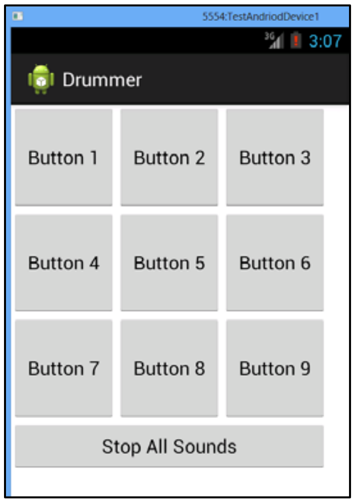
Mobile App Design Project
Drum Machine App



Description:

This App will take 9 Music samples and have buttons for each music sample on the User Interface. When the user touches a button, the corresponding music sample will play. We will use XML to design the user interface. The user interface will have nine Button objects in a table layout. The code for this App will consist of one Class called <name> which will hold the data and functions for the User Interactions with the Screen. We will also use arrays to store the Media Resources and Sound Objects.

User Interface Objects and instance names.

1. Button – button01
2. Button – button02
3. Button – button03
4. Button – button04
5. Button – button05
6. Button – button06
7. Button – button07
8. Button – button08
9. Button – button09
10. Button – stopSounds

User Interactions:

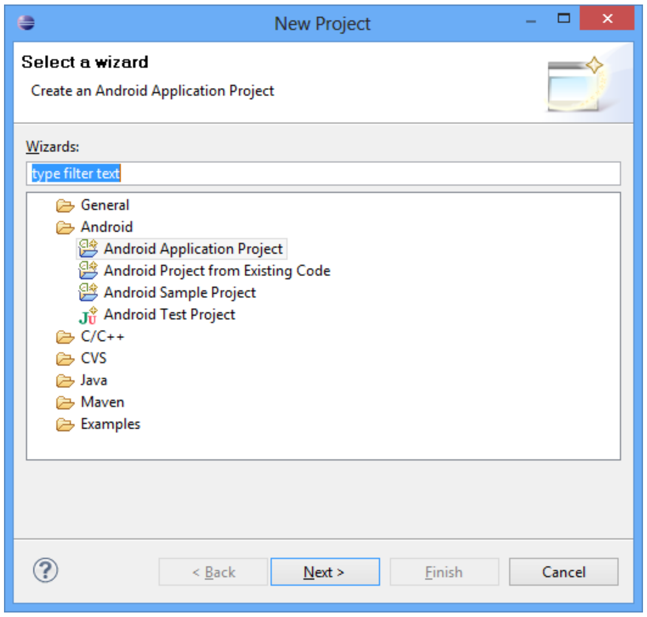
1. User touches a button and the corresponding sound plays.

App Resources:

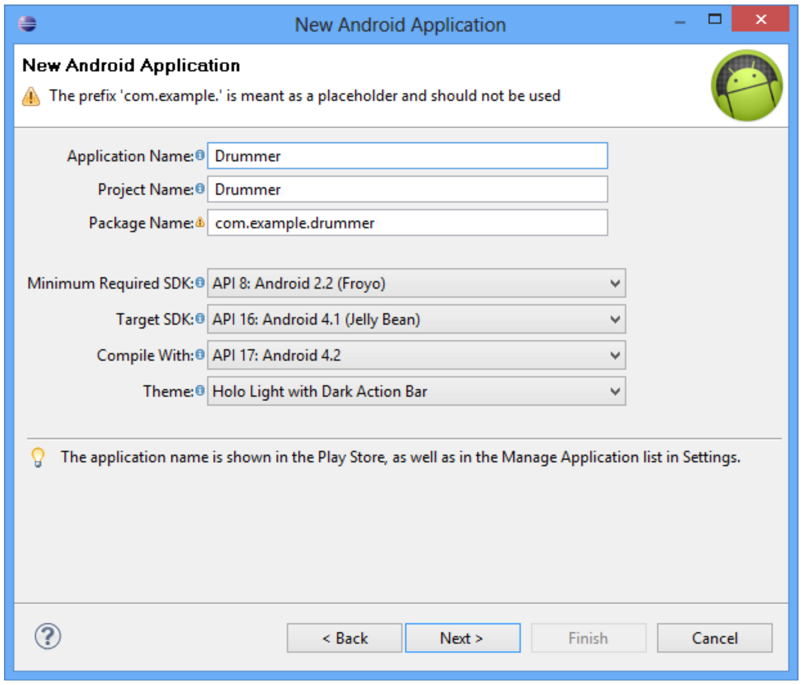
This App will use.wav files of various sounds. Apps and Applications store resources (Data, Sound, Images, Movies, and Other Data Objects) in folders within a directory called ‘res’. The ‘res’ stands for resources and in this project we will practice adding data to the ‘res’ folder and then calling this data using the ‘R.raw’ references.

Phase 1: Create the App Project

1. Start Eclipse and select New Project -> “Android Application Project”



2. Fill out the fields with the following:
    a. Application Name: Drummer
    b. Project Name: Drummer
    c. Package Name: com.example.drummer



New Android Application

The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: Drummer

Project Name: Drummer

Package Name: com.example.drummer

Minimum Required SDK: API 8: Android 2.2 (Froyo)

Target SDK: API 16: Android 4.1 (Jelly Bean)

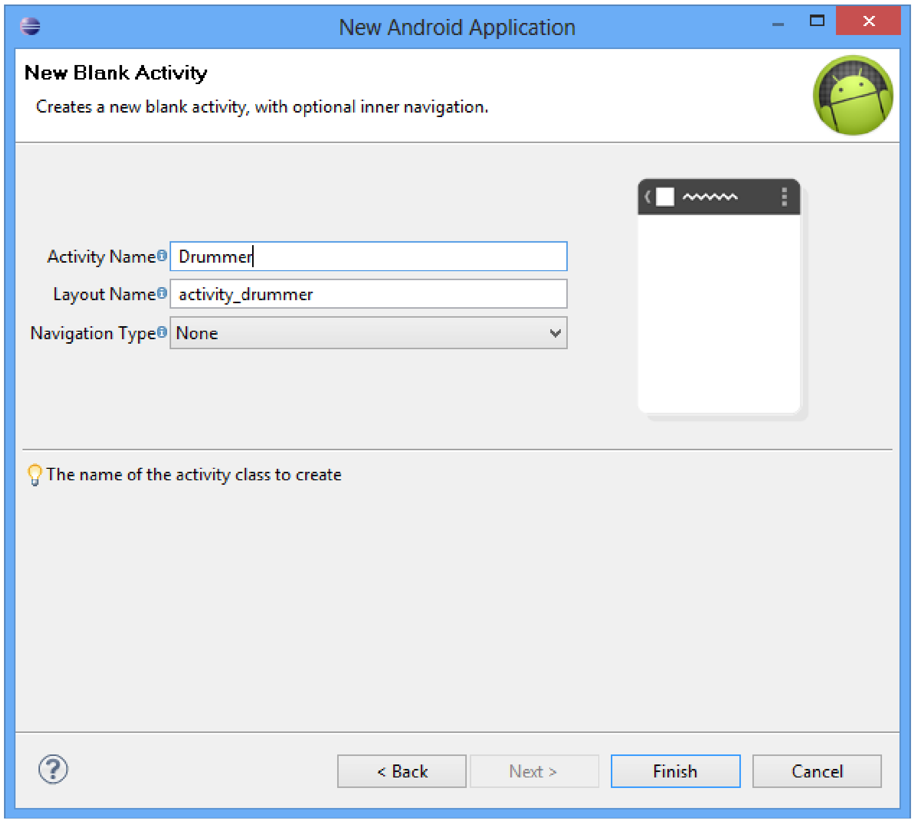
Compile With: API 17: Android 4.2

Theme: Holo Light with Dark Action Bar

The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

3. Click Next

4. Click Next at the Configure Project Screen
5. Click Next at the Configure Launcher Icon Screen.
6. Click Next at the Create Activity Screen.
7. Fill out the following fields on the New Blank Activity Screen
    a. Activity Name: Drummer
    b. Layout Name: activity_drummer
    c. Navigation Type: None



New Blank Activity

Creates a new blank activity, with optional inner navigation.

Activity Name: Drummer
Layout Name: activity_drummer
Navigation Type: None

The name of the activity class to create

8. Click "Finish"

Phase 2: User Interface and XML Design:

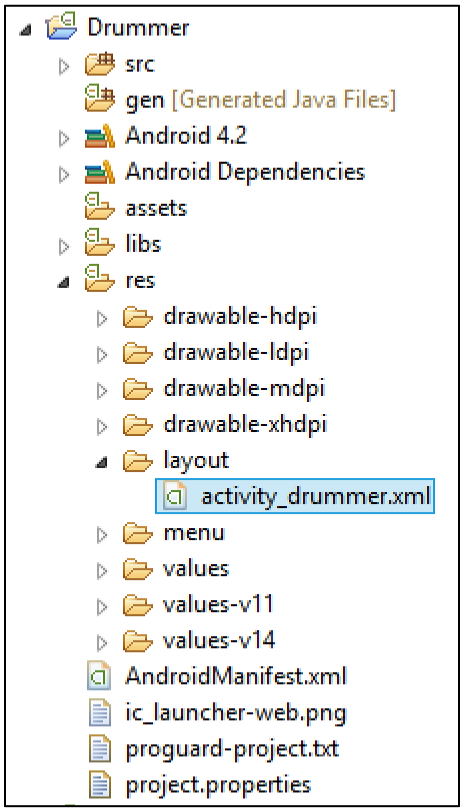
In Android Apps, we will 'create' objects three times:

1. In the XML User Interface View or XML Code.
2. In the Fields we define in the Activity Classes
3. In the Constructor or Functions within the Code (This is where we 'bind' them to the XML user interface).

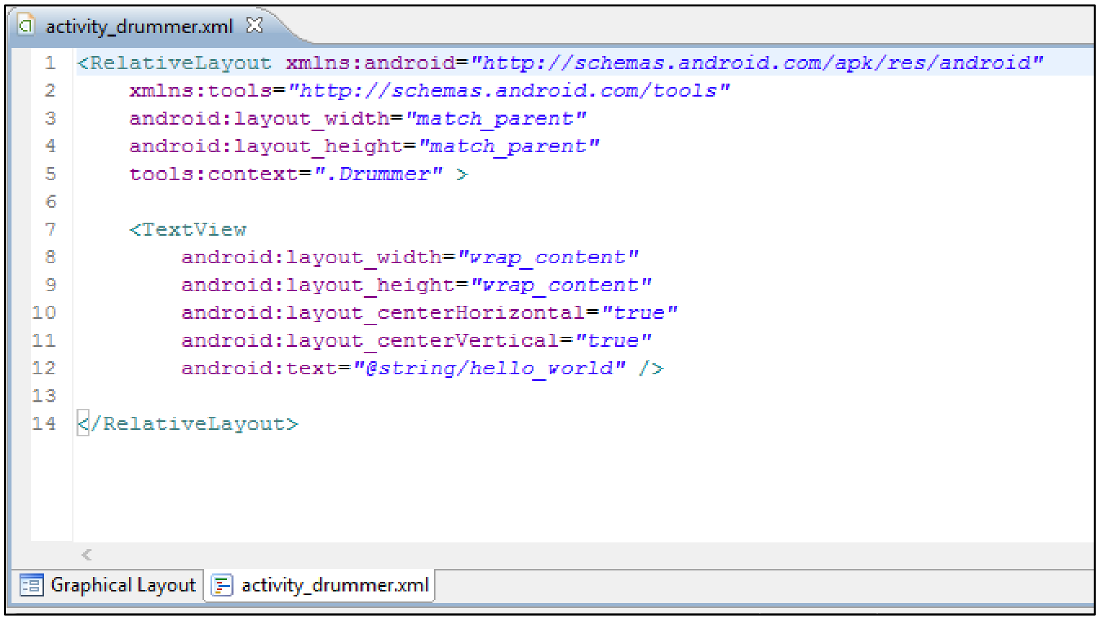
Any errors in these steps will cause the App to crash. (And sometimes you will not see in the stack trace the specific error message describing the nature of the crash.) If an App does not run, double check that you have done the "Big Three" definitions. (XML, Fields, Functions/Binding).

