

PROJECT 1.2.4

Create an Android Project

INTRODUCTION

Up until now, you have imported existing projects into Android Studio. You will now create your own new project that plays sounds when buttons are clicked.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator

RESOURCES

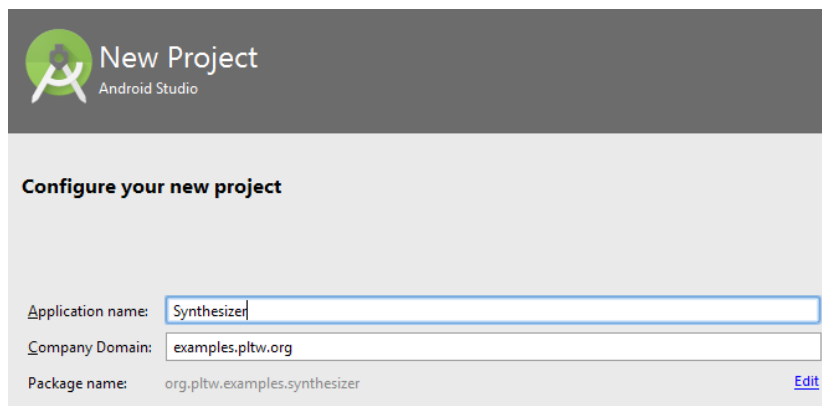


Lesson 1.2 Reference Card for Strings
Resources available online

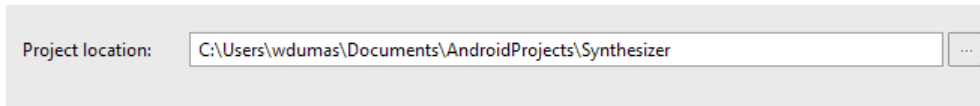
Procedure

Part I: Create Something New

- 1 Open Android Studio; the last project you were working on will be loaded.
 - a. From the menu bar, select **File > New > New Project...** and name your new project **Synthesizer**.



- b. To the right of the Project location text box, click the ellipsis (...) to specify a location for your new project.



- c. The path to this location should not contain any spaces. For example

`Users\wdumas\Documents\Android Projects\Synthesizer`

is not recommended, but

`\Users\wdumas\Documents\AndroidProjects\Synthesizer`

would be okay.

In the next window, you will be able to determine what percentage of Android users you can reach by targeting a given **API (Application Programming Interface)** and Android Operating System (OS) version. An API includes classes and methods to use while developing an app.

NOTE

As you view documentation from third parties and Google, you will likely notice that “**SDK** version” is often used synonymously with “API level”.

Software Development Kit (SDK)

Software that is part of the Android™ Studio IDE that provides functionality, including user interface items such as buttons and textboxes.

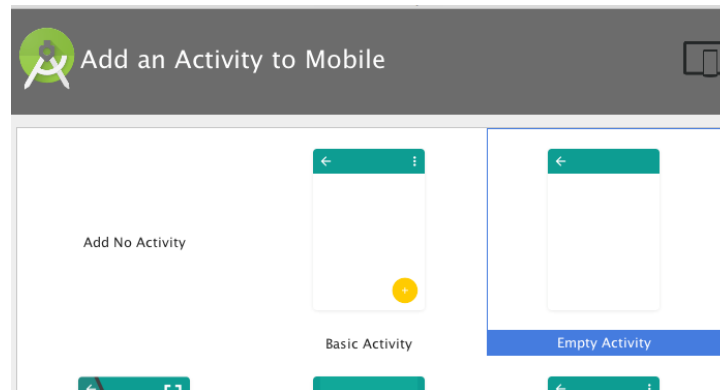
- d. From the drop-down menu labeled **Minimum SDK under Phone and Tablet**, select **API 22: Android 5.1 (Lollipop)**.

At the time of this writing, the message that appeared beneath this selection indicated that 18.4% of devices active on the Google Play Store would be able to use an app that targets this API level. This percentage is in bold as in the image below.

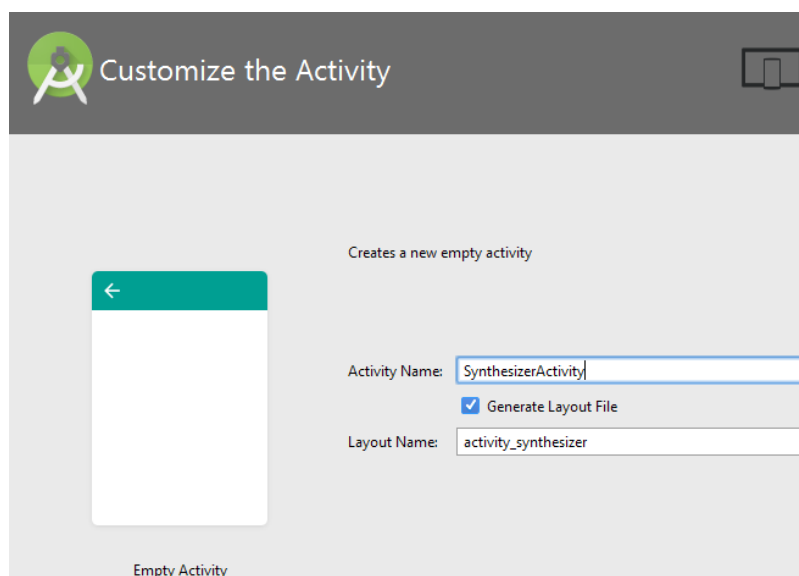


- e. Experiment with the drop-down menu from the previous step to determine the lowest numbered API level that you could target and reach 97% of the Android market. Record your answer.

