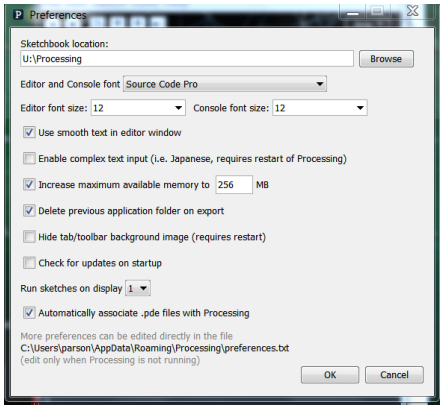
CSC 220 Object Oriented Multimedia Programming, Fall 2019

Dr. Dale E. Parson, Assignment 4, planetarium music server controlled by student Android client.
This assignment is due via **D2L Assignment 4 Android client** by **11:59 PM on 26 November**.

Important dates and notes on bonus points:

A. After going over this code on November 12, all sessions in November will be work sessions.
B. **November 19's work session will be in the KU Planetarium**. **Attendance is mandatory** (10%) unless you have a documented excuse from a physician's office or the health or counseling center.
C. Anyone who has Assignment 4 working on an Android device and can demo that to me by the end of the November 21 class earns **10% bonus points (a)** on this project. You must turn in your code by end of November 21, in addition to demoing to me. You can work in an optional team of 2, or solo. Make sure to put your name or the names of team members in the comments at the top of your sketch. The first 10 students or teams (5 per course section) who have Assignment 4 working on an Android device by November 21 and agree to be performers at the December 2 event **earn an additional 10% (b)**, but they must attend a practice session in the planetarium on Sunday December 1, and I will not accept students to perform who were late with Assignment 3.
D. **We will not have class on Tuesday November 26**. I will be traveling. I will post a Zoom video and announce its URL ahead of time. Thanksgiving break is on November 27-29.
E. Monday December 2 is our event in the planetarium with the Princeton Laptop Orchestra. **If you arrive by 6:40 PM and stay for the whole event, you will earn 10% bonus points on this assignment after signing in at the end**. If you cannot make Monday evening, you can help with setup on Sunday December 1 to earn the 10%. We can set a time to meet.
F. Tuesday December 3 we will meet in the planetarium for any last-minute work. December 5 we will review for a final exam.
G. The final exam for the 12:00 class is on Tuesday, Dec. 10, 2019, 8:00 a.m. – 10:00 a.m. The final exam for the 1:30 class is on Thursday, Dec. 12, 2019, 11:00 a.m. – 1:00 p.m. They will be in Old Main159. You must contact me with a documented medical excuse to miss the exam, or you will earn 0% on it.
H. The advertised 10% per day late penalty applies to this project 4.

When using Processing on the Kutztown campus Windows computers, make sure to start out **every time** by setting your Processing Preferences -> Sketchbook Location to U:\Processing. The U:\ drive is a networked drive that will save your work and make it accessible across campus. If you save it to your desktop or the lab PC you are using, you will lose your work when you log out. You must save it to the U:\ drive. If you do not have a folder called Processing under U:\, you must create one using the Windows Explorer. Processing Preferences is under the File menu on Windows.



If you will be downloading Processing 3.X and running it using an off-campus computer (do not use version [1]2.X for assignments), you can copy your project sketch to a flash drive on one machine, and then copy it from the flash drive to another Processing sketch folder.

Steps for getting 3D sketch **CSC220F19MIDIassn3parson** (Dr. Parson's MIDI keyboard solution to CSC220 Fall 2019 Assignment 3) to work as 2D sketch **CSC220F19Androidassn4Client** (OSC/UDP message interface on the Android) to play Dr. Parson's **CSC220F19MIDIassn4Server** on a laptop via wireless LAN.[2]

Pitfall: You will create sketch **CSC220F19Androidassn4Client** from handout sketch **CSC220F19MIDIassn3parson**, with editor tabs for **Musician**, **Note**, **CartesianPolar**, and **Menus**. You will also copy and save sketch **CSC220F19MIDIassn4Server** with its own, distinct **Musician**, **Note**, and **CartesianPolar** tabs. Be careful to copy & paste the correct tabs from my handouts on the course page.

