SENG2200 Programming Languages & Paradigms
School of Electrical Engineering and Computing
Semester 1, 2020

Assignment 3 (100 marks, 15%) - Due Jun 12, 23:59

1. Objectives

This assignment aims to do practices on OO design and Java programming. A successful completion should be able to demonstrate a solution of assignment tasks with correct Java implementation and report.

2. A Note

You will note that, for this assignment, you are not being given much of an indication of what the classes should be, where to use inheritance and polymorphism, and how to allocate responsibilities to your classes. You will need to spend some time thinking about how to design this program, and then doing actual design. There is at least one clean and efficient design for this program, and if you finish up having difficulties in working out which object should be doing a particular task, then it is possible that you need to go back to your design and check over its clarity and simplicity.

3. Problem Statement

This assignment at first glance may look like an application for concurrent programming, however, you are NOT to use it. Instead, you will use **Discrete Event Simulation** for this assignment.

A reference: https://en.wikipedia.org/wiki/Discrete-event_simulation

3.1 Notations

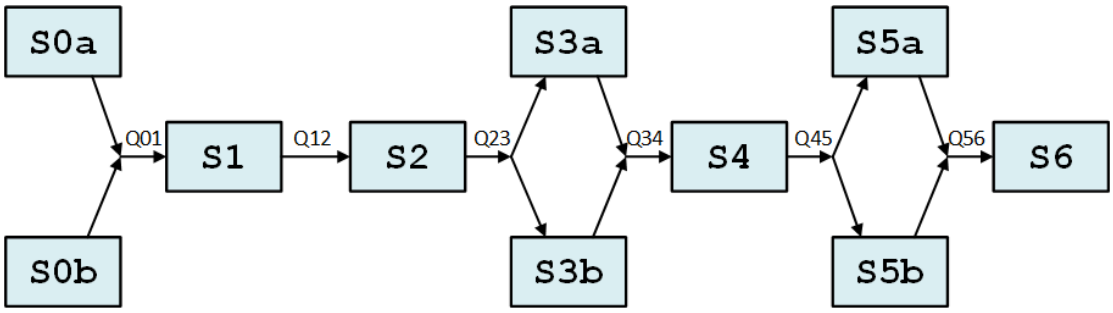
The table below shows a summary of notations that will be used in this assignment specification.

Parameter	Description
M	Average processing time of an item in a stage, given as a program input.
N	Range of processing time in a stage, given as a program input.
P	Processing time of an item
T1	Time of the starting of item production.
T2	Time of the completion of item production.
d	A random number in range 0 to 1.

Notations.

3.2 Production Line

For this assignment you will write a Java program to simulate the production of "*items*" on a production line. The production line consists of a number of production **stages**, separated by **inter-stage storage** in the form of queues of finite length (size *Qmax*). The inter-stage storages are necessary because the time taken to process an item at any production stage will vary due to random factors. The production line will be balanced in that the average time taken at any stage will be effectively equal.



The production line.

The production line can be seen from the above diagram that *S0a* and *S0b* are the beginning **stages**, while *S6* is the final stage. **Stages** *S3a/b* and *S5a/b* are *parallel stages* and share entry and exit storage queues. *Q01* is the **inter-stage storage** between *S0a/b* and *S1*, and *Q12* is the **inter-stage storage** between *S1* and *S2*, etc.

Production at any stage for this program will consist of

- taking an item from the preceding inter-stage storage;
- calculating how long it will take to process through this stage (using a random number generator);
- then after that amount of time, placing the item into the following inter-stage storage.

Production Line Parameters

- Stages *S0a*, *S3a/b*, and *S5a/b* will have mean and range valuese of *2M* and *2N*, respectively, while all other stages have the mean and range values of *M* and *N*, respectively.
- The inter-stage storage capacities (*Qmax*) will be *always* greater than 1.
- The simulation **time limit** will be 10,000,000 time units so that the production line will produce about 10000 items.

Note: you should not hardcode the values of M, N, or Qmax, as they are arguments of program execution, see Section 4.

3.3 Items

An item could be anything from cars to mousetraps. What is important in this simulation is the amount of idle time at each station, how long it takes a normal item to be produced, and what is the overall throughput of the production line.

For this assignment an item will simply be an object capable of storing simulation time values of the important events throughout its production, such as time entering, and leaving, each production stage, so that we can calculate things like the average time waiting in each queue, etc.

Item Creation

Item should be created in *S0a* and *S0b* stations. Each item should include a unique ID as a string, with the last letter being either 'A' or 'B'. The final digit will identify which start Stage produced the item ('A' for *S0a* and 'B' for *S0b*). For example, you may generate a unique string *x*, then the item's unique ID could be "*xA*" or "*xB*" dependeing on which stage (*S0a* or *S0b*) creates the item.

