Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Spring 2018-2019

Homework 2 - Production Simulation

Due date: 05 05 2019, Sunday, 23:59

1 Objective

This assignment aims to familiarize you with the development of multi-threaded applications and synchronization using Java. Your task is to implement a simulator for production of simple manufacturing materials by simulating the different agents within the scenario using different threads. Towards this end, you will be using multithreading and concurrency classes that are present in java.

Keywords— Thread, Semaphore, Lock, Executor, Runnable, Callable

2 Problem Definition

We want to simulate the production of manufactured material from elemental ingots. Specifically, *iron* and *copper* ingots are processed to produce copper wires and iron plates.

The processing is handled by four types of agents whose functions are described as below:

- **Smelters** produce ingots at a certain rate and have two types: iron, and copper. A smelter has a limited capacity to store the ingots it produced. It sleeps when its storage gets full and wakes up, when ingots are taken. There exists a maximum number of ingots that a smelter can produce. Once a smelter reaches this number, it quits.

- **Constructors** produce copper wire or iron plates at a certain rate. A copper constructor uses 3 ingots to produce 1 roll of copper wire. Iron constructors uses 2 ingots to produce 1 iron plate. Each constructor has a limited storage capacity for incoming ingots. Constructors can work while ingots are being loaded into their storage. Constructors quit if it cannot produce its products for a given duration (due to not receiving new ingots).

- **Transporters** carry *ingots* from specific *smelters* to specific *constructors*. A transporter can carry one *ingot* at a time. The transporters travels to their target smelter, and can load ingots from them if the smelter has ingots in its storage. If the smelter does not have any ingots in storage, transporter waits on that smelter until ingot becomes available or smelter stops. The transporter then travel to its target constructor and unload the

ingots to its storage if there is available space in its storage. If the constructor does not have space, it should wait on that until space becomes available or the constructor quits. Transporters work until their smelter stop producing and has no ingots left on its storage or its constructor stops.

You shall implement the three types of agents as threads and synchronize their activities. The number of smelter, transporter, and constructor threads to be used will be given, and the threads will be created at the beginning of the simulation. Initially, all smelters will be empty and transporters will have to wait for ingots to be produced at the smelters. After creation, your main thread should wait for all the agents to finish before stopping. The pseudo-code of simulation agent threads are given below:

Algorithm 1: Smelter main routine

Data: ID, IngotType, Capacity, Interval, TotalIngot
`WriteOutput(`*SmelterID, 0, 0,* **SMELTER_CREATED**`)`
while *there are remaining ingots in the smelter* **do**
    `WaitCanProduce` ()
    `WriteOutput(`*SmelterID, 0, 0,* **SMELTER_STARTED**`)`
    Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for production
    `IngotProduced()`
    `WriteOutput(`*SmelterID, 0, 0,* **SMELTER_FINISHED**`)`
    Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for the next round
end
`SmelterStopped` ()
`WriteOutput(`*SmelterID, 0, 0,* **SMELTER_STOPPED**`)`

The functions are explained below:

- `WaitCanProduce:` Wait until a storage space is cleared by a transporter and reserve a storage space for the next ingot before production.
- `IngotProduced:` Informs available transporters that there is available ingots in the smelter's storage.
- `SmelterStopped:` Signals the transporters waiting on the smelter that this smelter has stopped producing. Transporters can keep loading remaining ingots in the storage. If there is not available storage for all waiting transporters, extra transporters quit.

Algorithm 2: Constructor main routine

Data: ID, IngotType, Capacity, Interval
`WriteOutput(`*0, 0, ConstructorID,* **CONSTRUCTOR_CREATED**`)`
while *True* **do**
    `WaitIngots()` or timeout after 3 seconds
    if *timeout* **then**
        break
    end
    `WriteOutput(`*0, 0, ConstructorID,* **CONSTRUCTOR_STARTED**`)`
    Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for production
    `ContructorProduced()`
    `WriteOutput(`*0, 0, ConstructorID,* **CONSTRUCTOR_FINISHED**`)`
end
`ContructorStopped()`
`WriteOutput(`*0, 0, ConstructorID,* **CONSTRUCTOR_STOPPED**`)`

The functions are explained below:

- `WaitIngots:` Wait until required ingots arrive at its storage and reserve the storage spaces until the production is finished. There should be 3 ingots for copper and 2 ingots for iron. If storage of constructor already have required ingots, thread will directly continue, otherwise it will block.
- `ContructorProduced:` Signals available transporters that storage spaces have been opened in this constructor.
- `ContructorStopped:` Marks the constructor out of simulation so that transporters who are delivering this constructor can quit.

