Lab 2: Pure Data in Android

In this lab you are going to learn how to run a pd patch on an android phone. You will begin by using a template we have created. Then you will experiment with changing this template. You will also learn how to create graphical user interface elements such as buttons and sliders in the android app and then how to connect these to your patch.

Downloading the template

1.1. Download the zipped template project from github and place it into your Documents folder:

1.1.1. Go to https://github.com/hydroxate/lab02

1.1.2. Click the Green Clone or Download Button

1.1.3. Select Download Zip

1.1.4. Locate the downloaded zip file (its called lab02.zip) on your computer.

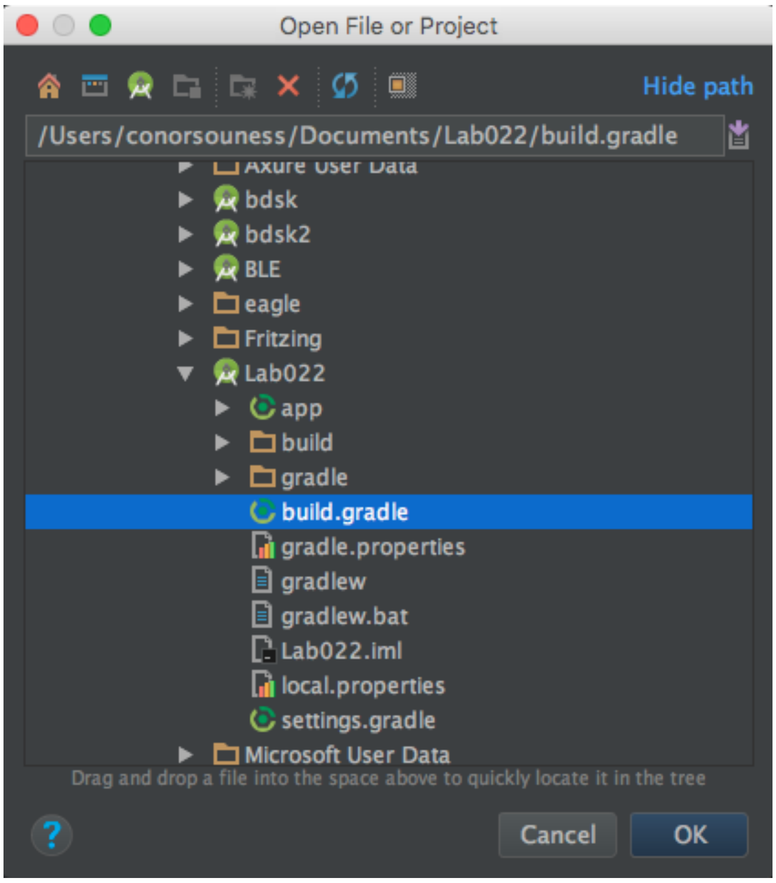
1.1.5. Move it into your Documents folder.

1.1.6. Unzip the lab02.zip file by double clicking it

1.1.7. You should now have a folder named “Lab02-master” in your documents folder. This contains all the files that make up the template android studio project.

Opening the Android Studio Template Project

1.2. Start Android Studio.
1.2.1. Open the template project you downloaded by: Clicking File>Open...
1.2.2. Then Locate the folder Lab02-master in Documents.
1.2.3. Double Click the Lab02-master/gradle/build.gradle

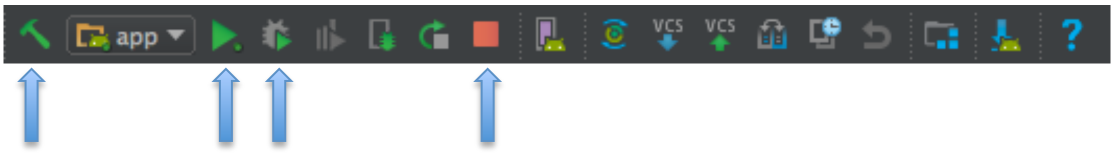


1.2.4. The build.gradle will auto generate all the project files needed.

Running an app

75% of your time will be spent running apps to debug errors.

The buttons are located above the editor.



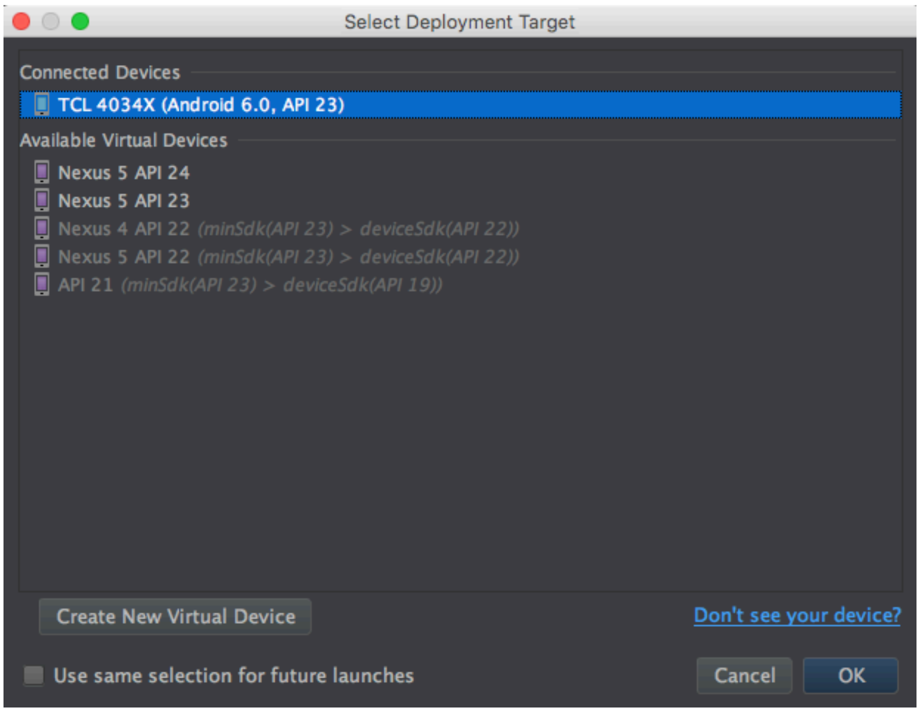
Build | Run|Run & Debug |Stop Application

The hammer icon is to build a package. The Play button is used to run the app on your device. The bug icon with a play triangle is to run the app in debug mode. Big red square = stop running app.

1.2. If you have an Android phone please connect it to the computer now using the charging cabe.

1.3. Run the app by clicking: Run.

The deployment target screen will pop up. This is where you choose where you want the app to run. You can select from phones that are connected to the computer via USB (connected devices) or running the app in an emulator on the computer (available virtual devices).



Select Deployment Target

Connected Devices

TCL 4034X (Android 6.0, API 23)

Available Virtual Devices

Nexus 5 API 24
Nexus 5 API 23
Nexus 4 API 22 *(minSdk(API 23) > deviceSdk(API 22))*
Nexus 5 API 22 *(minSdk(API 23) > deviceSdk(API 22))*
API 21 *(minSdk(API 23) > deviceSdk(API 19))*

Create New Virtual Device | Don't see your device?