- f. For now, select **API 22 (Lollipop)** as your minimum target, because this will allow you to ignore some of the more cumbersome aspects of Android development while you're learning.
- g. Select **Next** to advance to the next window and select **Empty Activity**.



- h. Click **Next** and change the Activity Name to **SynthesizerActivity**. Notice that the **Layout Name** changes automatically.

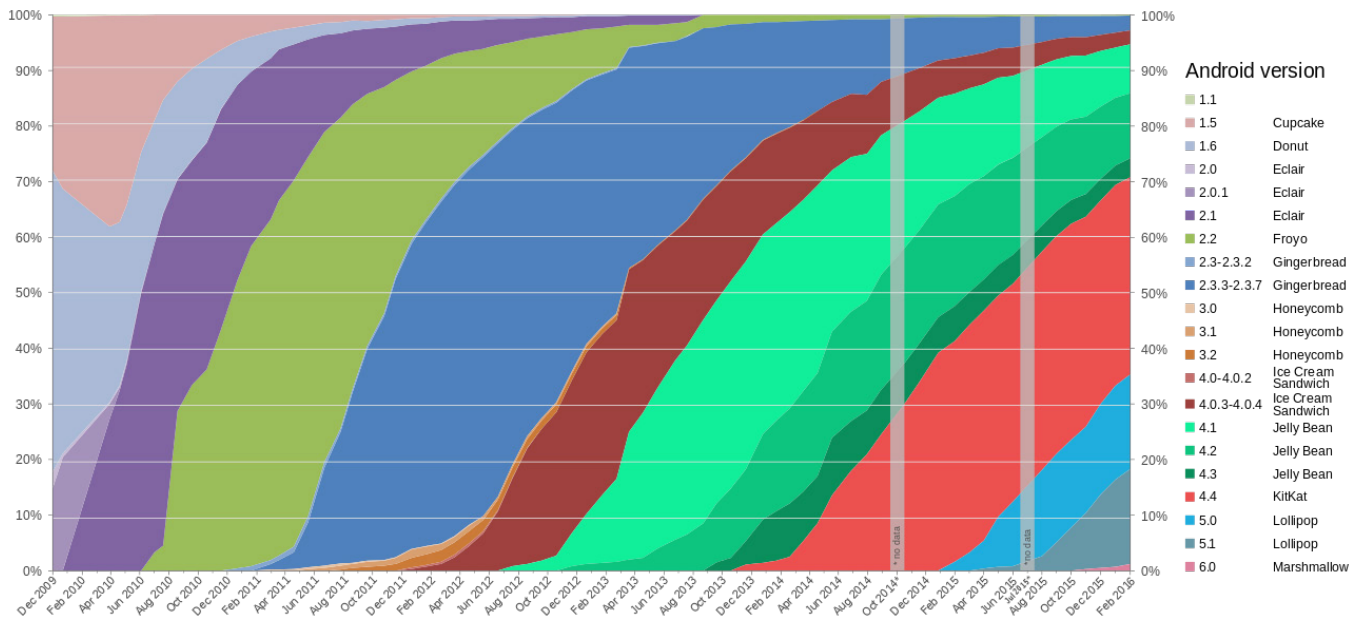


- i. Click **Finish**.

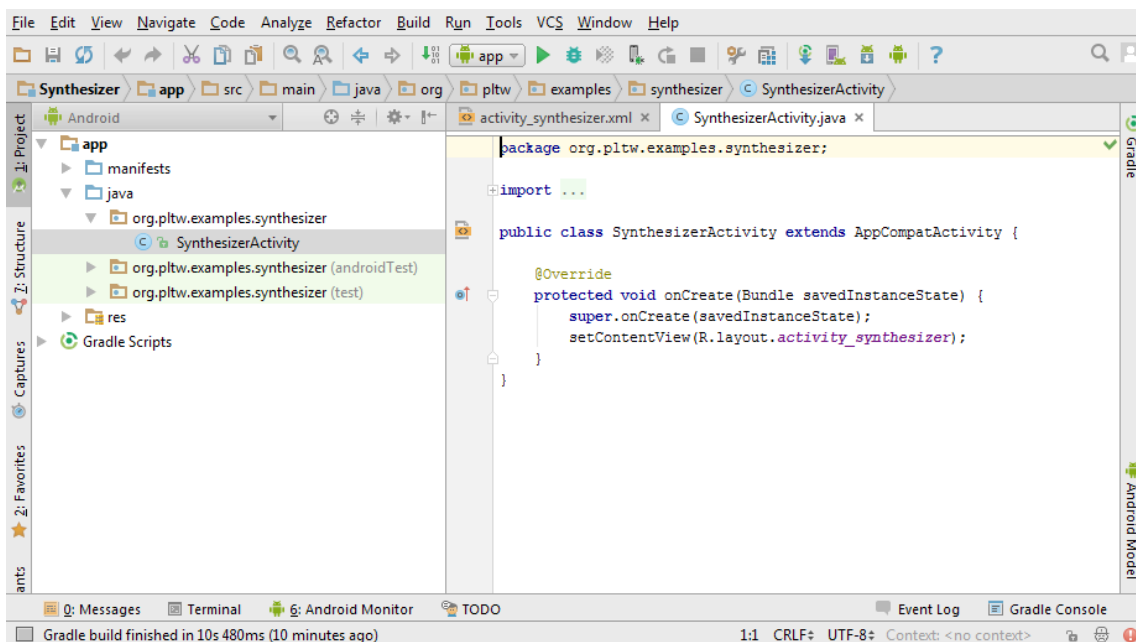
The Gradle project automation tool will now run to create your project.

