

Name:	Rohit Toshniwal
Student Number:	*****
Programme:	MCM1
Module Code:	CA670
Assignment Title:	Concurrent Programming
Submission Date:	18 March 2020
Module Coordinator:	Dr. David Sinclair

Java Threads - Sleeping Barbers Problem

1. Purpose

In this programming assignment, we will extend the original so-called sleeping-barber problem to a multiple sleeping barbers problem where many customers visit a barbershop and receive a haircut service from any available barber among barbers in the shop.

2. Sleeping-Barber Problem

A small barber shop has two 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.

3. Design of the program

In this problem, I have used 5 different classes to build a better solution. The classes are shop, Barber, Customer, Customer Generator and Sleeping Barber.

SleepingBarber.java: The SleepingBarber.java is a driver program that tests my sleeping barbers problem:

Receives parameters from end user at runtime such as:

Arg1	nBarbers	The number of barbers working in your barbershop
Arg2	nChairs	The number of chairs available for customers to wait on
Arg3	nCustomers	The number of customers who need a haircut service

In this class, there is a main method which is calling all the required methods created in other classes and I have used Multicore and Multi-threaded concept like Executor service and Thread Pool functions which are useful for achieving mutual exclusion, absence of deadlock in the solution.

The main method will instantiate the SleepingBarber

- › Then create a certain number of Customer threads that arrive at random times
 - Have the program sleep for a random amount of time between customer arrivals
- › Have the main method wait to finish executing until all of the Customer threads have finished

It will Start the SleepingBarber thread at the end of the constructor to execute the process.

Shop.java : The shop class have two methods and queue which is used as a waiting queue for the customers coming for getting a haircut.

Here I am using **ArrayBlockingQueue**, which stores the elements internally in FIFO (First In, First Out) order.

Method one is to add a customer to the waiting list:

- › If the number of waiting customers equals the number of seats in the waiting room, have the customer leave
- › Otherwise, add the customer to the waiting list
- › Print out all of the customers currently waiting
- › I have used synchronize in this method so that customers cannot be added to the list while iterating through it

Method two is to cut hairs of a customer:

- › Continue looping as long as more customer will be coming into the barber shop
- › As long as there are any customers waiting
 - Get the customer who has been waiting the longest and cut his hair for a certain amount of time
 - Let the customer know when you have started and when you have finished cutting his hair
- › If there are no customers waiting, go to sleep (i.e. get the lock and wait() on the sleeping condition)

Customer.java: This class is used to create methods which sets and gets the time and id of the customer who entered in the shop. It will instantiate the customer thread and The run() method of the Customer class will call goForHairCut() method which then add the customer to the waiting list.

CustomerGenerator.java : This class is used for generating customers at a random interval of time and the interval time is defined by mean and standard deviation which

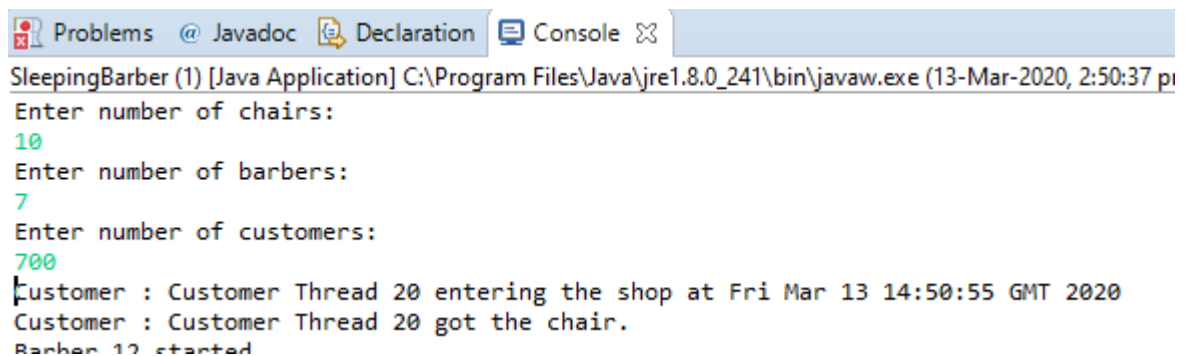
is then normally distributed using a function called **nextGuassain()**. It will start the customer thread and sleeps for a random interval of time for every customer coming for haircut.

Barber.java : This class is used to instantiate the barber thread and the the barber will wake up and get started for cutting the hairs of customers after getting notified that there are customers waiting, so it will call the method cutHair() to cut hairs.