1. Copy Dr. Parson's **CSC220F19MIDIassn3parson** sketch into a new Processing editor and Save As **CSC220F19Androidassn4Client**. Save it after each step.
2. One at a time, copy the **Musician**, **Note**, **CartesianPolar**, and **Menus** tabs from **CSC220F19MIDIassn3parson** and paste them as new tabs into **CSC220F19Androidassn4Client** tabs, similar to the start of Assignment 3. I have made changes since Assignment 3, so make sure to copy them from the Assignment 4 **CSC220F19MIDIassn3parson** section on the course page.
3. (11.25%) In **CSC220F19Androidassn4Client** setup(), change all occurrences of P3D to P2D for the 2D renderer. If you are testing on a PC or laptop that is also running the **CSC220F19MIDIassn4Server** sketch, use the Processing size() function rather than fullScreen() in setup(). When you start testing on separate devices, switch to fullScreen() in **CSC220F19Androidassn4Client** setup().
4. (11.25%) Try to run it on a PC or laptop, and correct any problems from trying to use 3D graphics. You will see error messages for the unsupported 3D operations. Any calls to the library translate() function that supply a Z, third argument must remove the third argument. Keep the translate call, but remove the third argument, unless the X and Y arguments are 0; when X and Y are 0, just remove that translate() call. Remove the 3D shape in Note.display(), along with anything else requiring a non-0 Z coordinate; remove any third argument to scale() *if there is one*. Also, *if there is* a rotateX() or rotateY() call, remove it, and change rotateZ() to rotate() *if there is one*.
5. (11.25%) **Save and run it in *Java mode***. It should still run & emit MIDI notes at this point, but there will be no 3D graphics. We are cutting out 3D to avoid clutter and CPU consumption on the Android. Testing on Android comes later.
6. Copy Parson's **CSC220F19MIDIassn4Server** sketch into a new Processing editor and Save As **CSC220F19MIDIassn4Server** (same name).

[1] If you receive an error message **The package oscP5 or netP5 does not exist** when attempting to run with OSC for the first time, go to **Sketch -> Import library -> Add library** on the editor's menu, **Filter** on **oscP5**, click and Install this networking library on your PC or laptop. Also, the first time you try to change from **Java mode** to **Android mode** (step 13 below) at the upper right corner of the editor, you will have to walk through **Add mode -> Android mode -> Install**. After **Install** completes, switching to Android mode the first time walks you through **Download** (android) **SDK automatically** and its installation. This SDK is the underlying Google library for programming in Java on Android.

[2] A few students running Macs with a recent version of OSX found that 3D Assignment 3 handout code would crash on the Mac. My older OSX works OK. One student enabled debugging via **Debug -> Enable Debugger** from the Processing editor. If you get cryptic crashes on handout code that should work, try enabling the debugger this way.

7. One at a time, copy the **Musician**, **Note**, **and CartesianPolar** tabs from **CSC220F19MIDIassn4Server** and paste them as new tabs into tabs, similar to steps 1 and 2 above. Save **CSC220F19MIDIassn4Server**. You will make **NO** further changes to **CSC220F19MIDIassn4Server**, but you must use it for later testing. I have made changes since Assignment 3, so make sure to copy them from the Assignment 4 **CSC220F19MIDIassn4Server** section on the course page
8. (11.25%) Back to **CSC220F19Androidassn4Client**: Insert these statements **startOSC(); orientation(LANDSCAPE);** as the very last statements in function setup() in the main Tab, before setup()'s closing curly brace. **Save and test**. It should work as before, and now you should see print statements like this at the bottom of your Integrated Development Environment (IDE – the editor) window, with your device's IP address:

