

CSA Readiness Training: Getting Started with Android

Introduction

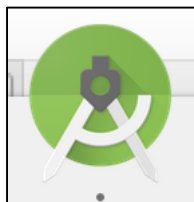
Project Lead The Way designed this AP Java course to connect to one of the most relevant uses for Java in the world today: developing mobile apps for Android. High school students rely on mobile devices for transportation assistance, entertainment, communication, schoolwork, and more. The ubiquity of mobile apps makes the content of the course attractive and engaging. To get there, however, you have to learn about the framework and tools for developing Android apps. This readiness training course will get you up to speed with the foundational knowledge you will need before joining core training.

When you have completed Parts 1 through 3, answer the questions in the **Getting Started with Android Readiness Training Quiz**.

Part I: Android Studio

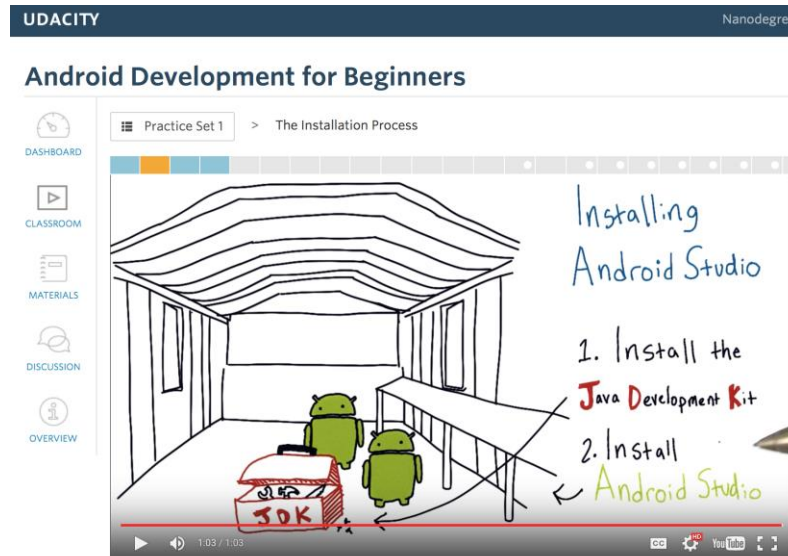
Android Studio is the **IDE** (Integrated Development Environment) created by Google for Android app development. It combines a code editor, a compiler, a graphical user interface (GUI) builder, and many other useful components that streamline the process of building apps for Android.

1. Open Android Studio on your computer.
 1. Refer to the Installation instructions in the [Getting Started with CSA Readiness Training course](#) if you have not yet installed Java and Android Studio.



2. You may also refer to the *Android Development for Beginners* course on Udacity.com. This free course offers a more in-depth look at Android

Studio and designing visual elements in apps.



Optional Resource: [Udacity.com Android Development for Beginners](https://www.udacity.com/android-development-for-beginners)

2. A welcome screen with several options will appear. Click "Start New Project...".
If you already have used Android Studio on this computer before then the welcome window will not appear. Instead you should select **File > New Project...** from the Android Studio menu.
3. In the "Create New Project" window, you will be prompted to set several fields.
 1. Change **Application name** to HelloWorld.
 2. Leave **Company Domain** as is.
 3. If you wish, you may change the **Project Location** to a location within your directory structure by clicking the ellipses (...) next to file path and then selecting a directory from the file chooser.
 4. Click the "Next" button.
4. The next window allows you to choose the SDK version and API level. From the dropdown menu labeled "Minimum SDK" select "API 22: Android 5.1 (Lollipop)". This means that you are choosing API 22, also known as Android 5.1 (Lollipop),

as your minimum target, or the lowest version of Android that is guaranteed to be able to run your app.

