Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Fall 2020-2021

Homework 2 - Production Simulation

Due date: 02 01 2022, 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 products 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, Condition Variable, Executor, Runnable, Callable

2 Problem Definition

We want to simulate the production of manufactured products from ores. There are three types of ores, namely *iron*, *copper* and *limestone*. Depending on the material, this production happens in one or two steps. Iron and cooper go through a two step process. First, *iron* and *copper* ores are smelted to *copper* and *iron* ingots. Second, *copper* and *iron* ingots are processed to produce *copper* sheets and *iron* rods. On the other hand, *limestone* ores are directly processed to produce concrete.

This simulation is handled by five types of agents whose functions are described as below:

- **Miners** produce ores at a certain rate and have three types: iron, copper, and limestone. A miner has a limited capacity to store the ores it mines. It sleeps when its storage gets full and wakes up, when ores are taken. There exists a maximum number of ores that a miner can mine. Once a miner reaches this number, it quits.

- **Smelters** produce ingots at a certain rate and have two types: iron, and copper. Copper smelters use 2 copper ores to produce 1 copper ingot. Iron smelters use 1 iron ore to produce 1 iron ingot. Each smelter has a limited storage capacity for incoming ores. This will be referred as incoming storage. Smelters can work while ores are being loaded into their incoming storage. A smelter has a limited capacity to store the ingots it produced. This will be referred as outgoing storage. It sleeps when its outgoing storage gets full and wakes up, when ingots are taken. Smelters quit when there are no more ores coming to their incoming storage and they have used all of the ores in their storage.

This can be determined by simple checking if transporters carrying ores to the smelter is still active (This will be clearer after Transporter section). **Note** that for copper smelters, there can be a single ore remaining in the incoming storage. It should still quit if there are no more incoming ores.

- **Constructors** produce products at a certain rate and have three types: iron, copper, and limestone. Copper constructors uses 1 ingot to produce 1 roll of copper sheet. Iron constructors uses 2 ingots to produce 1 iron rod. Limestone constructors uses 2 limestone ores to produce 1 concrete block. Each constructor has a limited storage capacity for incoming ingots. Constructors can work while ingots are being loaded into their storage. Constructors quit when there are no more ores or ingots coming to their storage and they have used all ores or ingots in their storage. This can be determined by simple checking if transporters carrying ores or ingots to the constructor is still active.

- **Transporters** carry cargo between two particular production agents. This is how ores or ingots are transferred. A transporter can carry one ore or ingot at a time. Each transporter has a single source and a single target. The source can be a Miner or a Smelter. The target can be a Smelter or a Constructor. The transporters travel to their source, and load 1 ore or ingot from them if the source has some in its storage. In smelters, this storage is outgoing storage and in miners there is only one type of storage. If the source does not have any in storage, the transporter waits on that source until one becomes available or the production agent stops. The transporters then travel to their target and unload their cargo to its storage if there is available space. In smelters, this storage is incoming storage and in constructors there is only one type of storage. If the target does not have space, it should wait until space becomes available. Transporters work until their source stops producing and there are no more ores or ingots in its storage. This can only happen after they unload their cargo to target and travel back to the source.

You shall implement the four types of agents as threads and synchronize their activities. The number of miner, 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 miners and smelters will be empty and transporters will have to wait for ores or ingots to be produced at their source. 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: Miner main routine

Data: ID, OreType, Capacity, Interval, TotalOre
`Log(`*MinerID, 0, 0, 0,* **MINER_CREATED**`)`
while *there are remaining ores to be mined* **do**
    `WaitCanProduce` ()
    `Log(`*MinerID, 0, 0, 0,* **MINER_STARTED**`)`
    Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for mining duration
    `Log(`*MinerID, 0, 0, 0,* **MINER_FINISHED**`)`
    `OreProduced()`
end
`MinerStopped` ()
`Log(`*MinerID, 0, 0, 0,* **MINER_STOPPED**`)`

The functions are explained below:

- `WaitCanProduce:` Wait until there is enough storage space and reserve a storage space for the next ore before production.
- `OreProduced:` Informs available transporters that there is available ore in the miner's storage.
- `MinerStopped:` Signals the transporters waiting on the miner that this miner has stopped producing. Transporters can keep loading remaining ores in the storage. If there is not enough ore for all of the waiting transporters, the extra ones should quit.

