

## Hello World (10 pts.)

### Lab Objectives:

- Create a new Android project in Android Studio
- Add a View to the layout
- Learn how to use buttons to call methods
- Learn how to programmatically modify Views
- Learn about resources

### What to Turn in:

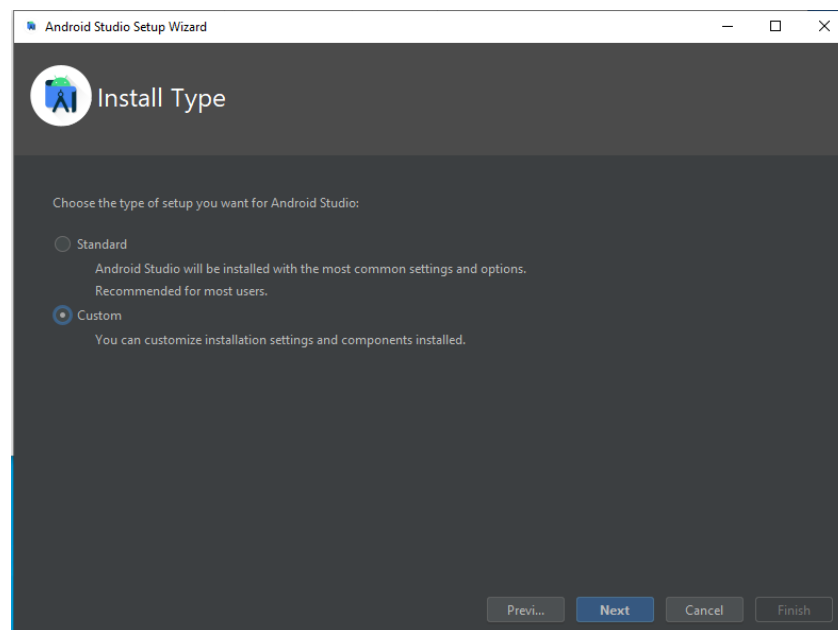
- Demonstrate your working app to a TA.

### Building Your First App

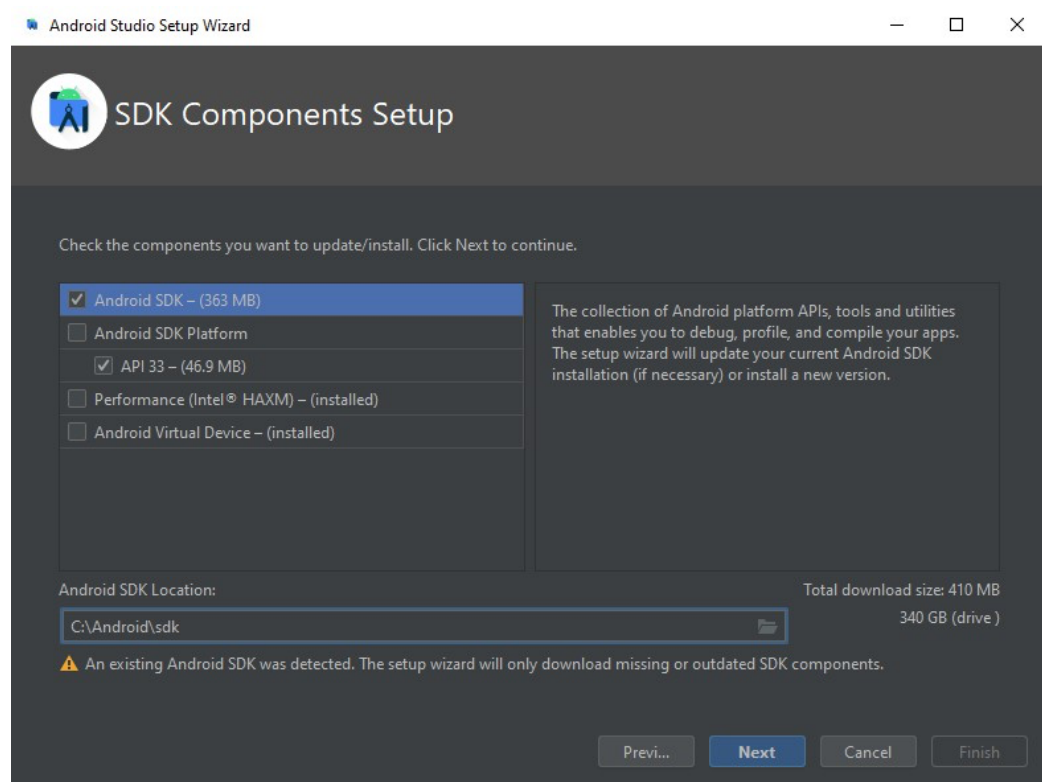
1. Use Android Studio on the lab computer, or install Android Studio on your own computer (instructions are posted on Canvas for both of these options). If you're opening it for the first time, you may have to go through a setup wizard. If you're on a lab computer, you may want to change the default save location to your U:.