Process:

1. Go to the res/layout/activity_drummer.xml file:

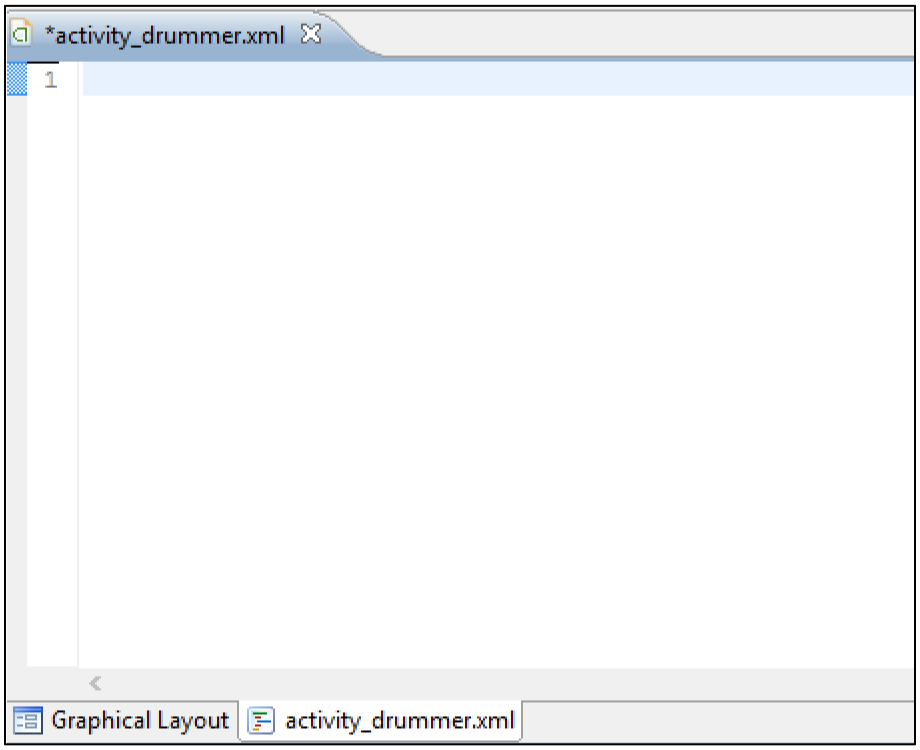


2. Click the activity_drummer.xml tab to view the xml code.

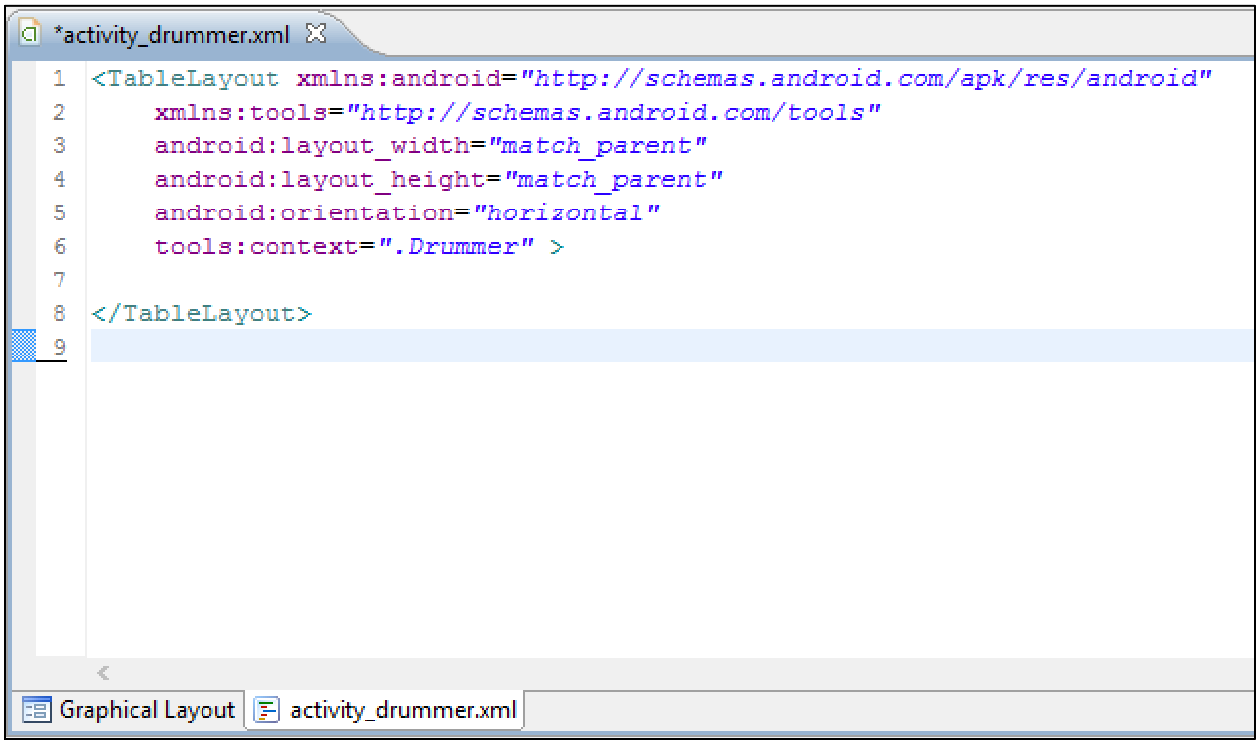


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Drummer" >

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

</RelativeLayout>
```

3. Delete all the existing xml code:

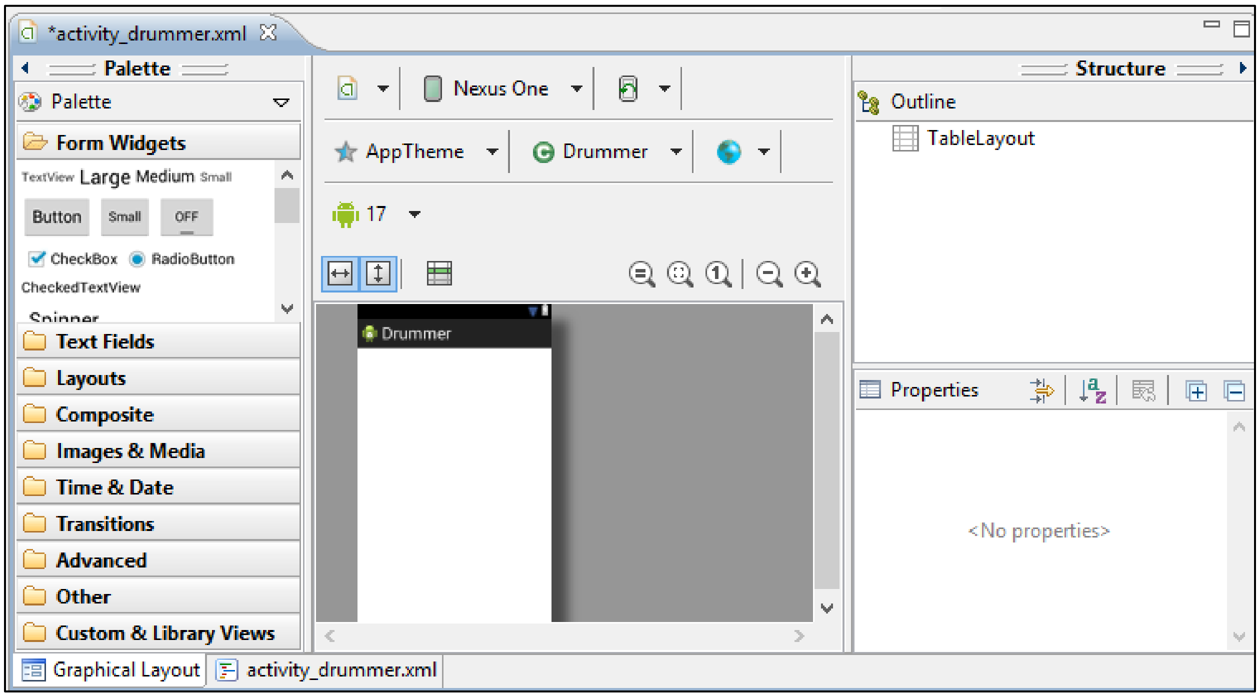


4. We will now begin to build the Table Layout. Type the following into the first 8 lines

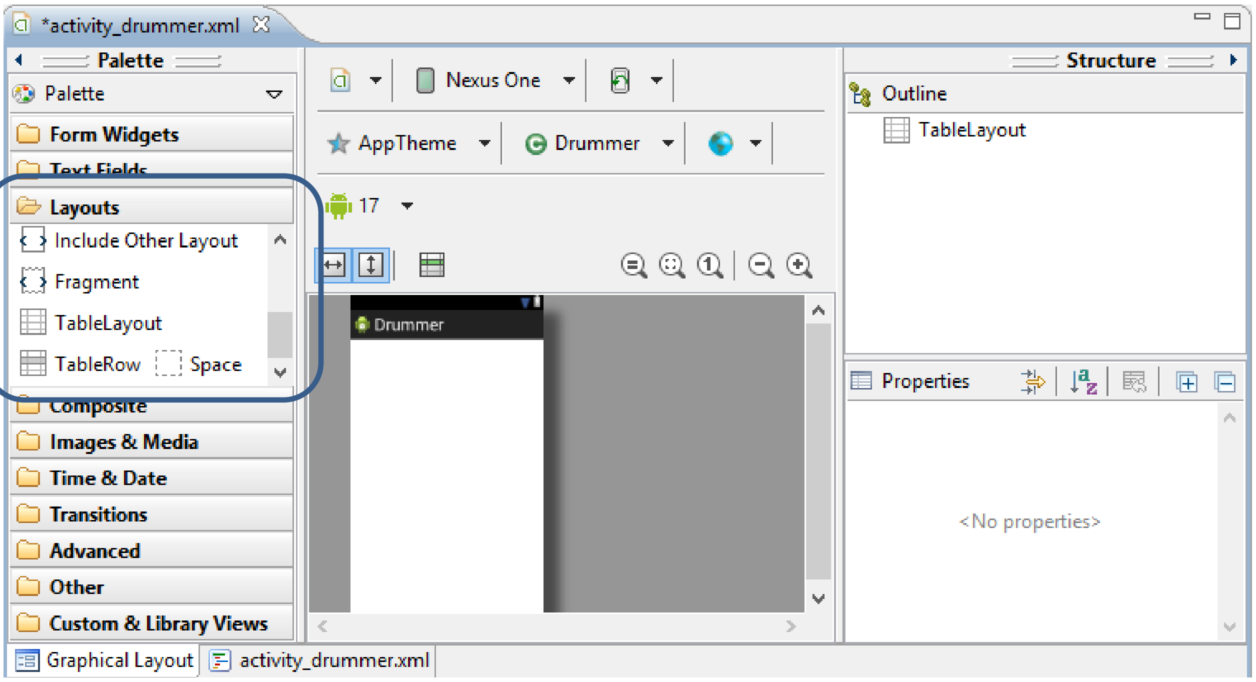

*activity_drummer.xml

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".Drummer" >

</TableLayout>
```

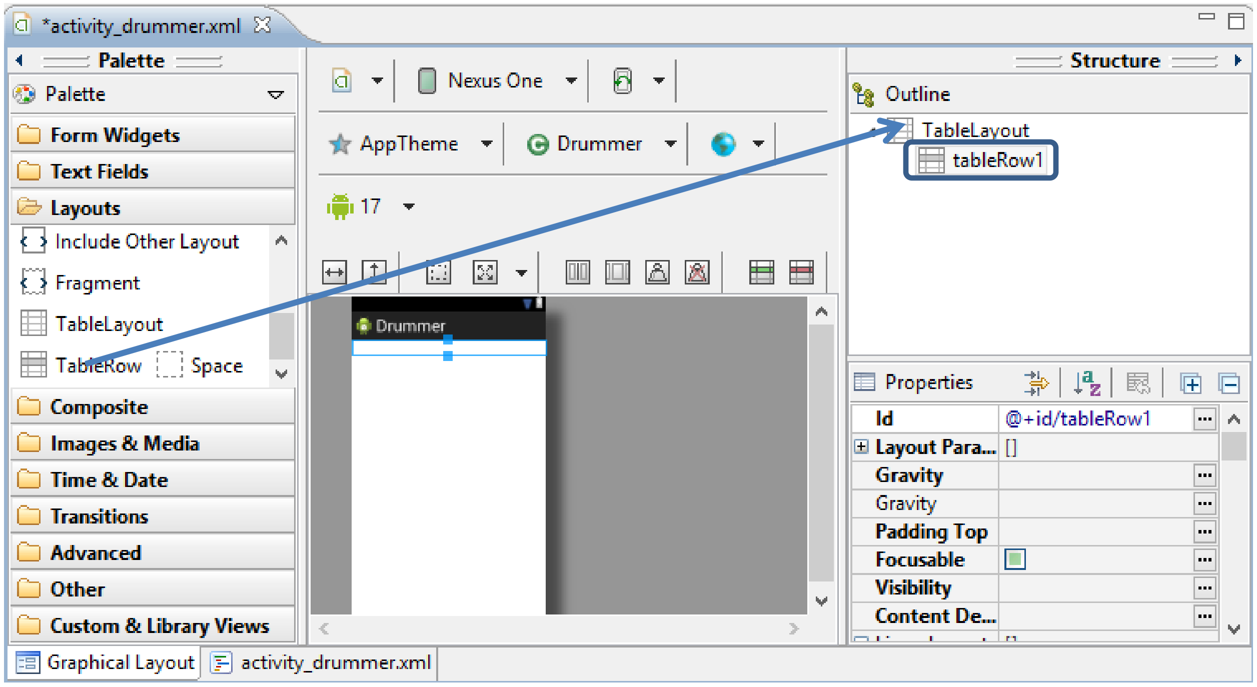
5. Switch to Graphical Layout in Eclipse by clicking on the Graphical Layout tab.