Android Studio uses Gradle 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. This Gradle run may take a few moments. Use this time to answer the following question:

- j. The following graph shows the percent of Android devices on the y-axis and date on the x-axis. In what year would you predict that app developers will no longer need to support Android versions lower than 5.1 (Lollipop)?



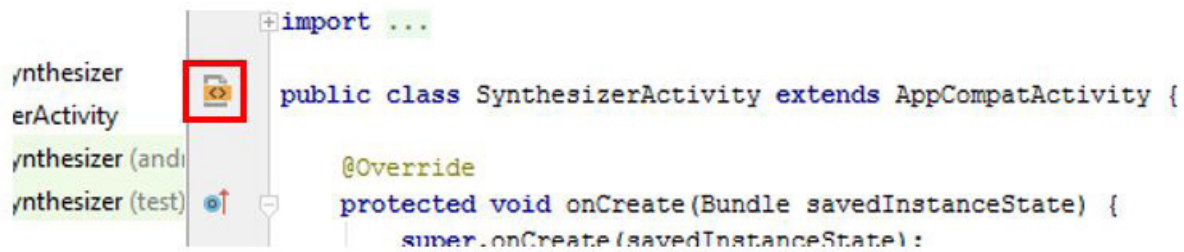
By now Gradle should have finished creating your project, and you should see a window very similar to the one below.



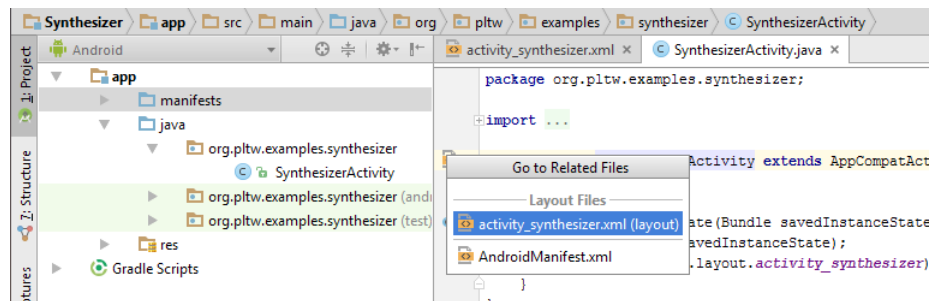
Part II: The User Interface

One convenient feature of Android Studio is the graphical tool. Android developers use a language called Extensible Markup Language (XML) to specify and organize the user interface (UI) components of their apps. The graphical design tool in Android Studio automatically generates the appropriate XML for the UIs that you create using it.

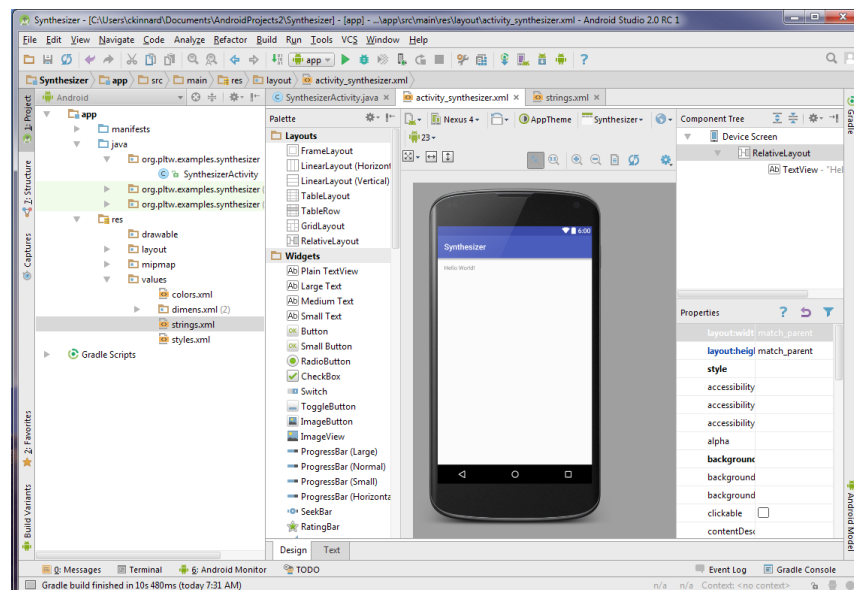
One of the ways to view the XML file for your project is to use the **Related XML file** menu indicated by the <> icon to the left of your class declaration:



- 2 Click the **Related XML file** icon to display the menu. Select **activity_synthesizer.xml (layout)** (layout).




The **Layout Panel** opens within the Android Studio IDE.

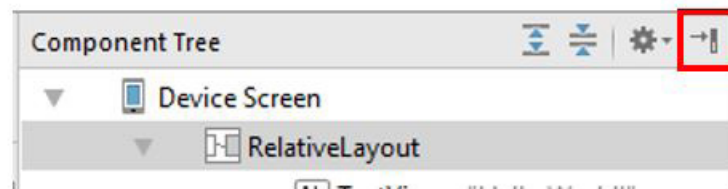


This view includes many panels. You may not always need all of them, and displaying them all can take up significant screen real estate. The **Component Tree** is a panel in Android Studio where you can visualize the relationships between your UI components.

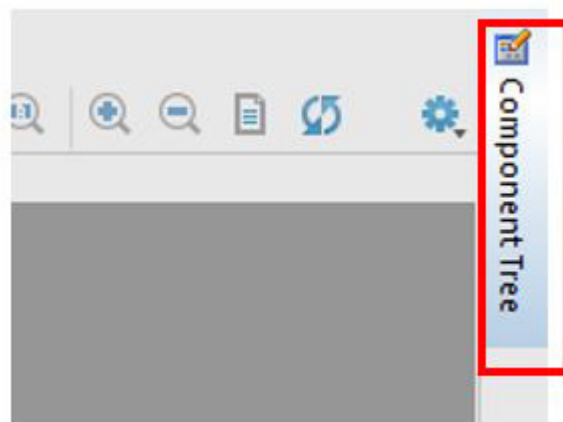
component tree

A structure that visually represents user interface components.

