# **CA-670 Concurrent Programming**

|  |  |
| --- | --- |
| Name | Nikhil Mittal |
| Student no. | 19210509 |
| Programme | MSc. Data Analytics |
| Module Code | CA670 |
| Assignment title | Java Threads – Sleeping Barber Problem |
| Submission Date | 18th 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: Nikhil Mittal Date: 18th March 2020

# ***Assignment 1 – Sleeping Barber Problem***

**Introduction:**

The program implements a multi-core and multi-threaded solution to the Sleeping Barber Problem. Whenever a customer enters a barbershop, the customers wake the barber if he is sleeping else waits and sits on a chair for this turn. When the turn comes of the customer, Barber is assigned, and the hair is Cut. If there are no chairs in the shop the customers exit the shop. When the Customer’s haircut is done, the customer exits the shop.

**Program Flow**

A screenshot of a cell phone

Description automatically generated

**Design:**

1. The program is designed in a very easy way. There are 3 Java Classes with different files i.e. Generator.java, Barber.java, and Customer.java.
2. Generator class, Barber class, and Customer class are extending Thread to get maximum efficiency.
3. The program is run by getting the input customer from the user and then there are three barbers defined which take the customers randomly. Once the input customer is done, then it is added to the queue so that the thread works simultaneously.
4. In Generator .java program has initiated three barbers and taken the number of customers. Only the after the input constructor calls the class of Customer of the Customer.java file and the customer sits on the chair and wakes barber if no one is there.
5. Customer .java see if there are any chairs left using the Array blocking queue where the chairs are fixed. If seats are full the customer leaves the barbershop.
6. Barber.java is where the barber is assigned to every Customer by calling the run() method and the hair is cut by calling the cutHair() method.
7. The Program only Terminates when input Customers are fetched by the three barbers.

**Correctness of the Program**

The Program is Running Correctly and there is no error and can be Verified using NetBeans IDE. The program achieves multithreading and multi Core and there is no Race around Condition. In Race around condition, there are no threads that are using the same resource to modify or share the same resource. Synchronization also helps in getting the working of multi-threading correctly and ensures multi-threading is working simultaneously which the program is making sure.

**Fairness and Starvation**

A program is said to be fair if all the threads get an equal chance to access the shared variable or access the critical section of the program. Starvation occurs when a thread never gets the chance to run.

In this program, all threads have equal access to every shared variable or can access the critical section of the program which makes it a fair program as the program is executed till the end otherwise it will be stopped.

Starvation never occurs as every thread in the program is running smoothly and no thread is using other threads variables or methods and lock are placed whenever the program is in execution mode.

**Deadlock**

Deadlock occurs when one thread is waiting for a thread to release a lock and the lock is acquired by the same thread. So, in that case, the program will get stuck as none of the threads will be able to acquire the lock proceed with their work.

There is only one critical section in this program and only one lock is used. When the crucial segment is accessed by one thread, the other threads awaits the lock in the queue. Only the thread at the top of the queue is notified, and then the lock released from the previous thread is acquired. There is no deadlock in the above program.

**Difficulties**

1. Making a thread class and assigning the queue for customer and barber was difficult but was achieved by Blocking queue
2. Handling queue, while some customers were waiting, was difficult and got solved by the array blocking queue

**Use of Sleeping Barber Problem**

1. One professor takes a lecture. Another professor cannot enter and teach at the same time. He/she has to wait to teach until the first professor leaves the classroom.
2. After the first professor leaves the classroom, the second professor can enter and teach the subject and leave after a specific period.
3. The lectures are synchronized one after the other and for a specific period.

**References**

[1]. Java - Thread Synchronization

https://www.tutorialspoint.com/java/java\_thread\_synchronization.htm.

[2].

Java Blocking Queue

https://www.journaldev.com/1034/java-blockingqueue-example

[3]. ArrayBlockingQueue Class in Java

https://www.geeksforgeeks.org/arrayblockingqueue-class-in-java/

[4]. https://www.quora.com/What-is-a-real-life-example-of-synchronization-in-Java

[5]. Teach Yourself Java in 21 Days

https://www.cs.cmu.edu/afs/cs.cmu.edu/user/gchen/www/download/java/LearnJava.pdf