6. Click on the Layouts Area of the Palette on the Left hand side.

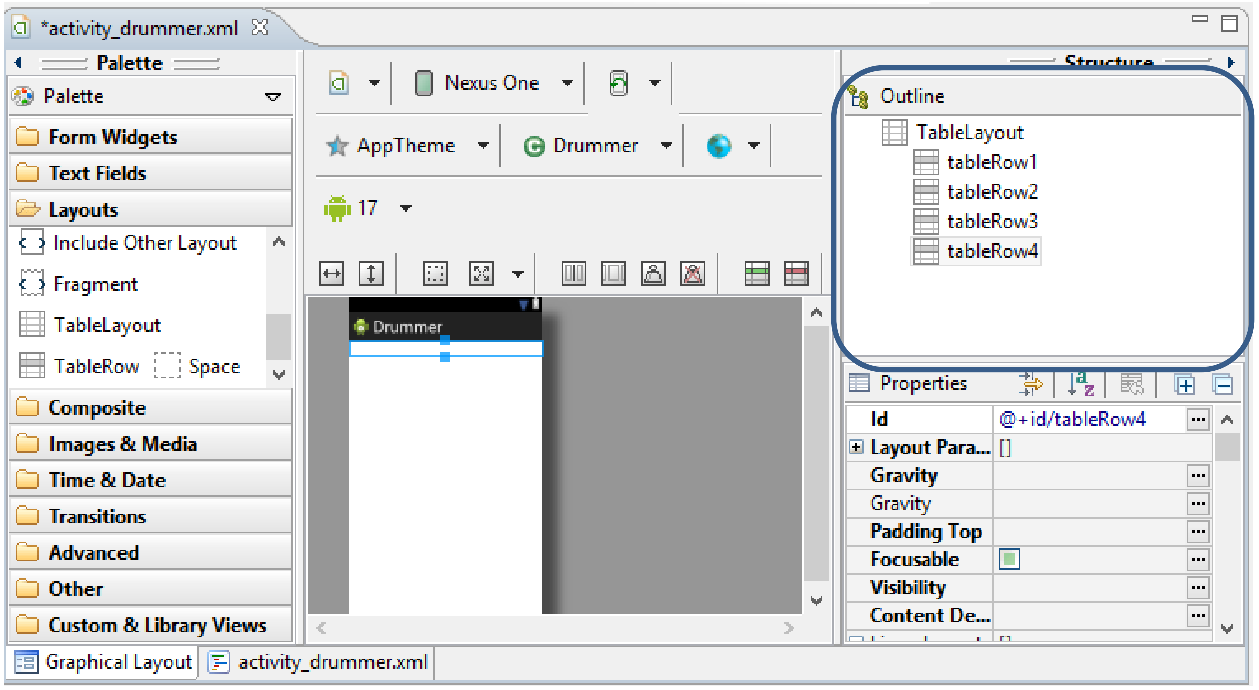


7. Drag a TableRow object and place it on top of the TableLayout Icon on the Outline Section. A tableRow1 instance should appear in the Outline underneath the TableLayout.



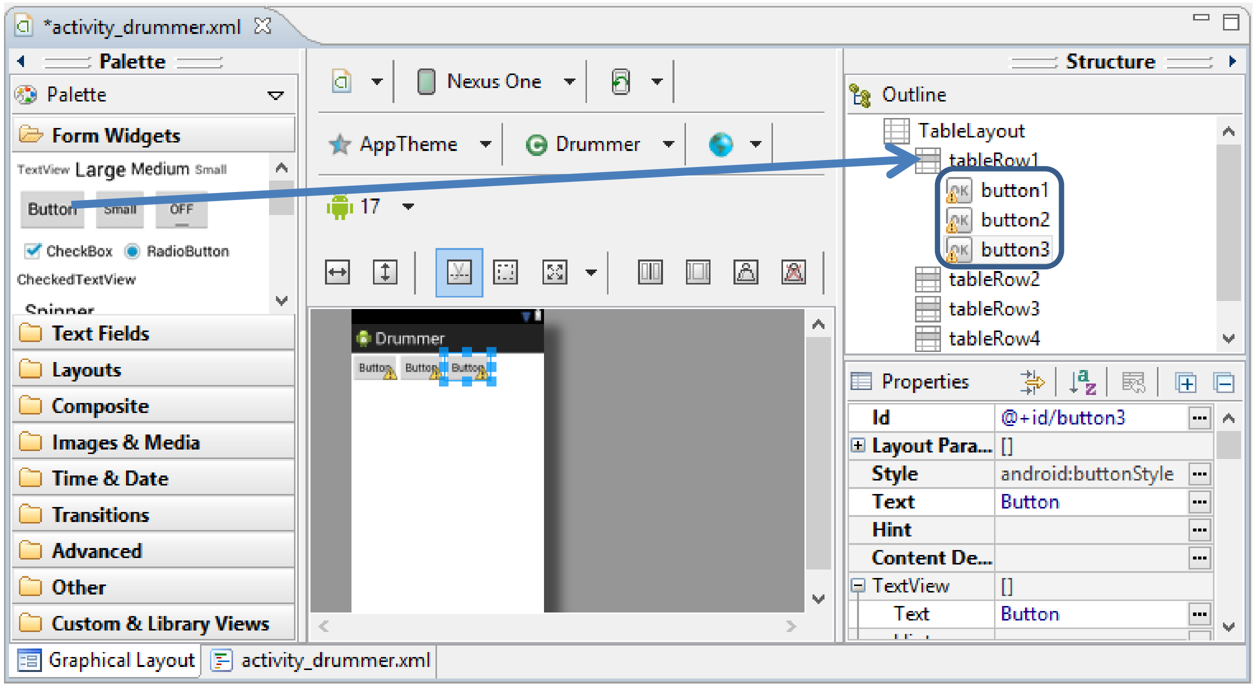
Id	@+id/tableRow1
Layout Para...	[]
Gravity	
Gravity	
Padding Top	
Focusable	☐
Visibility	
Content De...	

8. Drag three more TableRow Objects to the TableLayout in the Outline.



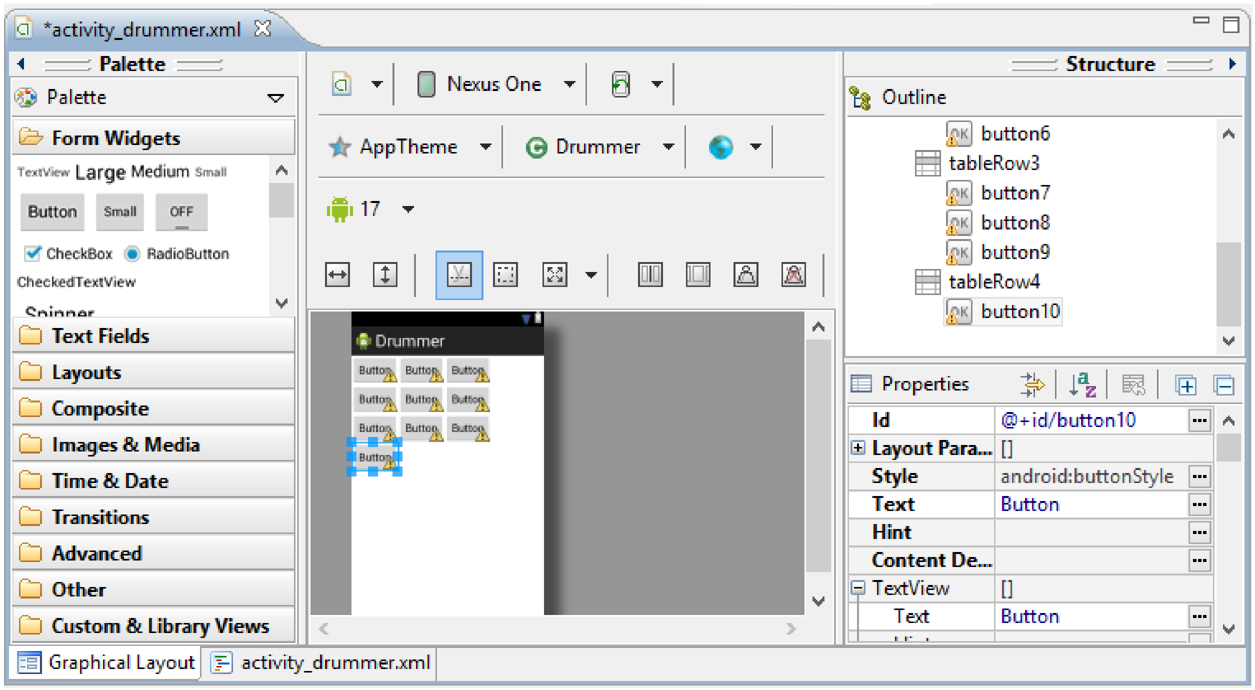
Id	@+id/tableRow4
Layout Para...	[]
Gravity	
Gravity	
Padding Top	
Focusable	
Visibility	
Content De...	

9. We now need to add the Button Objects. Each Table Row will get three Button Objects. Click on form Widgets on the Palette. Drag three Buttons to the tableRow1. Note the names of the Buttons (button1, button2, button3)



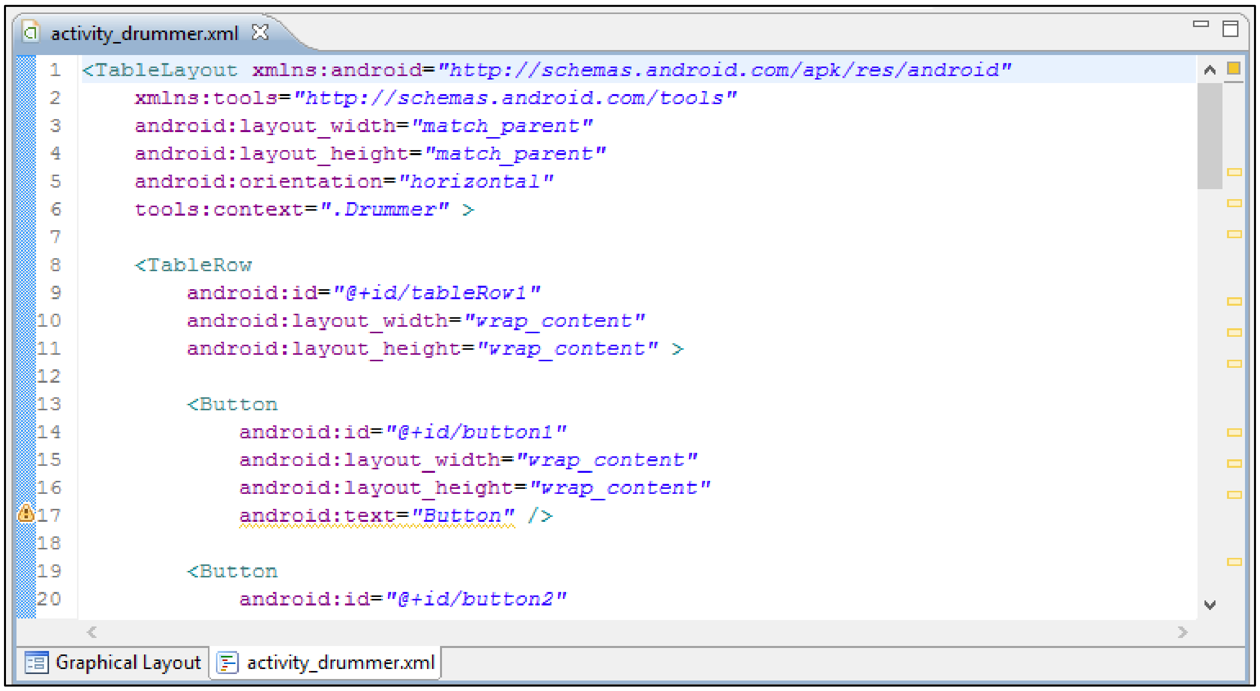
Id	@+id/button3
Layout Para...	[]
Style	android:buttonStyle
Text	Button
Hint	
Content De...	
TextView	[]
Text	Button

10. Place three Buttons in tableRow2, three Buttons in tableRow3, and one Button in tableRow4:



Id	@+id/button10
Layout Para...	[]
Style	android:buttonStyle
Text	Button
Hint	
Content De...	
TextView	[]
Text	Button

11. We have finished the framework for the User Interface. We will now go back to the XML code and refine the properties. Click on the activity_drummer.xml tab. Notice that the XML has automatically changed to reflect the new objects.