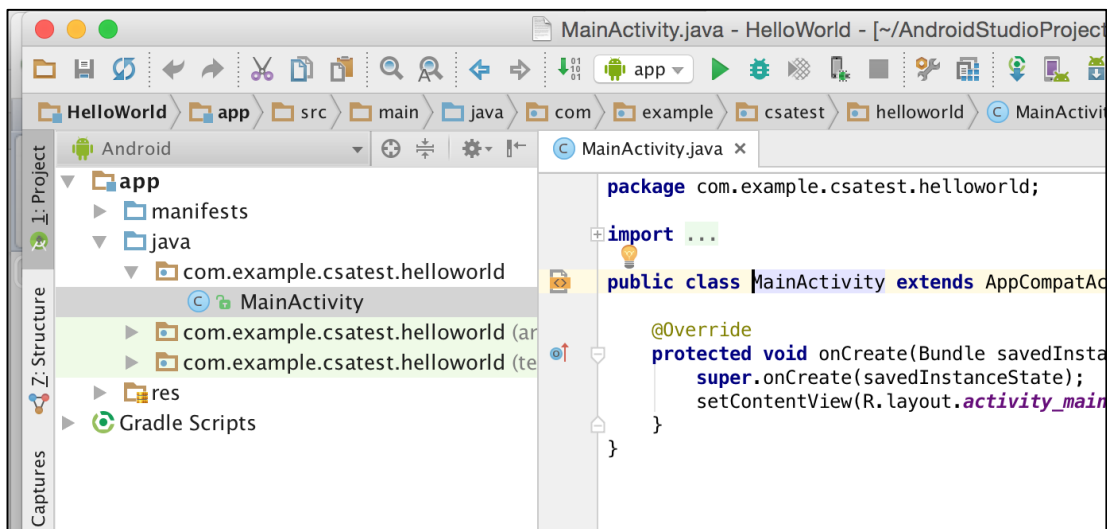
A given **API** (Application Programming Interface) and Android OS version determines the percentage of Android users reachable when you target an **SDK** (Software Development Kit). You can find out more about this by clicking “Help me choose.” In the CSA course students are asked to investigate what it means to choose an API level. We will not address this in readiness training but you can learn more in the Google Developer documentation:
<http://developer.android.com> (API Guides→Device Compatibility)

5. One of the benefits of using an IDE is that you are provided with project templates. Choose the “Empty Activity” template now and then click "Next".
6. The next screen prompts you to name the new activity's assets. Leave all of the defaults as they are and click "Finish".
7. Your computer will now run the Gradle project automation tool in order to create your project. This may take a few moments.

Gradle is the tool that Android Studio uses to manage all of your project's resources, directory structure, libraries upon which your project may depend, and the building of the binaries for your app.

PART II: Say hello to your our first app

1. You have now set up all of the files you need for your new app in Android Studio. If you do not see anything on your screen, click “1: Project” on the left side as shown below. Open the java folder and double click the MainActivity file to open it in the viewer on the right. **MainActivity.java is a Java file.**