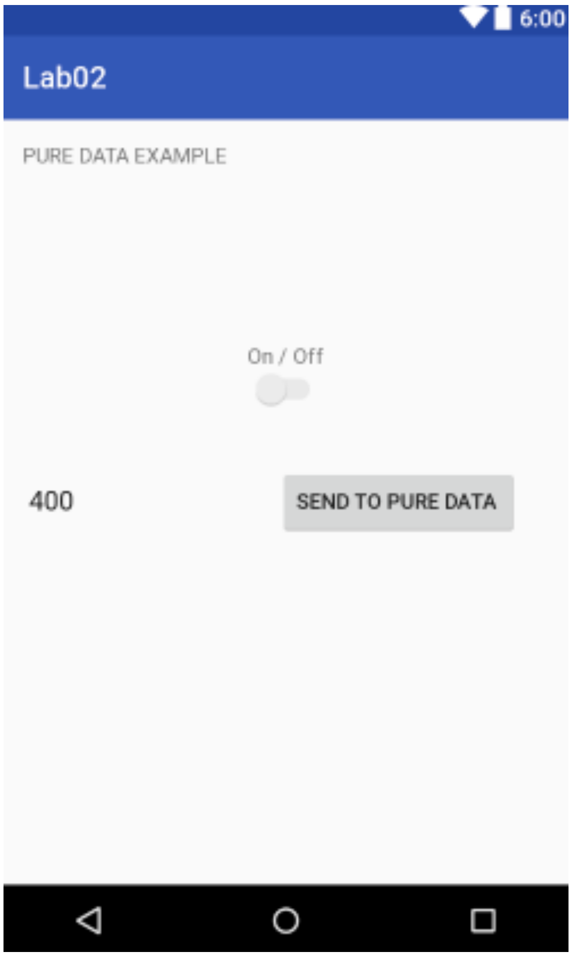
Use same selection for future launches | Cancel | OK

1.4. If you have a phone connected then it should appear as a connected device. Select the connected device and then click OK.

If you do not have an android phone select a virtual device. If no virtual devices are present, raise your hand.

1.5. The app will build, install the apk and then launch on your device.

You will see the following on your phone or on the emulator:



1.6. Tap the on/off switch on the app to turn on the synth.

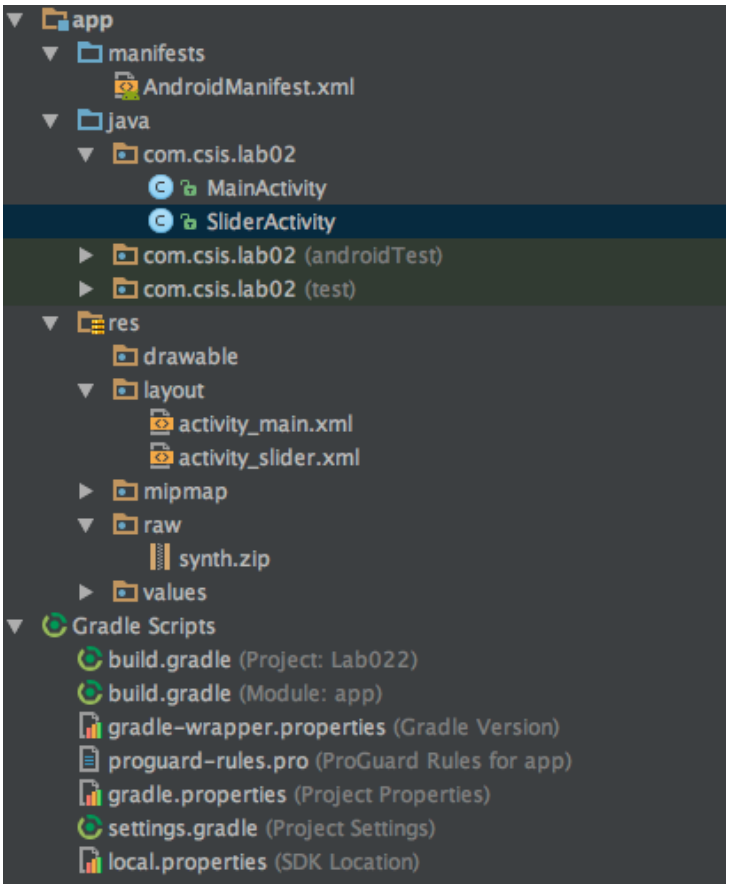
There is a textfield to the left of the button that will send a desired frequency to pure data upon the click of the “Send to Pure Data” button.

1.7. Test different frequencies.

The sound you hear is generated by a PD patch that the app loads on startup. The GUI objects (the on/off switch, textfield and button) are used to pass data into the PD patch.

Examining the template.

1.8 In Android Studio examine the project structure to the left. If it is hidden, there is a button on the left marked "Project". Click this to reveal the project structure.



```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.csis.lab02
        MainActivity
        SliderActivity
    ▶ com.csis.lab02 (androidTest)
    ▶ com.csis.lab02 (test)
  ▼ res
      drawable
    ▼ layout
        activity_main.xml
        activity_slider.xml
    ▶ mipmap
    ▼ raw
        synth.zip
    ▶ values
▼ Gradle Scripts
    build.gradle (Project: Lab022)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

As you can see there are two main sections; "app" and "gradle scripts". Gradle is what is used to build the project. App contains all the files needed for, well, the app.

App has three folders in this project. Manifest is used to dictate activity properties and permissions used by the app. Java contains the code that defines the behaviour of your app. Res contains all resources used by the app!

Resources include layouts, the visual screens in the app. These layouts are in a markup language called XML, similar to, but not the same as, HTML.

At project creation you can specify the launcher screen.

Part 2: Pure Data

Lets look at how a pd patch is integrated into an android app in more detail. Following this we will practice editing the pd patch and updating our app. The aim here is that after this lab you will be able to create your won pd patches and run them on an android device.

Where in the Project is the PD Patch?

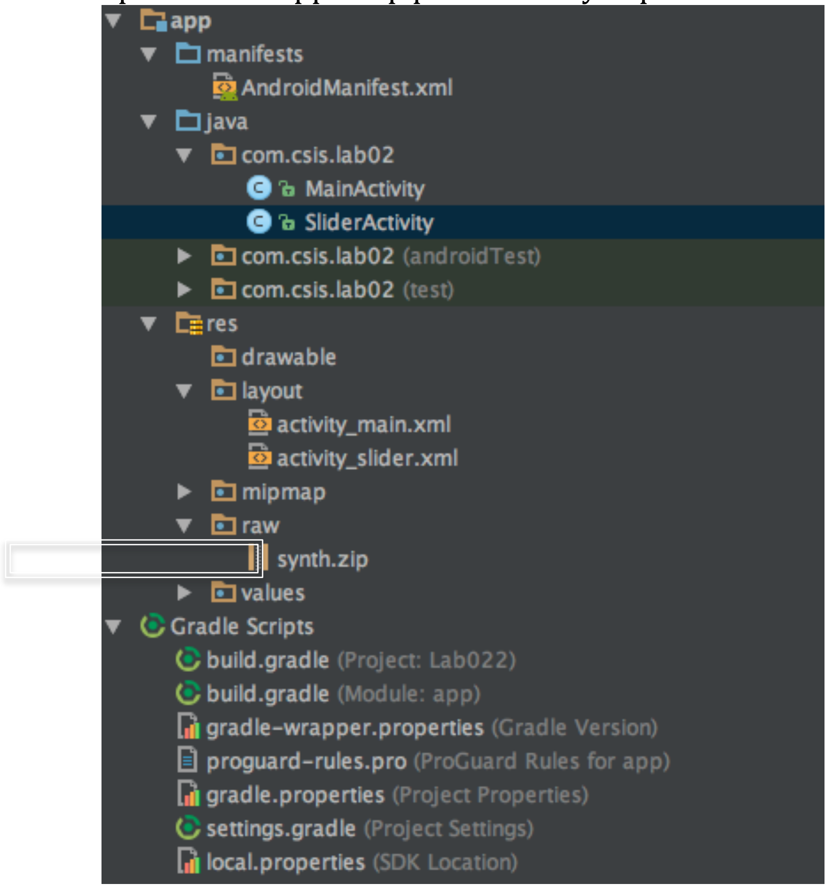
2.1. Locate the “res” section in the project pane. Then locate the folder named “raw”.

This is the folder that holds the pd patch that the app runs.

The patch must be stored as a zip file. This is a limitation of the way pd is implemented in Android. We cant just drop in our patch.pd file. Instead it must be zipped first and the resulting .zip file added to the project.

When the app runs on the phone it unzips the zip file and loads the patch.

2.2. The template has a zipped up patch already in place. It is called synth.zip.

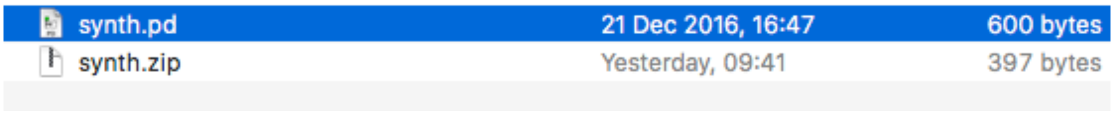


