The Written Report:

Your Written Report is worth 20% of the assignment’s value, and will address each of the following:

# 1) Keep and present records of how much time you spent on designing, reviewing, coding, testing, and correcting (within these various phases of) your assignment development.

Designing

3 hours 15 minutes

Reviewing

45 minutes

Coding

13 hours 30 minutes

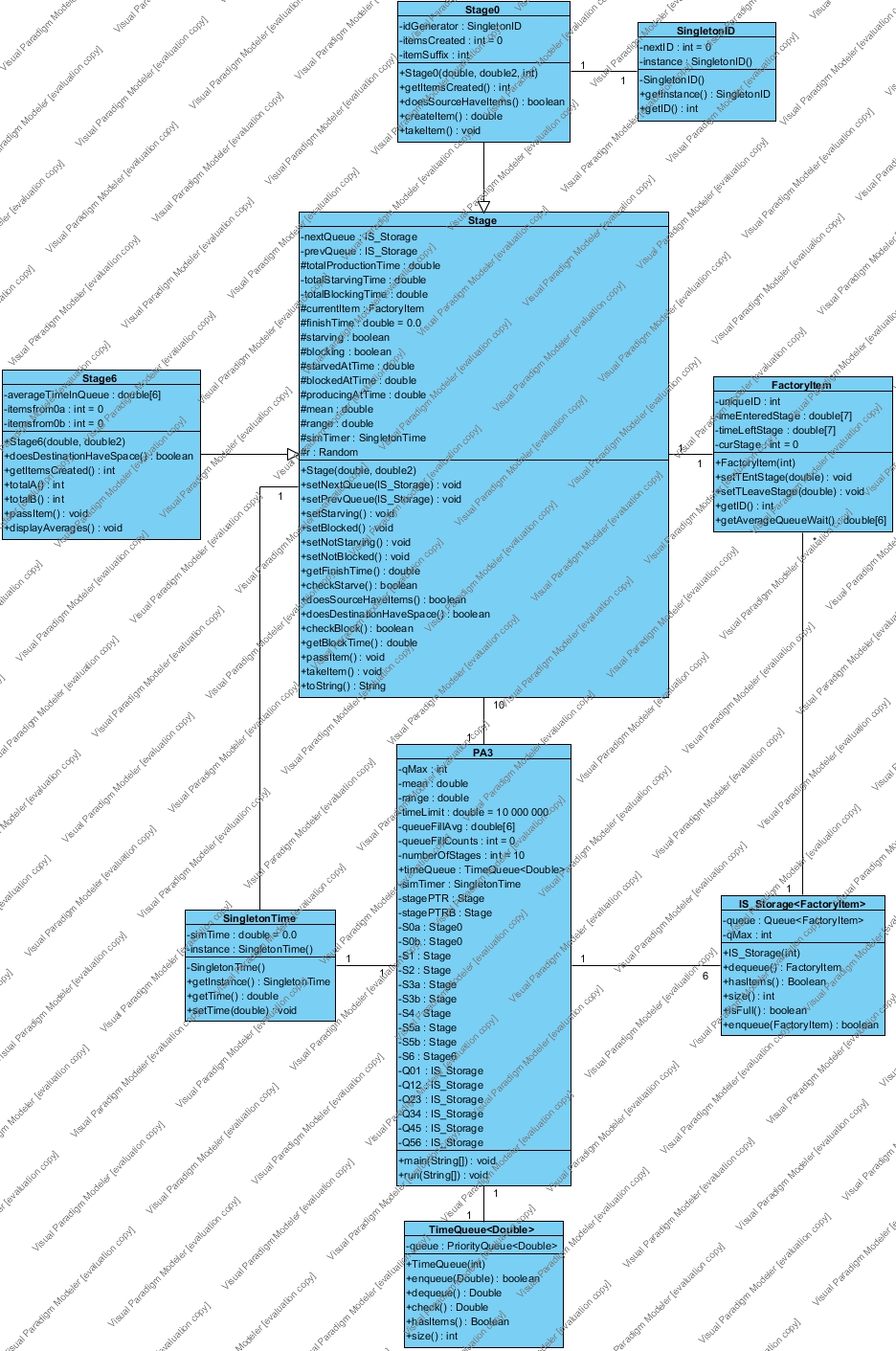
Testing

2 hours

Correcting

2 hours 30 minutes

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



Please Note: a full-size copy of this image is included in the base directory as Class Diagram.jpg