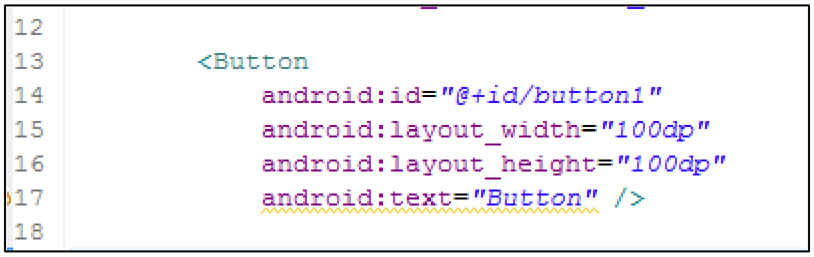
```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".Drummer" >

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />

        <Button
            android:id="@+id/button2"
```

12. The standard properties are:

    a. `android:id` -> This is the name of the object within the Resources (R.) system.
    b. `android:layout_width` -> Defines the width of the object
    c. `android:layout_height` -> Defines the height of the object
        i. wrap_content will match what is typed or contained within the object
        ii. match_parent will math what container holds the object
    d. android:text -> The printed text that will show to the user

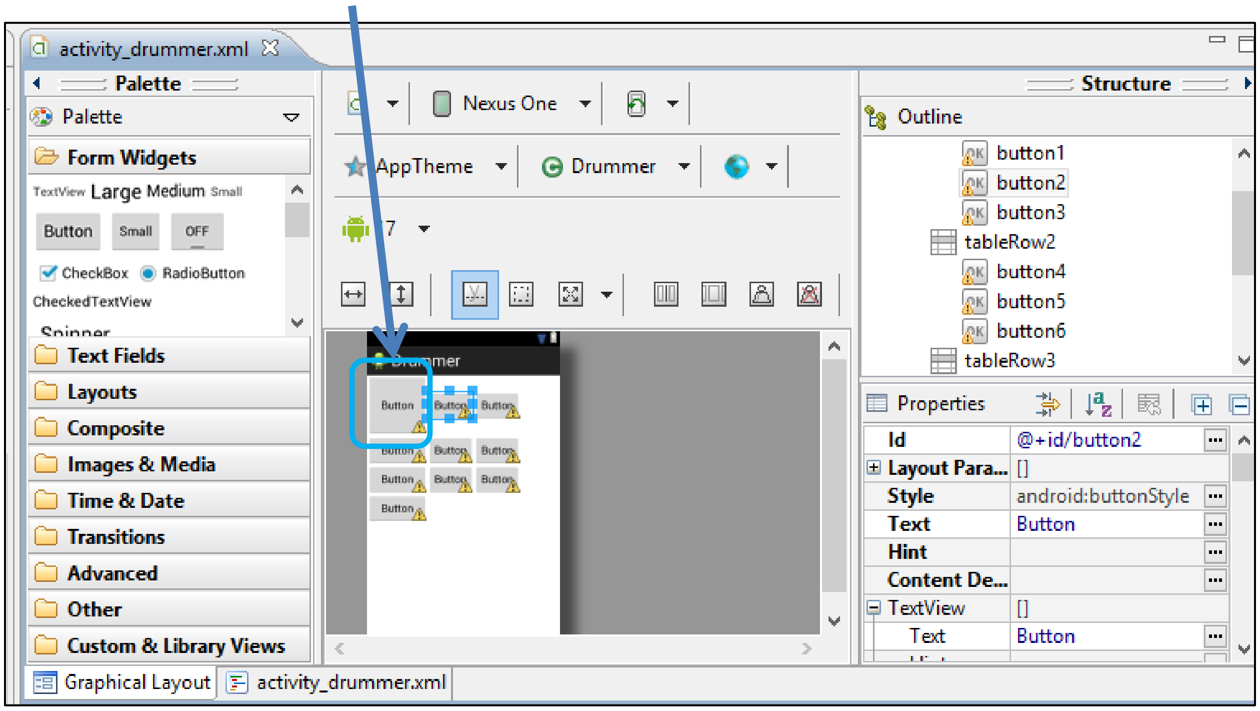
    There are many other properties that can be set within an Android object in XML. For this App, we will change the height and the width of the Buttons to 100dp (100 pixels).

13. Go to line 15 and 16 and make these changes to the button1 objects width and height:



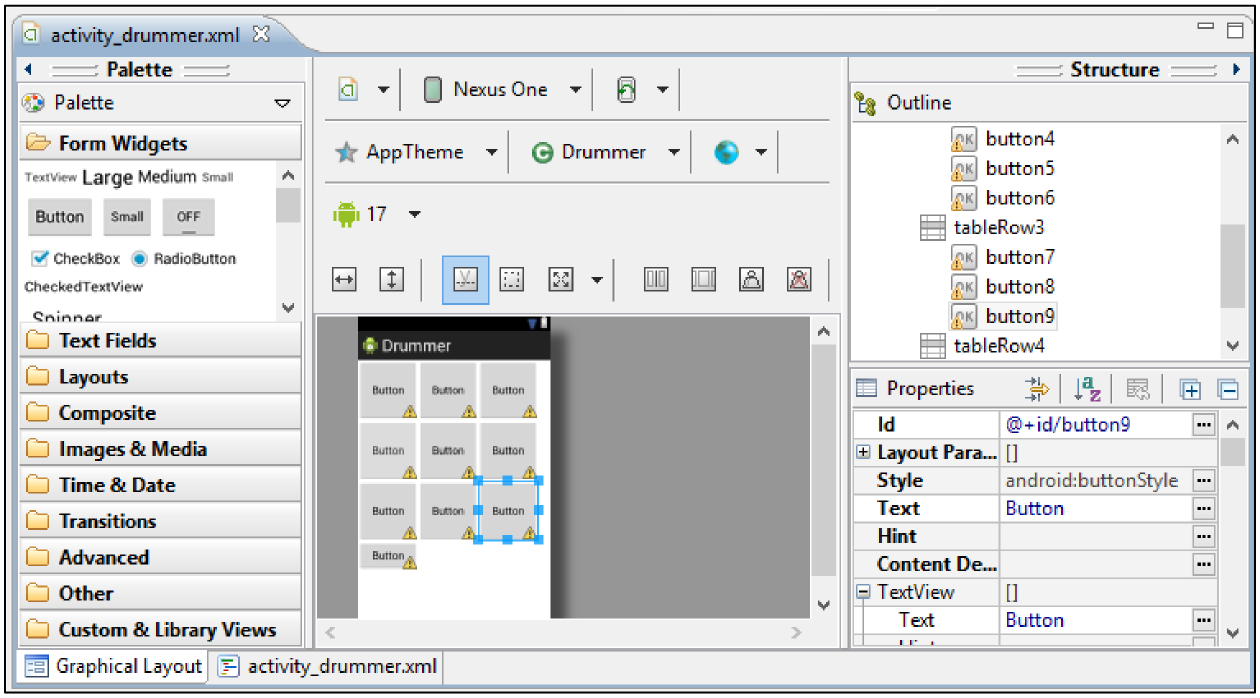
```
12
13          <Button
14              android:id="@+id/button1"
15              android:layout_width="100dp"
16              android:layout_height="100dp"
17              android:text="Button" />
18
```

14. Click on the Graphical Layout Tab and note how the button1 changed size:



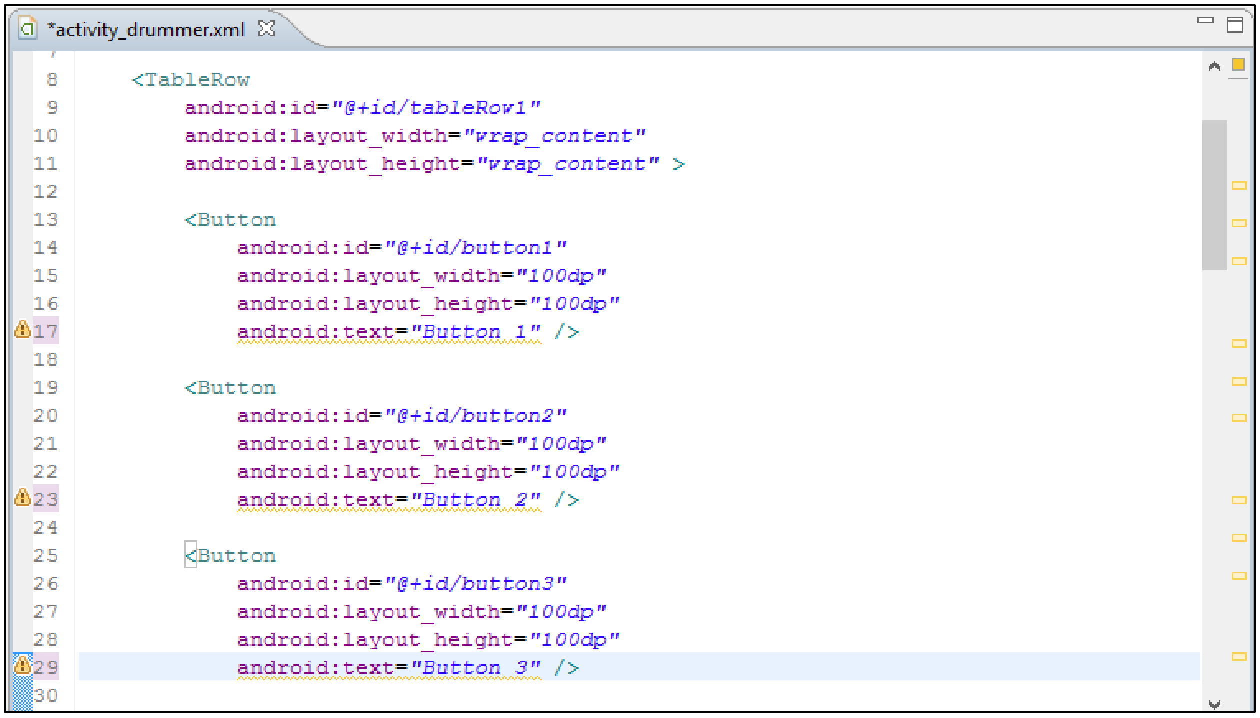
Id	@+id/button2
Layout Para...	[]
Style	android:buttonStyle
Text	Button
Hint	
Content De...	
TextView	[]
Text	Button

15. Go back to the XML tab and edit the code for buttons 1 through 9 so the width and height are set to "100dp" (Copy and Paste are very useful with XML)