```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.csis.lab02
        MainActivity
        SliderActivity
    ▶ com.csis.lab02 (androidTest)
    ▶ com.csis.lab02 (test)
  ▼ res
      drawable
    ▼ layout
        activity_main.xml
        activity_slider.xml
    ▶ mipmap
    ▼ raw
        synth.zip
    ▶ values
▼ Gradle Scripts
    build.gradle (Project: Lab022)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

2.3. You can locate this folder on the mac by right-clicking synth-zip in android studio and selecting "Reveal in Finder". This makes copying and pasting easier!

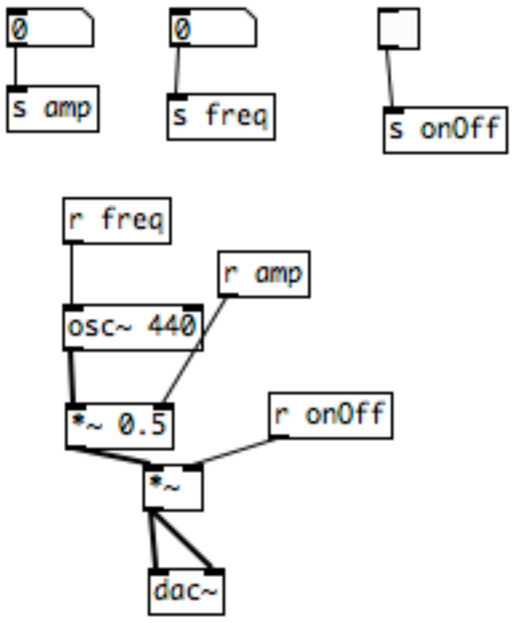
Open the folder that contains synth.zip in finder now.

2.4. In Finder, unzip the synth.zip file, by double clicking it. When it unzips you should now see a file named synth.pd. This is the pd patch that our app runs.



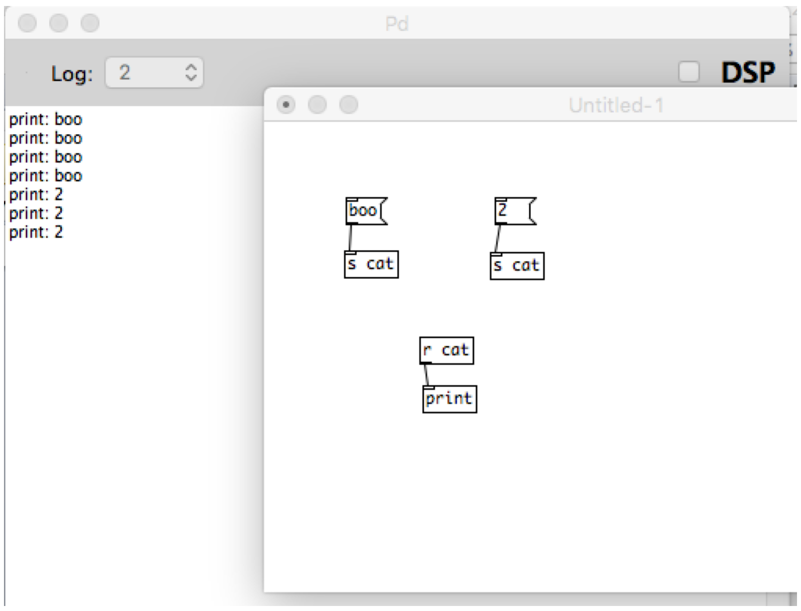
synth.pd	21 Dec 2016, 16:47	600 bytes
synth.zip	Yesterday, 09:41	397 bytes

In finder Double click synth.pd to open it in PD (or in PD select file, open and navigate to the folder and open it). It should like the below.



A small aside: Send and Receive in PD

Note the [s] and [r] objects. S is short for send and r is short for receive. In pd we can use a [send] and [receive] pair instad of a patch chord. Look at the next figure. Any messages sent into [s cat] will arrive at the outlet of [r cat].



```
print: boo
print: boo
print: boo
print: boo
print: 2
print: 2
print: 2
```

If this patch were loaded into an android app we could send data into it by addressing the receive objects. Another way to say this is that the pd patch receives data from the android app that it is loaded in via receive objects.

Lets look at out synth.pd patch again. Note the [r freq] object. The GUI of the app can send data to this receive object. This data will emerge from the [receive] objects outlet.

What other [r] objects are there in synth.pd? Which GUI object on the app sends data to [r amp] ? Which GUI object on the app sends data to [r onOff] ?

Editing the synth.pd patch and testing it in the app

2.5. Change the default frequency of the oscillator to 100Hz.
2.6. Save the patch.
2.7. In Finder select the existing synth.zip file and delete it. (it contains the old unedited copy of synth.pd).
2.8. Now create a new synth.zip file by right clicking on the synth.pd file, and choosing "Compress synth.pd".

This will create a zip file called "synth.pd.zip", we want to name it "synth.zip" as this is what the android app expects to find.

2.9. Click once on synth.pd.zip and when the text field appears, rename it to "synth.zip".
2.10. Delete synth.pd
2.11. Go back to Android Studio.
2.12. Run the app. The default tone that we hear when the on/off switch is clicked on should now be 100hz. Is it?