Algorithm 3: Transporter main routine

Data: ID, Interval, Smelter, Constructor
`WriteOutput(`*0, TransporterID, 0,* **TRANSPORTER_CREATED**`)`
while *Smelter is active or has ingots in storage and Constructor is active* **do**
  Smelter = `WaitNextLoad()`
  `WriteOutput(`*SmelterID, TransporterInfo, 0,* **TRANSPORTER_TRAVEL**`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for travel
  `WriteOutput(`*SmelterID, TransporterID, 0,*
  TRANSPORTER_TAKE_INGOT`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for loading
  `Loaded(Smelter)`
  Constructor = `WaitConstructor()`
  `WriteOutput(`*0, TransporterID, ConstructorID* **TRANSPORTER_TRAVEL**`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for travel
  `WriteOutput(`*0, TransporterID, ConstructorID,*
  TRANSPORTER_DROP_INGOT`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for unloading
  `Unloaded(Constructor)`
end
`WriteOutput(`*0, TransporterID, 0,* **TRANSPORTER_STOPPED**`)`

The functions are explained below:

- `WaitNextLoad:` Wait if the smelter has no ingots in storage. Continue if there are available ingots to be transported. Reserves the storage space so that other transporters cannot take the ingots.
- `WaitConstructor:` Waits if the constructor has no storage space available. Continue if there is space storage for the ingot. Once unblocked, storage is reserved so that no other transporters can fill that storage space.
- `Loaded:` Signals the smelter to inform that new storage space is available.
- `Unloaded:` Signals the constructor to inform that new ingots have arrived so that if the constructor has reached the required number of ingots, it can continue production.

The simulation will be subjected to these constraints.

1. Initially, smelters, transporters, and constructors have no ingots in their storage.
2. The duration a smelter takes to produce an ingot and the idle time between each production is given as input.
3. Each smelter has limited capacity to store the ingots it produced. If it becomes full, it should wait for a space in its storage to be opened.
4. Each smelter has a limited amount of ingots it can produce. When that number is reached, it should quit.
5. Transporters can load a ingot from a smelter, while the smelter is producing ingots. Similarly, they can unload an ingot to a constructor while the constructor is in production. Smelter reserve a storage during production for its output. This storage is considered empty until smelting finishes. Similarly constructors keep their ingot input storage occupied until production of material (iron plate or copper wire) finishes.

6. No two transporters can load an ingot from a smelter or drop an ingot to a constructor at the same time.

7. Transporters take a certain amount of time to travel between smelter and constructor agents in the simulation. They also take certain amount of time when loading/unloading. This duration is given to you as input.

8. Transporters should quit if their smelter is inactive and has no ingots in its storage or their constructor is inactive.

9. If a transporter loads a copper ingot from a smelter, it can only carry it to a copper constructor. If it is a iron ingot, it can only carry it to a iron constructors. Non-matching smelter constructor pairs will not be given to you as input.

10. Transporters have no priority (between each other) when loading or unloading ingots.

11. The copper constructors require three ingots to produce a roll of wire. They should wait for ingots to be deposited without busy waiting.

12. The iron constructors require two ingots to produce a iron plate. They should wait for ingots to be deposited without busy waiting.

13. Constructors can produce their outputs while a transporter is unloading an ingot into their storage. Similarly a transporter can unload an ingot into its storage while the constructors are producing their outputs, as long as there is space in their input storage.

14. It takes a certain duration for the constructors to produce their materials. This duration is given to you as input.

15. A constructor quits if it can not produce its material (due to the lack of incoming ingots) for a certain duration. Note that, they may still have have left-over ingots in their input storage (e.g. one or two copper ingots in a copper constructor or a iron ingot in an iron constructor).