- 3 Hide the Component Tree by clicking the  icon in the title bar of that panel.

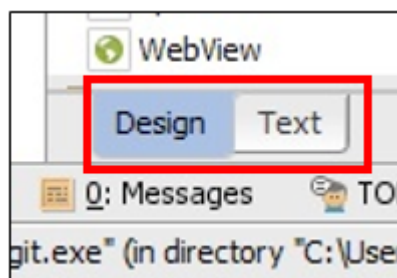


You can show the Component Tree again by clicking its name on the right side of the Android Studio window.



By default, Android Studio shows the XML file in Design mode. You can switch between the graphical (Design) and XML (Text) views using the two tabs at the bottom of the *activity_synthesizer.xml* panel.

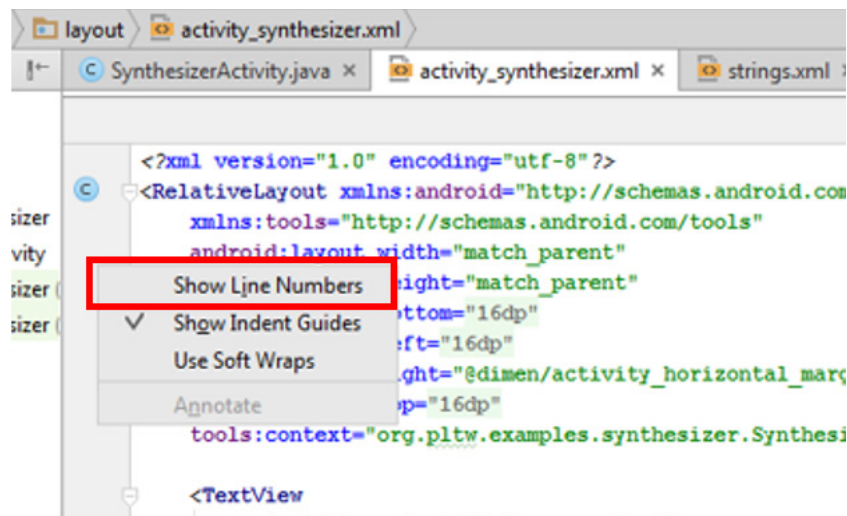
- 4 Select the tab to switch to the **Text** view.



You should see the following XML code in the *activity_synthesizer.xml* panel.

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout
3:     xmlns:android="http://schemas.android.com/apk/res/android"
4:     xmlns:tools="http://schemas.android.com/tools"
5:     android:layout_width="match_parent"
6:     android:layout_height="match_parent"
7:     android:paddingBottom="16dp"
8:     android:paddingLeft="16dp"
9:     android:paddingRight="@dimen/activity_horizontal_margin"
10:    android:paddingTop="16dp"
11:    tools:context="org.pltw.examples.synthesizer.
    SynthesizerActivity">
12:
13:    <TextView
14:        android:layout_width="wrap_content"
15:        android:layout_height="wrap_content"
16:        android:text="Hello World!" />
17: </RelativeLayout>
```

- 5 To display line numbers in the Text view of a source file, **right-click** in the gray area to the left of the source and select the menu item **Show Line Numbers**.

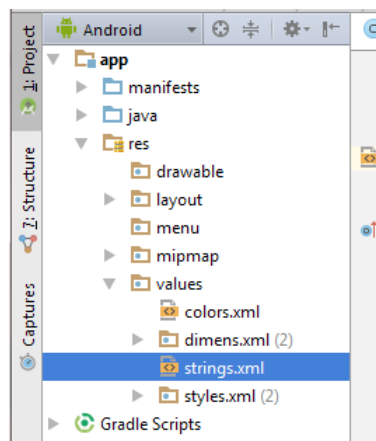


- 6 In any of the lines 3–6, hover your mouse over the characters "16dp". Record what you see.

Android Studio has hidden the path to a resource. Instead, it displays only its value. This is called *folding* and can be toggled on or off by right-clicking the text selection and choosing **Folding > Expand** or **Folding > Collapse**.

As you create Android applications, you'll want to reference string resources and other resources in your code instead of literal values. That way, if you decide you want to change that value in some way, you don't have to hunt through all of your code to find every time you used that value. Instead you will just change the resource once to fix every occurrence.

- 7 To find where these string resources exist, so that you can modify them when necessary, use the Project panel shown below. Click the right arrow next to **res** and then the right arrow next to **values** to expand the contents of the directories.

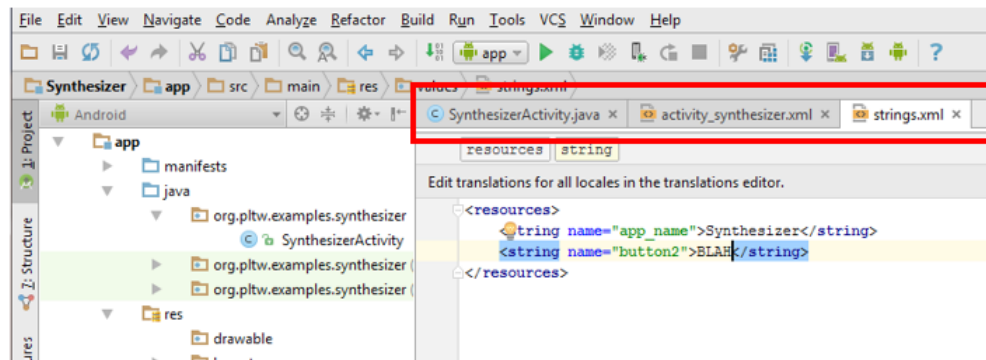


- 8 Double-click the *strings.xml* file to open the file.
- 9 You are about to create a button in your app and the button needs text to display. On line 3 of *strings.xml*, add the following line of code: `<string name="button1">First Button</string>`

This string resource will tell your button what text to display at runtime.

Now that you have created a string resource for your button, you need to create the button itself, and reference this string resource from it.

- 10 Use the file navigation tabs to view *activity_synthesizer.xml* again.



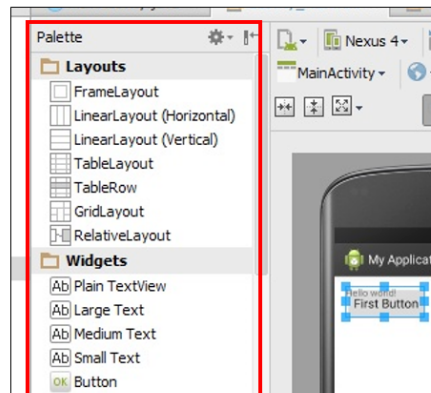
- 11 Add lines 17–22 to modify *activity_synthesizer.xml* to read as follows:

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <RelativeLayout
3:     xmlns:android="http://schemas.android.com/apk/res/android"
4:     xmlns:tools="http://schemas.android.com/tools"
5:     android:layout_width="match_parent"
6:     android:layout_height="match_parent"
7:     android:paddingBottom="@dimen/activity_vertical_margin"
8:     android:paddingLeft="@dimen/activity_horizontal_margin"
9:     android:paddingRight="@dimen/activity_horizontal_margin"
10:    android:paddingTop="@dimen/activity_vertical_margin"
11:
12:    tools:context="org.pltw.examples.synthesizer.
13:    SynthesizerActivity">
14:    <TextView
15:        android:layout_width="wrap_content"
16:        android:layout_height="wrap_content"
17:        android:text="Hello World!" />
18:    <Button
19:        android:layout_width="wrap_content"
20:        android:layout_height="wrap_content"
21:        android:text="@string/button1"
22:        android:id="@+id/button1" />
23: </RelativeLayout>
```