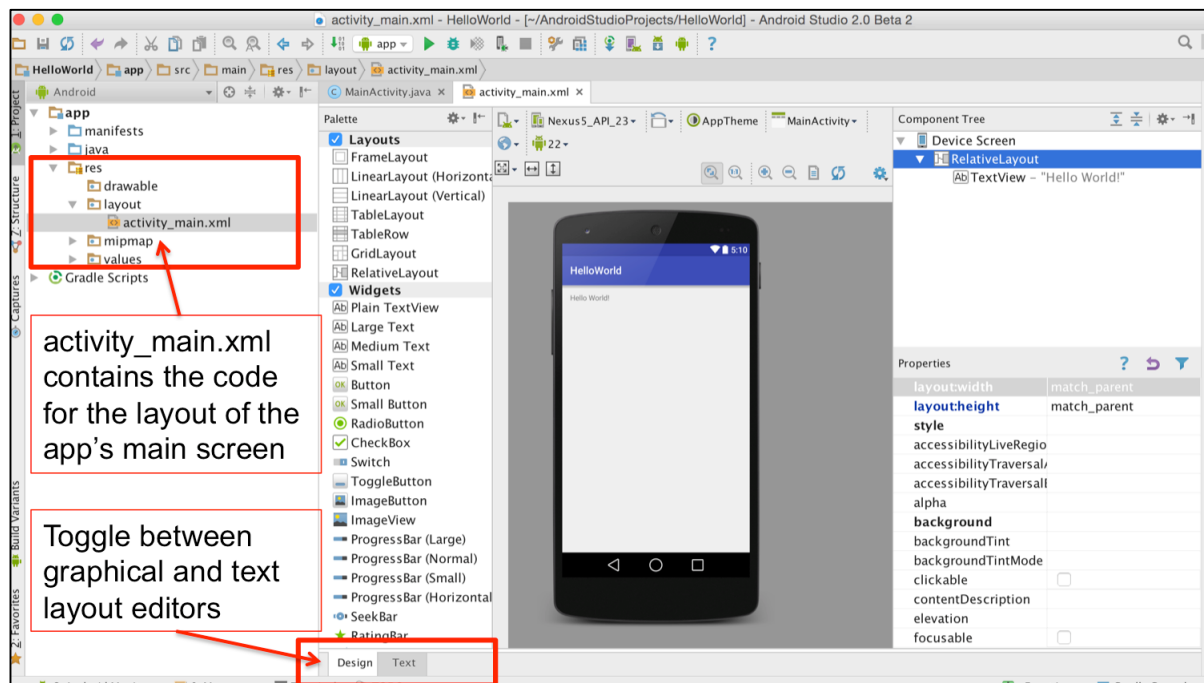


As you can see, Android Studio has set up some of the necessary code for your app. MainActivity.java already has a MainActivity class. The code for this class sets the ContentView to something called `R.layout.activity_main`.

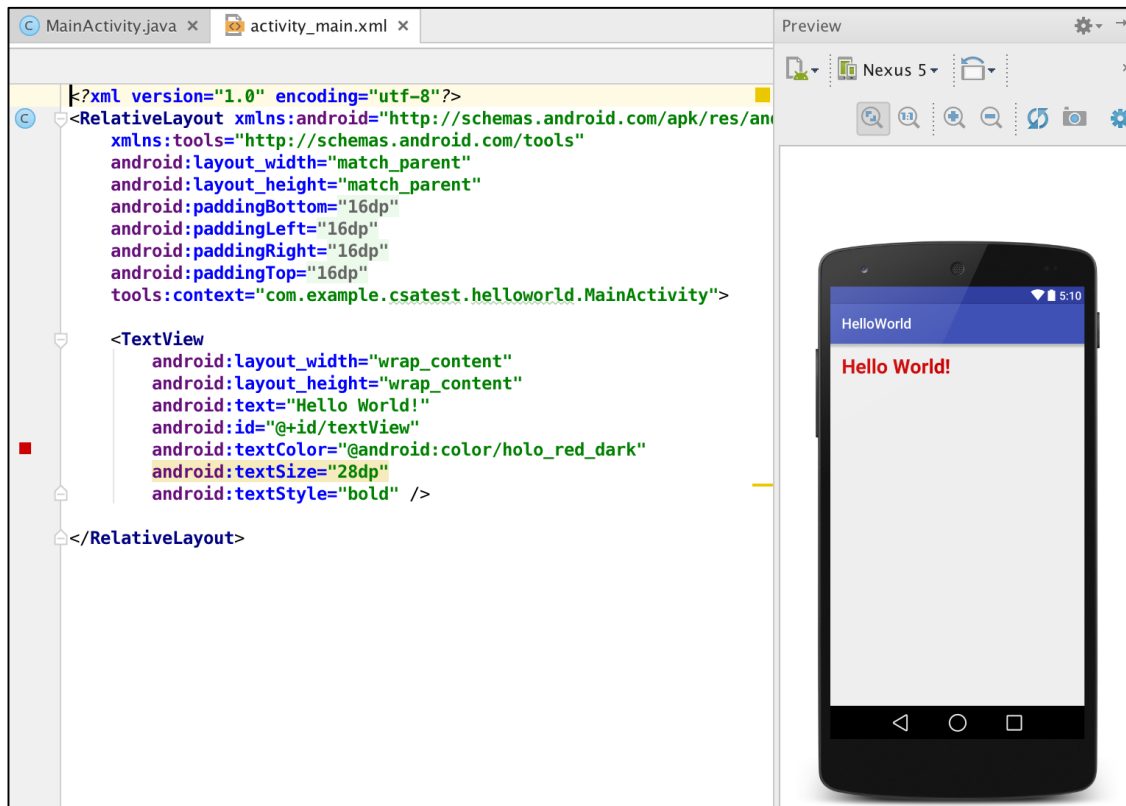
Don't worry if you don't understand all of the Java code. You will learn more in the next readiness training module. What you need to understand now is that **activity_main.xml is an XML file** (eXtensible Markup Language) and that this file dictates the graphical interface for the app. MainActivity.java is simply pointing to this XML file as a resource.

