Assessment 3 Report

Development on this assessment can be roughly divided into the following areas:

* drawing the design: I took 1 hour to clearly and repeatedly read the specifications. Once reasonably certain that all parts were understood, I spent 30 minutes drawing a pseudo-UML diagram of the classes, their components and how they link to each other (a more complete version will be presented in section 2.) Throughout the code I revised this diagram to account for unconsidered variables – cumulatively taking roughly 20 more minutes.
* building the classes: Using the design, I built the classes – the initial **Startup**; Item to be created and processed through the production line; **ProductionStage** to process and manage the Items; and **StorageQueue** to store the Items between Stages. I initially created a **ProductionLine** class to represent the whole process but decided that it was redundant, moving its functionality into Startup.

I made the getters and setters, and some obvious functions. In addition, I extended the **Object** class for the **Item** class, and the **PriorityQueue** class for the **StorageQueue** class – although in hindsight, extending the simple **Queue** class would have worked for my program. In this initial step, building the classes took 15 minutes and reviewing/fine-tuning them took another 35.

* basic functionality: I decided to create a simple production line first – 7 Stages and a StorageQueue between each, plus 1 Queue at the start and 1 at the end, for 8 in total. Eventually it became clear that the first and last Queues were useless – the first was redundant and the last, which stored ALL of the finished items, was a waste of memory with functionality that could be condensed to a 4-size integer array (prodPaths).   
  After spending 5 minutes writing the console output code for the production stages (neglecting the queue output here cost me a lot of time), I spent 3 ½ hours on coding, testing and correcting the flow of Items through the line. By this, I mean figuring out what was causing the millionth NullReferenceException. In total, this stage took roughly 4 hours.
* splitting the paths: The next step was to split the path at stages 2 and 4. I had committed to making the code flexible – the developer could change the number of stages by changing a single variable at the beginning of the code, and the stages would split and merge in the pattern established by the specifications: 1, 2, 1, 2, 1, etc. In doing so, they could have any number of stages. Additionally, every split stage would have the required 2000 units of work time. Unfortunately, this also took another 1 ½ hours to get working.  
  Ensuring that each stage collected from and deposited to the correct queue was the next, surprisingly simple step. This was a simple for-loop and an implementation of the LinkedList concept – each stage would have two **int** values that denoted the position of the appropriate queues. This took 15 minutes.
* finalising the program: The program was basically finished, with only the output left to complete. After taking 5 minutes to format the output,   
  At this point, it was obvious that I had made the following errors throughout development – I had designed the program for adaptability, which made certain parts of the output more difficult; and more seriously, I had neglected the Queue output as an afterthought, which threatened to require some major code restructuring to fix. It took another 3 hours of development to implement the Queue output, before I decided on a simple but clumsy solution – calculate the time each item spent in a queue when it left, add that figure to a total, count each item as they left, and divide the two variables.  
  But there was still more to do. The output was… not right. Everything was passing through the queue seemingly too well and nothing was blocking. Some negative times were being recorded. Eventually I discovered that, due to an oversight, the production time was essentially nil.   
  I spent roughly 5 hours fine-tuning each part of the Stage output until it was somewhat expected – high peaks of starvation before the split paths and smaller peaks of blocking afterwards, no starvation in the first stage and no blocking in the last stage.   
  I spent a further 2 hours on the Queue output as the averages were far too random.  
  In total I spent close to 10 hours finalising the program.

1. Polymorphism in this program is restricted to extending classes in the standard Java library. I designed Item as a subclass of the generic Object to take advantage of its versatility, and StorageQueue as a subclass of PriorityQueue since the useful add() and poll() methods were included.
2. The program was designed with this sort of flexibility in mind. The developer can change the ProductStage[8] variable to any ProductStage[i+1] and the corresponding for-loop will output the desired amount of stages, properly split.   
   They will need to update the StorageQueue[6] to their required StorageQueue[i+1] and alter the output of the Production Lines – the only supported Production Lines at the moment are (2a -> 4a); (2a -> 4b); (2b -> 4a); and (2b -> 4a).  
   Altering the production line so that it splits into 3 or more stations would similarly be relatively easy – it would only require expanding the for-loop in Startup (line 37-39). Depending on whether the developer would like to end the line on a split station or a single station, this may become more complicated – ending on a single station would be much easier.
3. The program would need to undergo a significant overhaul to accommodate that functionality. It

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

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?