Algorithm 2: Smelter main routine

Data: ID, IngotType, Capacity, Interval
`Log(`*0, SmelterID, 0, 0,* **SMELTER_CREATED**`)`
while *there are active transporters (carrying ores to the smelter) and ores in the incoming storage* **do**
  `WaitCanProduce` ()
  `Log(`*0, SmelterID, 0, 0,* **SMELTER_STARTED**`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for production
  `Log(`*0, SmelterID, 0, 0,* **SMELTER_FINISHED**`)`
  `IngotProduced()`
end
`SmelterStopped` ()
`Log(`*0, SmelterID, 0, 0,* **SMELTER_STOPPED**`)`

The functions are explained below:

- `WaitCanProduce:` Wait until required ores arrive at the incoming storage and reserve the outgoing storage space until the production is finished. There should be 2 ores for copper and 1 ore for iron. If storage of constructor already have required ingots and enough space in the outgoing storage, thread will directly continue, otherwise it will block.

- `IngotProduced:` Informs waiting transporters that there is available ingots in the smelter's outgoing 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 enough ingot for all waiting transporters, the extra ones should quit.

Algorithm 3: Constructor main routine

Data: ID, Type, Capacity, Interval
`Log(`*0, 0, 0, ConstructorID,* **CONSTRUCTOR_CREATED**`)`
while *there are active transporters (carrying ores or ingots to the constructor) and ores in the incoming storage* **do**
  `WaitCanProduce()`
  `Log(`*0, 0, 0, ConstructorID,* **CONSTRUCTOR_STARTED**`)`
  Sleep a value in range of $Interval \pm (Interval \times 0.01)$ milliseconds for production
  `Log(`*0, 0, 0, ConstructorID,* **CONSTRUCTOR_FINISHED**`)`
  `ConstructorProduced()`
end
`Log(`*0, 0, 0, ConstructorID,* **CONSTRUCTOR_STOPPED**`)`

The functions are explained below:

- `WaitCanProduce:` Wait until required ingots or ores arrive at its storage and reserve the storage spaces until the production is finished. There should be 1 ingot for copper, 2 ingots for iron and 2 ores for limestone. If the storage of constructor have the required ingots or ores, thread will directly continue, otherwise it will block.

- `ConstructorProduced:` Signals available transporters that a storage space has been opened in this constructor.