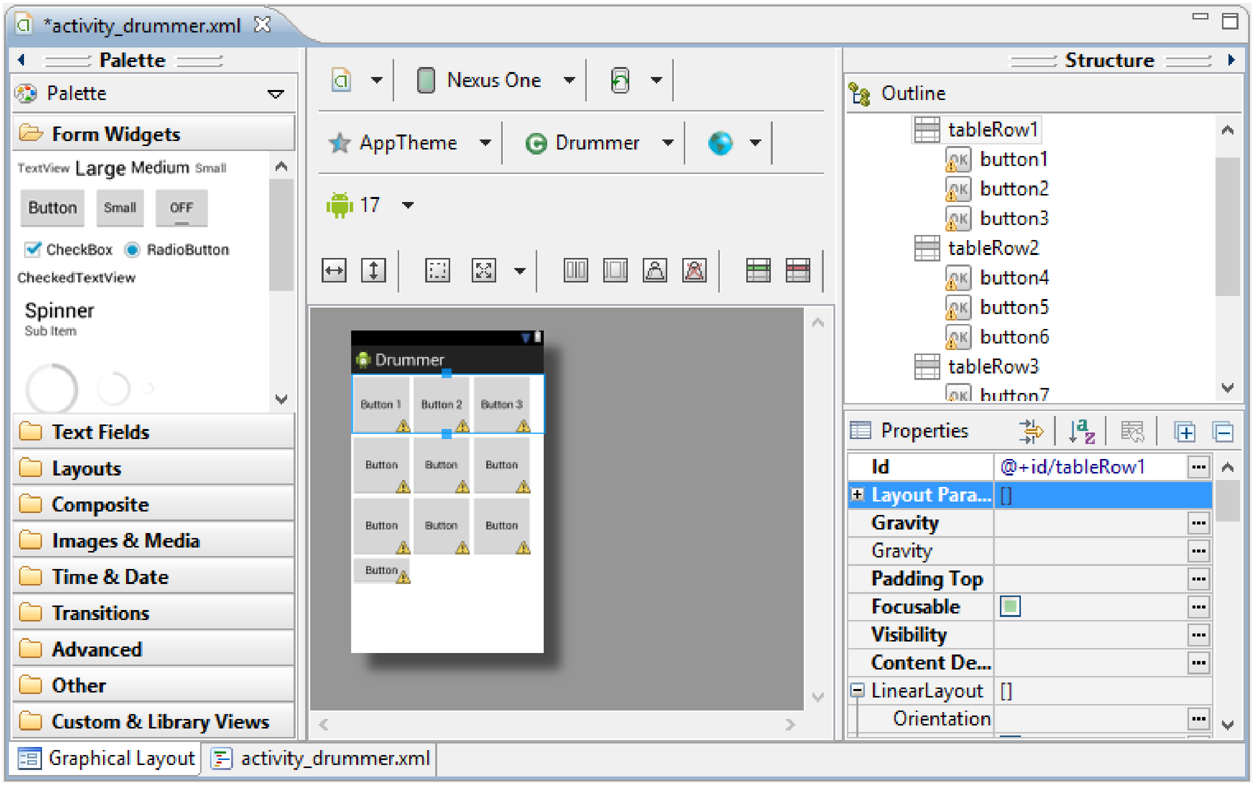
Id	@+id/button9
Layout Para...	[]
Style	android:buttonStyle
Text	Button
Hint	
Content De...	
TextView	[]
Text	Button

16. In the XML code, change the android:text="Button" to reflect the name of the Button. ("Button 1", "Button 2", "Button 3") (Lines 17, 23, and 29).



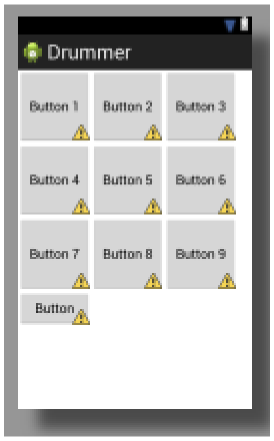
```
 8      <TableRow
 9          android:id="@+id/tableRow1"
10          android:layout_width="wrap_content"
11          android:layout_height="wrap_content" >
12
13          <Button
14              android:id="@+id/button1"
15              android:layout_width="100dp"
16              android:layout_height="100dp"
17              android:text="Button 1" />
18
19          <Button
20              android:id="@+id/button2"
21              android:layout_width="100dp"
22              android:layout_height="100dp"
23              android:text="Button 2" />
24
25          <Button
26              android:id="@+id/button3"
27              android:layout_width="100dp"
28              android:layout_height="100dp"
29              android:text="Button 3" />
30
```

Note the yellow ! point warnings. Usually strings in XML and Android are coded as @string references in a separate file. (These eases tasks such as translation or locality as changes in the @string XML file will reflect across the objects in the App.) For the purposes of this lesson, we will hard code the strings into XML.

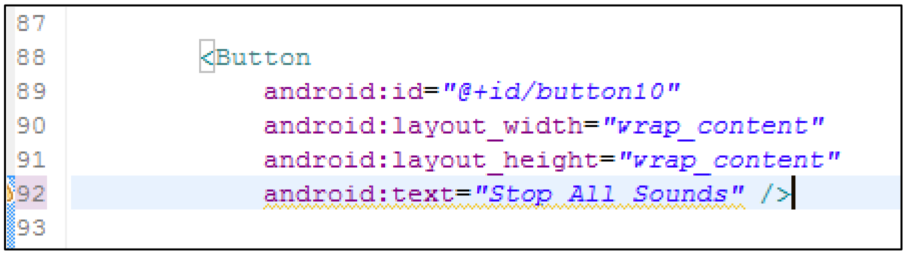


Id	@+id/tableRow1
Layout Para...	[]
Gravity	
Gravity	
Padding Top	
Focusable	☐
Visibility	
Content De...	
LinearLayout	[]
Orientation	

17. Change Buttons 4 through 9 android:text fields to reflect their button name:

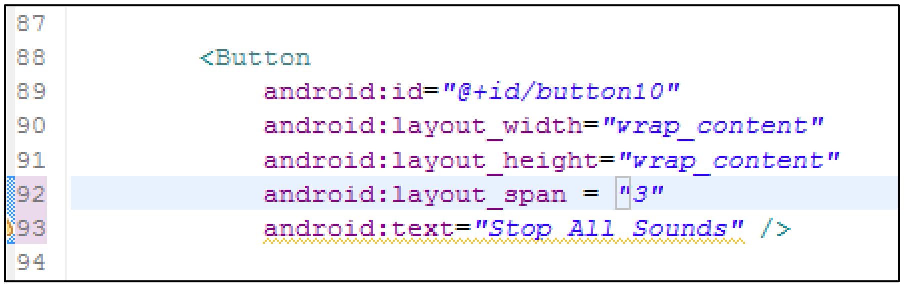


18. Change the android:text property in button10 to the string “Stop All Sounds” (Line 92).

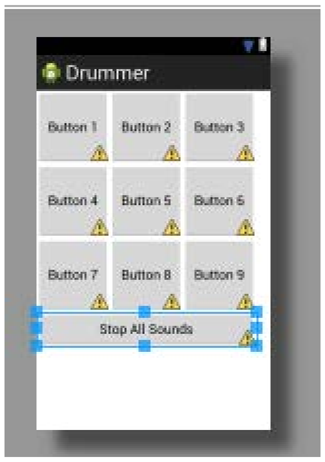


```
87
88          <Button
89              android:id="@+id/button10"
90              android:layout_width="wrap_content"
91              android:layout_height="wrap_content"
92              android:text="Stop All Sounds" />
93
```

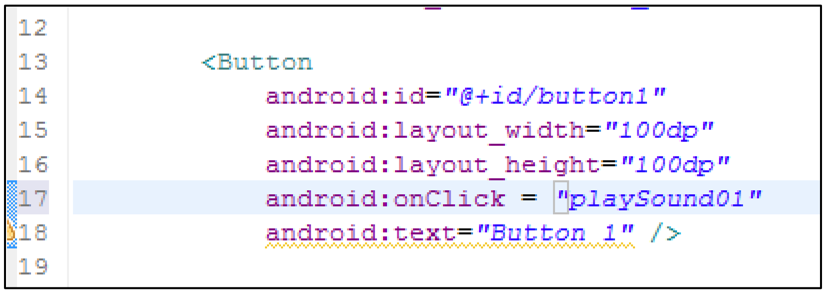
19. Add a line of code `android:layout_span = “3”` after line 91 to button10’s properties to have the button span three columns.



```
87
88          <Button
89              android:id="@+id/button10"
90              android:layout_width="wrap_content"
91              android:layout_height="wrap_content"
92              android:layout_span = "3"
93              android:text="Stop All Sounds" />
94
```



20. To bind a button to a specific function, we will use the android:onClick property. After line 16 in button1, add the following code `android:onClick = “playSound01”`



```
        <Button
            android:id="@+id/button1"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:onClick = "playSound01"
            android:text="Button 1" />
```

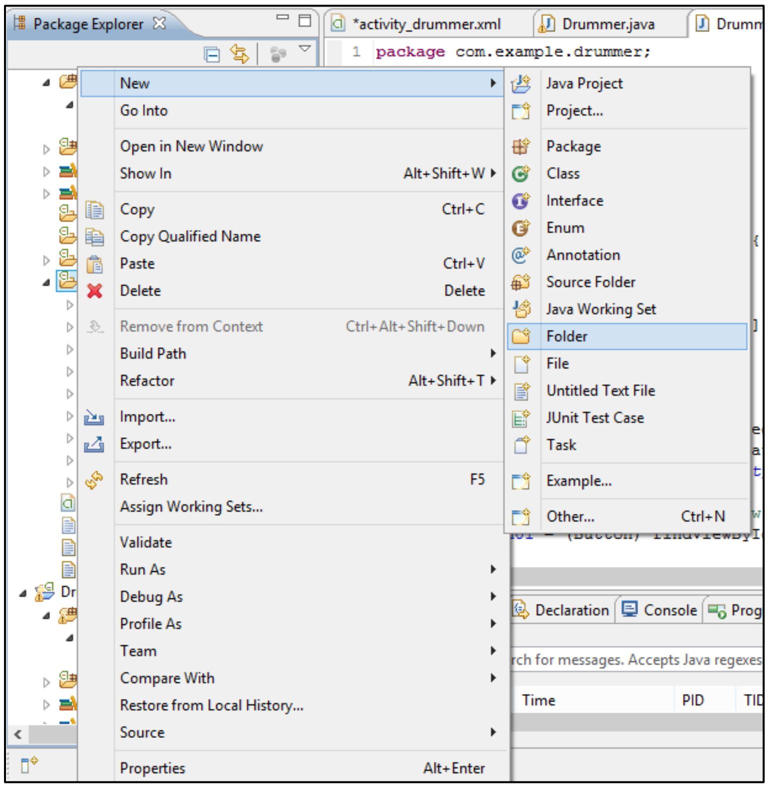
playSound01() will be a public void function we will create in the Drummer Activity. We will add the other android:onClick properties later.

Phase 3: Importing sound files

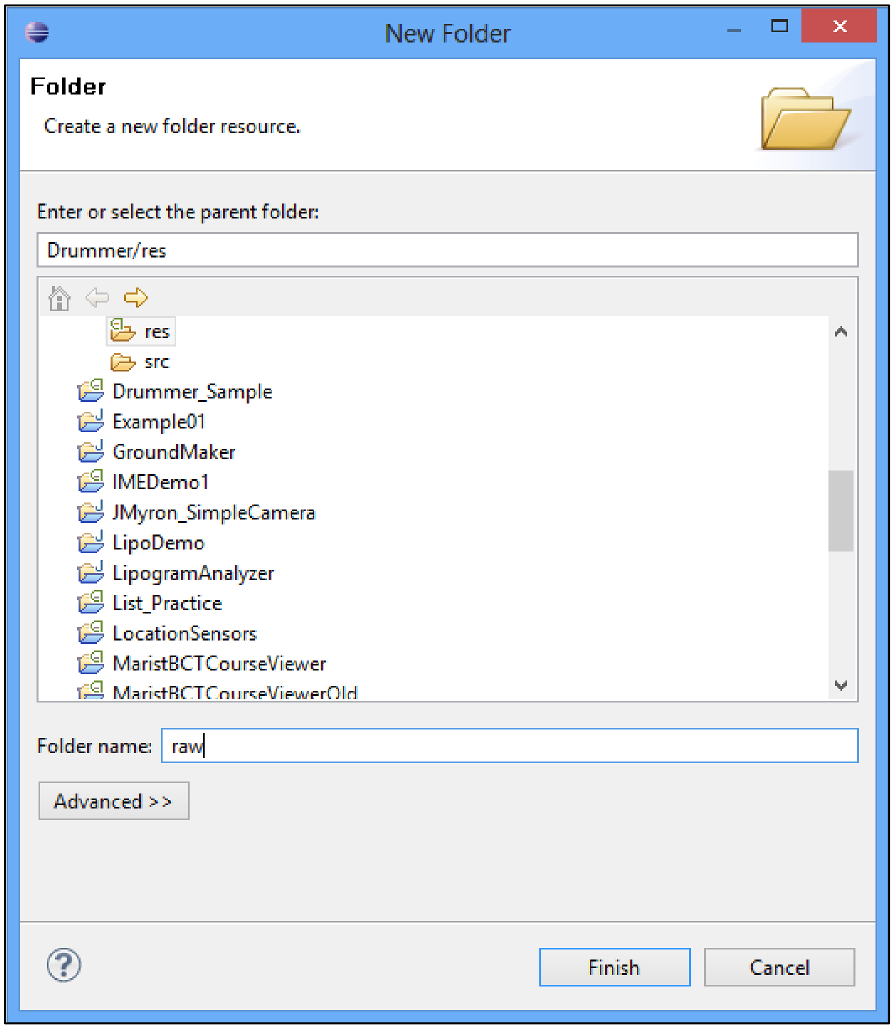
This App needs sound files to function. Images, Sounds, and other Data are stored in the res in folders. Images are stored in a folder named ‘drawable.’ Sounds are stored in a folder named ‘raw’.