Essentially, the MainActivity class has a method named setContentView that connects the java file with the XML file. Android Studio allows you to view and edit this file easily, as you will see in the next step.

2. In the project structure on the left side, open the “res” folder and then the “layout” folder. Choose “activity_main.xml” and you will see it appear on the right side of your screen, as shown below.



3. If for some reason you do not see the Palette, Designer, Component Screen, and Properties as shown above, click the **Design** tab at the bottom of the screen as shown in the image above.
4. What you are seeing is Android Studio's Graphical Interface designer. This tool allows you to set up your app visually without writing code. Toggle to the Text view to see the XML code that represents what you are seeing on your device preview. Your screen should look like the image below.




5. Toggle back to the **Design view** using the tab at the bottom of the screen, as shown in step 2 above.
6. Click once on the label “Hello World” that is showing in the mobile phone preview. The **Properties pane** on the lower right side of the window will display the properties for this visual element. Scroll through the properties until you find the **font size** and **font color**. Change the font size and font color and watch what happens on the screen. The image above shows the preview after the `textsize` has been changed to 28dp and the `textcolor` has been set to `holo_red_dark`. Notice that these changes are shown in the screenshot above step 5.
 - a. Find other colors in the Android documentation and try them out:
<https://developer.android.com/reference/android/R.color.html>
7. After making changes in the Design view, switch back to the text view using the tabs at the bottom of the screen and try to see where your changes to the text properties show in the XML code.

8. It's time to test your app on a device. Follow the steps below to set up and run an Android Virtual Device (AVD). AVD emulates an actual device with specific options that you define in the AVD Manager.
9. Launch AVD Manager from Android Studio by selecting **Tools > Android > AVD Manager**. (You may also click the AVD Manager icon in the toolbar.)



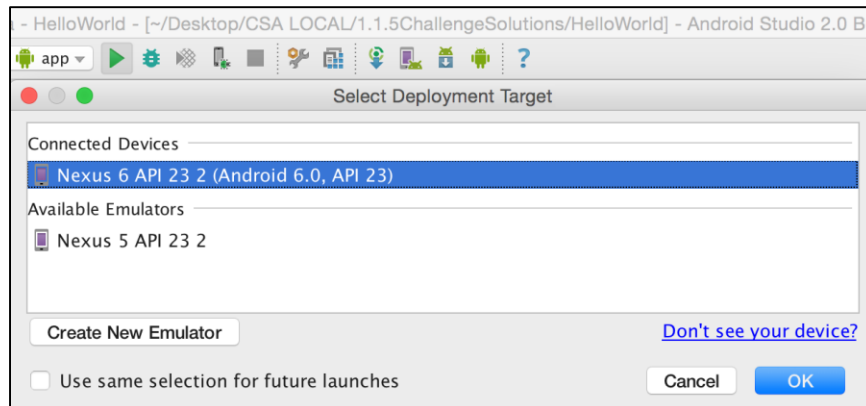
AVD Manager Icon

- a. The AVD Manager main screen shows your current virtual devices.
- b. Click **Create Virtual Device**.
- c. In the Select Hardware window, select a device configuration, such as Nexus 6, then click **Next**.
- d. The next screen asks you to choose the system image version for the AVD. Select whatever is recommended and click **Next**. Devices with higher API's can run apps created for lower API levels. You set this project up with a minimum target level 22 so anything higher than that will work.
- e. On the next screen click **Finish**.
- f. To launch the AVD in the Android Emulator, click the green launch arrow  in the list of AVDs. After a short wait you will see an image of an



Android device on your screen.

10. Run your app on the emulator: Click the green arrow in the toolbar, or select **Run→Run app** and wait for your app to load on the AVD.