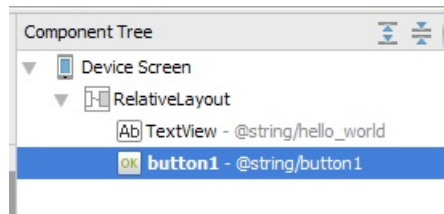
Which line of code tells the button to use the string resource that you created in Step 9 as its display text?

- 12 Switch back to the **Design** tab for *activity_synthesizer.xml* and you will see the Palette panel appear.

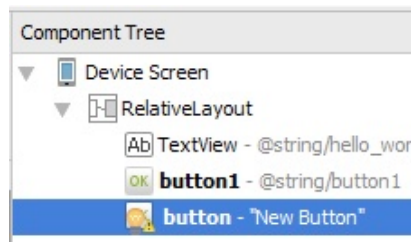
In combination with the Component Tree, you will use this panel to create a second button using the graphical UI design tools that are part of Android Studio. Functionally, you will do the same thing that you did in the previous step. In these steps, you will use the graphical layout tool, whereas in the previous step you typed text to create the button.



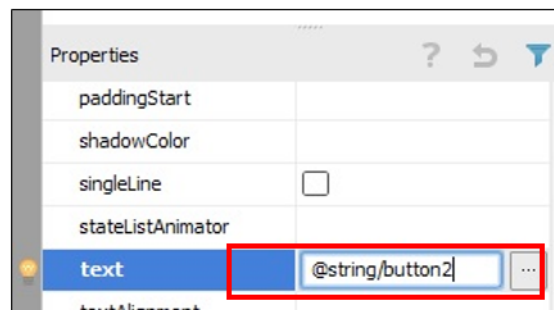
- 13 In the Component Tree, you should see the first button that you just created by programming it in XML.



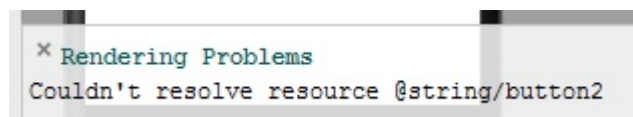
- 14 To create a second button, drag the Button widget from the palette onto the **RelativeLayout** in the Component Tree.



- 15 You will want to change the default label “New Button” to something useful to your app.
- With the new button selected in the Component Tree, scroll down through the Properties panel until you find “text” and change its value to `@string/button2` as shown.



The following rendering error occurs.



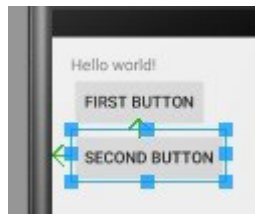
- To fix the problem, you need to create a new string resource, just as you did in Step 9. Name this new resource `button2` and give it a value of `Second Button` in the appropriate XML file. (If you need help, refer to Step 9.)
- You also need to change your second button's id.

The id is used to access the button from within your Java code. Referencing the button's id is the only way to respond to click events. Because your first button was assigned the id of `button1`, the graphical UI designer gave your second button the default id `button`.

- d. Either in the Properties panel or directly in `activity_synthesizer.xml`, change the id to `button2`.

If you have followed the previous steps correctly, your Android device preview shows a garbled mess of buttons and text. You can switch to the Design view to reposition these button and text elements by dragging them around with your mouse.

- 16 Try to arrange them as shown below.



NOTE

You cannot drag components on the Preview pane while in Text view. You must select the **Design** view of the `activity_synthesizer.xml` panel.