Process:

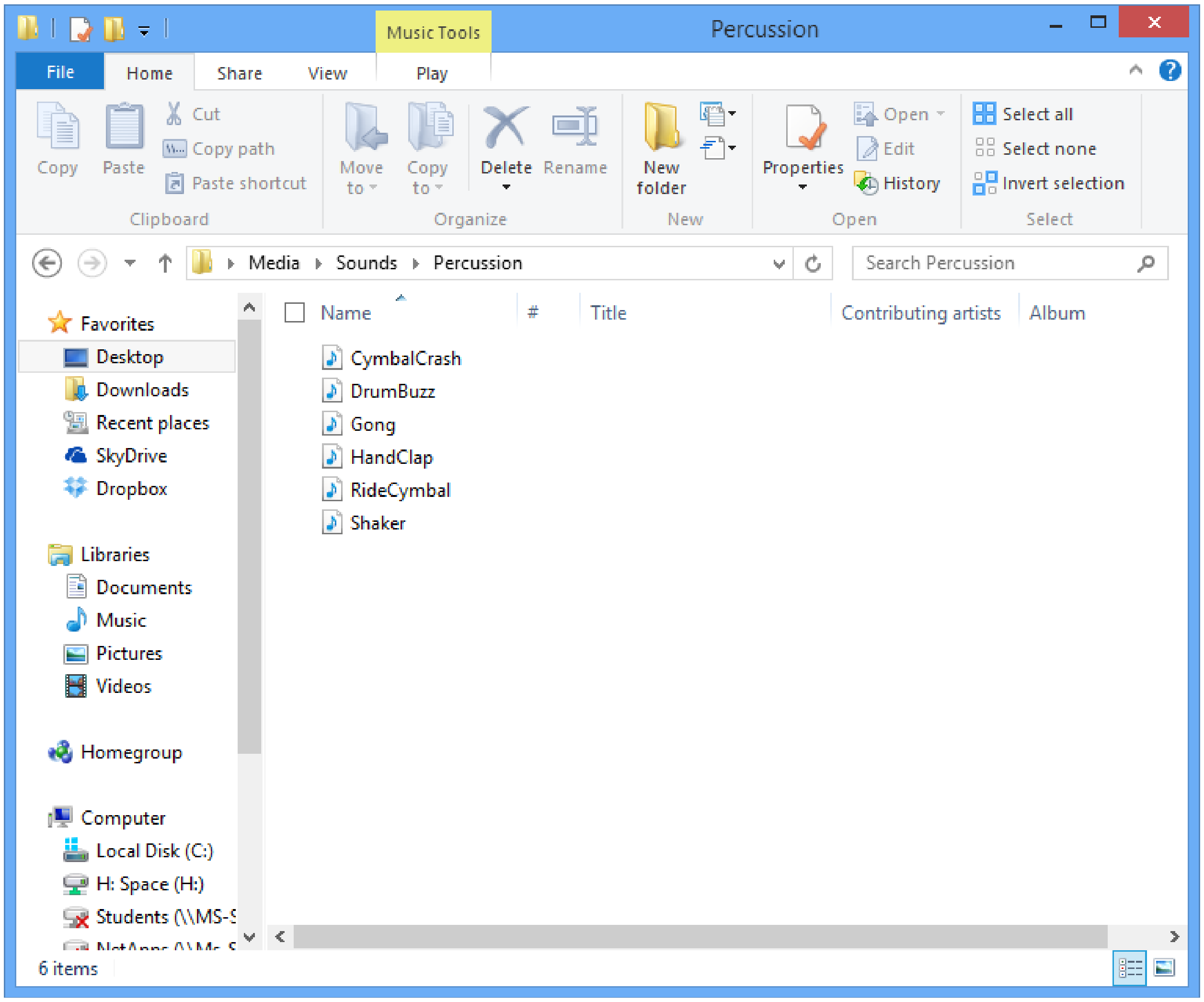
1. Right click on the res folder and select New -> Folder



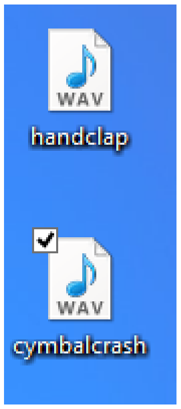
2. Name the Folder 'raw' and click Finish.



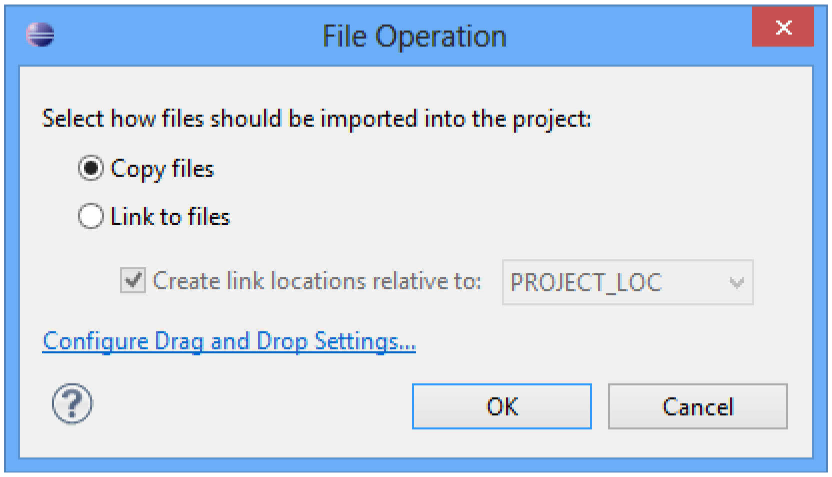
3. On the desktop of your computer, you have a folder called Media. Minimize the Eclipse IDE and find the Media/Sounds/Percussion folder.



4. Copy the HandClap.wav and the CymbalCrash.wav folder to the desktop. Rename the files so they are ‘handclap.wav’ and ‘cymbalcrash.wav’. (No upper case letters or spaces.)



5. Drag these files to the res/raw folder and select ‘Copy Files’ at the dialog window.



File Operation

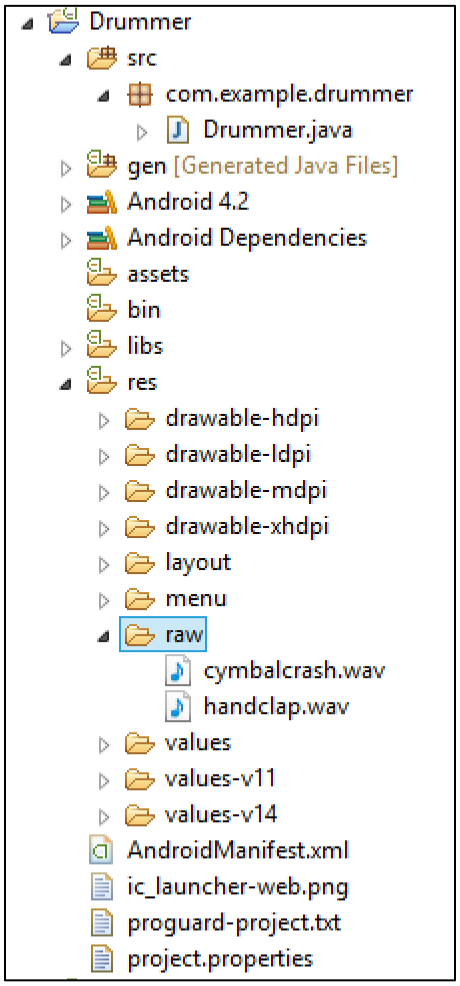
Select how files should be imported into the project:

(●) Copy files
( ) Link to files

[✓] Create link locations relative to: PROJECT_LOC

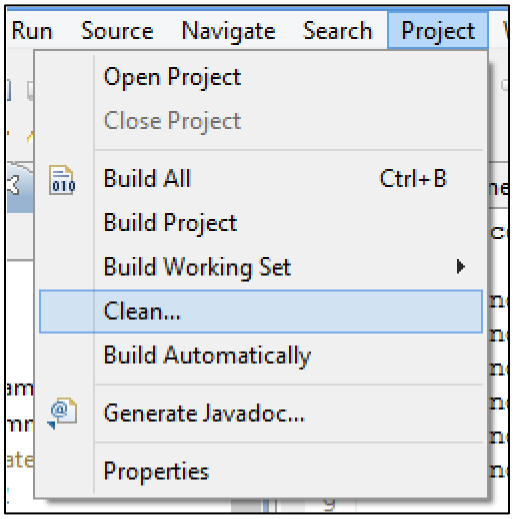
Configure Drag and Drop Settings...