On the lab computers, **do a custom install so you can edit the SDK path.**

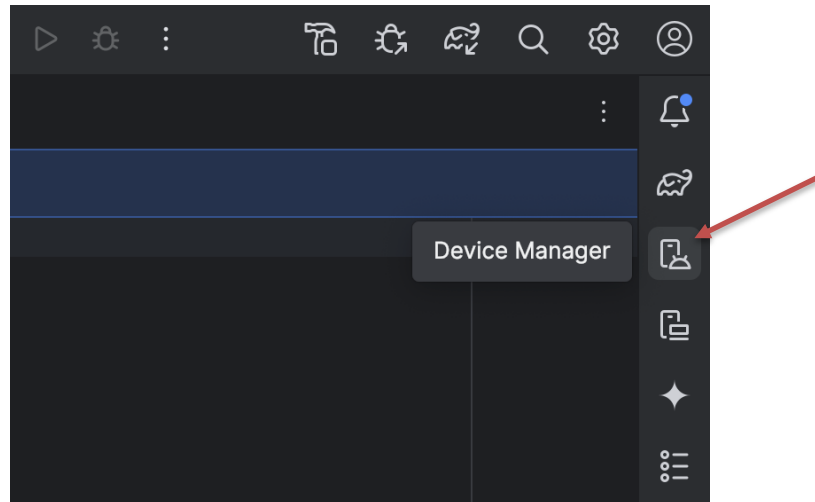
- a. Choose custom on the Install Type screen.



- b. Take the defaults until you reach the SDK components setup screen. On this screen, change the Android SDK Location to C:\...\Android\sdk (which should already exist on the lab machine).



- c. Take the defaults for the rest of the wizard.
  - d. **If you get a Windows prompt to enter an admin username and password, ignore it (click No).**
2. In general, when opening Android Studio, if you are prompted to update SDK components or tools or Gradle, it is usually a good idea to do so (though we have seen Gradle updates break things in the past).
3. Create a new Project in Android Studio. Use the following settings:
  - a. Empty **Views** Activity (NOT Empty Activity)
  - b. Name: Hello World
  - c. Save location: use your U: on a lab computer, otherwise the default is fine
  - d. Language: Java (unless you plan to use Kotlin in this course)
  - e. Minimum SDK: Just use the default values for now
4. After you click finish, a number of things will happen. You'll see some messages about Gradle in the lower-right. Gradle is the build system (similar to Make) used by Android Studio.
5. While you're waiting for the Gradle Sync to complete, you can install an Android Virtual Device (AVD) for use in the emulator. Click the Device Manager button in the upper-right and select Create Device on the Virtual tab (yours may look a little different).



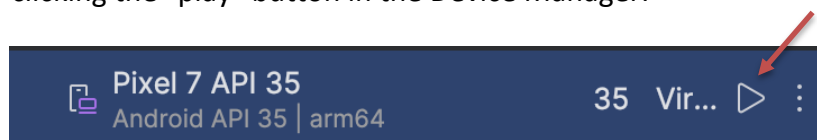
- a. We recommend selecting a device with the Play Store included in the image, such as the Pixel 7 shown below.