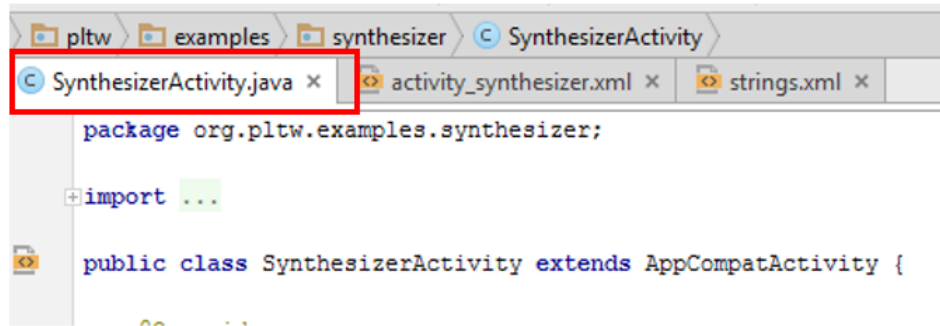
You can learn a lot of handy XML tricks by rearranging your app's components here and then examining the XML that Android Studio generates.

How does the `activity_synthesizer.xml` file change each time you rearrange the components? (Remember to switch between Design and Text views using the tabs at the bottom.)

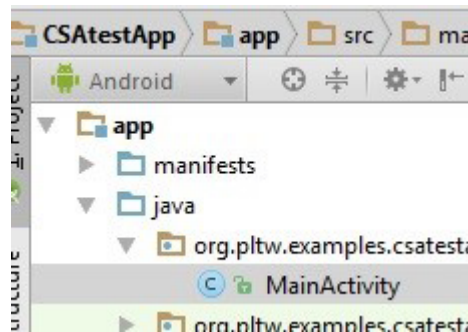
Part III: Java in Android Studio

So far you have explored several of the Android Studio panels associated with layout design and created some components using XML. While XML is an important part of Android device programming, most of the functionality of any Android app is programmed in Java, as you've seen in previous activities.

When you first create a new project in Android Studio, the primary class for that project is displayed by default in your file tabs.



You can find this file, *SynthesizerActivity.java*, and any other Java files, in the Project panel by navigating to **app > java > your.package.name**.



What is your project's package name?

Reminder: Whenever you are asked to add lines of code to an existing file in this course, the lines immediately before and after the new lines will be provided to help you with context. The line numbers are only for reference within the instructions and do not necessarily match the line numbers in your code.

- 17 Add the following lines of code in *SynthesizerActivity.java* on lines 2, 3, and 4 as shown.

```

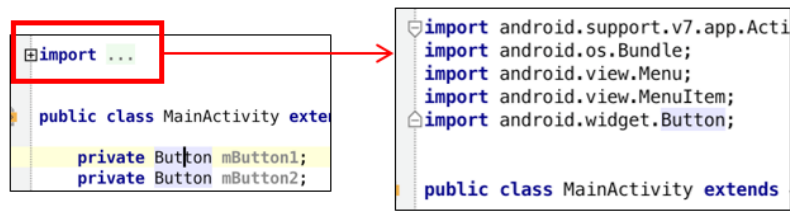
1: public class SynthesizerActivity extends ActionBarActivity {
2:     private static final String TAG =
3: SynthesizerActivity.class.getName();
4:     private Button button1;
5:     private Button button2;
6:
7:     @Override

```

Button is a class in Android's widget library. Android Studio will indicate that you have an error on each of these lines because you have not added the appropriate import statement that tells Android Studio to include the widget library. You can manually enter `import android.widget.Button;` after your other imports, or have Android Studio automatically import the appropriate library by hovering over the error until Android Studio prompts you to hold the **Alt** key and press **Enter**.


NOTE

If you see `import ...` as shown below, click the + to expand the code.



The `private` keyword in lines 2 through 4 helps enforce encapsulation. It is called a "modifier". The `private` modifier can be applied to variables or methods within a class to ensure that other objects will not be able to access them.

Why is encapsulation important?

Lines 2, 3, and 4 are **declaration statements**. Java is a **statically typed** language, unlike *Python*. Any time a programmer wants a new variable, they must explicitly tell the Java compiler what type of variable that will be. The type could be a primitive type, such as `int`, `float`, `char`, or `boolean`, or it could be a reference type. In this case, both `button1` and `button2` are declared as being of type `Button`. Some programmers follow naming conventions, such as making instance variables begin with an `m` prefix; this is a naming convention indicating that these variables are members of the class in which they were declared. You will see this convention used later in this course. You can find out more about Android coding conventions at:  <http://source.android.com/source/code-style.html#follow-field-naming-conventions>



18 Add lines 2–4 as shown.

```
1:     setContentView(R.layout.activity_synthesizer);
2:
3:     button1 = (Button)findViewById(R.id.button1);
4:     button2 = (Button)findViewById(R.id.button2);
5: }
```

Line 2 is left intentionally blank as a way to provide visual space between two different parts of the `onCreate` method. In the previous step, you added two assignment statements, one for each `Button`. These assignments wire the buttons from your layout file to `button1` and `button2`, connecting the View and Controller layers of this app. The table below explains the parts of the expression on the right-hand side of the equal sign.

<code>(Button)</code>	A type cast, like in <i>Python</i> . Takes the result of <code>findViewById(R.id.button1)</code> and ensures that the compiler treats it as an object of type <code>Button</code> .
<code>R</code>	An object that exists in every Android project. It stands for “Resources” and it contains a Java representation of all of the resources in your project. These could be images, sound files, or data in XML files.
<code>id</code>	A member of the <code>R</code> class of this project that contains the <code>int</code> value of the <code>id</code> for any project resource.
<code>button1</code> and <code>button2</code>	The names of the resources whose <code>ids</code> you want to retrieve.
<code>findViewById</code>	A method that takes as its parameter an <code>int</code> representing the <code>id</code> of a widget and returns that widget as an object of type <code>View</code> .