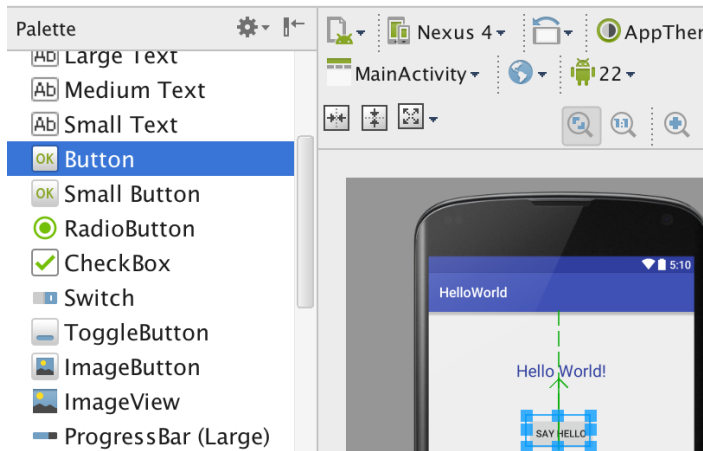
Testing on a physical Android device

It is possible to connect an actual Android device but you will need to figure out the exact steps for your computer and phone. The Android Developer documentation will be helpful: <http://developer.android.com/tools/device.html>

11. Technically you've made an app! But it's not very exciting yet, is it? In the next part you'll add a button and program the app to respond when the button is clicked.

PART III: Your first app says hello to you!

1. Switch to Design view. Go to the Palette and find the component named "Button".
 - a. Click and hold on the Button component while dragging it over to the device preview, then drop it onto the device's screen. Android Studio will automatically give the button default properties such as "New Button" for its display text.



2. The first step in programming the button to do something is to set up a *listener*. In this case, it is called an *onClick* listener and it gets triggered when someone clicks the button.

A button click is just one of many different types of *events* that happen in Android. The *onClick* listener calls some code in the Java program called a *method*. Follow the steps below to set up the *onClick* listener and the corresponding method.

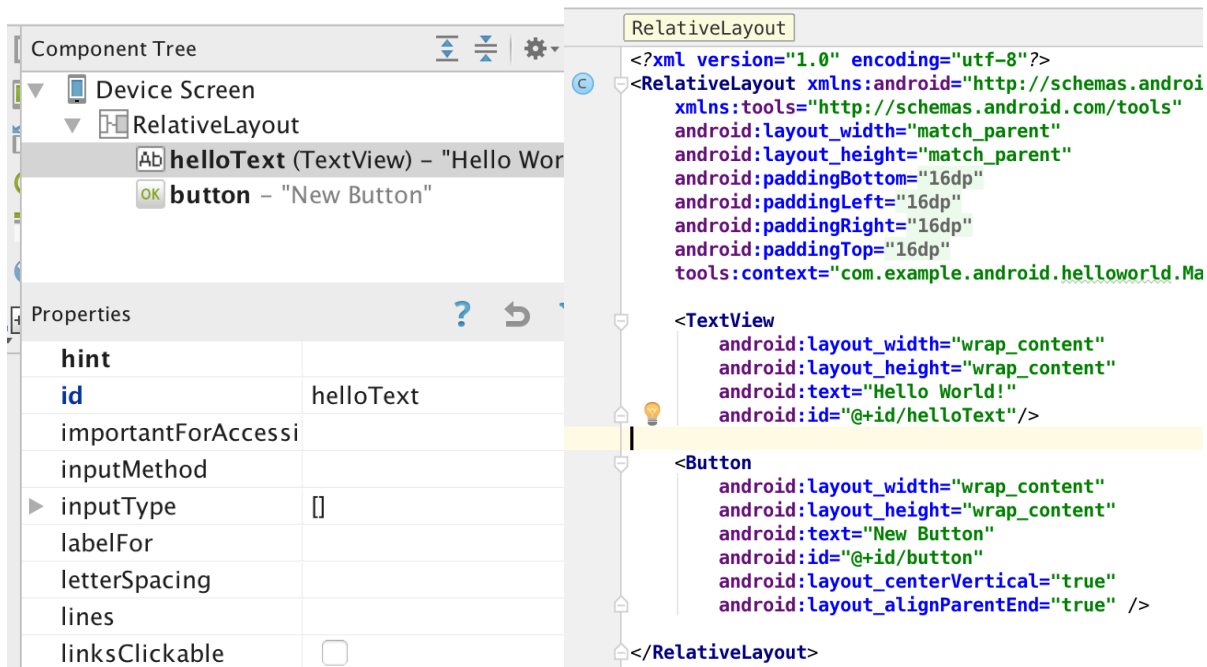
3. Set the *id* property of the *TextView* component to “helloText” so that you can refer to it from the Java code.