Algorithm 4: Transporter main routine

```
Data: ID, Interval, Source, Target
Log(0, 0, TransporterID, 0, TRANSPORTER_CREATED)
if Source is a Miner then Log(MinerID, 0, TransporterInfo, 0,
 TRANSPORTER_GO)
else Log(0, SmelterID, TransporterInfo, 0, TRANSPORTER_GO)
while The source has ores or ingots in storage or is active do
    Sleep a value in range of Interval ± (Interval × 0.01) milliseconds for travel
    if Source is a Miner then Log(MinerID, 0, TransporterInfo, 0,
     TRANSPORTER_ARRIVE)
    else Log(0, SmelterID, TransporterInfo, 0, TRANSPORTER_ARRIVE)
    WaitNextLoad()
    if Source is a Miner then Log(MinerID, 0, TransporterID, 0,
     TRANSPORTER_TAKE)
    else Log(0, SmelterID, TransporterInfo, 0, TRANSPORTER_TAKE)
    Loaded(Source)
    if Source is a Miner then
        if Target is a Smelter then Log(MinerID, SmelterID, TransporterInfo, 0,
         TRANSPORTER_GO)
        else Log(MinerID, 0, TransporterInfo, ConstructorID,
         TRANSPORTER_GO)
    else
        Log(0, SmelterID, TransporterInfo, ConstructorID,
         TRANSPORTER_GO)
    end
    Sleep a value in range of Interval ± (Interval × 0.01) milliseconds for travel
    if Target is a Smelter then Log(0, SmelterID, TransporterInfo, 0,
     TRANSPORTER_ARRIVE)
    else Log(0, 0, TransporterInfo, ConstructorID, TRANSPORTER_ARRIVE)
    WaitUnload()
    if Target is a Smelter then Log(0, SmelterID, TransporterID, 0,
     TRANSPORTER_DROP)
    else Log(0, 0, TransporterInfo, ConstructorID, TRANSPORTER_DROP)
    Unloaded(Target)
    if Source is a Miner then
        if Target is a Smelter then Log(MinerID, SmelterID, TransporterInfo, 0,
         TRANSPORTER_GO)
        else Log(MinerID, 0, TransporterInfo, ConstructorID,
         TRANSPORTER_GO)
    else
        Log(0, SmelterID, TransporterInfo, ConstructorID,
         TRANSPORTER_GO)
    end
end
Log(0, 0, TransporterID, 0, TRANSPORTER_STOPPED)
```

The functions are explained below:

- `WaitNextLoad`: Wait if the source has no ores or ingots in storage. Continue if there are available ores or ingots to be transported. Reserves the storage space so that other transporters cannot take the same ores or ingots.
- `WaitUnload`: 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 source to inform that new storage space is available for production.
- `Unloaded:` Signals the target to inform that new ingots or ores have arrived so that if the target has reached the required number of ingots or ores, it can continue production.

The simulation will be subjected to these constraints.

1. Initially, miners, smelters, transporters, and constructors have no ores or ingots in their storage.
2. The duration a miner takes to produce an ore is given as input.
3. Each miner has limited capacity to store the ores it produced. If it becomes full, it should wait for a space in its storage to be opened.
4. Each miner has a maximum amount of ores it can produce. When that number is reached, it should quit.
5. The duration a smelter takes to produce an ingot is given as input.
6. 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.
7. The copper smelters require two ores to produce a copper ingot. They should wait for ores to be deposited without busy waiting.
8. The iron smelters require one ore to produce an iron ingot. They should wait for ores to be deposited without busy waiting.
9. Transporters can load from their source, while the source is producing. Similarly, they can unload to their target while the target is in production. The source reserve a storage during production for its output. This storage is considered empty until production is finished. Similarly targets keep their input storage occupied until production is finished.
10. Transporters take a certain amount of time to travel between source and target agents in the simulation. This duration is given to you as input.
11. If a transporter loads a copper ore from a miner, it can only carry it to a copper smelter. If it is a iron ore, it can only carry it to an iron smelter. If the ore is limestone, it can only carry it to a limestone constructor. Similarly, if the package is ingots, it can be carried to a matching constructor. Non-matching source target pairs will not be given to you as input.
12. Transporters have no priority (between each other) when loading or unloading ingots.
13. Transporters should quit if their source is inactive and has no nothing in its storage.
14. The duration a constructor takes to produce a product is given as input.
15. The copper constructors require one ingot to produce a roll of copper sheet. They should wait for ingots to be deposited without busy waiting.
16. The iron constructors require two ingots to produce an iron rod. They should wait for ingots to be deposited without busy waiting.
17. The limestone constructors require two ores to produce a concrete block. They should wait for ores to be deposited without busy waiting.
18. Constructors can produce their outputs while a transporter is unloading to their storage. Similarly a transporter can unload while the constructors are producing their outputs, as long as there is space in their storage.

19. A constructor quits if it can not produce its material due to the lack of incoming ingots or ores and it has none left in storage to continue producing. Lack of incoming ingots or ores can be determined by checking if all of the transporters unloading to the constructor is inactive and not carrying any at the moment.