Bonus Task: you will earn TEN bonus marks if generate a unique ID by using a ***Singleton*** *class (using the Singleton Creational Design Pattern), with a* ***getID()*** *class that returns a unique string. Then, S0a and S0b will append 'A' or 'B' to this value, and store it as the item's unique ID.*

Note: the bonus marks will be given if and only if you correctly implement the ID generation process by using a Singleton class. The possible maximum mark of this assignment remains in 100.

3.4 Time

Time will be simulation time, as calculated below and will step forward according to which production stage finishes its current item next.

This will therefore be a simple discrete event simulation, which can be controlled by time events that are placed into a *priority queue* (You can use standard Java container, such as *PriorityQueue* and *ArrayList*).

The simulation time will start at **zero** and proceed until the **time limit** of the simulation. Time is probably best stored as a *double*, that way the chance of two time values being equal to within 13 significant figures is so remote that it may be ignored. *If two time values are however equal, you may simply assume first come, first served.*

How to calculate (simulate) a random processing time of an item?

Given mean *M* and range *N,* the processing time of an item is calculated as

$$P = M + N \times (d - 0.5).$$

In a stage, if an item processing starts at T1, then you will know that this stage completes production on this item at time

$$T2 = T1 + P.$$

Random number generation

For this simulation, production times are calculated according to a uniform probability distribution. This is easily catered by using the standard Java random number generator from **java.util**. After setting up a random number generator with

```
Random r = new Random();
```

Then, you can obtain the next pseudo-random number *d* (in the range 0 to 1) from `r` using

```
double d = r.nextDouble();
```

3.5 Blocking and Starving

If a stage finishes processing an item but finds that its destination storage is full, then that stage must block until its successor stage takes an item from the storage, thereby allowing it to re-submit its item to the storage and continue with its next item.

If a stage wishes to proceed to produce its next item but finds its predecessor storage empty, then it must starve, until its predecessor stage completes production and places that item into the intervening storage, for this stage to take it and continue.

The beginning stage(s) of a production line is considered to have an infinite supply of (raw) items and so can never starve. The final stage(s) are considered to have an infinite sized warehouse following, and so can never be blocked. The simulation begins with all stations (except the beginning one) in a starved state, and all the inter-stage storages empty.

3.6 Unblocking and Unstarving

When a stage completes production on an item, it must check whether stages **either** side of it, which are currently blocked or starved, may now be able to resume production. For example, if a stage is blocked it is because the storage following it is full, so if the stage after it starts production on a new item, it must have taken that item from the storage between these two stages, so there is now a free space for the earlier stage to place its item into the storage and so it can do so, and go ahead and try to begin production on its next item. Similarly, if a stage is starving and the stage before it has just completed processing an item, then the later stage can resume production by taking that item from the storage between the two stages.

3.7 Input and Output

1. The values of *M (e.g., 1000)*, *N (e.g., 1000)*, and *Qmax (e.g., 7)* are to be read in **from command line**, in form of (using example values):

   ```
   java PA3 1000 1000 7
   ```

2. All results are to be produced on **standard output**.
3. The program should **produce statistics** on the amount of time **each stage** spends in
   a. **actual production** (as a percentage)
   b. how much time is **spent starving**
   c. how much time is **spent blocked**.

4. For the **inter-stage storages**, calculate
    a. the **average time** an item spends in each queue
    b. the **average number** of items in the queue at any time (this statistic will require some thought).
5. You will also keep a **total number of items** created by *S0a* and *S0b* that arrive at the end of the line.
6. Lastly, you will keep a total of the number of items that followed **each path** through **Stage 3a/b** and **Stage 5a/b**, that arrive at the end of the line.

<u>An as example of how you could format your output:</u>
Note: These values are completely made up for format demonstration, and should not be considered as test values.

```
Production Stations:
--------------------------------------------
Stage:  Work[%]      Starve[t]      Block[t]
S0a      47.96%           0.00    103,601.39
S0b      48.65%           0.00    101,127.90
S1       99.87%       4,672.64     48,409.24
S2       98.87%      94,264.24     18,346.11
S3a      72.40%   2,236,949.88    522,533.24
S3b      63.02%   2,943,353.53    754,801.25
S4       98.61%     136,951.46      2,206.19
S5a      62.96%   2,957,251.09    746,649.33
S5b      72.32%   2,226,450.41    540,988.14
S6       98.82%     118,226.16          0.00

Storage Queues:
-----------------------------
Store   AvgTime[t]    AvgItems
Q01       3,674.47        3.12
Q12       3,734.88        3.11
Q23         515.32        0.40
Q34       4,085.28        3.39
Q45         414.71        0.29
Q56       4,011.52        3.31

Production Paths:
-----------------
S3a -> S5a:  2,296
S3a -> S5b:  1,321
S3b -> S5a:  3,973
S3b -> S5b:  2,272

Production Items:
-----------------
S0a:  2,296
S0b:  1,321
```

Obviously using random numbers will result in different values for you, but there is an expected range into which your results should fit; deviation from these expected ranges will result in a loss of marks.

You are not restricted to using this format, but whatever you choose to do must meet the criteria mentioned previously; the above example meets the specification.