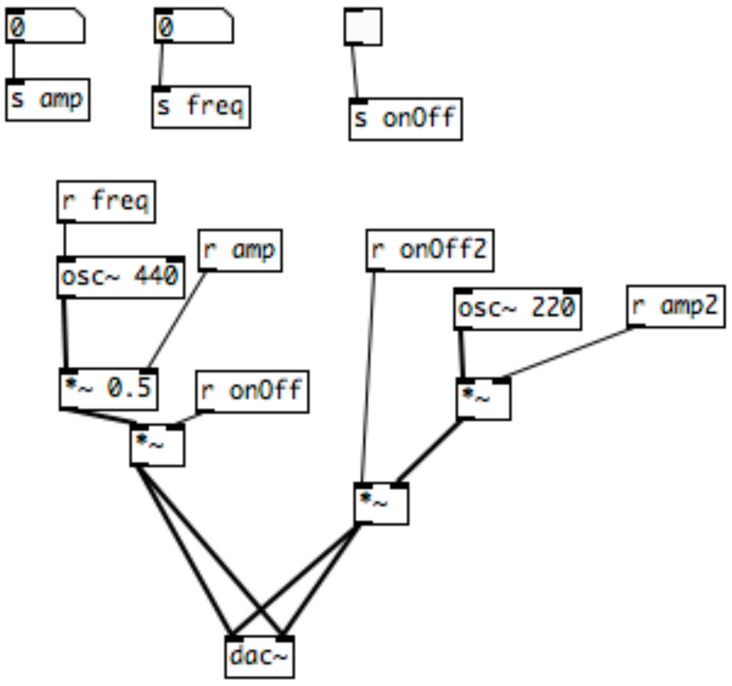
Part 2b: Extending our PD Patch and App

Lets extend the app so that we can control two oscillators. We'll need to edit the PD patch. We'll also need to add some new GUI objects to our app to control the new parts of our patch.

Lets edit the PD patch first.

2b.1. Locate the raw folder in Android Studio.
2b.2. Right click and locate the synth.zip in finder.
2b.3. Unzip the archive.
2b.4. Delete the archive.
2b.5. Open the synth.pd file in pure data.
2b.6. Edit the patch with the following changes:

2b.6.1 Create a new oscillator with a default frequency of 220Hz.
2b.6.2. Create a new [*~] to control its amplitude.
2b.6.3. Create a new receive object called [r amp2] and connect it to the multiplier
2b.6.4. Connect the output of the [*~] to the existing [dac~]
2b.6.5. Test your patch in pd using toggles.



2b.7. Save the patch
2b.8. Close the patch
2b.9. Right click the synth patch, and click "Compress synth.pd".

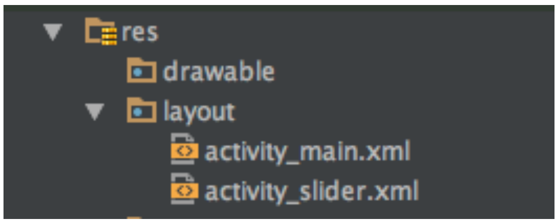
This will create a zip file called "synth.pd.zip". Rename it "synth.zip".

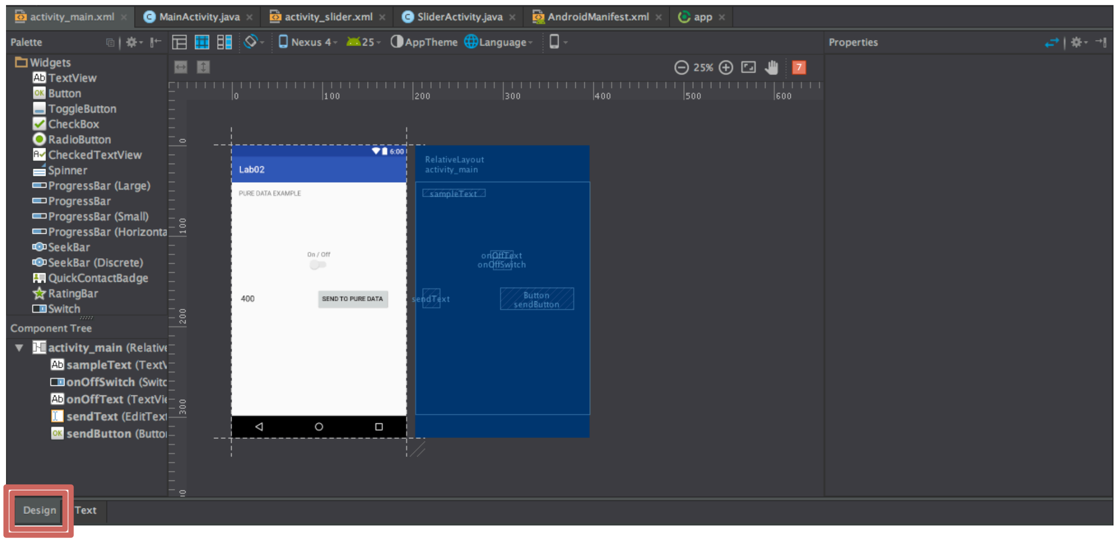
2b.11. Delete synth.pd

Part 3a Examining and Creating GUIs

Now we want to create some new GUI objects in our app that will communicate with the new [receive] objects in our PD patch.

3.1. Click on activity_main.xml in the project pane, it is located under res/layout.

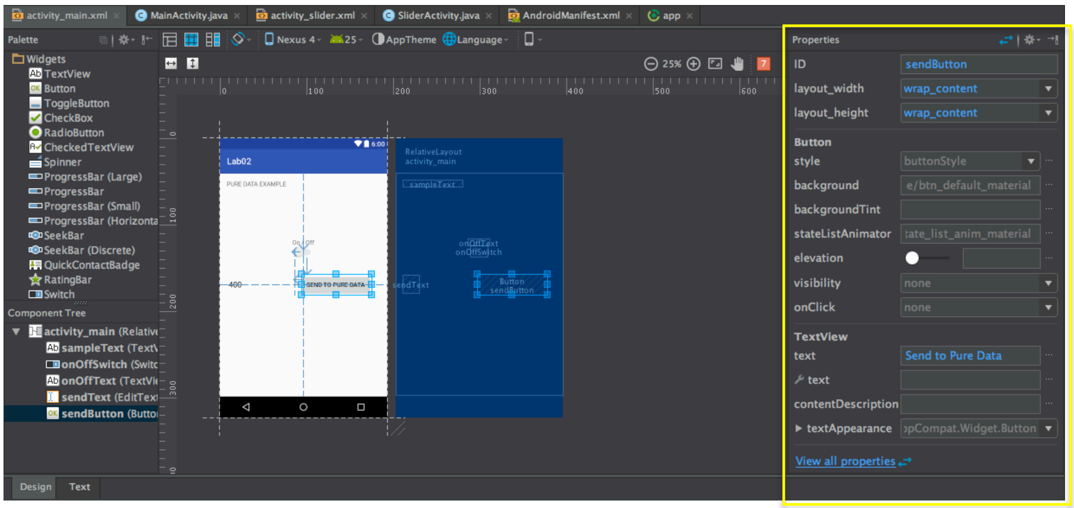




The design view is the default view upon opening a new project. If the text view was the last opened view, it will be the default view. If the view is defaulted to text view, you can switch to design view by pressing the highlighted button above.