6. The Object tree should look like this:

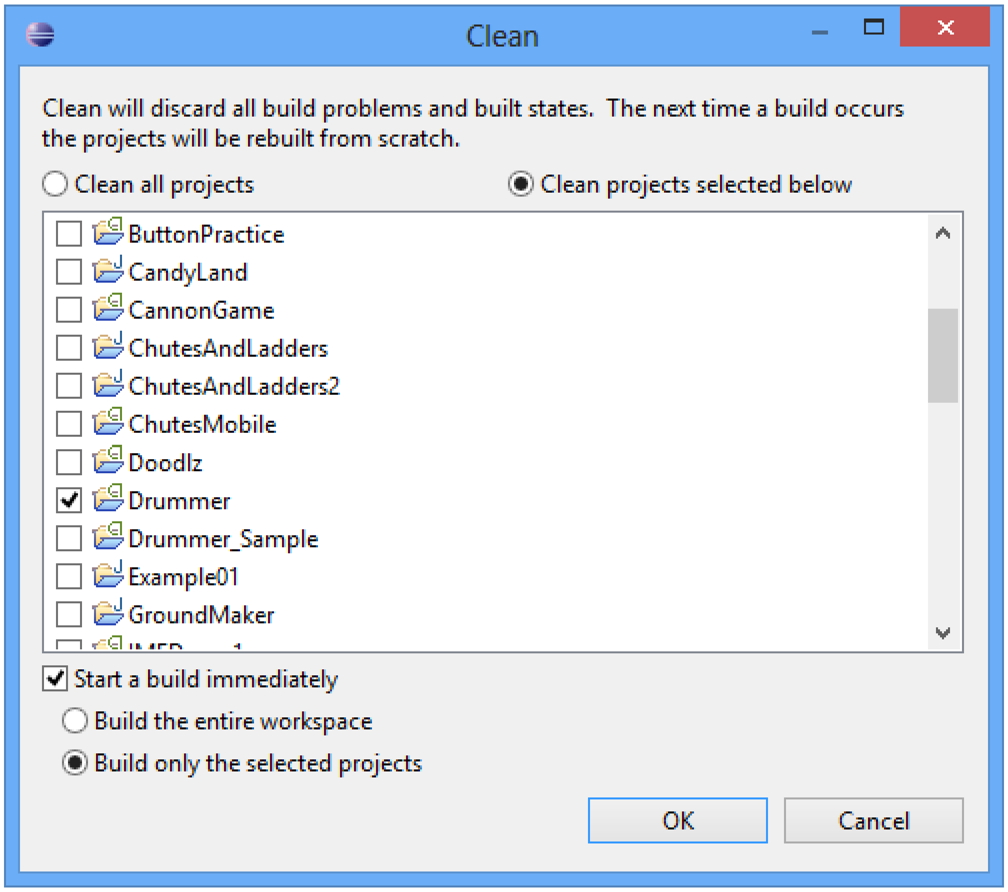


```
Drummer
  src
    com.example.drummer
      Drummer.java
  gen [Generated Java Files]
  Android 4.2
  Android Dependencies
  assets
  bin
  libs
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    layout
    menu
    raw
      cymbalcrash.wav
      handclap.wav
    values
    values-v11
    values-v14
  AndroidManifest.xml
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

7. After adding resources to the project, you need to 'Clean' the project to make sure all the R. references are set. Select Project -> Clean from the menu bar.



8. Make sure your project, Start build immediately, and Build only the selected projects are checked. Click OK.


Clean

Clean will discard all build problems and built states. The next time a build occurs the projects will be rebuilt from scratch.

○ Clean all projects [✓ ● Clean projects selected below]

- ☐ ButtonPractice
- ☐ CandyLand
- ☐ CannonGame
- ☐ ChutesAndLadders
- ☐ ChutesAndLadders2
- ☐ ChutesMobile
- ☐ Doodlz
- ☑ Drummer
- ☐ Drummer_Sample
- ☐ Example01
- ☐ GroundMaker

☑ Start a build immediately

○ Build the entire workspace [✓ ● Build only the selected projects]

9. You are now ready to proceed to writing the Code for the Drummer.java class.

Phase 4: Writing the Java for the Drummer Activity

The Drummer.java class will extend the Android Activity Class and have three types of fields:

private Buttons -> One for each Button in the XML User Interface
private array of MediaPlayer objects to hold and play the sounds
private array of integers of mediaReferences to hold the R. resources pointing to the sound files.

The Drummer.java class will have the following functions:

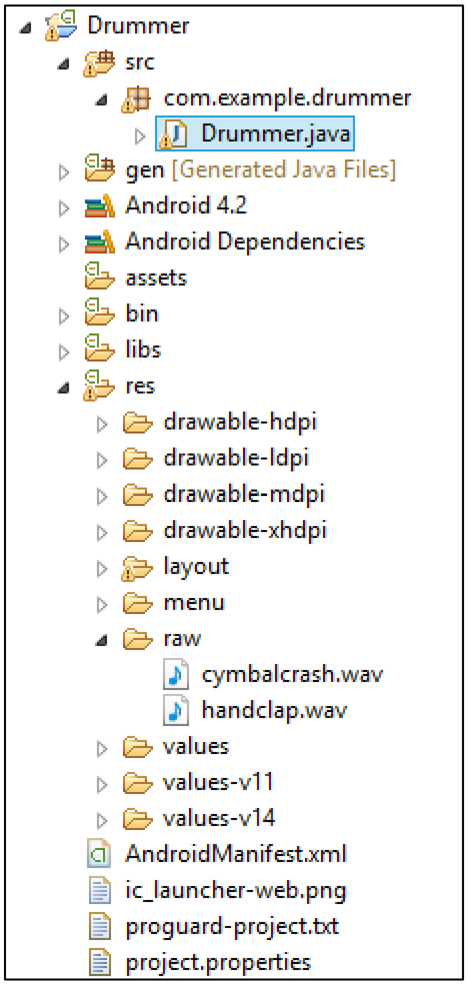
`playSound01(View v)` -> This function will set the index and then call the playSound() function (and functions for the other 9 buttons)

`playSound(int i)`

`stopAllSounds(View v)`

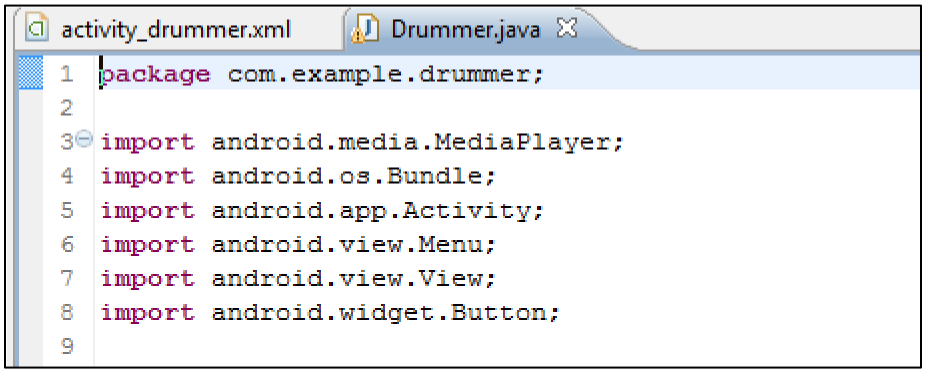
Process:

1. Go to the src/com.example.drummer/Drummer.java file.



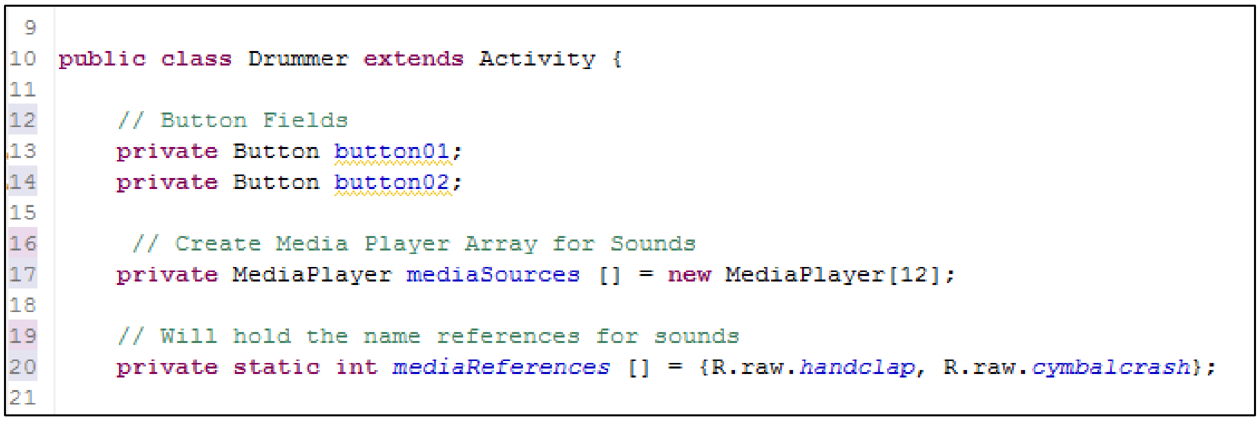
```
Drummer
  src
    com.example.drummer
      Drummer.java
  gen [Generated Java Files]
  Android 4.2
  Android Dependencies
  assets
  bin
  libs
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    layout
    menu
    raw
      cymbalcrash.wav
      handclap.wav
    values
    values-v11
    values-v14
  AndroidManifest.xml
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

2. There will already be code set by Eclipse. We will modify the code for our activity, adding our fields, constructors, and functions. In lines 1 through 9, add the import resources needed for this App.



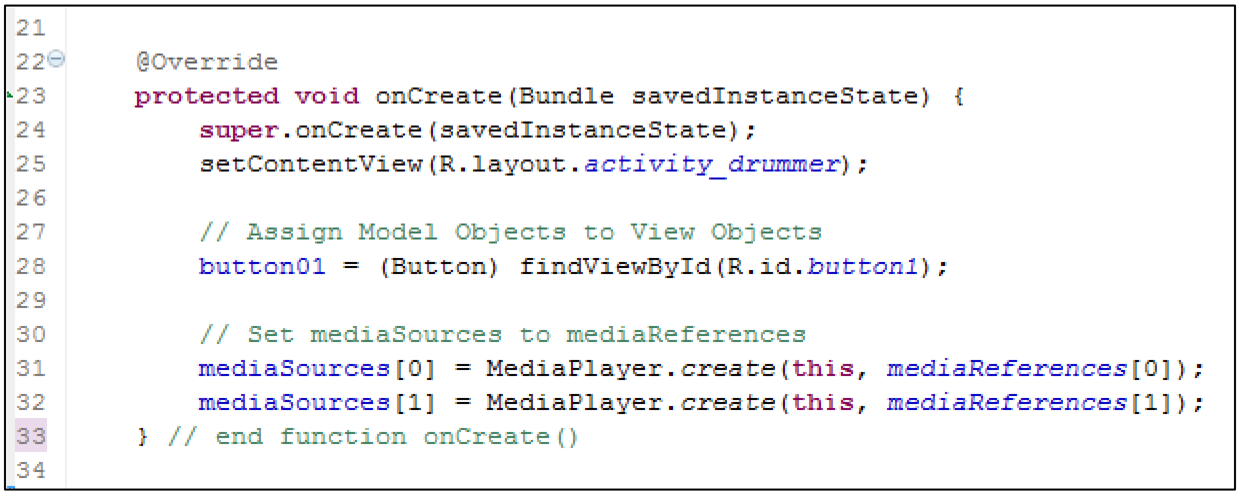
```
1 package com.example.drummer;
2
3 import android.media.MediaPlayer;
4 import android.os.Bundle;
5 import android.app.Activity;
6 import android.view.Menu;
7 import android.view.View;
8 import android.widget.Button;
9
```

3. Under the class declaration in line 10, add the fields for this class: (Note for now, we are only adding two Button fields)



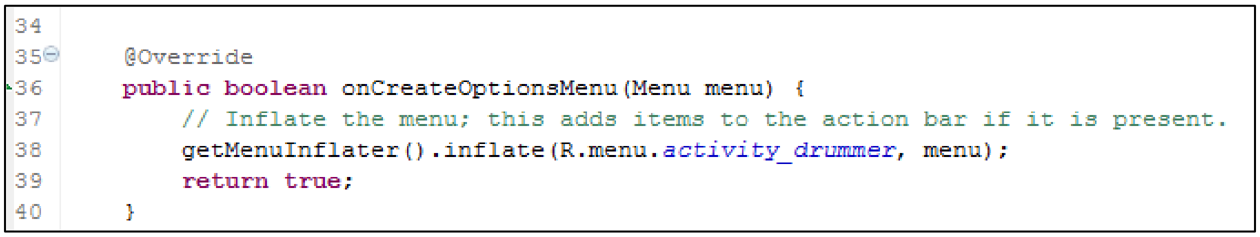
```
 9
10 public class Drummer extends Activity {
11
12     // Button Fields
13     private Button button01;
14     private Button button02;
15
16      // Create Media Player Array for Sounds
17     private MediaPlayer mediaSources [] = new MediaPlayer[12];
18
19     // Will hold the name references for sounds
20     private static int mediaReferences [] = {R.raw.handclap, R.raw.cymbalcrash};
21
```

4. Android Activities do not have traditional Java constructors. Instead, they have the “onCreate” function that defines the fields and initializes the class when the activity is called within the App. Modify the onCreate() class to assign the Fields the XML objects. (This is the binding process where we connect the XML interface to the model of the Java Activity.



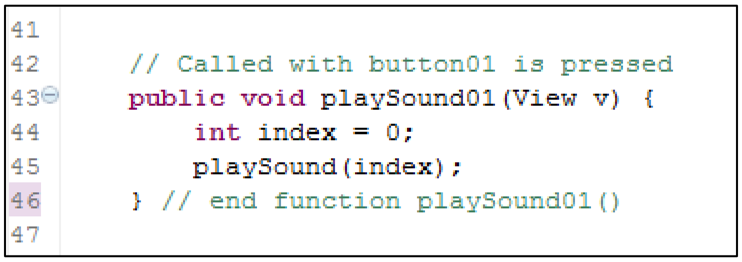
```
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_drummer);
26
27         // Assign Model Objects to View Objects
28         button01 = (Button) findViewById(R.id.button1);
29
30         // Set mediaSources to mediaReferences
31         mediaSources[0] = MediaPlayer.create(this, mediaReferences[0]);
32         mediaSources[1] = MediaPlayer.create(this, mediaReferences[1]);
33     } // end function onCreate()
34
```

5. The boolean onCreateOptionsMenu() function exists to call up a ‘settings’ screen to modify App parameters. Leave this function as written.



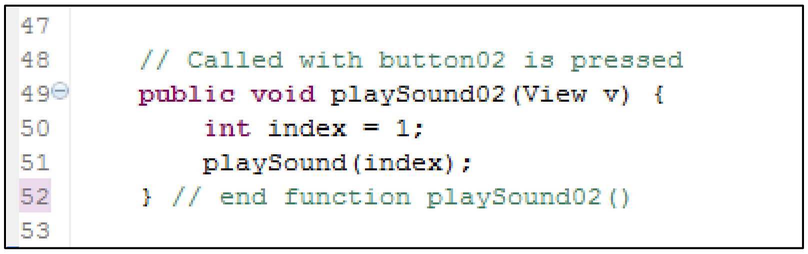
```
34
35     @Override
36     public boolean onCreateOptionsMenu(Menu menu) {
37         // Inflate the menu; this adds items to the action bar if it is present.
38         getMenuInflater().inflate(R.menu.activity_drummer, menu);
39         return true;
40     }
```

6. The next function is called with Button 01 is pressed. The playsound(View v) function sets an integer to correspond to the index of the desired sound. The function then calls the playSound(int i) function with index as the parameter.