`button1` now stores a reference to an object of type `Button`.

- 19 In a browser, navigate to  <http://developer.android.com/reference/packages.html>.
This will be one of your most important resources as you learn to develop for Android devices.
- 20 In the upper right-hand corner of the page, click the search icon  and search for **android.widget.Button**.
- 21 Click the first link and navigate to the page for that class.
Here you will find some examples of how to use this class, as well as reference information about all the methods available to the class. `Button` itself does not contain very many methods; however, one of its super classes, `View`, does.
- 22 To see all of the `View` class methods that are also available to `Button`, find the Inherited Methods section and click the arrow next to `View`.



In previous steps you created two `Button` variables (Step 18) and assigned values to them (Step 19). To be specific, you assigned the variable `button1` to refer to the `button1` object in your app and the variable `button2` to refer to the `button2` object. Objects `button1` and `button2` are now objects in the `Button` class, and they inherit methods from the `android.view.View` class. `button1` and `button2` automatically have functionality based on their class type. For example, you could call the method `button1.onTouchEvent` since that method is defined in the `View` class.

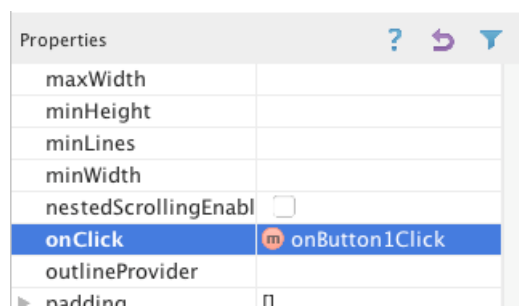
- 23 Record the names of two methods of the `View` class.

Android code relies heavily on the **listener pattern**. If you've taken Computer Science Principles (CSP), you should remember listeners from Scratch and App Inventor like the two shown below. Each "listens" for an event and executes some code when that event is detected.



The Android operating system will provide to your programs most of the events that you will use listeners for in this course.

- 24 Add code in `SynthesizerActivity.java` to set up listeners for when a user touches the buttons in your app. Create a public method of return type `void` named `onButton1Click` with a single parameter of type `View` that contains only this in its body:
`Log.i(TAG, "Button 1 Clicked");`
- 25 Create a similar method for `button2`.
- 26 To wire these two methods to their corresponding layout objects, return to the Design view of `activity_synthesizer.xml` and select **button1**. Scroll through the properties until you find **onClick**.
- 27 Change the value to `onButton1Click`, the name of the method you created in Step 25. This wires the `onClick` functionality of the layout element `button1` to the method you created in `SynthesizerActivity`.



- 28 Repeat this process for `button2`.
- 29 Run your app and record what appears in the logcat panel when you click **First Button**.

Part IV: Adding Sound

In this part, you will program buttons to play sounds. You need to create a directory in your file structure to house the sound files for this project.

- 30 Within the Project view, right-click the **res** directory and select **New > Android resource directory**. From the **Resource type** menu, select **raw** and click **OK**.

Copy the sound files with .ogg extensions from *1.2.4Sounds* provided by your teacher to the newly created **raw** directory within your project folder. Hint: You can search for the **raw** directory starting in your Android Studio project directory to find its location.

- 31 Add two declaration statements to `SynthesizerActivity`—both `private` variables, both of type `MediaPlayer`, one identified by `mpE` and the other by `mpF`.
- 32 Add the following immediately after your `button2` initialization statement:

```
1:         button2 = (Button)findViewById(R.id.button2);
2:         mpE = MediaPlayer.create(this, R.raw.scalee);
3:         mpF = MediaPlayer.create(this, R.raw.scalef);
4:     }
```

- 33 Within the `onButton1Click` method, add line 3 as shown.

```
1:         public void onButton1Click(View v) {
2:             Log.e(TAG, "Button 1 clicked");
3:             mpE.start();
4:         }
```

This call to the `start` method of this instance of `MediaPlayer` should have an audible effect!

- 34 Add the corresponding code for `mpF` in the appropriate location.
- 35 Test your app as directed by your instructor.

What role do `mpE` and `mpF` serve in causing music to play in your app?

None of the identifiers for the buttons have descriptive names, so you should change them. The **refactoring** feature of Android Studio automatically changes the name of that identifier everywhere that it appears in your project. You won't have to hunt down each individual occurrence of that identifier and change it manually.

- 36 Follow these steps to refactor your project:
- Within `activity_synthesizer.xml`, right-click the text that reads `@string/button1` (your text may read "First Button" instead of `@string/button1`) and select **Refactor > Rename...**

- b. In the prompt that appears, type `eButton` to denote that this is the button that plays the low E note and select **Refactor**.
- c. Repeat this process, changing the text that reads `@+id/button1` to `@+id/eButton`.
- d. **Refactor** and **Rename** `button1` in `SynthesizerActivity.java` to `mEButton`. To complete this renaming, press Enter.
- e. Change the display text for your buttons manually in `strings.xml`.
- f. Finally, change the strings in your logging messages in `SynthesizerActivity.java` to match the other changes to this project.

With this refactoring tool, all references to an item are changed throughout your entire project.

- 37 Refactor your second button as you did in the previous step. This button should indicate that it plays the note F.

In Step 34, you may have noticed that a button will not play a sound a second time until it has finished playing the entire sound clip.