There are two ways to do this: either in the XML code or in the Component Tree/Properties manager. The screenshots below demonstrate both ways. Choose one. Note that the XML must have `@+id/` before the name, as shown below. This prefix enables Android to keep all of the component *id*’s together in one place in memory.

- 1) set **id** to `helloText` in Properties of the Component Tree

—OR—

- 2) add `android:id="@+id/helloText"` to **Button** in `activity_main.xml`



4. Open MainActivity.java so that you can write the method that will be called when the button click event occurs. Insert the code **before the last curly brace** in the class MainActivity, as shown below. Notice that Android Studio auto-completes as you type. The name “changeText” is a made-up name for the method. The method takes one argument, an object of type View named “v”. This simply means that when the button is clicked, the method will receive a reference to the view that contains the button.

```
package com.example.android.helloworld;

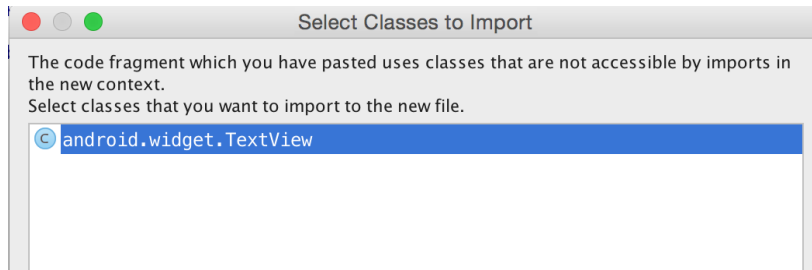
import ...

public class MainActivity extends AppCompatActivity {

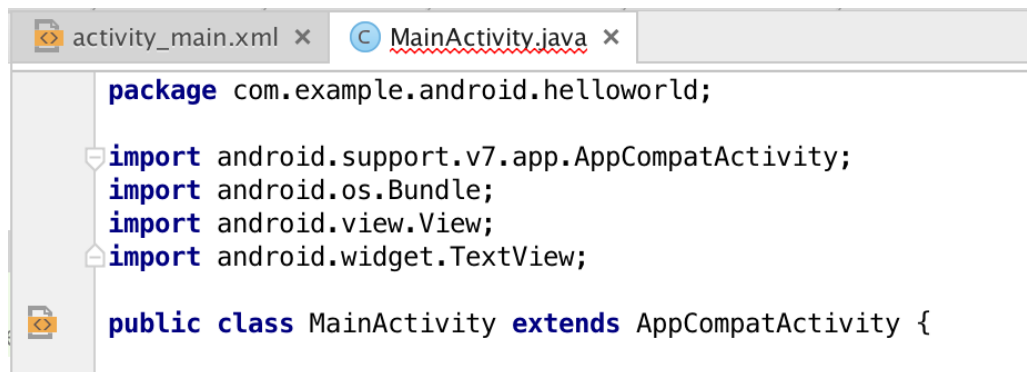
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /* change the text displayed in helloText component */
    public void changeText (View v) {
        TextView
        TextView (android.widget)
        TextViewCompat (android.support.v4.widget)
        TextureView (android.view)
    }
}
```

5. As you type in `TextView`, Android Studio will notice that you are referring to a class library that has not been added to the project yet. Select the `android.widget.TextView` class to import it.



6. To see the effect of importing this class, expand the import section by clicking the small white arrow next to “import”. You will see that `android.widget.TextView` has been added. This automatic import is an example of something the IDE does for you that makes Android development easier.