3.8 Hints

- Spend time on designing your suite of classes, a good design will more than halve your workload. Design the stages and storages first. Look for similarities between the different stages and storages to decide how inheritance and polymorphism can be best used in your design.
- You can obtain a repeatable set of pseudo-random numbers from `r` by providing a particular seed value as a parameter to the construction of `r`, for example,

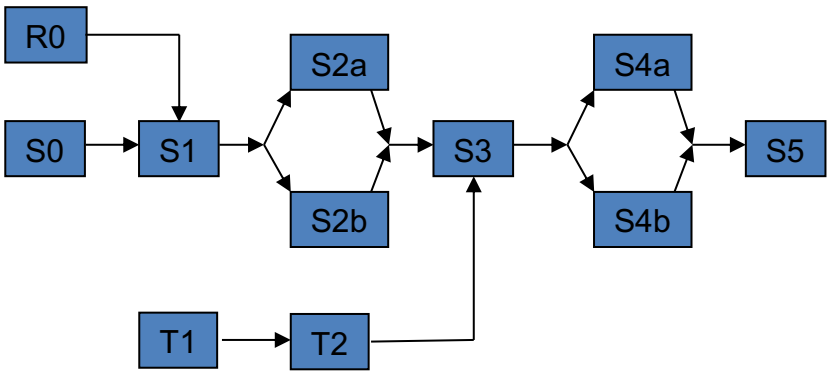
```
Random r = new Random(100);
```

  This allows you to run your simulation many times with the same sequence of random numbers by way of the r.nextDouble() method call given above.

3.9 Written Report

(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.)

1. Produce a UML class diagram that shows the classes (and interfaces) in your program and the relationship(s) between them.
2. Comment on your use of Inheritance and Polymorphism and how you arrived at the particular Inheritance/Polymorphic relationships you used in your program.
3. 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?
4. 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?



4. Submission

Submission is through the Assessment tab for SENG2200 on Blackboard under the entry Assignment 3. If you submit more than once then only the latest will be graded. Every submission should be ONE ZIP file (*not a .rar, or .7z, or etc*) named **`c9999999.zip`** (where c**`9999999`** is your student number) containing:

- Assessment item cover sheet.
- Report (PDF file): addresses the tasks in Section 3.9.
- Program source files.

Programming is in Java, and will be compiled against **Java 1.8**, as per the standard university lab environmen. Name your startup class PA3 (capital-P capital-A number-2), that is, the marker will compile your program with the command:

```
javac PA3.java
```

…and to run your program with the command (example):

```
java PA 1000 1000 7
```

…within a Command Prompt window.

Note that your program will have to handle different arguments for testing.

5. Marking Criteria

This is a draft guide only, and subject to change once marking begins.

Project Aspects:

Production Line Design and Implementation **40%**

Valid OO design for the Production Line, mapping the requirements to Objects, Objects to the required Structure and using OO design prociples; making valid use of Library Data Structures, etc.

Discrete Event Simulation Design and Implementation **25%**

Valid Discrete Event Simulation Design, using valid OO design principles, Patterns were applicable, making valid use of Library Data Structures, etc. Handling Simulation Events such as Block, Starve, etc, correctly.

Calculations and Statistics Output **20%**

Valid Statistics Calculations and Output formatting.

Report Task **15%**

Valid Responses to Report Questions, formatted as a Report.

Note: Your design should demonstrate the use of Inheritance and Polymorphism, Interfaces, Library Data Structures, and good Object Oriented Design Principles and Programming Practices.

Deductions:

Grouped on Feedback as *Other*, with the exception of *Late* and *Coversheet* issues:

- Does not use Discrete Event Simulation *up to* -50%
    - *instead the implementation is based on Java concurrency, or multi-threading, and/or works in real time or time slices*
- Does not compile *up to* -40%
    - *including does not compile as per specification, or does not compile due to errors*
- Crashes on execution *up to* -40%
    - *including does not execute as per specification, crashes with unhandled exceptions, infinite loop, hang, etc*
- Resultsvary beyond expected ranges *up to* -30%
    - *The program outputs statistics that are not present or outside of the expected range*
- Violate or Poor OO-design principles *up to* -30%
    - *such as no 'extends', use of concurrency instead of DES, etc*
- The program provides a menu/interface for user input. *up to* -20%
    - *there is no specification requirement for a menu or any user interaction beyond the command line*
- Code form and Commenting Standards *up to* -10%
    - *valid and sufficient commenting, consistent formatting*

** If your program cannot be compiled and/or executed (incl. run-time errors) by using the above commands, you may receive ZERO mark without being examined.*

** The mark for an assessment item submitted after the designated time on the due date, without an approved extension of time, will be reduced by 10% of the possible maximum mark for that assessment item for each day or part day that the assessment item is late. Note: this applies equally to week and weekend days.*

** The assignment (code and report) will be checked for plagiarism. A plagiarized assignment will receive a zero mark and will be reported to SACO.*

Nan and Dan
2020-04-25 v1.0