**Choose a device definition**

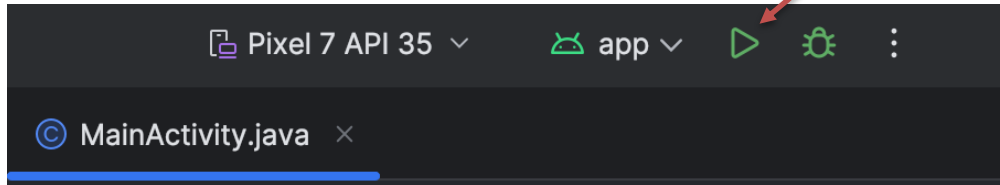
Q-

Category	Name	Play Store	Size	Resolution	Density
Phone	Small Phone		4.65"	720x1280	xhdpi
Tablet	Medium Phone		6.4"	1080x2400	420dpi
Wear OS	Pixel Fold		7.58"	1840x2208	420dpi
Desktop	Pixel 8 Pro		6.7"	1344x2992	xxhdpi
TV	Pixel 8		6.17"	1080x2400	420dpi
Automotive	Pixel 7a		6.1"	1080x2400	420dpi
Legacy	Pixel 7 Pro		6.71"	1440x3120	560dpi
	<b>Pixel 7</b>		<b>6.31"</b>	<b>1080x2400</b>	<b>420dpi</b>
	Pixel 6a		6.13"	1080x2400	420dpi
	Pixel 6 Pro		6.7"	1440x3120	560dpi

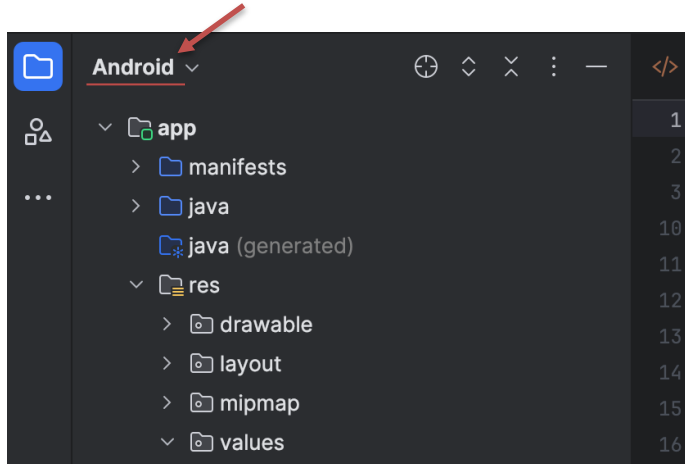
- b. For the system image, you can either choose one of the existing images or download a new one. (API 35 is recommended in the current version)
- c. Other settings can remain as the default. Click finish and boot up your new AVD by clicking the "play" button in the Device Manager:



- Hopefully Gradle is done with the project setup by now. Once the device boots, run the Hello World app on the device by selecting the device in the dropdown in the upper-right and clicking the play button. This may take a minute. If it fails the first time, try it again.

