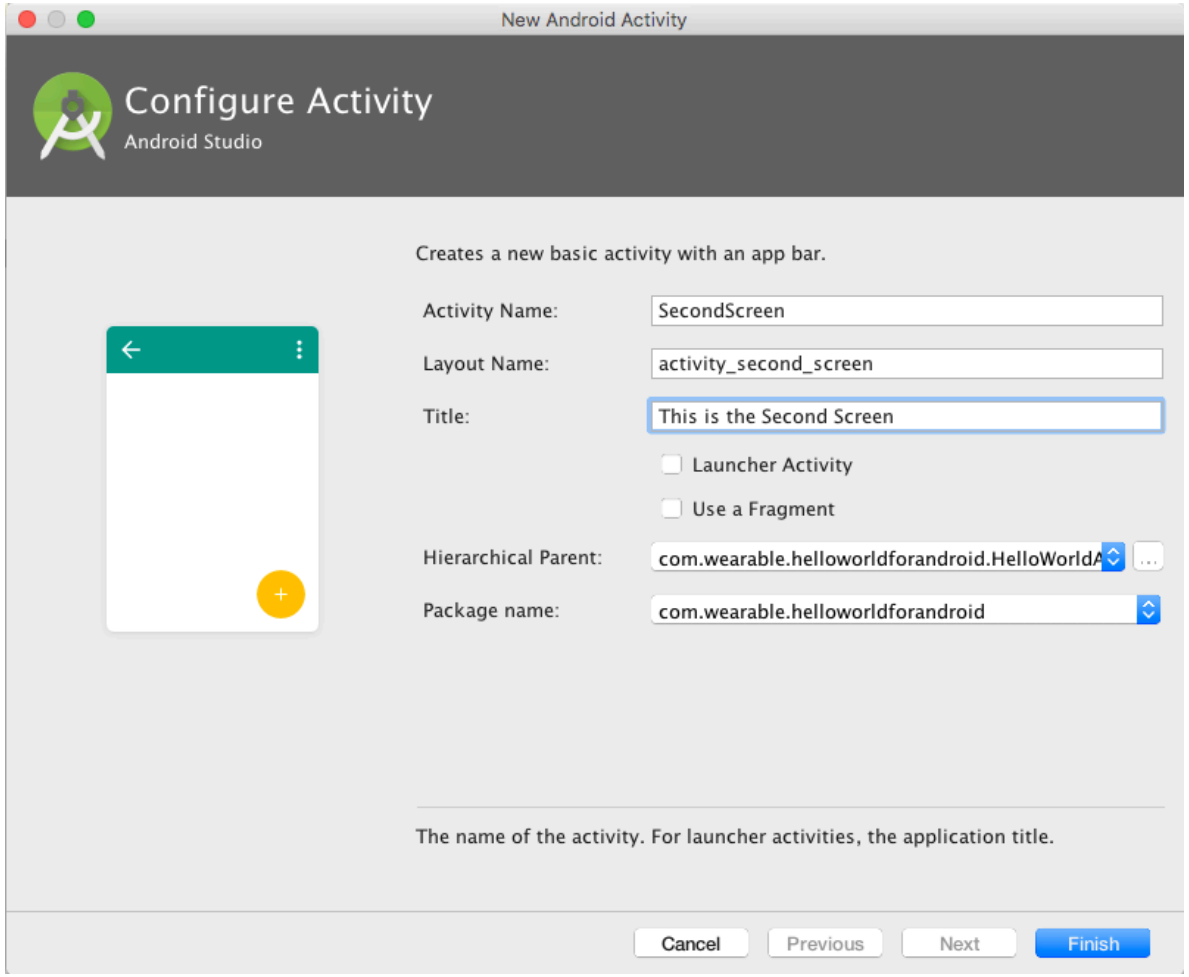
Right-click on com.wearable.helloworldforandroid in the **Project** window and select New -> Activity -> Basic Activity as shown in the figure.