7. Finish typing in the code as shown below.

PLTW | Computer Science

```
package com.example.android.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

    /* change the text displayed in helloText component */
    public void changeText (View v) {
        TextView greetingText = (TextView) findViewById(
            R.id.helloText);
        greetingText.setText("Hello There!");
    }
}
```

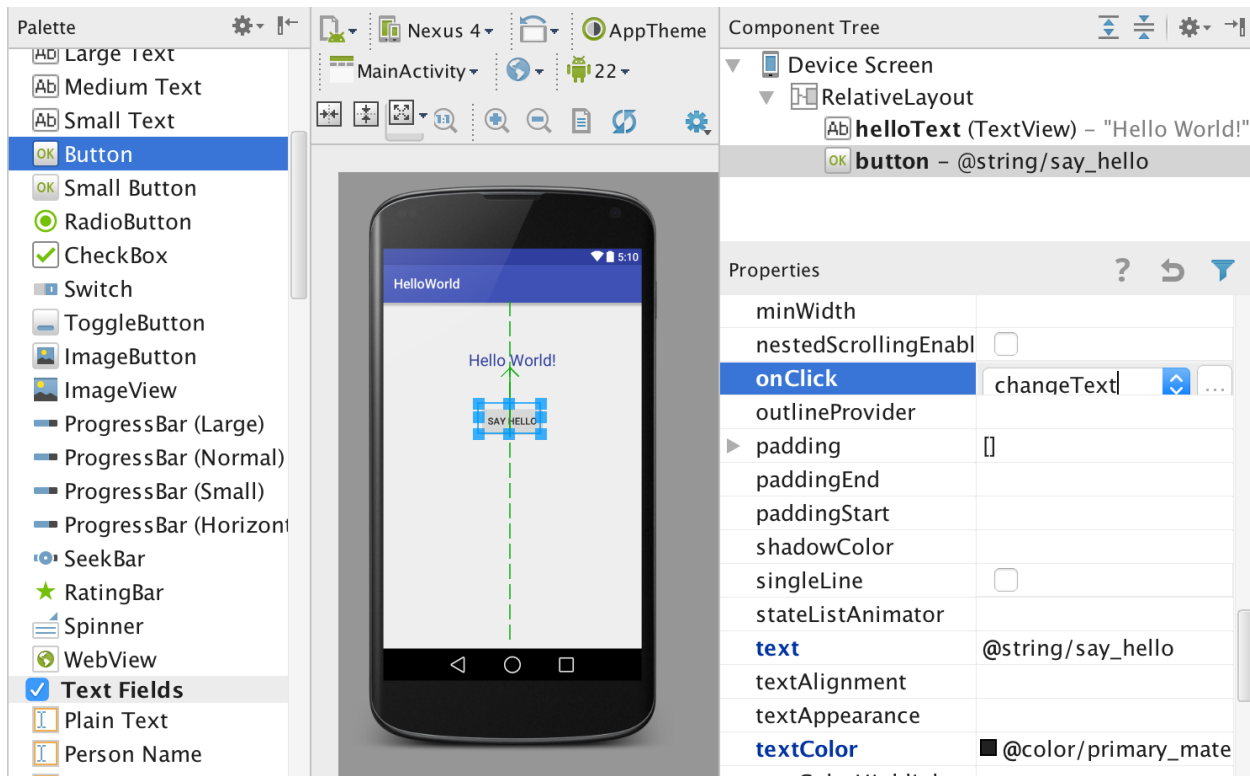
The app developer chooses a name for the new method. In this case, "changeText".

findViewById searches for the resource whose id name is "helloText".

greetingText is a variable of type TextView. The app developer also chooses this name.

greetingText refers to the helloText component. This allows the setText method to change the value of the text property in the helloText component.

The last step before you can test your new functionality is to tell the onClick listener that the name of the method to call is changeText. The image below shows how you can use the dropdown box next to the onClick property in the ComponentTree's Properties pane to see the available methods. The new method called changeText should be in the list. If it is not, be sure you have completed the earlier steps in MainActivity.java.



8. Run your app on the Android Virtual Device by clicking the green arrow.
9. Now that you've programmed the button to call the "changeText" method, pressing the button should cause the text on the screen change from "Hello World!" to "Hello There!".
10. Congratulations on building your first app! The next way to improve your app would be to accept text input so that a user can enter a name and the app can say hello to that person by name. You can learn how to do this in Part V (optional).