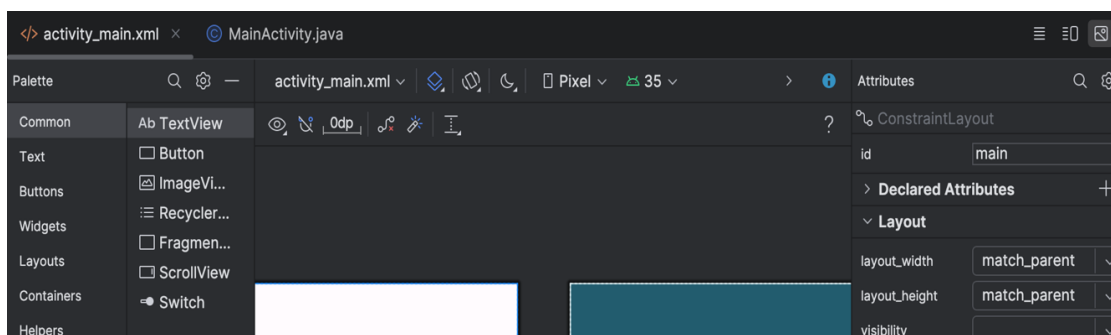


- You should now have a running app with a single "screen" that displays the string "Hello World".
- Let's look at how the layout for this screen is defined. In the Project explorer on the left side of the screen, make sure you're looking at the Android view of the project files.

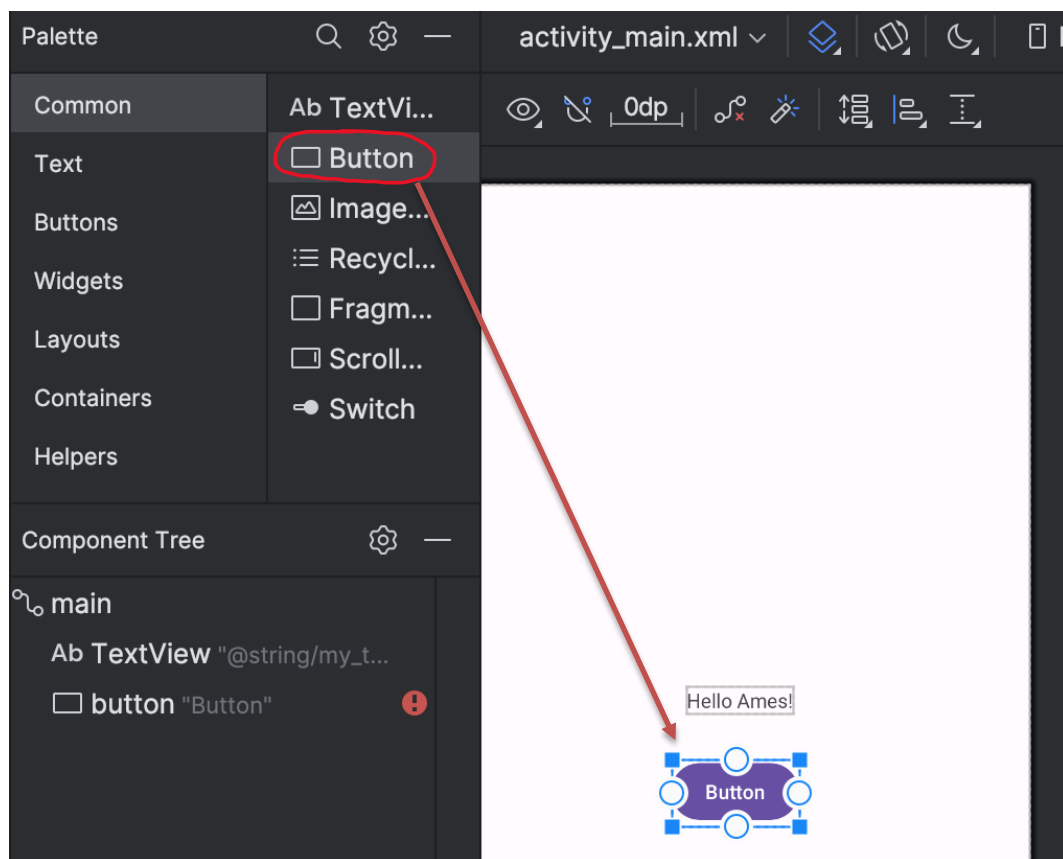


- From here, open the res folder. This folder contains the app's *resources*, which are static non-Java files bundled into the app - mostly XML and media like images.
- In the layout subfolder, open activity\_main.xml. This XML file defines the initial state of the layout for the Activity called Main in the project (more on Activities later). There's a lot to look at here. Note that you can edit the layout graphically, or you can edit its XML directly, using the buttons in the upper right of the pane.