4. Justifications as to the correctness and fairness properties

Correctness: In the solution I've worked out I have four threads: for Barber, Customer, Customer Generator and Shop. One barber can take care of one customer at a time, when done checks the waiting room for other customers, and if there are none, the barber sleeps.

This problem implementation has no deadlocks and shows the absence of starvation, below are the proof for the same, the thread for it can be seen in the screenshot captured via Yourkit Java profiling tool.



```
Problems @ Javadoc Declaration Console
SleepingBarber (1) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (13-Mar-2020, 2:50:37 pm)
Enter number of chairs:
10
Enter number of barbers:
7
Enter number of customers:
700
Customer : Customer Thread 20 entering the shop at Fri Mar 13 14:50:55 GMT 2020
Customer : Customer Thread 20 got the chair.
Barber 12 started
```

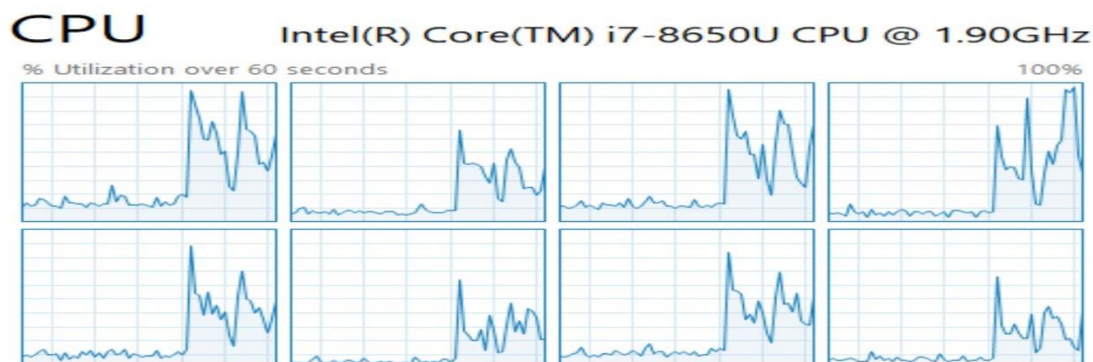
Input taken from the end user at Runtime

Fairness: The program implements the strong fairness in the program as the barber infinitely waits for the customer to come for getting a haircut and the request is granted access as the customer comes in the line.

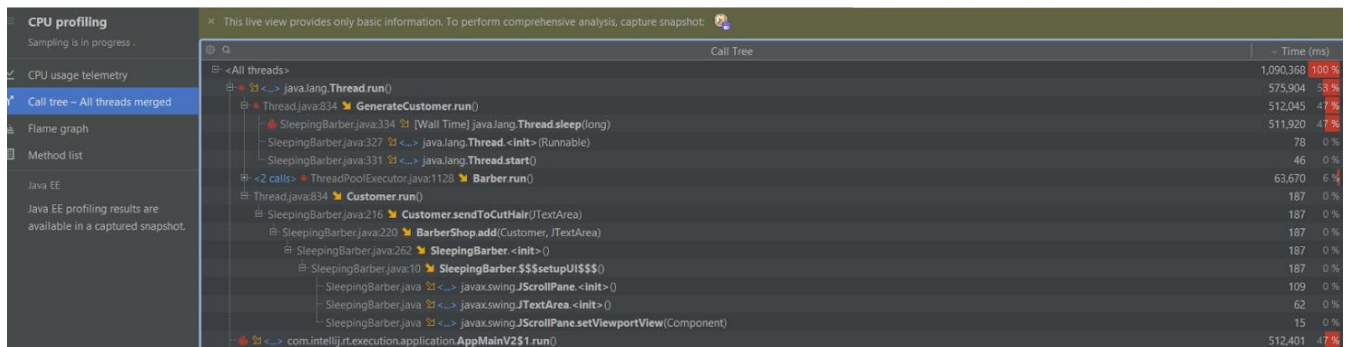
```
Problems @ Javadoc Declaration Console
SleepingBarber (1) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (11-Mar-2020, 1:04:52 pm)
Enter number of chairs:
2
Enter number of barbers:
2
Enter number of customers:
3
Customer : Customer Thread 15 entering the shop at Wed Mar 11 13:05:03 GMT 2020
Customer : Customer Thread 15 got the chair.
Barber 12 started..
Barber 11 started..
Barber 11 waiting for lock.
Barber 12 waiting for lock.
Barber 11 found a Customer Thread 15 in the queue.
Barber 12 is waiting for customer.
Cutting hair of Customer Thread 15 by barber 11
Customer : Customer Thread 16 entering the shop at Wed Mar 11 13:05:15 GMT 2020
Customer : Customer Thread 16 got the chair.
Barber 12 found a Customer Thread 16 in the queue.
Cutting hair of Customer Thread 16 by barber 12
Completed Cutting hair of Customer : Customer Thread 15 in 9 seconds by Barber 11
Barber 11 waiting for lock.
Barber 11 is waiting for customer.
Customer : Customer Thread 17 entering the shop at Wed Mar 11 13:05:27 GMT 2020
Customer : Customer Thread 17 got the chair.
Barber 11 found a Customer Thread 17 in the queue.
Cutting hair of Customer Thread 17 by barber 11
Completed Cutting hair of Customer : Customer Thread 16 in 17 seconds by Barber 12
Barber 12 waiting for lock.
Barber 12 is waiting for customer.
Completed Cutting hair of Customer : Customer Thread 17 in 14 seconds by Barber 11
Barber 11 waiting for lock.
Barber 11 is waiting for customer.
```