One client IPADDR: 10.0.1.17 (**Your IP address will likely be different**. USE **127.0.0.1** when both on same PC.)
SETTING MYIPADDR to 10.0.1.17
SETTING MYIPADDR to 10.0.1.17
One client IPADDR: 127.0.0.1
OscP5 0.9.9 infos, comments, questions at http://www.sojamo.de/oscP5
[2019/11/8 15:46:27] PROCESS @ OscP5 stopped.
[2019/11/8 15:46:27] PROCESS @ UdpClient.openSocket udp socket initialized.
[2019/11/8 15:46:28] PROCESS @ UdpServer.start() new Unicast DatagramSocket created @ port 12000
[2019/11/8 15:46:28] PROCESS @ UdpServer.run() UdpServer is running @ 12000
[2019/11/8 15:46:28] INFO @ OscP5 is running. you (10.0.1.17) are listening @ port 12000

9. (11.25%) In **CSC220F19Androidassn4Client**, insert this sequence of statements at the very top of function draw() in the main Tab, after draw()'s opening curly brace. **Save and test**. In order to run this test, you must first start **CSC220F19MIDIassn4Server** from its editor window, and then start your **CSC220F19Androidassn4Client** to communicate with the server.

```
void draw() {
  background(0, 0, 0);  // black for planetarium
  if (! connectionReady()) { // STUDENT ADD THIS SECTION TO CLIENT FOR ASSN4.
    return ;
  }
```

It should run, but now **CSC220F19Androidassn4Client** requires the user to walk through a series of text menus. The first displays the screen resolution; click the mouse once to step through it. The second sets the MIDI channel for the client device from 1 through 15 (I have reserved channel 0 for keyboard-based testing on the server), and the third requires the user to connect to the IP address[3] and port (usually port 12001) printed at the bottom of the server's editor window.[4] Note that when you are running 2 sketches simultaneously, println() debug messages may appear at the bottom of the other window. This is a quirk of Processing.

[3] When you are testing the client and server sketches on a single PC or laptop, enter IP address **127.0.0.1** in the client menu user interface, since you are not using a wireless network. **127.0.0.1** is "localhost", meaning client-server network messages stay on that single computer.
[4] If there are two network connections on the server device, the server println() may report the wrong IP address for the wireless network used by the client device. On Windows you can query the server IP address by running the DOS **cmd** interpreter and typing **ipconfig**. On Linux and Mac the command is **ifconfig**.

Figure 1: Client prints display resolution. Click through using the mouse.

Figure 2: Set your MIDI channel and click Submit.

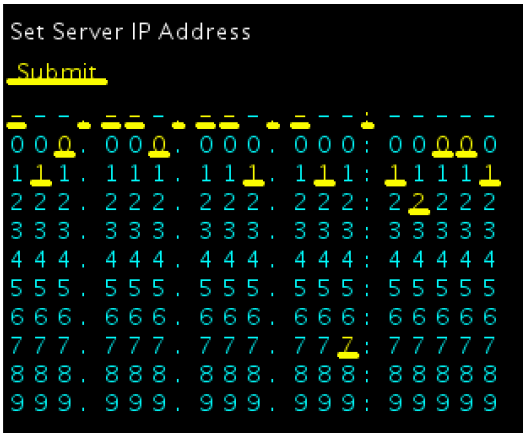


Figure 3: Set the server's IP address and port number, then click Submit.

After submitting a valid server address and port, the client displays the 2D musical keyboard using rectangles. The server's display on the dome uses 3D. I plan to add many graphical effects on the server for the December 2 event. The handout server is bare bones.

10. (11.25%) In **CSC220F19Androidassn4Client** , remove or comment out all code for connecting to MIDI devices and sending MIDI messages. Block comments are useful. To find code depending on MIDI libraries, I first commented out this line in the client's main tab:

```
// import javax.sound.midi.* ;
```

Then I followed the error messages. For a large block of code that does not already use /* … */ block comments, you can use them:

```
/*
// MIDI OUTPUT DEVICE SELECTION:
```

```
final int midiDeviceIndex = 0 ;  // setup() checks for number of devices. Use one for output.
// NOTE: A final variable is in fact a constant that cannot be changed.
MidiDevice.Info[] midiDeviceInfo = null ;
// See javax.sound.midi.MidiSystem and javax.sound.midi.MidiDevice
MidiDevice device = null ;
// See javax.sound.midi.MidiSystem and javax.sound.midi.MidiDevice
Receiver receiver = null ;
// javax.sound.midi.Receiver receives your OUTPUT MIDI messages (counterintuitive?)
// SEE https://www.midi.org/specifications/item/gm-level-1-sound-set but start at 0, not 1
*/
```