<2.2> CREATE A NEW ACTIVITY

Configure the new activity using the values shown in the following screen shot.

Click the **Finish** button.



<2.3> CREATE A NEW ACTIVITY

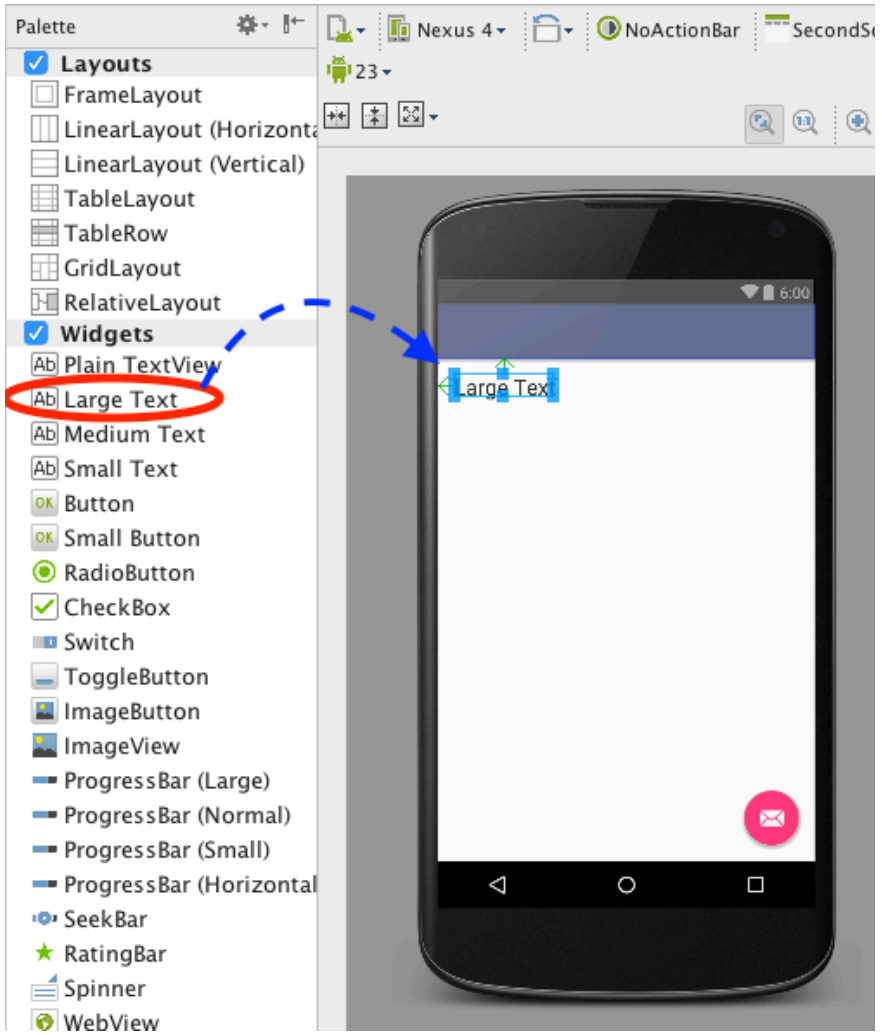
Android Studio creates a new Java file SecondScreen.java.

```
package com.wearable.helloworldforandroid;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;

public class SecondScreen extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second_screen);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }

}
```

<3.1> ADD A NEW TEXTVIEW ELEMENT

Double-click content_second_screen.xml in the **Project** window.

Drag a **Large Text** widget from **Palette** to the phone screen, as shown below.



<3.2> ADD A NEW TEXTVIEW ELEMENT

Click on the **Text** tab and you will see the new **TextView** element added to the XML file.