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:

   - **Miner:**
     `Log`(MinerID, 0, 0, 0, **MINER_CREATED**)
   - **Smelter:**
     `Log`(0, SmelterID, 0, 0, **SMELTER_CREATED**)
   - **Transporter:**
     `Log`(0, 0, TransporterID, 0, **TRANSPORTER_CREATED**)
   - **Constructor:**
     `Log`(0, 0, 0, ConstructorID, **CONSTRUCTOR_CREATED**)

2. You need to call `Initialize` function of HW2Logger 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:

   - **Miner:**
     `Log`(MinerID, 0, 0, 0, **MINER_STOPPED**)
   - **Smelter:**
     `Log`(0, SmelterID, 0, 0, **SMELTER_STOPPED**)
   - **Transporter:**
     `Log`(0, 0, TransporterID, 0, **TRANSPORTER_STOPPED**)
   - **Constructor:**
     `Log`(0, 0, 0, ConstructorID, **CONSTRUCTOR_STOPPED**)

5. Simulator should use `Log` function to output information, and no other information should be printed. Do not modify the file as it will be overwritten.

4 Input Specifications

Information related to simulation agents will be given through standard input. First line will contain number of miners (N_M) in the simulation. Following N_M lines contain the properties of the miners with i^{th} ID (All IDs start from 1) in the following format:
- I_M C_M T_M R_M where

- I_M is an integer representing the production interval of the miner. It is given in milliseconds. The smelter will sleep this amount during production of each ore with slight deviation.
- C_M is an integer representing the storage capacity of the miner.
- T_M is an integer representing the ore type of the miner. Ores have corresponding values:
  - **IRON:** 0
  - **COPPER:** 1
  - **LIMESTONE:** 2
- R_M is an integer representing the total amount of ores that can be produced from the miner.

Next 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 L_S R_S T_S where

- I_S is an integer representing the production interval of the smelter. It is given in milliseconds. The smelter will sleep this amount during production of each ingot with slight deviation.
- L_S is an integer representing the incoming storage capacity of the smelter.
- R_S is an integer representing the outgoing storage capacity of the smelter.
- T_S is an integer representing the ingot type of the smelter. Ingots have corresponding values:
  - **IRON:** 0
  - **COPPER:** 1

Next line contains the number of constructors (N_C) in the simulation. Following N_C lines contain the properties of the constructor with i^{th} ID (All IDs start from 1) 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 product with slight deviation.
- C_C is an integer representing the storage capacity of the constructor.
- T_C is an integer representing the type of the constructor. It can be one of the following values:
  - **IRON:** 0

  - **COPPER:** 1
  - **LIMESTONE:** 2

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 (All IDs start from 1) in the following format:
- I_T SM_T SS_T TS_T TC_T

- I_T is an integer representing the travel time of the transporter. It is given in milliseconds. The transporter will sleep this amount during travel with slight deviation.
- SM_T is an integer representing the source miner ID that this transporter should load ores from. If the value is 0, then the source is not a miner.
- SS_T is an integer representing the source smelter ID that this transporter should load ingots from. If the value is 0, then the source is not a smelter. Only one of SM_T and SS_T values can be zero.
- TS_T is an integer representing the target smelter ID that this transporter should unload ores to. If the value is 0, then the target is not a smelter.
- TC_T is an integer representing the target constructor ID that this transporter should unload ingots or ores to. If the value is 0, then the target is not a constructor. Only one of TS_T and TC_T values can be zero.

5 Specifications

- Your code must be written in Java (version 8).
- Late Submission is allowed. A penalty of $5 \times Lateday^2$ will be applied for submissions that are late at most 3 days.
- You are allowed to use 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 `Log` 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 if it does not involve code or solution.

6 Submission

Submission will be done via ODTUClass. Create a zip file named `eXXXXXXX.zip` (eXXXXXXX is your student id) 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
```