Part 1 (FCFS - Tobias Solvang Nesvik):

How it works:
The first come first serve algorithm is a simple stack algorithm where the first to arrive is the first to be processed. I sorted it based on arrival time to ensure that the process with the lowest arrival time gets to be first. I use a scanner to take inputs that I then place in arrays for each relevant variable, being processID, arrivalTime and BurstTime. These are then received in a for-loop that gets each input and places it in their respective arrays. I then have everything I need to sort the processes before calculation, so I make a nested loop that compares processes and sorts based on arrival time. After that I make another loop to determine completionTime, turnaroundTime and waitingTime. The completion time is the total time it took for the specific process to finish since the algorithm was run so it is calculated by adding together the current time it starts at and the burst time. The turnaround time is the total time from when a process arrives till it completes and is subsequently calculated by taking the completion time minus the arrival time. Lastly is the waiting time, which is essentially the wait between the last process completing and the new one arriving, and is calculated by taking the turnaround time and subtracting the burst time. Then I just do a simple average calculation of turnaround and waiting time by adding all their values together and dividing by the amount of processes requested. Finally I print all values except the current time because it is for temporary use.

How to test:
I used Java to code up the algorithm, you can find it in the FCFS Java class in the project repository and run it. When you run it you are prompted to enter in how many processes you would like to input, after that you manually input the arrival and burst time for said processes and then the rest will be calculated and printed out.

What I learned:
I learned a lot about the inner workings of the algorithm, about what burst time, arrival time, waiting time and turnaround time was in scheduling etc. I also got good practice using Java.

Part 2 (Pre-emptive Priority Scheduling - Nikolai Mork Aambø):

How it works:
The Pre-Emptive Priority Scheduling algorithm, shortened to PEPS, is similar in that it is to start with also first come first serve. The main difference is tasks are given priorities sorted from low number and high priority to high number and low priority. When a task with a higher priority appears, the lower priority has to save its progress and be put back into the queue to wait until there are no tasks with higher priority.

Using objects from a class called Process to act as tasks for the CPU to do. The Process object stores information as id, time of arrival, priority, time of completion, burst time, and remaining burst time for when the process has been interrupted. The main code uses PriorityQueue, to keep all the processes ordered in time of arrival, and generic lists to keep order of what processes are processed, have arrived for processing, and are underway for processing.

How to test:
The PEPS algorithm is written in java just like the FCFS algorithm, contained in the same maven project.

Two jar files should be in the folder ‘target’. One named FCFS.jar which runs the FCFS algorithm, and the other name PEPS.jar which runs the PEPS algorithm. Either jar files can be run on windows in the command prompt by using the command ‘java -jar <directory/file-name>.jar’. Two batch files have been provided, and when run, will each run their respective jar file without the need to open the command prompt.

When you run it you are prompted to enter in how many processes you would like to input, after that you manually input the arrival and burst time for said processes and then the rest will be calculated and printed out.

What I learned:
Causing interruptions at the right time, whilst retaining both wanted information and correct behavior within the application, is very difficult.