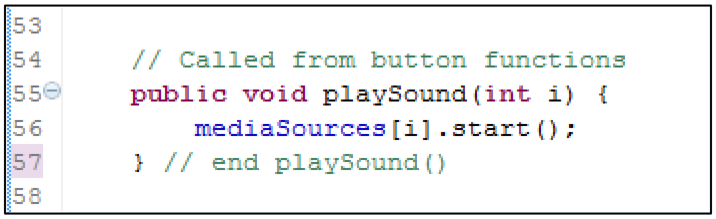
```
41
42      // Called with button01 is pressed
43      public void playSound01(View v) {
44          int index = 0;
45          playSound(index);
46      } // end function playSound01()
47
```

7. Write the function for playSound02(). It follows a similar format:



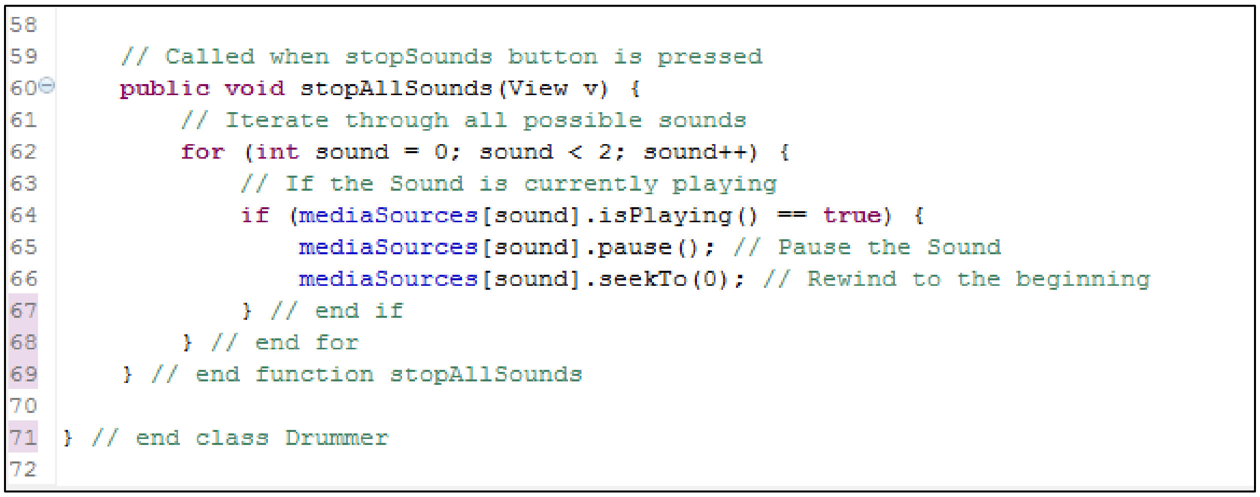
```
47
48      // Called with button02 is pressed
49      public void playSound02(View v) {
50          int index = 1;
51          playSound(index);
52      } // end function playSound02()
53
```

8. Write the playSound() function to have the mediaSources[] objects start their sounds.



```
53
54      // Called from button functions
55      public void playSound(int i) {
56          mediaSources[i].start();
57      } // end playSound()
58
```

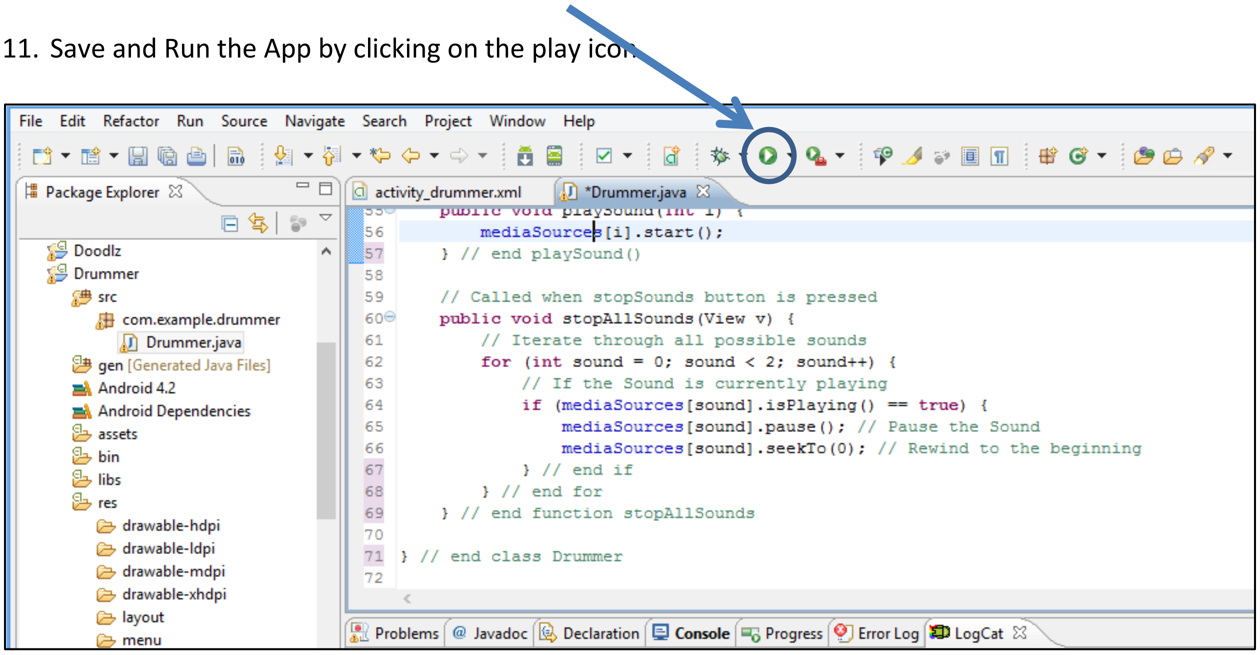
9. Write the function to stop the sounds when the stopButton pressed. Line 71 closes the Drummer class.



```
58
59      // Called when stopSounds button is pressed
60      public void stopAllSounds(View v) {
61          // Iterate through all possible sounds
62          for (int sound = 0; sound < 2; sound++) {
63              // If the Sound is currently playing
64              if (mediaSources[sound].isPlaying() == true) {
65                  mediaSources[sound].pause(); // Pause the Sound
66                  mediaSources[sound].seekTo(0); // Rewind to the beginning
67              } // end if
68          } // end for
69      } // end function stopAllSounds
70
71  } // end class Drummer
72
```

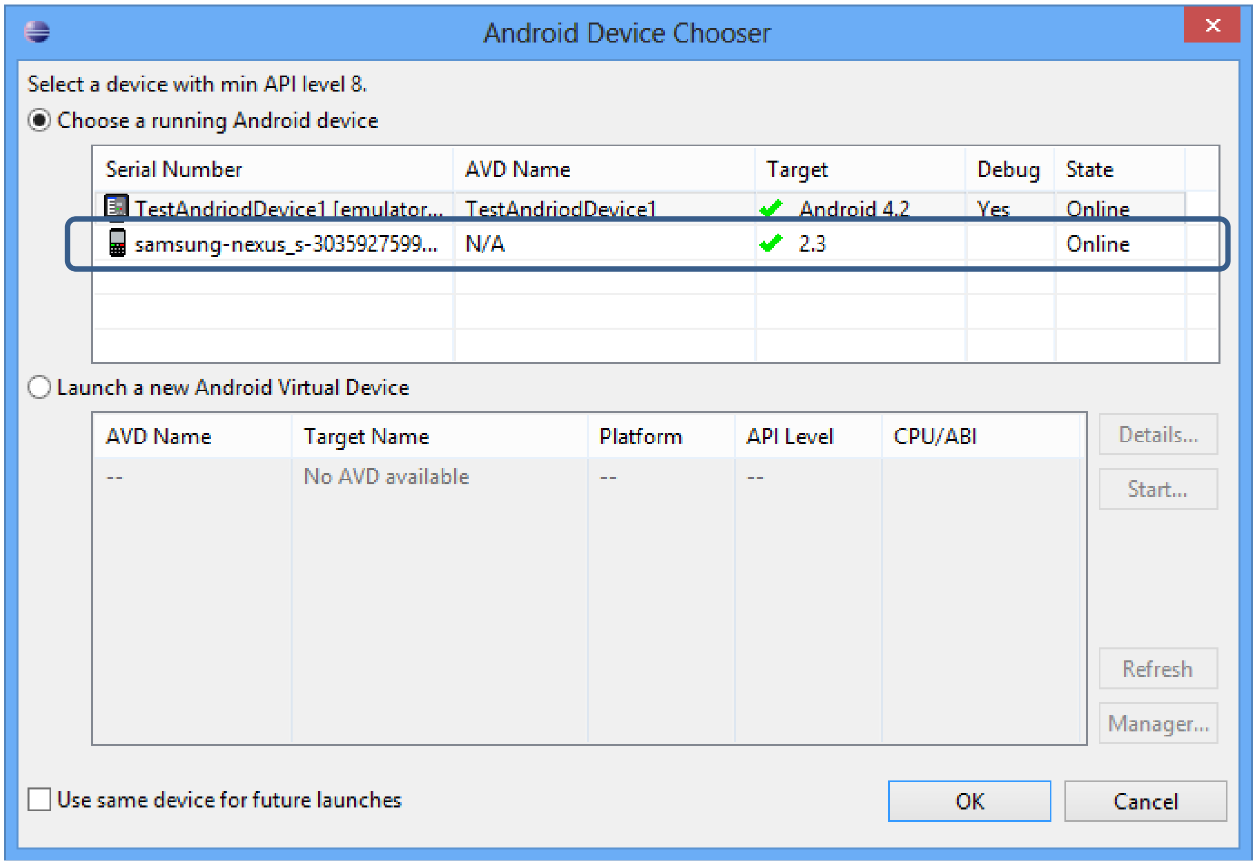
10. Plug in your Android Phone and turn it on.

11. Save and Run the App by clicking on the play icon.



```
    public void playSound(int i) {
        mediaSources[i].start();
    } // end playSound()

    // Called when stopSounds button is pressed
    public void stopAllSounds(View v) {
        // Iterate through all possible sounds
        for (int sound = 0; sound < 2; sound++) {
            // If the Sound is currently playing
            if (mediaSources[sound].isPlaying() == true) {
                mediaSources[sound].pause(); // Pause the Sound
                mediaSources[sound].seekTo(0); // Rewind to the beginning
            } // end if
        } // end for
    } // end function stopAllSounds

} // end class Drummer
```

12. Select your Android device from the available connected devices (Mine is the Samsung-nexus). Click OK.


Android Device Chooser

Select a device with min API level 8.

(●) Choose a running Android device

Serial Number	AVD Name	Target	Debug	State
TestAndriodDevice1 [emulator...	TestAndriodDevice1	✓ Android 4.2	Yes	Online
samsung-nexus_s-3035927599...	N/A	✓ 2.3		Online

( ) Launch a new Android Virtual Device

AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

☐ Use same device for future launches

Phase 5: Finish the App
Maximum Points: 100

This provides a framework for you to finish the App. You will need to do the following to earn an 85.

1. Select 9 Sounds. They must be in .wav, .mp3, or .ogg format. Place these sounds in your rex/raw folder. Make sure they are named with only lowercase letters, numbers, and no spaces.
2. Modify the XML code for the buttons to call their corresponding functions with the android:onClick setting.
3. Clean and Build the Project.
4. Add the references to the sound files in the mediaReferences[] integer array. (All the references need to be located in that array.)
5. Set the mediaSources[] to the references. See lines 31 and 32 in Drummer.java for examples. (A For Loop would be an elegant way to accomplish this)
6. Write the functions for each button in the Drummer.java class.
7. Finish Binding the stopButton to the stop sounds function.

Extensions (Up to 15 points)

1. Modify the Properties of the buttons, text, and background to make the App look more interesting or appealing.
2. Note – There is an ImageButton Object. You could put pictures on Buttons instead of text – research this usage in XML and make changes with picture files.
3. There are several assistive technology Apps or sound effect Apps. You could change your Drum Machine into a Sound Effect Player. Research, search, or record your own sounds for use in the App.