**Final Project Documentation**

**Data Structures:**

*D – ArrayList*

The data structures I chose to use in my Process Scheduling Final Project included Java’s ArrayList and PriorityQueue. I used an ArrayList, D, to store Process instances that were read into my program from the “process\_scheduling\_input.txt” file. I used an ArrayList to store the Process instances because this data structure made it easy to add and remove Process instances with the .add() and .remove() methods, respectively. It should be noted that the .remove() method removes the first element from an ArrayList which worked for my program because the Process Objects needed to be removed from D in order of their arrival time which was the given order of the processes in the text file.

*Q – PriorityQueue*

I used Java’s PriorityQueue data structure, Q, to store the Process instances that were removed from the ArrayList, D, in. The PriorityQueue, Q, represents an execution queue where processes arrive to be executed or run. This data structure was useful in this application because the decision to run a Process instance in the PriorityQueue, Q, was based on the priority of a Process instance. A Process instance waiting in Q with the highest priority, or lowest priority number, would be chosen for execution before the other processes. I was able to define a comparator to be used by the PriorityQueue, Q, to sort Process instances based on their priority number during operations. The PriorityQueue data structure implements Java’s Iterable interface which I used to update the priority and wait time of Process instances.

**Observations:**

After completing this project, I tested different sets of output on my code to see how it would handle different situations. One observation I made was that Java’s PriorityQueue data structure arbitrarily chooses an element when there are two equivalent values available. When Process instances in my program had the same priority number, Java’s PriorityQueue arbitrarily removed one of these Process instances. This is an interesting feature of Java’s PriorityQueue and should be taken into consideration when writing program with this data structure or unwanted results could occur.

I also realized that I could have implemented this program with only one while loop that kept running until both the ArrayList, D, and PrioirityQueue, Q, were empty. This method would have cut down on the number of lines of code, but was a bit more difficult to implement from my perspective as keeping track of local and global variables proved to be difficult. I implemented the code with two while loops; one that executed until D was empty and another that executed until Q was empty. I was able to reuse code from the first while loop in the second while loop which made this implementation quite simple.

**What I Learned:**

I feel like I learned a lot from this project. Prior to this project, I had a very basic understanding of comparators and did not know how to implement a comparator. As I worked on this project, I eventually came to understand comparators and was able to create a user-defined comparator class for the PriorityQueue, Q.

I also learned more about the Iterable interface. I was able to use an iterator to iterate through the PriorityQueue, Q, to update priorities and wait times of Process instances. This was exciting because I was able to iterate through Q using something other than a for loop in as few of lines.

Lastly, I learned that is important to take a break from coding when you are stuck on an issue for a while. I was stuck on an issue for two hours and felt like I was experiencing tunnel vision, so I decided to take a break. When I came back with fresh eyes, I was able to identify and fix the problem in twenty minutes.