This is the design tab. This is where you can drag and drop items to create your GUI. The widgets (indiidual GUI objects like buttons etc) are listed on the left, these can be dragged and dropped onto the screen.

Click on the “send to pd’ button widget that is already in place .
Its properties should now be shown in the properties tab to the right hand side.



ID	sendButton
layout_width	wrap_content
layout_height	wrap_content
Button	
style	buttonStyle
background	e/btn_default_material
backgroundTint	
stateListAnimator	tate_list_anim_material
elevation	
visibility	none
onClick	none
TextView	
text	Send to Pure Data
text	
contentDescription	
textAppearance	›pCompat.Widget.Button

View all properties

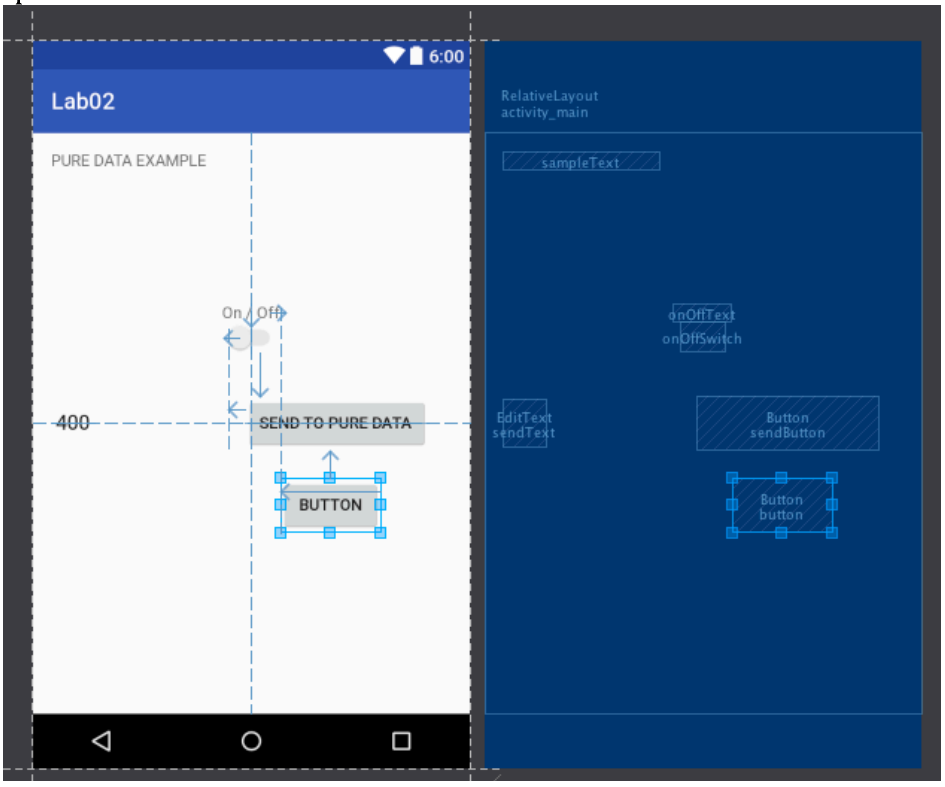
The most important property to note is ID. This ID is what we use to connect behaviours to the widget. We create behaviours or functionality in code and then attach this behaviour to the widget by specifying the ID that the behaviour should be connected to.

There are also items to set the text of the buttons, width, and height too.

Creating Functionality for Widgets

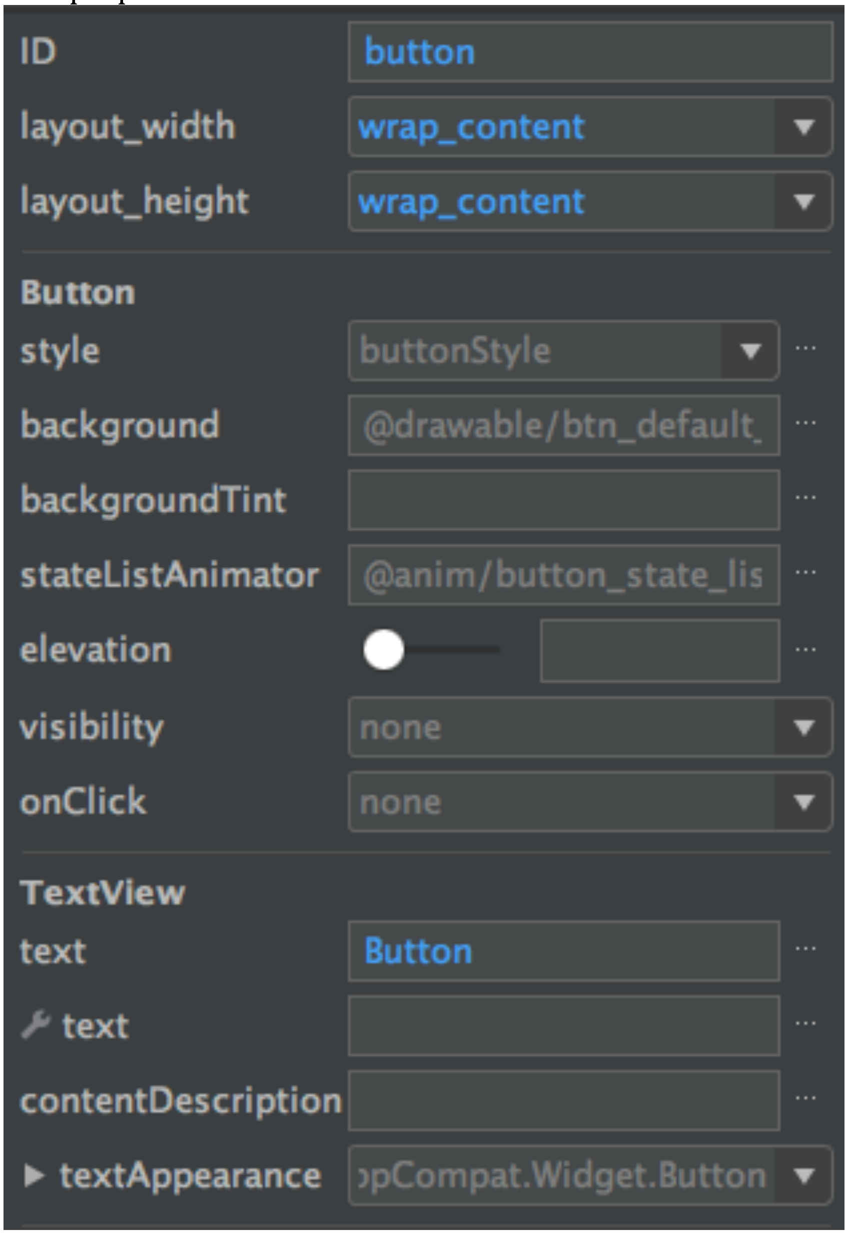
3.2. Double-click activity_main.xml like before.