3 Implementation Specifications

1. Each agent should be implemented as separate thread. When a agent thread created, following function call should be made for every agent:

   - **Smelter:**
     `WriteOutput`(SmelterID, 0, 0, **Smelter_CREATED)**
   - **Transporter:**
     `WriteOutput`(0, TransporterID, 0, **TRANSPORTER_CREATED)**
   - **Constructor:**
     `WriteOutput`(0, 0, ConstructorID, **CONSTRUCTOR_CREATED)**

2. You need to call InitWriteOutput function before creating your threads.

3. You can use this code snippet to sleep in $[0.99 * time, 1.01 * time]$ milliseconds range:

```
Random random = new Random(System.currentTimeMillis());
DoubleStream stream;
stream = random.doubles(1, interval-interval*0.01, interval+interval*0.02);
Thread.sleep((long) stream.findFirst().getAsDouble());
```

4. Main thread should wait for every thread to finish before exiting. Each agent should make the following call before exiting:

   - **Smelter:**
     `WriteOutput`(SmelterID, 0, 0, **Smelter_STOPPED)**
   - **Transporter:**
     `WriteOutput`(0, TransporterID, 0, **TRANSPORTER_STOPPED)**
   - **Constructor:**
     `WriteOutput`(0, 0, ConstructorID, **CONSTRUCTOR_STOPPED)**

5. Simulator should use WriteOutput function to output information, and no other information should be printed.

4 Input Specifications

Information related to simulation agents will be given through standard input. First line will contain number of smelters (N_S) in the simulation. Following N_S lines contain the properties of the smelter with i^{th} ID (All IDs start from 1) in the following format:
- I_S C_S T_S R_S where

- I_S is an integer representing the production and wait interval of the smelter. It is given in milliseconds. The smelter will sleep this amount during and between production of each ingot with slight deviation.
- C_S is an integer representing the storage capacity of the smelter.
- T_S is an integer representing the ingotType of the smelter. Ingots have corresponding values:
  - **IRON:** 0
  - **COPPER:** 1
- R_S is an integer representing the total amount of ingot that can be produced from the smelter.

Next line contains the number of constructor (N_C) in the simulation. Following N_C lines contain the properties of the constructor with i^{th} ID in the following format:
- I_C C_C T_C

- I_C is an integer representing the production interval of the constructor. It is given in milliseconds. The constructor will sleep this amount during production of each ingot with slight deviation.
- C_C is an integer representing the storage capacity for ingots of the constructor.
- T_C is an integer representing the ingotType of the constructor. It can be IRON or COPPER with 0 and 1 values respectively.

Next line contains the number of transporters (N_T) in the simulation. Following N_T lines contain the properties of the transporters with i^{th} ID in the following format:
- I_T S_T C_T

- I_T is an integer representing the travel and load/unload time of the transporter. It is given in milliseconds. The transporter will sleep this amount during travel or load/unload operation slight deviation.
- S_T is an integer representing the target smelter ID that this transporter should load ingots from.
- C_T is an integer representing the target constructor ID that this transporter should unload ingots to.

5 Specifications

- Your code must be written in Java.
- You are allowed to use any of the java multithreading classes. Your solution should not employ busy wait.
- Submissions will be evaluated with black box technique with different inputs. Consequently, output order and format is important. Please make sure that calls to WriteOutput function done in the correct thread and correct step. Also, please do not modify `HW2Logger` class given to you as your submission for this file will be overwritten.
- Submissions will also be evaluated with white box to make sure you are not busy waiting and your code adheres to Object Oriented Programming principles.
- There will be penalty for busy waiting and non-compliance with OOP principles. Non terminating simulations will get zero from the corresponding input.
- Everything you submit should be your own work. Usage of binary source files and codes found on internet is strictly forbidden.
- Please follow the course page on ODTUClass for updates and clarifications.
- Please ask your questions related to homework through ODTUClass instead of emailing directly to teaching assistants.

6 Submission

Submission will be done via ODTUClass. Create a zip file named `hw2.zip` that contains all your java source code files without any directory. All your code should be under default package. Your code should be able to compile and run using this command sequence.

```
> javac *.java
> java Simulator
```

If there is a mistake in any of the steps mentioned above, you will lose 10 points.