# **CA670 Concurrent Programming**

|  |  |
| --- | --- |
| Name | Aditya Gupta |
| Student no. | 19210195 |
| Programme | MSc. Cloud Computing |
| Module Code | CA670 |
| Assignment title | Assignment 1 - Java Threads |
| Submission Date | 17th March, 2020 |
| Module Coordinator | David Sinclair |

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name: Aditya Gupta Date: 17 March 2020

# 

# **Assignment 1 - Java Threads**

**Introduction:**

The program implements Java Threads on a sleeping barber problem which has a small barber shop having 2 doors, an entrance and an exit. Inside is a set of M barbers who spends all their lives serving customers, one at a time. Each barber has chair in which the customer sits when they are getting their hair cut. When there are no customers in the shop waiting for their hair to be cut, a barber sleeps in his chair. Customer arrive at random intervals, with mean mc and standard deviation sdc. If a customer arrives and finds the barber asleep, he awakens the barber, sits in the barber’s chair and sleeps while his hair is being cut. The time taken to cut a customer's hair has a mean mh and standard deviation sdh. If a customer arrives and all the barbers are busy cutting hair, the customer goes asleep in one of the N waiting chairs. When the barber finishes cutting a customer’s hair, he awakens the customer and holds the exit door open for him. If there are any waiting customers, he awakens one and waits for the customer to sit in the barber's chair, otherwise he goes to sleep.

**Design:**

The program is designed in a very simple way. There are three classes which have been made in the program. The main class is “Shop.java” and the other two classes are “Barber.java” and “Customer.java”. In the Main class we simply used ‘Double Ended Queue’ which just directly blocks the threads trying to insert or remove elements from the deque this is the case in which it is not possible to enter elements or remove any elements from the deque. The main idea of using BlockingDeque is that it waits while inserting or deleting elements if necessary. We initially assumed waiting seats as 15. In our main method we also used a new type of thread pool which is introduced in Java 8 as ‘newWorkStealingPool()’, it is based on a work-stealing algorithm, where a task can spawn other tasks and the smaller tasks which will be added to queues of parallel processing threads. If one thread has finished its work and has nothing to do it can ‘steal’ the work from other threads queue. Then we created an object of “barbersGroup” of barber class and used for loop which initiates from 0, for waiting area customers, barbers and customers and from there on our thread starts as now our ‘shop opened’. We furthermore used try and catch method for barber as when there is no customer the barber goes to sleep until next customer comes and then further we used normal if else statement if more customers are there in waiting room then the customers exits as there are no waiting seats.

In “Barber.java” class I have made a barber thread and imported “java.util.logging.logger” this package provides the logging capabilities via the logger class and java contains the Java Logging API. The logger that is created is a hierarchy and a ‘.’ indicates a level in the hierarchy. Then we have a barber function in this thread to check the number of active barbers. In barber function I used BlockingDeque so that it checks the waiting area seats, number of barbers available and number of customers available then accordingly it sends in the queue. It is generally used to avoid inconsistency in operations, and used Blocking Deque so that there is no situation deadlock. In this program try and catch method checks the situation of deadlock. Then lastly used “getHaircut” function basically checks that if no customer is there in waited area then it will let barber to sleep, or if there is any customer then will cut hair. I used “math.round” because we initially didn’t decide that how much time it will take a barber to cut the hairs of customers and hair cutting time should be random so used this function so that it generates. So, the standard mean is 2 and standard deviation is 3.

**Correctness of the Program:**

The program is working correctly or not can be verified by running the same code and spawning only one thread at a time. The problem of race condition occurs when multiple threads are accessing the same shared data. In this program as we can see that everything is working perfectly as when we enter number of barbers in shop in console it works exactly like the number of barbers present there and customer then getting hair cut or leaves the shop or just waits in waiting area as it can be inferred from output. Whenever a new customer enters shop or goes to waiting area it works perfectly fine. Situation of deadlock or starvation is solved, and program is not going in deadlock.

**Fairness and Starvation:**

The program is fair as all the thread got equally accessed to shared resources. Starvation occurs when any one of the threads didn’t get to use acquire the shared resources

In the program a combination of semaphores and synchronization is used to implement fairness as all threads were able to use the shared resources and none of the threads were in the situation of starvation or deadlock as showed in the output in the txt file. Starvation was handled in the program by using semaphores and logger as they initiated the execution of the program by making shared resources available to all threads and avoiding deadlock.

**Deadlock:**

When one thread waiting for another thread to come out of the critical section by releasing shared resources, but the thread already acquiring the resource. In the program none of the threads will be acquiring shared resource and ultimately reach deadlock.

In the program, one thread is using the shared resource and in the critical section, the other threads waiting to enter critical section, the thread on the topmost queue will be able to share the resource.

**Result:**

The program’s output is written in a text file. The program has been tested with variation in number of barbers available i.e. no of threads created at a time. The output shows exactly what it was expected. ‘Output.txt file’ is attached in the folder.