In the design tab, drag a button widget onto the screen under the "send to pure data" button.



We need to add unique properties to the button for our code to access.

3.3. Go to the properties tab.



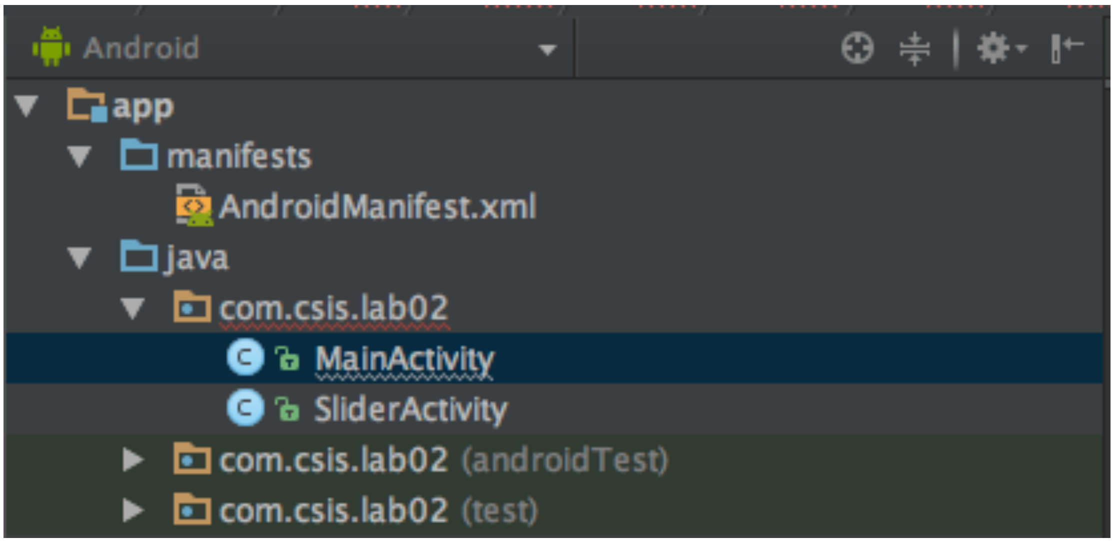
ID	button
layout_width	wrap_content
layout_height	wrap_content
Button	
style	buttonStyle
background	@drawable/btn_default_
backgroundTint	
stateListAnimator	@anim/button_state_lis
elevation	
visibility	none
onClick	none
TextView	
text	Button
text	
contentDescription	
textAppearance	›pCompat.Widget.Button

3.4. Rename the button ID to popUp.

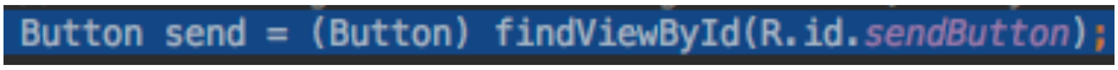
3.5. Rename the button text to Pop Up

3.6. Now that we have placed the button and given it an id we will add functionality to it.

3.7. Double click the MainActivity.java file in the project pane. This is located under the java folder.



3.8. Go to line 36.



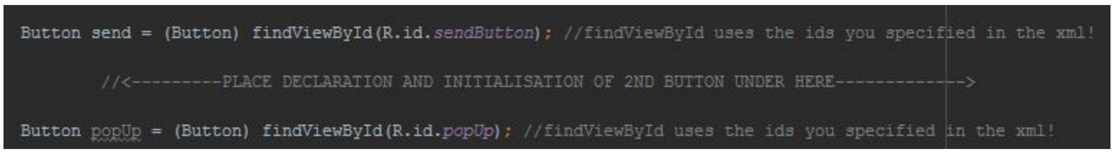
```
Button send = (Button) findViewById(R.id.sendButton);
```

This line is creating an instance of the button from the GUI that the code can work with.

R.id.sendButton is the ID for the "send to pure data" button. That was typed in the properties section of the button in the GUI design tab.

3.9. Underneath this line add the following:

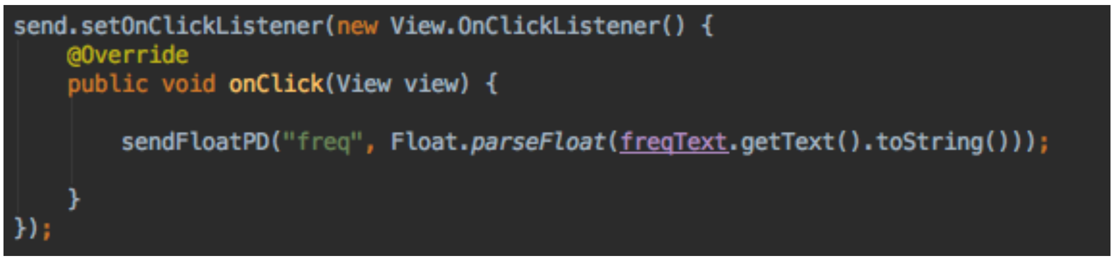
Button pop = (Button) findViewById(R.id.popUp);



```
Button send = (Button) findViewById(R.id.sendButton); //findViewById uses the ids you specified in the xml!

    //<---------PLACE DECLARATION AND INITIALISATION OF 2ND BUTTON UNDER HERE------------>

Button popUp = (Button) findViewById(R.id.popUp); //findViewById uses the ids you specified in the xml!
```

This now points to the button you created with the id specified in the properties section of the Design Tab.

To add a listener that does something when the button is clicked, we will look at the code for the "send to pure data" button.



```
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        sendFloatPD("freq", Float.parseFloat(freqText.getText().toString()));

    }
});
```

Here that button is assigned a listener that will activitate on a click (hence, onClick).

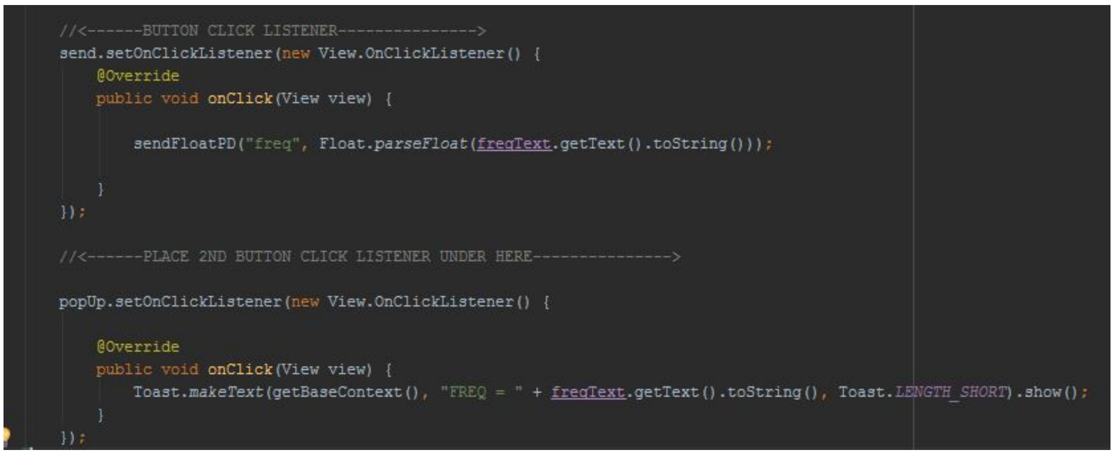
3.10. Create the listener for the button **popUp**.