Part IV (Optional): Taking User Input

To accept a user's name and make the app say a personal hello, you'll need to

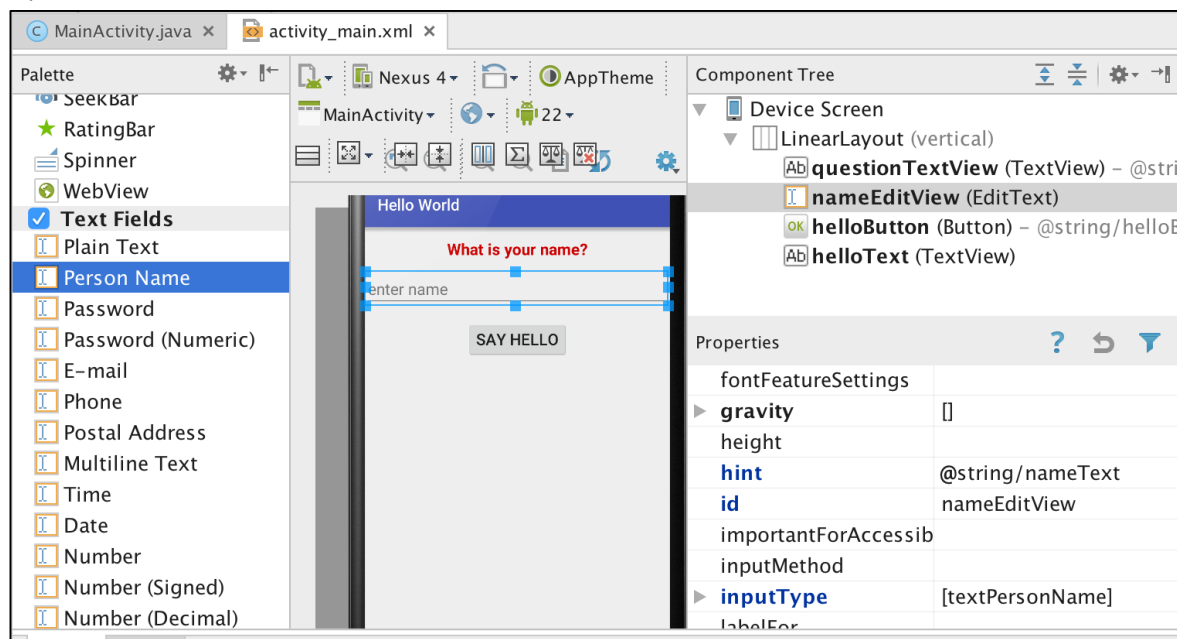
- An EditText component to the UI. This is a field on the screen into which the user can type new text.
- Write code to tell the changeText method to take the text from the EditText field and add it to the "Hello" greeting. The screen shots below show an example of a working app. Below that you will find instructions for adding the new component and coding the new behavior.

You can copy, or cut and paste, the code shown below into your project, but you must use the exact same names.

1. Add a new EditText component named nameEditView. Type into the XML directly (A) or use the Design view (B).



B.



2. Write code in the `changeText` method to retrieve the text that was entered into the text box and concatenate it in between the strings “Hi ” and “ , nice to meet you.” See sample code below.
 - a. Add a new variable called `nameEntered` of type `EditText`. Assign the variable a reference to the `EditText` component. (line #26)
 - b. Set the variable `greetingText` to a new string. (line #29)
 - i. The get method that you use is called `getEditableText()`. The `nameEditView` component has this functionality because it is an object of the class `EditText`. This method is often referred to as a *getter*. (line #31)
 - ii. The set method that you use is called `setText()`. The `helloText` component has this functionality because it is an object of the class `TextView`. This method is often referred to as a *setter*. (line #31)
3. Run your program. Type your name in, click the button, and enjoy a personal hello message!

```
20  /**
21   * Take the name entered in the nameEditView component and insert it into the greeting that
22   * is displayed in the helloText component. Called by SayHello button's onClick attribute.
23   */
24
25  public void sayHello(View v) {
26      EditText nameEntered = (EditText) findViewById(
27          R.id.nameEditView);
28      /* Set the Greeting in helloText with this new name */
29      TextView greetingText = (TextView) findViewById(
30          R.id.helloText);
31      greetingText.setText("Hi" + " " + nameEntered.getEditableText() + ", nice to meet you!");
32  }
33
34  }
35
```