Note down the Android ID for the **TextView** element. It should be called textview2.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..."
    tools:context="com.wearable.helloworldforandroid.SecondScreen"
    tools:showIn="@layout/activity_second_screen">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/textView2"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

<4.1> DISPLAY THE STRING

This step adds code to the onCreate() method of the **SecondScreen** activity to capture the string message from the **Intent** originated from the **HelloWorldActivity** and post to the **TextView** element. Add the following code to the onCreate() method.

```
public class SecondScreen extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        . . .

        // Get the intent and assign it to a local variable
        Intent intent = getIntent();

        // Extract the string message delivered by HelloWorldActivity
        String msg = intent.getStringExtra(HelloWorldActivity.EXTRA_MESSAGE);

        // Reference to the TextView item using its ID textview2
        TextView tx2 = (TextView) findViewById(R.id.textView2);

        // Put the string message on the TextView item
        tx2.setText(msg);
    }
}
```

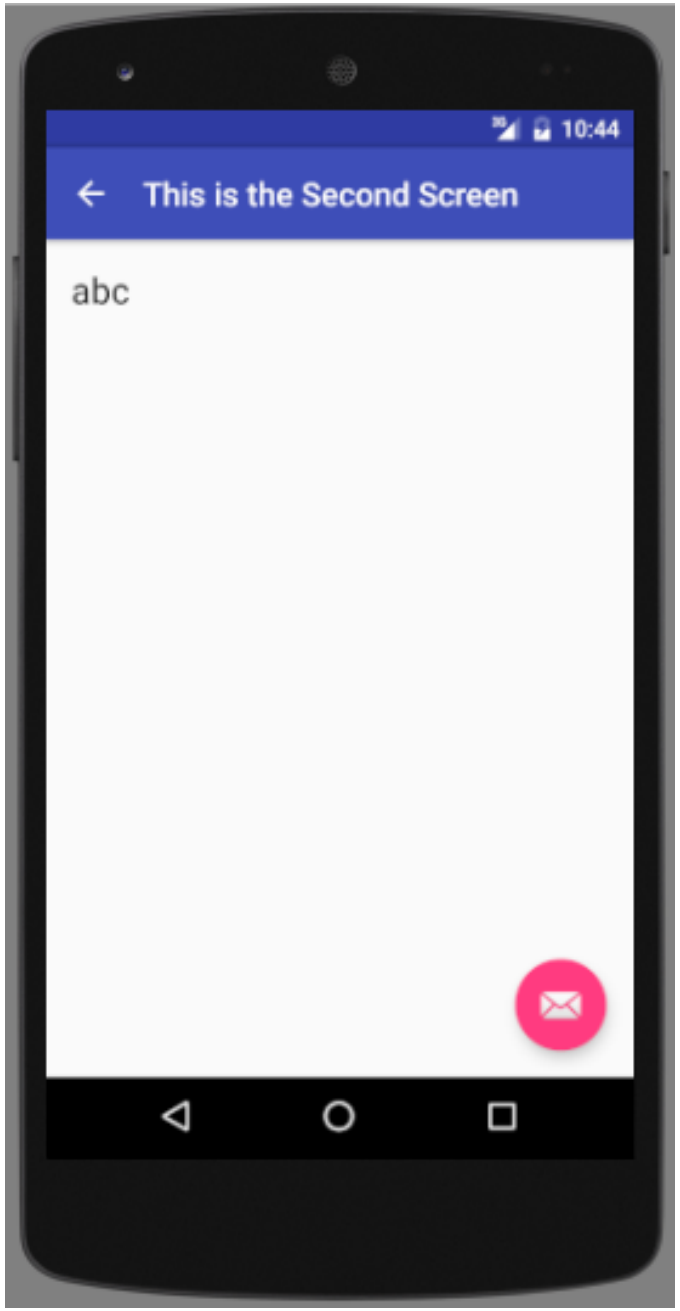
<4.2> RUN THE APP

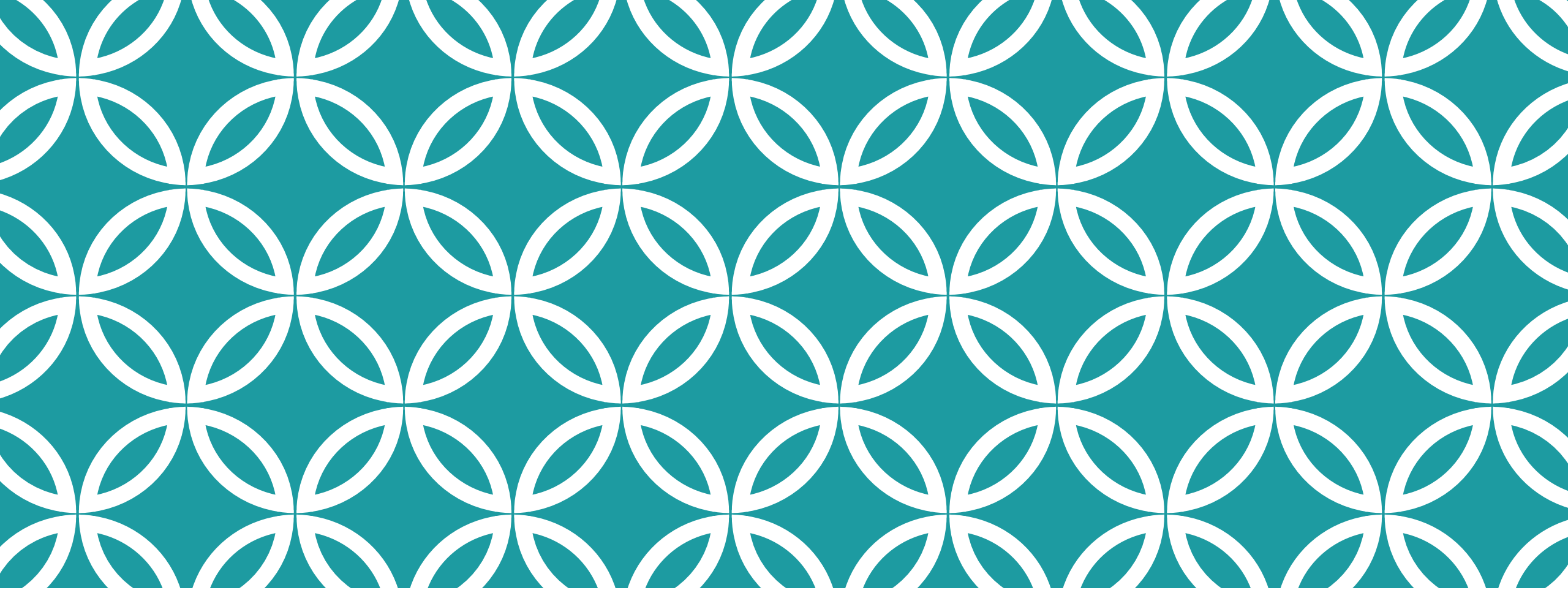
Run the app and fix any errors you have in your project.

Type “abc” in the **Text Edit** field and click on the **Send** button.

The phone screen will switch from the **HelloWorldActivity** screen to the **SecondScreen**.

The string message “abc” is shown in the **SecondScreen**.





TRANSITIONS

CS 175 Mobile Software Dev

by Dr. Angus Yeung

EXTRA READING: TRANSITIONS

Use overridePendingTransition() to customize the transitions between activities. Read the TEXTBOOK Chapter 3.6 for transitions between activities, and Chapter 3.7 for scene transitions.

ActivityB.java

```
1  @Override
2      protected void onStart() {
3          super.onStart();
4          overridePendingTransition(R.anim.transition_in,
5                                    R.anim.transition_out);
6      }
```

TRANSITIONS BETWEEN ACTIVITIES

R.java

```
1  public final class R {
2      public static final class anim {
3          public static final int transition _in=0x7f040000;
4          public static final int transition _out=0x7f040001;
5  }
6  .
7  .
8  .
```