We'll copy the format of the "Send to Pure Data" button!

Type this at line 78.

```
pop.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View view) {
                        Toast.makeText(getBaseContext(), "FREQ = " +
        freqText.getText().toString(), Toast.LENGTH_SHORT).show();
        }
});
```

It will look like this:



```
//<------BUTTON CLICK LISTENER--------------->
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        sendFloatPD("freq", Float.parseFloat(freqText.getText().toString()));

    }
});

//<------PLACE 2ND BUTTON CLICK LISTENER UNDER HERE--------------->

popUp.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View view) {
        Toast.makeText(getBaseContext(), "FREQ = " + freqText.getText().toString(), Toast.LENGTH_SHORT).show();
    }
});
```

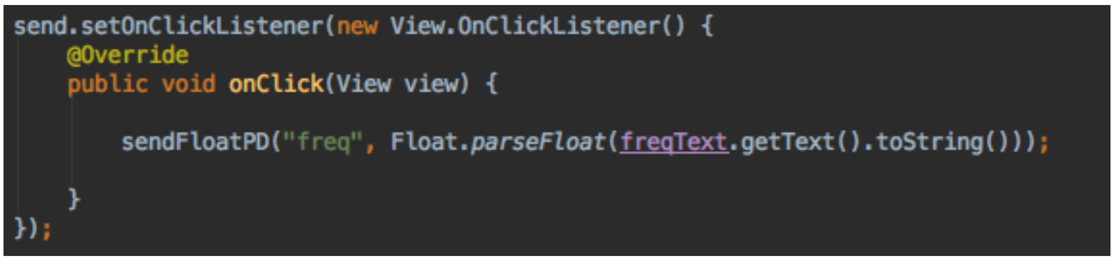
3.11. Build and run app and test!

Any errors raise your hand.

Things to note:

I have code here to send floats to puredata. This sendFloatPD method can be called. It needs the name of the receiveEvent in the patch and a float!

In the sendButton it goes:



```
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        sendFloatPD("freq", Float.parseFloat(freqText.getText().toString()));

    }
});
```

sendFloatPD needs two Strings: receiveEvent name and a float value to send.

So if you added a editText in the GUI for amplitude called ampText, you could then send it to a receiveEvent called amp with:

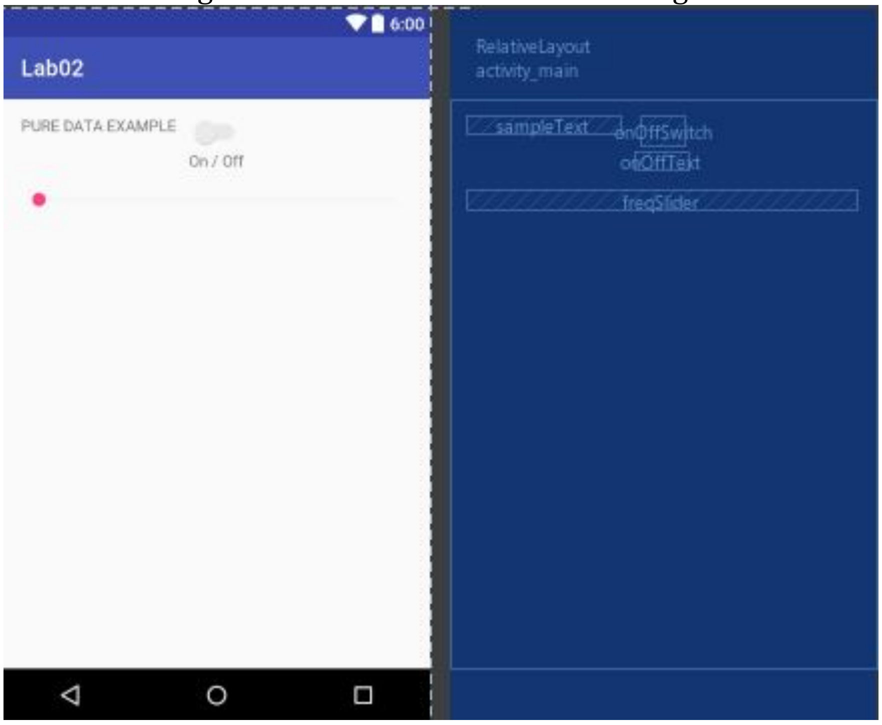
sendFloatPD("amp", Float.parseFloat(ampText.getText().toString()));

Adding a Slider to the GUI

4.1. In the project pane, there is another xml and java file regarding sliders. These are called seekbars in Android.

4.2. Double click the activity_slider.xml.

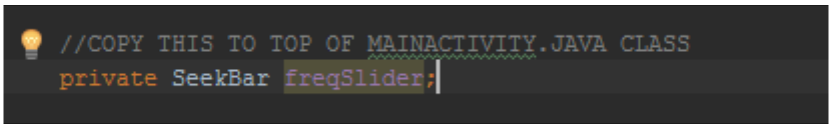
4.3. Enter the Design Tab. This will show the following:



4.4. Your task is to bring the slider into your activity_main.xml and have that send the frequency to pure data each time it updates the value!