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

I decided to use inheritance as extending a base Stage class into a specified Stage0 and Stage6 class seemed to be the ideal relationship for my program. Extending the functionality of my base class seemed much more straight-forward and efficient for me, rather than implementing an interface, and then making 3 classes that implement that interface. By using inheritance, I can still use all the same functions, and overwrite their default methods, but I can also implement my base stage class, rather than having an interface, and then yet another class having to implement that interface.

# 4) 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 3 a/b/c rather than just 3 a/b?

I believe that my program is very easy to modify for different sized production lines. Currently, to increase the number of stations processed, all one needs to do is increase the number\_of\_stations variable, initialise the new stations, add them into the queue linking process, slightly modify my loops that loop through all stations, and finally, add some more if statements to my blockchains that assign a pointer to a station in my PA3.java file.

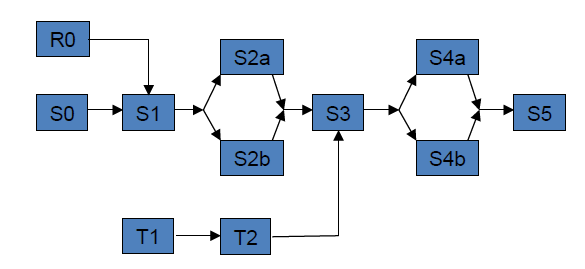
However, adapting my program to have additional substations is not as easy as simply adding additional stations and would quickly ramp up in complexity. The only real changes that would occur in creating a 3rd (or more) substation, would be to create and initialise a third pointer, let’s say stagePTRC (then D etc.), add the third station to our loops and if-then(pointer select) chains (very similar to adding an extra station), and then finally re-write our “check blocked” loop that tests all stations for being blocked, and then unblocks in order of blocking. This final section is where it gets a little tricky.

As it is currently, the code checks to see if both stations are blocked, and then proceeds to unblock the first station, then checks each station individually to see which one is still blocked, before unblocking it. Adding a third substation would cause it to have to check for 2+ of the stages Substations being blocked, then in a worst-case scenario, if all stages were blocked, check which of the three were blocked first, then check which of the two were blocked second, and so on.

With how I have written that section of code, increasing the number of substations would start creating larger and larger if-chains that would quickly become unfeasible, requiring a re-write of the section. I’m very happy with my solution, but I’m aware that the accessibility of adding additional substations is a definitive flaw in it.

# 5) 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?*



(*You will probably need several pages to do this properly – you will also need a significant amount of time to think through and answer the later questions properly*.)

I believe it would be extremely easy to modify my program to work like this. All I would have to do is write a new class extending my default stage class, that overwrites the default “starvation check” function and the “take item” function. The starvation check would simply say that the station was starved until both pre-ceding queues were ready, and then takeItem() would simultaneously take an item out of both queues and transform them into a new Item.

I would of course also have to increase the number\_of\_stations variable and introduce the new stations into my loops and pointer selections, so that they were successfully checked for production finishing, starvation and blocking, but it would hopefully take very little time to expand the system to hold these extra stations, and could be increased to easily work from any number of input queues to a single station.

Unlike adding more substations, changing the topology of the entire line is very accessible, even at higher levels of complexity. Had I known that the above could have been a possibility, I don’t believe I would’ve made any major changes to how I’ve laid out my program, because as it is – it is very easy to modify to add other stations. Although I may have taken some more time to consider implementing an interface, rather than using Inheritance, but I’m confident I still would have used Inheritance, for the same reasons I addressed earlier.