- 38 To improve the functionality of this app, modify your code by adding line 2 as shown below.

```
1:      public void onButton1Click(View v) {
2:          mpE.seekTo(0);
3:          Log.e("SynthesizerActivity", "Button 1 clicked");
4:          mpE.start();
5:      }
```

- a. Add a similar line for `mFButton`. Use the official Android Developer documentation to discover what this call to `MediaPlayer`'s `seek` method is doing and record what you find.
- a. Create new `Buttons` and `MediaPlayers` for the other sound files in your `raw` directory. You may find it helpful to reference previous steps in this activity. Test your app.

NOTE

If you tested this app on a phone instead of a tablet, you'd notice that the layout forces some of your buttons off the viewable area of your app.

- b. Open `activity_synthesizer.xml` and use the UI design tool to create a solution to this problem. Share solutions with the class as directed by your instructor.

In the next activity, you will add more functionality to your Synthesizer app.

CONCLUSION

1. In your own words, summarize how the View layer of an app is connected to the Controller layer.

PROBLEM 1.2.5

Synthesizer

INTRODUCTION

This problem will pose several challenges for you to complete within the context of a synthesizer app. The idea is that the synthesizer will play different musical notes depending on the challenge you are completing. Your synthesizer will have buttons for you to use to produce “music”.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator

Procedure

Part I: The Challenges

You will be adding functionality to your Synthesizer app in the form of challenges. The challenges are designed to help reinforce your skill with some of the most common Java programming constructs. You may not complete all of the challenges in the time allotted by your instructor. If this is the case, you are encouraged to return to them whenever you finish other activities or projects early as well as outside of class.

As you attempt these challenges, you will find it necessary to cause your app to delay for a time in between playing sounds, so you will set a time delay in your code.

- 1 Open your Synthesizer project in Android Studio.
- 2 In SynthesizerActivity, add line 3 as shown.

```
1: public class SynthesizerActivity extends AppCompatActivity {  
2:  
3:     private final int WHOLE_NOTE = 1000;
```

WHOLE_NOTE is a **constant** value, one that never changes after it is created. Here, WHOLE_NOTE will be used to determine how long to wait between playing one sound and the next. You will call the method defined in lines 3–9 below whenever you require a pause between playing sounds.

- 3 Add it to your code just before the definition of `onButton1Click(View v)`.

```
1:  ...
2:  }
3:
4:  private void delayPlaying(int delay) {
5:      try {
6:          Thread.sleep(delay);
7:      } catch (InterruptedException e) {
8:          Log.e("SynthesizerActivity", "Audio playback
          interrupted");
9:      }
10: }
11:
12: public void onButton1Click(View v){
13:  ...
```

Details about the `try` and `catch` statements will be covered later in the course. For now, know that you may need to replace their code with your own.

Below is an example of how a `Button` identified by `mChallenge1` could be set up to play an E, delay for a whole note, and then play an F.

- 4 As you attempt the challenges, you will replace the code between the `try` and `catch` statements (lines 5–7 below) with your own.

```
1: mChallenge1.setOnClickListener(new View.OnClickListener() {
2:     @Override
3:     public void onClick(View v) {
4:         Log.e("SynthesizerActivity", "Challenge 0 Button
         clicked");
5:         mpE.start();
6:         delayPlaying(WHOLE_NOTE);
7:         mpF.start();
8:     }
9: });
```

- ❑ CHALLENGE 1: Create a button that plays a scale from low E pitch to high E pitch with a half-note delay in between each note. Use the following sequence of notes.

E, F Sharp, G Sharp, A, B, C Sharp, D Sharp, E

Hint: A half note can be expressed as `WHOLE_NOTE/2`.

- ❑ CHALLENGE 2: In this challenge you will add functionality so that a user can select a note and the number of times to play that note. When the user presses their Challenge 2 button, the app should play that note, that number of times. To set up for this challenge, add two `NumberPickers` to your app. You can do this by declaring and initializing them in a similar manner to how you declared and initialized your `Buttons`. *Hint: you will need to use `NumberPicker`'s `setMinValue`, `setMaxValue`, and `getValue` methods, as well as `MediaPlayer`'s `seekTo`, and `start` methods to complete this challenge.*
- ❑ CHALLENGE 3: Store each of your `MediaPlayers` in an array and solve Challenge 1 again by iterating through each element in the array.
- ❑ CHALLENGE 4: Solve Challenge 2 again by accessing a new array like the array you created in Challenge 3.
- ❑ CHALLENGE 5: The following sequence of notes is the first line of a version of “Twinkle, Twinkle, Little Star”. Create a `Button` that plays this tune when pressed.

A, A, High E, High E, High F Sharp, High F Sharp, High E, D, D, C Sharp, C Sharp, B, B, A

- ❑ CHALLENGE 6: Complete Challenge 5 without referencing any object of type `MediaPlayer` within the call to `setOnClickListener`. *Hint: you may create a helper method that references one or more `MediaPlayers`.*
- ❑ CHALLENGE 7: Complete Challenge 6 using an array to retrieve each note.
- ❑ CHALLENGE 8: The timing of the notes playing is not quite right. Fix it.
- ❑ CHALLENGE 9: Following are the notes that make up the second line of this version of *Twinkle, Twinkle, Little Star*. The second line repeats itself once. The third line is the same as the first line. Create a `Button` that plays the entire song when clicked.

High E, High E, D, D, C Sharp, C Sharp, B

- ❑ CHALLENGE 10: Complete Challenge 9 and allow the user of the app to select how many times the second line repeats.
- ❑ CHALLENGE 11: Create a `CheckBox` that toggles whether or not the second line will play.
- ❑ CHALLENGE 12: Create a `Button` that plays a version of your favorite song.