SENG 2200 - Assignment 3: Written Report

Harrison Rebesco C3237487

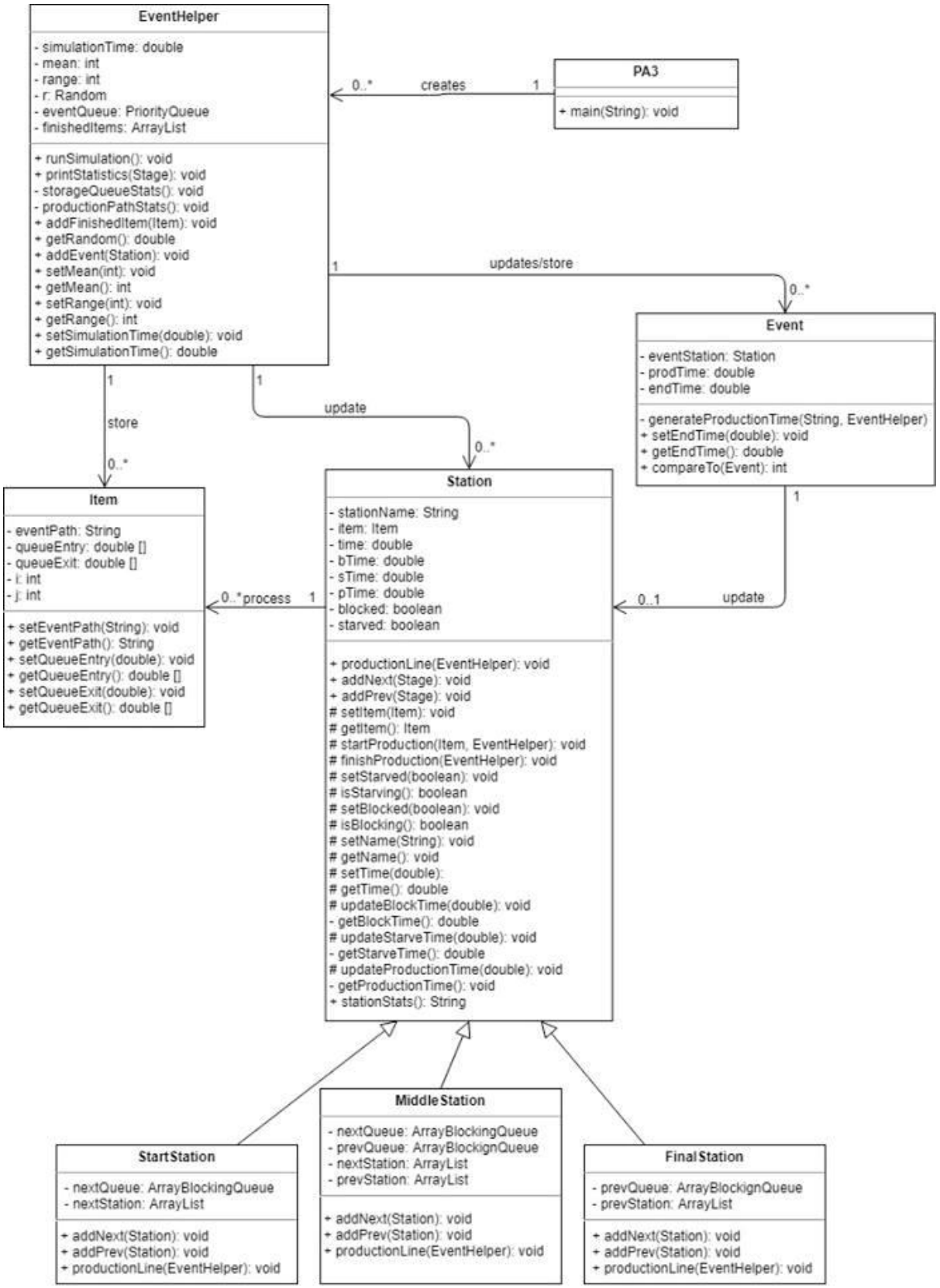
I. *For each of the programs, keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct.*

For this assignment I spent 21 hrs on design and research, and 13 hrs coding and debugging. Which is 62% and 38% respectively. I found it very hard to get the logic correct, however the implementation wasn't too bad once I had a good design figured out.