Make sure to keep both of these statements at the bottom of setup():

```
musicians[0] = new Musician(programNumber, controlEffectNumber, controlEffectData2, MIDIchannel);
startOSC();  orientation(LANDSCAPE); // added 11/12, see footnote on page 6
}
```

The MIDI code within classes Musician and Note must be commented out or removed, but DO NOT CHANGE THE BLOCK STRUCTURE OF THEIR FUNCTIONS, other than commenting out or removing "try...catch" statements (and their closing "}") for InvalidMidiDataException. The next steps require you to add OSC messages to the server within Note.display(), in place of the MIDI NOTE_ON and NOTE_OFF messages.

11. (11.25%) Replace **channel** with **oscClientMidiChan** everywhere that **channel** appears **within function** Note.display(). **DO NOT CHANGE THE FUNCTION DECLARATION ITSELF**:

    void display(boolean isHighlighted, boolean isBeingPressed, int **channel**) { // keep this as channel

    oscClientMidiChan is the global channel variable in Menus.pde that the user selected via the channel selection menu. It is the channel dedicated to this client device. If you change it in the function declaration, Note.display() just uses **oscClientMidiChan** as a parameter, which works like a local variable, thereby hiding the global variable of the same name. Every other occurrence of **channel within** Note.display() other than the declaration must change to **oscClientMidiChan**.

12. (11.25%) Where the handout code sent NOTE_ON and NOTE_OFF messages, call sendOSCMessage instead. Here is the structure of the section of Note.display() that sent MIDI NOTE_ON and NOTE_OFF in assignment 3. This code comes immediately after my commented-put 3D call to box() and popMatrix(); I commented out the entire pushMatrix()-3D-popMatrix() section after rect().

```
if (! isSounding.get(oscClientMidiChan)) {
  sendOSCMessage("noteon", oscClientMidiChan, pitch, velocity);
  // NOTE_ON CODE was here.
  isSounding.put(oscClientMidiChan, true);
 }
} else { // This else is counterpart to "if (isBeingPressed) {" further up.
 if (isSounding.get(oscClientMidiChan)) {
  sendOSCMessage("noteoff", oscClientMidiChan, pitch, velocity);
  // NOTE_OFF code was here.
  isSounding.put(oscClientMidiChan, false);
```

```
    }
  }
//} catch (InvalidMidiDataException dx) {
```

At this point you should be able to start the server on a PC or laptop, start the client on the same machine, and play notes on the server via the client user interface.

13. Running on the Android device is not required before turning in your assignment unless you want the November 21 and optional performer bonus points listed in bullet C on the first page, or you just want to use an Android device. To run on the Android after you have it working using a laptop or PC, do the following steps.

a. Save and start **CSC220F19MIDIassn4Server** running in Java mode. Once you start running any sketch in Android mode, Processing tends to want to run everything in Android mode.
b. Enable your Android tablet or phone for USB debugging by going to *Settings -> About device*, and then tapping the *Build number* 7 times to enable USB debugging. On the KU Samsung tablets the sequence is *Settings -> About tablet -> Software information -> Build number* 7 times. If *Developer options* show up under Settings, this has already been done. Also, make sure your Android screen is unlocked and be ready to enable USB debugging during the following steps.
c. In Processing's Sketch menu, go to Import Library -> Add Library for Android mode, if it is not already there. You may have to auto-update the Android SDK on your computer.
d. With the server running per step (a), and with your most recent changes to **CSC220F19Androidassn4Client** saved (make sure setup() uses fullScreen(P2D)), plug the Android device into the PC via its USB cable. If any other Android apps pop up on the PC/laptop, such as Android File Transfer, quit out of them. They compete with Processing for the connection.
e. Enable Android Internet access for the sketch via the **Android -> Sketch Permissions** menu as seen in Figure 4.