The three icons on the top right helps to switch between design view, code view and split view. Take a peek at the code view, but we'll use the Design view for now.



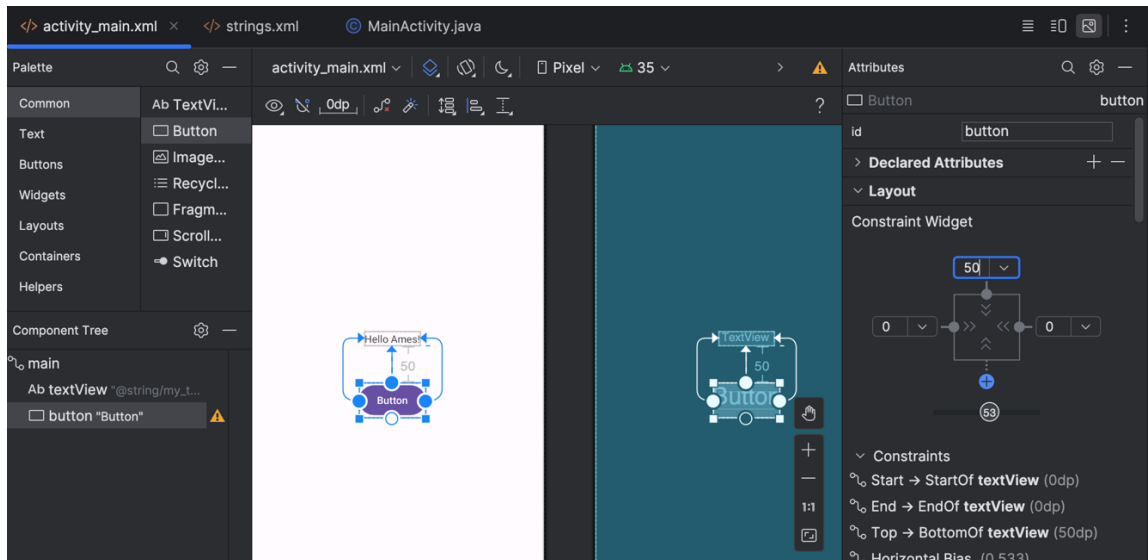
11. In the design view, select the "Hello World" text by clicking on it. If you look at the Component Tree in the lower-left part of this pane, you'll note that you've selected a TextView that is a child of a ConstraintLayout. When this layout is loaded at runtime ("inflated"), objects of those classes will be created and given that parent-child relationship.
12. Look at the Attributes on the right side of the pane. Find the attribute of the TextView that determines the content of the TextView (i.e. the string "Hello World").
13. It's generally bad practice to hardcode a string literal directly into a layout. What if you want to use this value in multiple places, or what if you want to be able to easily translate your app into multiple languages? Android encourages developers to think of string values as *resources*. Open `res->values->strings.xml` and add a new `<string>` element named `"my_text"` with the value `"Hello Ames!"`.
14. Now use this resource in your layout instead of the hardcoded string. In `activity_main.xml`, change the text attribute for the TextView to `"@string/my_text"` (it should autocomplete). This demonstrates how to reference one resource (the string) from another (the layout).
15. Next, let's add a button. Click and drag a Button from the Palette onto the layout.



16. The Button now also shows up as a child of the ConstraintView in the Component Tree, but there's a warning icon that says the Button is "missing constraints" when you hover over it. All children of ConstraintLayouts need to have a position defined relative to other elements (i.e., constraints).

17. Define some constraints for the Button. One way to do this is to select the Button, click and hold on one of the white circles that appear on the Button's edges, and drag it to a blue circle on the constraining element (or the edge of the layout). You can set the distances between the elements, for each constraint, in the Constraint Widget.

Note that the Button needs to be both horizontally and vertically constrained. There are a variety of interesting rules that can be used. In the below screenshot, we are vertically constraining the Button to be 50dp (device-independent pixels) below the bottom of the TextView. We are horizontally constraining the left side of the Button to the left side of the TextView, and the right side to the right side. These balancing forces result in the Button being horizontally centered on the TextView.



