SENG2200: Assignment 3

**A3: Discrete Event Simulation**

*QiTao Weng*

**Student ID: C3352830**

*Date submitted: 26/06/2020*

1. **Introduction and Background**

Objective: Practice on OO design and Java programming

Problem Statement: Use Discrete Event Simulation to stimulate a production line

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

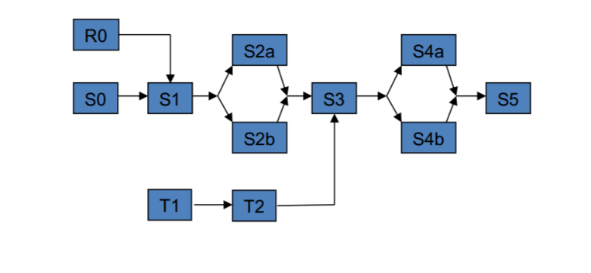
Attached below

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

I started on creating classes that I knew were necessary. So, the first classes were Item, UniqueID, and Storage. An Event class was needed for the PriorityQueue used to keep track of changes on the simulation. I added minimum required member variables and methods. I initially had the wrong idea of the production Stages; first I created the Production class and assumed that the class would act as the Stage and the Event moving along the line, but it could not be inserted into the PriorityQueue with a time in this manner, then I thought that the items would move between Storages by the events and deleted the Production class, but that was an issue as blocking and starving wasn’t possible. I ended up using Events to facilitate moving items between the Stages and Storages; however, at this point, I needed to consider how the different Stages were going to interact with the Storage. The Storage cannot be the class that facilitates movement, as there were parallel Stages, so the Stages must do so. If the Stages facilitate movement, they must consider from where the item is taken and to where it is delivered. There needed to be different cases for the beginning, ending, and intermediate Stages. Thus I created 3 cases of Stage classes with an abstract Parent class. This was not entirely successful as I was overzealous in adding generics to the classes which caused an issue as CreateStage cannot instantiate a generic type. Thus, abstract Stage can only be used to force the implementation of required methods for its children. Removing the generics may allow all the child Stages to be treated as the Stage class, but my implementation had numerous flaws and it seems that having CreateStage, InterStage, and FinishStage, as Stages will not change how it operates.

1. 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 S3a/b/c rather than just S3a/b?

This should be relatively simple. It would require creating more Stage or Storage instances and adding in the conditionals in the Factory class. If the generics were removed as per the previous response and I found a way to treat the different Stages as the Stage class effectively in the Factory, then the conditionals can be made into a function, which may eliminate the need for adding it directly to the Factory.

1. 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?

It seems it would be quite challenging. The good thing in this case is that since my program includes generics, it can handle moving more than one class type through the production line, thus allowing the addition of the assembled item as long as it follows the Item archetype. The stages would need to be redesigned for allowing the specification for the items required to assemble the new item and how it will treat waiting for all prerequisite items to arrive.

