Section 4

In this section I made two main classes, `Server` and `Client,` and another class, `ServerAccept,` that simply accepts new client connections and adds the client input and output streams to two a static data structures on the server. Since the call to `Server.accept()` is synchronous and blocking I put it in a new thread to listen while the main Server class did the work to communicate with the clients. Otherwise the server couldn't read or write until it found a new connection. I chose the thread safe `CopyOnWriteArrayList` data structure because the array list is simple to work with for small amounts of elements, easy to loop over, and doesn't require manual use of synchronization blocks.

The main job of `Server` is to never stop looking to read from clients until it is manually killed. The while loop will continuously read from the various client input streams, in a for loop, it got from `ServerAccept`. Putting the IO in the while loop allows the server to smoothly do IO while clients connect asynchronously. Also, looping this way and using the concurrent arraylist will allow future implementations to disconnect from clients and update the IO stream data structure when the client tells the server to disconnect.

For future requirements I started to use `ObjectInputStream` and `ObjectOutputStream` because you can simply pass serializable Objects through these instead of making commitments to just passing strings, or whatever, through the read buffers and print buffers. Thus promoting a more generic design. I didn't get to pass in more complicated things other than `ints` but initializing these streams correctly and seeing how you check for empty streams will be a good start for future requirements.

To comment on this is a game engine subsystem: in the future I anticipate making the server a singleton class and so start it before the main game loop runs, or when during the run of the game a it's needed. But for this assignment having another class to instantiate the server wasn't necessary.

The client instantiates a tcp socket and attempts to connect to the server socket over the same port. It writes to the server using an object stream. Then it reads in a while loop for server responses until the client times out after not reading anything for a few seconds, I hardcoded the timeout to 5 seconds. I also used `Thread.Sleep()` to keep the thread from busying waiting for too long and to count the timeout. Reading in the while loop until timeout allows the client to read any amount from the server without prior knowledge of what it's going to get which enables flexible communication for future requirements.

The communication between server and client are just some simple `println`s of strings and ints. The server prints what message the client sent along with the index the client stream is in the stream data structure and sends the message back to the client with a modification. Then the client prints the modified message from the server to it's own output console. Currently I'm reading and writing with prior knowledge of the data type but in the future I will use the object streams to simply write objects to promote the generality of a game engine.

Section 1

In the interest of just “get something working” the code in this section is fairly messy. The animation consists of a small rectangle, `agent`, moving horizontally across another rectangle that fills the bottom, like a platformer. For the objects in the animation I’m keeping an array of floats to represent their positions and dimensions. However, `agent` has a few other properties associated with it like it’s jumping angle and starting positions that are stored in the single main class. If future requirements make it useful this information may have to be moved to it’s own class or object. Because right now the only cohesion between these properties is happening in my head, programmatically there’s nothing to associate them with each other. I tried to extend PShape to promote some modularity but ran into some odd compilation issues. However, I know realize I could have just used aggregation with another class and PShape, I’ll probably do this in the next assignment. Or put this in a game object in the world object model.

My collision detection algorithms were taken from github.com/jeffThompson/CollisionDetection. I think I like using this library better because they are more flexible than going through method of translating Processing shape information into java.awt rectangles even if some of the object weren’t rectangles, like a boundary line. This is another component that’s still coupled to the main big class. Again, in the future I may put this logic in it’s own component for modularity, but for now it just works.

Section 2

For the first modification: I demonstrated a way to use the same data structure for two threads. The “writer” thread writes a few thing to the queue, with intermittent sleeps, and then notifies the other thread to quit by writing -1. The “reader” thread prints the content of the queue and waits for other messages until it reads -1.

For the second modification: I demonstrated how to change the flow of control to basically “finish” where the program decides it’s done processing. Instead of printing “done” within a created thread I used `Thread.join()`, a blocking synchronous call, to block in the main thread and print “done” there.

For the third modification: I demonstrated this same functionality could be achieved with Semaphores. In fact using Semaphores decreased the amount of lines written and didn’t require synchronization blocks. Although, while this decreases code it might introduce a little more complexity as you have to keep track of the semaphore count and you have to remember to release a semaphore while using a synchronized block there’s no burden to remember this.

For the fourth modification: I demonstrated a way to switch which thread uses the notifies and waits. This also requires to change which thread the sleeps occur in. As both threads at some point need a blocking call. Ane thread use the `.wait()` blocking call and the other uses the `Thread.sleep()` blocking call.

Section 3

For this part I have a `Server` and `Client` class. Since the requirements say the server only has to read I have the client writing to the server. Since this is a synchronous server I have it calling `Server.accept()` three times in succession before it starts reading. The Client writes a few integers to the server and then the server prints to the console the content of the client's messages. I didn’t put much thought into this section because I knew the next section is what would continue to the next assignment.

There’s no appendix because there’s nothing interesting to see that would increase clarity, at least from what I can tell

Now that you have designed and implemented a number of engine components, write a 2–3 page paper **summarizing your design**. That is a minimum of 2 FULL single-spaced pages. Think creatively about what you have done. **Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? What didn’t work out the way you had hoped, and why**? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have **gone beyond what is written in the assignment**. As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 2–3 page requirement. Your points for the writeup will represent 1/2 of the points allocation for each of the above sections (i.e., 25 of the 50 points for the "Putting it all Together" section will come from your writeup).

https://github.com/cblupodev/csc481hw1
https://moodle-courses1617.wolfware.ncsu.edu/mod/resource/view.php?id=203231