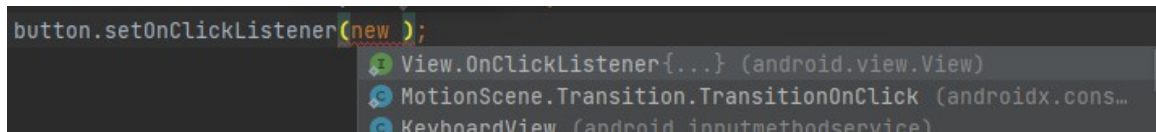
18. Now the icon in the Component Tree tells us that the Button has hardcoded text. Replace the Button's hardcoded text with a string resource that says "Change it!" or something similar.
19. If you run the app now, you'll find that clicking the Button doesn't do anything. Guess what - it's up to us to define what the Button does! To do this, we need to implement a "listener" method and attach it to the Button.
20. A listener should be implemented in Java, but where's the Java code for this screen? Open java->com.example.helloworld->MainActivity.java. This is the MainActivity class definition. Note that MainActivity is a subclass of [AppCompatActivity](#), which is itself a subclass of many things. Activities are top-level *controllers* in Android - that is, they contain the code to define the behavior for a single screen.
21. MainActivity currently just has an override for onCreate(), a lifecycle callback that we'll talk about in class. Basically, this method is called when the Activity is first created, so it's **kind of** like a main() method or a constructor for this screen. We can define our button listener here. At the end of onCreate(), add this line:

```
Button button = findViewById(R.id.button);
```

When you type "Button", it will initially be highlighted in red because the Button class has not been imported in this file. You can automatically import it by pressing alt+enter with your cursor on (or just after) the class name.

The `findViewById()` method comes from Activity. It searches the view tree of the Activity for a View with the given ID. You can reference the ID of a component defined in XML via "R.id.<whatever>" (which is really referencing an ID resource that is autogenerated based on the IDs given to components in XML). If you look at `activity_main.xml`, you'll find that the Button was given the ID "button" by default, so `R.id.button` refers to the Button you added to the layout.

22. Now you have a reference (or "handle") to the instantiated Button. You can dynamically attach a listener to the Button by calling `setOnClickListener` on it, passing in a `View.OnClickListener` object (i.e. an object that implements the [View.OnClickListener](#) interface). You could write yourself a custom class that implements this interface, but more typically you'll use an *anonymous* class. The syntax for this is painful - we recommend letting Android Studio autocomplete it for you by pressing enter when you get to the point shown below (making sure that `View.OnClickListener` is highlighted).



```
button.setOnClickListener(new );
    View.OnClickListener{...} (android.view.View)
    MotionEvent.Transition.TransitionOnClick (androidx.cons...
    KeyboardView (android.inputmethodservice)
```

23. This gives you code that defines, in-line as a parameter to `setOnClickListener()`, an instance of a nameless class that just implements the `onClick()` method of the `View.OnClickListener` interface. Within that `onClick()` method, add the below code to get a handle to the `TextView` and change its text:

```
TextView textView = findViewById(R.id.textView);
textView.setText("Hello ISU!");
```

24. Run your app again - you should now find that clicking the button changes the text.
25. Finally, a little Java practice - update your app so that clicking the button causes the text to randomly cycle through some number of messages. These messages can be hardcoded (e.g. in an `ArrayList<String>`), or string resources, or randomly generated, or whatever seems the most interesting to you.

If completed correctly, the app should:

1. Display some text and a button. (4 pts.)
2. Change the text when the button is pressed. (4 pts.)
3. Change the text to a random message every time the button is pressed. (2 pts.)

**Demonstrate your working application to one of the TAs during lab time or TA student hours.**

For documentation on the Android API you can go to:  
<http://developer.android.com/reference/packages.html>.

The Android API guides are also a good place to look for ideas and can be found at:  
<https://developer.android.com/guide>