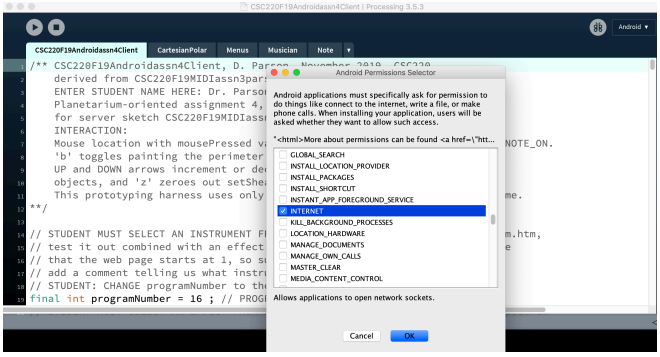


Figure 4: Enable Android Internet permission

f. Run **CSC220F19Androidassn4Client** on the Android device, and unplug the USB cable.[5] It will work in either vertical or horizontal orientation – tall or wide, respectively – but I have found that horizontal (wide) works best to avoid interacting with Android hot locations. **Changing orientation**

[5] You can leave the Android USB cable plugged in to see println() debugging statements on your PC or laptop, but you should do final testing with the USB cable unplugged.

after the sketch has started running on the Android restarts the sketch, so be careful not to do that.[6]

g. You will have to go into Android's **Settings -> Connections** to connect to the wireless LAN the first time for that router. This working sketch stays on the Android device after disconnecting the USB cable.

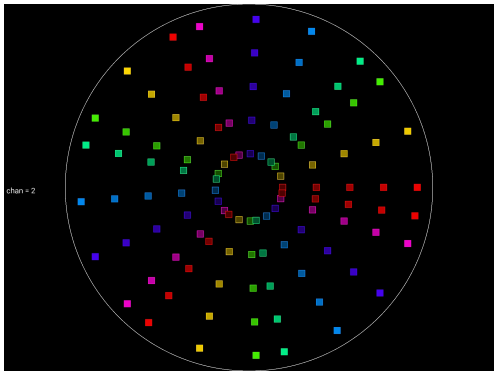


Figure 5: An Android screen shot while running the sketch

[6] Correction added 11/12/2019 – insert this statement immediately after startOSC(); at the bottom of function setup(): **orientation(LANDSCAPE);** it has no effect when running the client on the PC, but it keeps the Android client in landscape orientation, and avoid restarts on re-orientation. Restarts still occur if you minimize the sketch on Android. Also, in the Menus tab, change the statement near the top of function startOSC() from **globalPointSize = height / 48 ;** to **globalPointSize = height / 24 ;** for the Galaxy tablet in landscape mode. Adjust this statement to suit your device.

GRADING:

Mandatory attendance at the November 19 class in the planetarium is worth 10% of the project as documented in bullet B on the first page. See other bonus point opportunities on page 1.

Each of steps 3, 4, 5, 8, 9, 10, 11, and 12, starting on page 2, is worth 11.25% of the project.

You will need to **zip your CSC220F19Androidassn4Client into a standard zip archive and turn that in via D2L** in order for me to get all of your .pde files from the editor tabs. Use the zipping instructions here to create a zip file. "Here is how to compress your sketch folder into a .zip file (NOT 7z) on our Windows network. Right click the sketch folder.":
http://faculty.kutztown.edu/parson/spring2018/CSC120Spring2018.html
Mac and Linux have command line "zip -r CSC220F19Androidassn4Client.zip CSC220F19Androidassn4Client" command line zippers, as well as zip utilities available from their command GUIs.
You can use the 7z utility on Windows, but you must choose the "Add to CSC220F19Androidassn4Client.zip" option to create a standard .zip file archive of the sketch folder. If you turn in a .7z or other compressed format that requires me to go fishing for an extractor, I will deduct points. You can substitute your sketch directory name for CSC220F19Androidassn4Client, but this must be in a standard .zip format. Turn that .zip file in via D2L by the deadline.