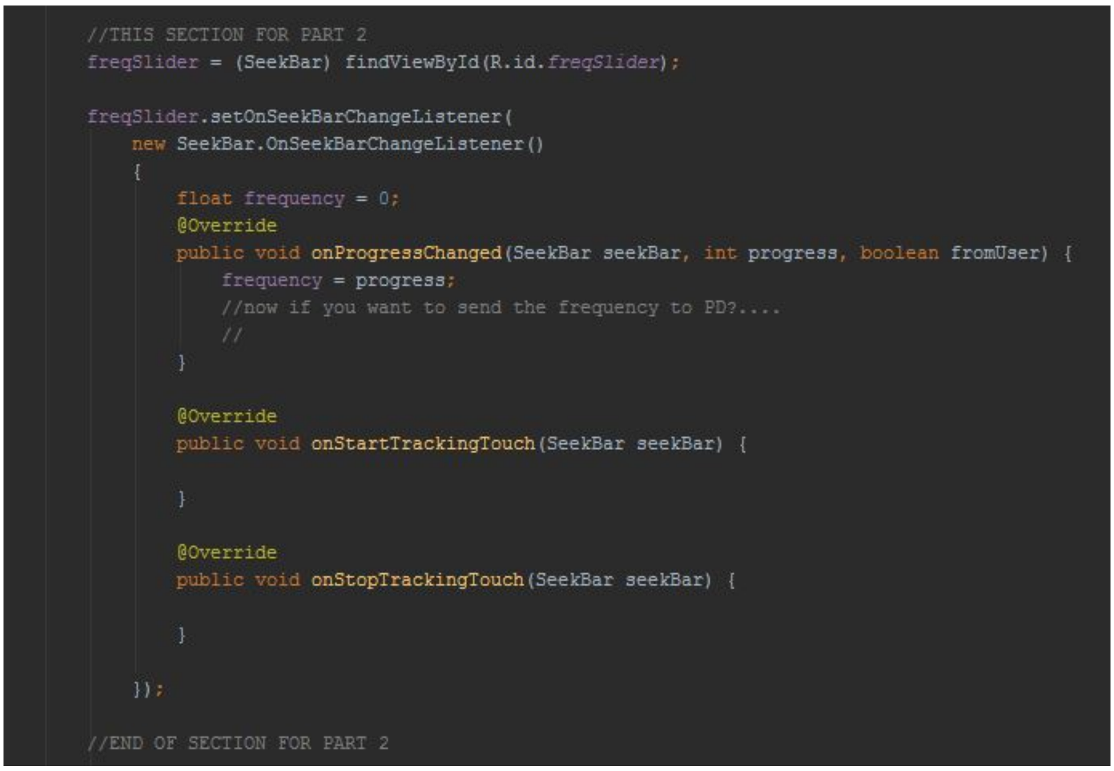
4.5. Double click SliderActivity.java

4.6. Similar to before, the slider must be declared and initialised in onCreate(). Copy the following code into Main Activity above the OnCreate() method.



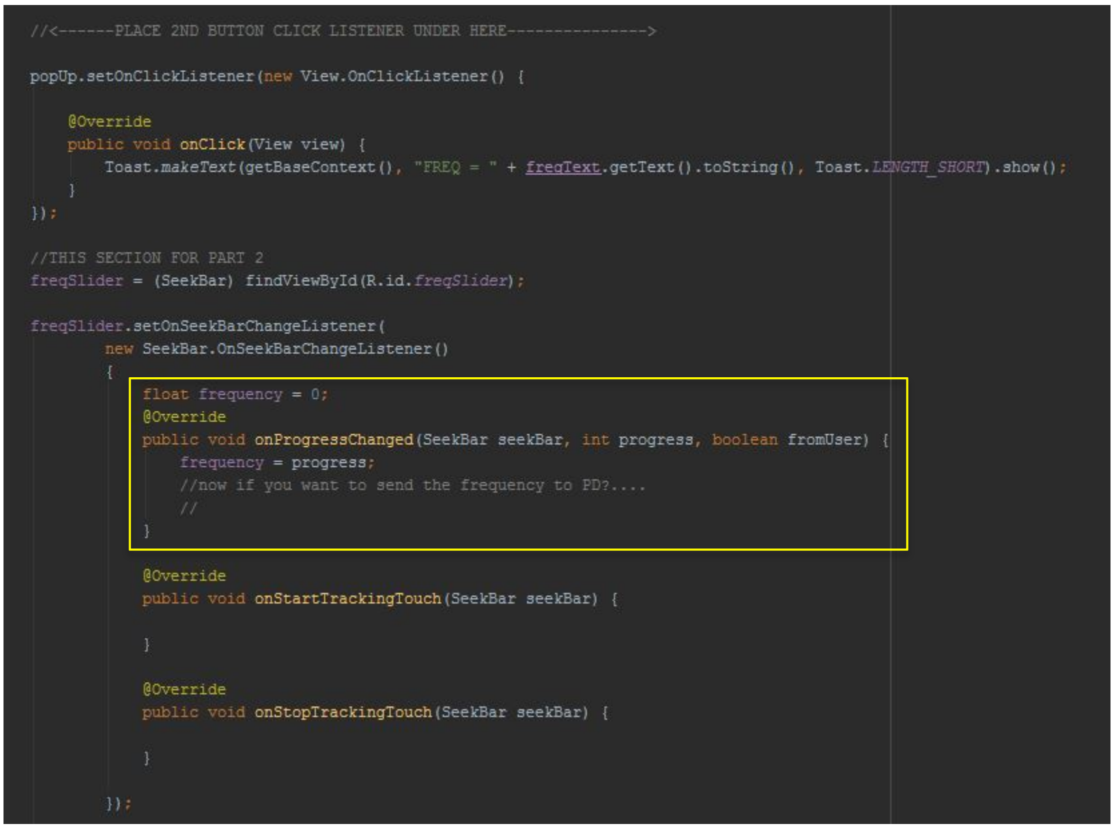
```
//COPY THIS TO TOP OF MAINACTIVITY.JAVA CLASS
private SeekBar freqSlider;
```

4.7. The following code needs to be copied and pasted into MainActivity.java underneath the buttons created in the previous section.



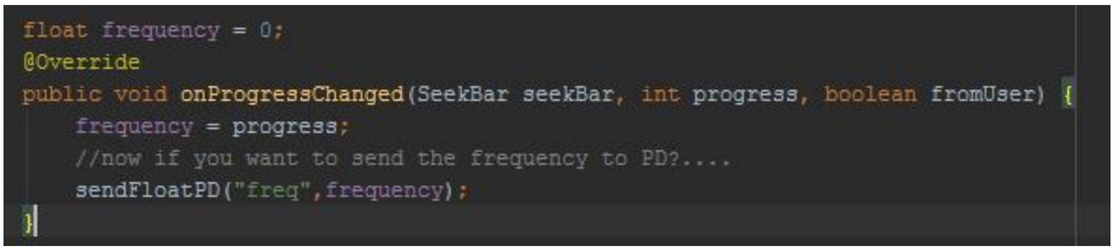
```
//THIS SECTION FOR PART 2
freqSlider = (SeekBar) findViewById(R.id.freqSlider);

freqSlider.setOnSeekBarChangeListener(
    new SeekBar.OnSeekBarChangeListener()
    {
        float frequency = 0;
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
            frequency = progress;
            //now if you want to send the frequency to PD?....
            //
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {

        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {

        }
    });

//END OF SECTION FOR PART 2
```

It should look like this:



```
//<------PLACE 2ND BUTTON CLICK LISTENER UNDER HERE--------------->

popUp.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View view) {
        Toast.makeText(getBaseContext(), "FREQ = " + freqText.getText().toString(), Toast.LENGTH_SHORT).show();
    }
});

//THIS SECTION FOR PART 2
freqSlider = (SeekBar) findViewById(R.id.freqSlider);

freqSlider.setOnSeekBarChangeListener(
        new SeekBar.OnSeekBarChangeListener()
        {
          float frequency = 0;
          @Override
          public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
              frequency = progress;
              //now if you want to send the frequency to PD?....
              //
          }

          @Override
          public void onStartTrackingTouch(SeekBar seekBar) {

          }

          @Override
          public void onStopTrackingTouch(SeekBar seekBar) {

          }

        });
```

4.8. To send the slider value as a frequency to the patch, add the following code to the onProgreeChanged Method highlighted above:



```
float frequency = 0;
@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    frequency = progress;
    //now if you want to send the frequency to PD?....
    sendFloatPD("freq",frequency);
}
```

4.9. Test app to make sure it works.

Mandatory Homework:

You'll need to edit the pure data patch for more functionality.
Add a second oscillator to the patch.

In total, have at least 6 widgets for both oscillators including the amplitude and frequency (get creative!).

Use buttons, switches, sliders and text fields to make a more complicated synthesiser!