Declaration on Plagiarism

This form must be filled in and completed by the student(s) submitting an assignment

Name:	Ajit Kumar
Student Number:	19210438
Programme:	Concurrent Programming
Module Code:	CA670
Assignment Title:	Java Threads - Sleeping Barbers Problem
Submission Date:	16 March 2020
Module Coordinator:	Dr. 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. 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 at http://www.dcu.ie/info/regulations/plagiarism.shtml, https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines

Name: Ajit Kumar
Date: 16th March 2020

Java Threads - Sleeping Barbers Problem

Introduction

The program implements multi-thread and multicore concept on Sleeping Barbers Problem. There are two doors to a small barbershop, an entrance, and an exit. Within the shop, there is a group of N barbers who spend their entire time serving clients, one at a time. Each barber has a chair where the client sits when their hair is cut. A barber sleeps in his chair when there are no customers in the shop waiting to get their hair cut. When a customer enters the shop and finds the barber sleeping, he wakes up the barber, sits in the barber's chair, and sleeps while his hair is cut.

Design

- This program is designed in a very simple way, it contains five different classes named Barber.java, Customer.java, CustomerGenerator.java, SleepingBarber.java, WaitingRoom.java, in which SleepingBarber.java is main class.

- Barber, Customer and CustomerGenerator implements runnable, it does not extend threads, so I must build threads manually and allocate the object to each thread.

- This program can be run in two ways, one in which we can pass parameters for count of barbers, count of waiting room capacity and count of customers and if we don't pass any parameters the default value(3,5,30) is assigned to count of barbers, count of waiting room capacity and count of customers arriving respectively.

- If any of the value passed in parameters is zero or negative, the default value is considered, and application will throw errors if any alphabets or special characters are passed as arguments.

- In the main class (SleepingBarber.java) an object of WaitingRoom, CustomerGenerator, and threads of all barbers and customers are created and the same has been started.

- In Barber.java, it takes WaitingRoom and barbers thread id as an argument and the main run() method which is used to start the thread runs continuously until the shared waitingRoom object is interrupted by the main thread to signal the completion of the application, thus killing the thread.

- In CustomerGenerator, waiting room and customer count is passed as an argument and customers are generated at random and added to the WaitingRoom customer list if the condition satisfies i.e. if the WaitingRoom seats are available else the customer thread is terminated for that specific customer and "Customer Exits..." is printed.

- Random customer generated from CustomerGenerator method is assigned a unique identifier with the help of Customer.java class. It has one method that is goForHairCut(), the thread calls the run() method which calls goForHairCut() which adds the customer to the WaitingRoom if the above mentioned condition is fulfilled. goForHairCut() is a synchronized method to avoid race condition.

- In WaitingRoom method, different customers are assigned to waiting room based on the availability of chairs. The cutHair() method for barber uses synchronized block over customer list to avoid inconsistency of multiple barbers selecting the same customer. Each barber thread acquires lock on the customer list and checks if customer is present in the customer list and picks one for hair cutting with random duration of hair cutting(basically calling sleep on the barber thread); else if no customers are present, the barbers thread goes to sleep(customerList.wait()) until a new customer is added and (customerList.notify()) method is called.

- To terminate the application, terminating the barber thread is essential and to identify when to kill the barber thread, I check if the (customerList == 0) and all the barbers are sleeping and finally System.exit(0) is used to terminate the program.

Correctness of Program

The program is running successfully and the same can be verified by running the same code in eclipse and there is no problem of race condition. Java race condition occurs in a multithreaded system when more than one thread concurrently attempts to access a common resource (modify, write). Since multiple threads try to race each other to finish executing a method thus the name race condition. In this program, different barber threads access the shared waitingRoom for picking up the customer from the waiting area for hair cutting if they are free and this is done by using the synchronized method which is used to lock the shared resource between different barbers. When a thread invokes a synchronized process, it acquires the lock for that object automatically and removes it when the thread completes its job. Therefore, it means the application is right. The output file contains results.

Fairness and Starvation

Starvation defines a condition where a thread is unable to access shared resources regularly and cannot make any progress. This happens when a thread does not receive any CPU time for a longer period. The solution to starvation is called fairness - threads are given a chance to perform equally. Since the threads are generated at random and it is handled by JVM and all the three barbers access the shared WaitingRoom which is in synchronization with the help of synchronized keyword, there is no evidence of starvation which leads to fairness in this programme.

Difficulties Faced

- Handling race condition was challenging and the same was handled using the synchronized keyword.
- The biggest challenge faced in the assignment was to decide when to terminate the barber's thread because customers are generated at random and it is handled by JVM Due to this, I was not able to get the exact customer-id who was in last waiting for a haircut and hence, I was not able to stop the barber's thread. So, in order to solve this, I have used Boolean variable isSleeping() in each barber object and after fixed interval of time I keep checking if all barbers isSleeping() method return 'true' and the customer waiting list is empty.Finally as no customer is waiting for haircut. System.exit(0) is used to terminate the application.

Practical use of sleeping barber problem

- It is being used in inventory management system.
- If any user is accessing any stock related to material, then it should not get changed or updated by any other user's sitting at same or different geographical location otherwise it will create inconsistency in the inventory.
- This problem is being overcome by using the synchronization technique so that different user can access the same stock in the inventory, hence avoiding the inconsistency.

References

[1] Synchronization in Java
https://www.javatpoint.com/synchronization-in-java

[2] Race Condition in Java Multi-Threading
https://netjs.blogspot.com/2015/06/race-condition-in-java-multi-threading.html

[3] Starvation and Fairness
http://tutorials.jenkov.com/java-concurrency/starvation-and-fairness.html

[4] The Complete Reference Eighth Edition