Middle East Technical University | Department of Computer Engineering

CENG 443

Introduction to Object-Oriented Programming Languages and Systems

Spring 2023-2024

Lab 2 Preliminary - Construction Site

(Ver 1.0)

1 Introduction

Keywords: *Concurrency*

In this assignment, you are tasked to write a program to manage a construction site.

2 Overview

In this construction site, there are n tools and n workers (n will be given as a command line argument). Every worker and tool are numbered from 0 to and including $(n-1)$.

i^{th} worker needs to do the following steps repeatedly without stopping:

- Try to take both i^{th} and $((i+1)\%n)^{th}$ tools (% is the modulo operator) until both tools are acquired.
- Complete the current job.
- Put both tools back.

Your program should manage the tools such that every worker could operate productively.

3 Example Output

Due to indeterminism, your output will likely be different from the one below. What is important is that your output should not violate the specification e.g. a specific tool should not be possessed by more than one worker etc.

First few lines of an output of

```
java ConstructionSite 5
```

can be seen below. Note that your app should run **indefinitely** until terminated:

```
Worker 0 tries to take tool 0 (on Thread 9)
Worker 0 takes tool 0 successfully (on Thread 9)
Worker 0 tries to take tool 1 (on Thread 9)
Worker 0 takes tool 1 successfully (on Thread 9)
```

```
Worker 1 tries to take tool 1 (on Thread 10)
Worker 1 fails to take tool 1 (on Thread 10)
Worker 2 tries to take tool 2 (on Thread 11)
Worker 2 takes tool 2 successfully (on Thread 11)
Worker 2 tries to take tool 3 (on Thread 11)
Worker 2 takes tool 3 successfully (on Thread 11)
Worker 3 tries to take tool 3 (on Thread 12)
Worker 3 fails to take tool 3 (on Thread 12)
Worker 4 tries to take tool 4 (on Thread 13)
Worker 4 takes tool 4 successfully (on Thread 13)
Worker 0 completes the job (on Thread 9)
Worker 4 tries to take tool 0 (on Thread 13)
Worker 4 fails to take tool 0 (on Thread 13)
Worker 2 completes the job (on Thread 11)
Worker 0 puts tool 0 back (on Thread 9)
Worker 4 puts tool 4 back (on Thread 13)
Worker 2 puts tool 2 back (on Thread 11)
Worker 0 puts tool 1 back (on Thread 9)
Worker 1 tries to take tool 1 (on Thread 10)
Worker 1 takes tool 1 successfully (on Thread 10)
Worker 1 tries to take tool 2 (on Thread 10)
Worker 2 puts tool 3 back (on Thread 11)
Worker 1 takes tool 2 successfully (on Thread 10)
Worker 3 tries to take tool 3 (on Thread 12)
Worker 3 takes tool 3 successfully (on Thread 12)
Worker 3 tries to take tool 4 (on Thread 12)
Worker 3 takes tool 4 successfully (on Thread 12)
Worker 4 tries to take tool 4 (on Thread 13)
Worker 4 fails to take tool 4 (on Thread 13)
Worker 0 tries to take tool 0 (on Thread 9)
Worker 0 takes tool 0 successfully (on Thread 9)
Worker 0 tries to take tool 1 (on Thread 9)
Worker 0 fails to take tool 1 (on Thread 9)
Worker 1 completes the job (on Thread 10)
Worker 3 completes the job (on Thread 12)
Worker 0 puts tool 0 back (on Thread 9)
Worker 1 puts tool 1 back (on Thread 10)
Worker 3 puts tool 3 back (on Thread 12)
Worker 2 tries to take tool 2 (on Thread 11)
Worker 2 fails to take tool 2 (on Thread 11)
Worker 1 puts tool 2 back (on Thread 10)
Worker 3 puts tool 4 back (on Thread 12)
Worker 4 tries to take tool 4 (on Thread 13)
Worker 4 takes tool 4 successfully (on Thread 13)
Worker 4 tries to take tool 0 (on Thread 13)
Worker 4 takes tool 0 successfully (on Thread 13)
```

....

4 Some Remarks

Although your code will not be graded, and you have freedom in your design, your code should utilize concurrency nonetheless, since the upcoming lab will be graded accordingly.

- Your main class name should be **ConstructionSite**.

- All workers should run as a separate thread and act independent of each other.
- For every action (trying to take a tool, dropping a tool, doing a job etc.), you should use the corresponding action method in **Actions** class.
- When ensuring coordination between workers, you can use any synchronization primitive other than atomics.
- You should not make any assumptions based on timings of actions since different timing intervals may be used during the evaluation of the upcoming quiz.
- You should make sure that no deadlock, no starvation,no busy-wait or no unnecessary wait will occur in runtime.
- To expand the remarks above, solution models below (but not limited to) are not valid solutions:

  Running all logic on a single thread

  Workers waiting for each other to do their jobs even when necessary tools are available

  Threads sleeping and waking up periodically to poll a variable

  Threads waking up more than necessary

  Threads causing other threads to block while sleeping
- Atomicity of the output depends on implementation. Hence, you may need to add synchronization between print statements in **Actions** Class.
- Use Java 8 or a newer version