NAME OF CLASS	**TIME SPENT**	**ERRORS & SUMMARY**
Station	***Design/Research:*** 5 hrs ***Coding/Debugging:*** 2 hrs ***Total Time:*** 7 hrs	***Errors:*** - basic syntax errors ***Summary:*** Didn't run into any real problems with code implementation, the hardest part was thinking of a coherent way to make station work and deciding on Data Structure imports to use.
StartStation	***Design/Research:*** 2 hrs ***Coding/Debugging:*** 1 hr ***Total Time:*** 3 hrs	***Errors:*** - basic syntax errors - null pointer exception ***Summary:*** This class is just an extension of the class Station, so I didn't require as much planning. Implementation was simple once I had a good idea of what I was supposed to be doing.
MiddleStation	***Design/Research:*** 2 hrs ***Coding/Debugging:*** 1 hr ***Total Time:*** 3 hrs	***Errors:*** - basic syntax errors - null pointer exception ***Summary:*** This class is just an extension of the class Station, so I didn't require as much planning. Implementation was simple once I had a good idea of what I was supposed to be doing.
FinalStation	***Design/Research:*** 2 hrs ***Coding/Debugging:*** 1 hr ***Total Time:*** 3 hrs	***Errors:*** - basic syntax errors - null pointer exception ***Summary:*** This class is just an extension of the class Station, so I didn't require as much planning. Implementation was simple once I had a good idea of what I was supposed to be doing.

Event	***Design/Research:*** 4 hrs ***Coding/Debugging:*** 2 hrs ***Total Time:*** 6 hrs	***Errors:*** - basic syntax errors - null pointer exception ***Summary:*** When designing event, I realised I would need a helper method to assist with tracking the time, mean, range, etc. this helped a lot with all my other classes and saved a lot of time in the end. I didn't run into much implementation issues, just designing.
EventHelper	***Design/Research:*** 4 hrs ***Coding/Debugging:*** 5 hrs ***Total Time:*** 9 hrs	***Errors:*** - basic syntax errors - null pointer exception - endless looping ***Summary:*** I decided to implement this class when creating event, so some of the designing was already done. However, I decided to make this a helper method for the whole program, this class displays statistics, and runs the event simulation. Because of all the class interactions I had a fair bit of trouble making it work well with other classes, which impacted coding/debugging time
PA3	***Design/Research*** 0 hrs ***Coding/Debugging*** 0.5 hrs ***Total Time:*** 0.5 hrs	***Errors:*** - basic syntax errors ***Summary:*** PA3 originally had a lot of the functionality that EventHelper ended up getting, so it has become more of a driver class that just initializes Stations and Queues, then runs the program.
Item	***Design/Research*** 2 hrs ***Coding/Debugging*** 0.5 hrs ***Total Time:*** 2.5 hrs	***Errors:*** - basic syntax errors ***Summary:*** this class for the most part was straight forward, however coming up with a way to track the queue entry/exit times turned out to be a big challenge, most of the design and implementation time was spent on figuring out a way to do this effectively.

II. *Produce a UML class diagram that shows the classes (and interfaces) in your program and the relationship(s) between them.*



III. *Comment on your use of Inheritance and Polymorphism and how you arrived at the particular Inheritance/Polymorphic relationships you used in your program.*

For this assignment I have used Inheritance and Polymorphism to make the different stations of the production line work without having to rewrite massive amounts of code. The Station class has child classes StartStation, MiddleStation, and FinalStation - these subclasses inherit all the functionality of Station, allowing them to use all of the attributes and methods provided by the parent class Station. Polymorphism is also utilized by adapting the implementation of productionLine(), addNext() and addPrev(), for instance StartStation can never be starving, and will not have a previous station, so using polymorphism, I have overridden the methods to prevent the addition of a previous station, and made productionLine() unable to be starved. The same has been done for MiddleStation (has a next and previous station, can be blocked and starved) and FinalStation (only has a previous station, can't be blocked). By using these techniques, I saved many hours of coding by just inheriting and adapting code.

IV. *How easy will it be to alter your program to cater for a production line with a different topology – e.g. one with 4 stations or 10 stations, or one that has stations 2 a/b/c rather than just 2 a/b?*

The way my stations are set up (with StartStation, MiddleStation, and FinalStation) it would be very easy to add more stations – I could have as many or as little MiddleStations as I want without affecting the functionality or flow of the program, would just have to alter how I declare them in my driver class

Adding a 3rd parallel station (a/b/c) would require a little more work, but would only alter MiddleStation, I would simple allow a third prev/next Station, and would need to put a check in there to see if it's a 2 or 3 station parallel to avoid null pointer exception (might not even have to do this).

V. *How easy will it be to alter your program to cater for a production line that is more complicated than the "straight line" item processing that your program does – e.g. one that involves taking two different types of items and assembling them to make a new type of item? Would you design your program differently if you had known that this might be a possibility? E.g. the following production line?*

To adapt my program to facilitate the production line shown in assignment specs, I would consider adding a 4th Stage, (possibly called AssemblyStage) that has two storage queues feeding into it. Seeing as it requires two parts, I would implement a ItemA and ItemB classes, and also a blockItemA/blockItemB, starveItemA/starveItemB system to accept/reject the correct queues while waiting on parts. I think this would be the easiest way to implement such a system, and once that AssemblyStage is working I don't think it would be much of an issue adding it to the current system I have created.