Output snapshot for Sleeping Barber Problem

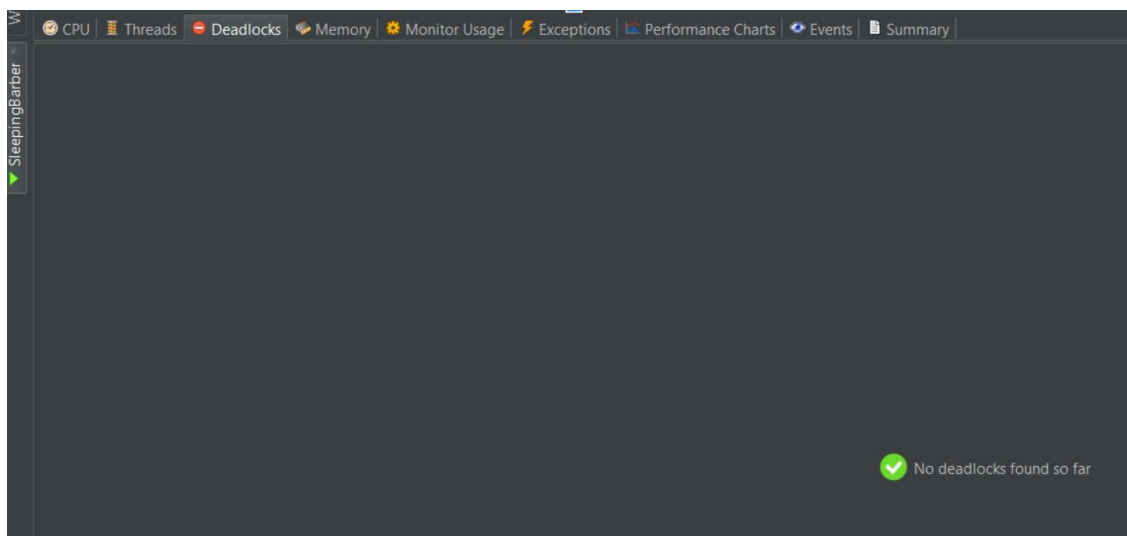
This program also has multicore and multiprocessing, this can be verified from the below screenshots of Jvisual and CPU task manager, as we can see the different number of thread pool started for the barber and customers. We can see the waiting time for each barber when there are no customers in the queue.



All the processors are in use (multiprocessing) to solve the problem



Number of threads used for cutting hair of customer



No deadlocks found in the solution

Event	Time Range	Time (ms)	Thread	Stack Trace	De
Thread.Create #1		0	main	SleepingBarber.main	Name="pool-1-thread-1" Class="java.lang.Thread"
Thread.Create #2		0	main	SleepingBarber.main	Name="pool-1-thread-2" Class="java.lang.Thread"
Thread.Create #3		0	main	SleepingBarber.main	Name="pool-1-thread-3" Class="java.lang.Thread"
Thread.Create #4		0	main	SleepingBarber.main	Name="pool-1-thread-4" Class="java.lang.Thread"
Thread.Create #5		0	main	SleepingBarber.main	Name="pool-1-thread-5" Class="java.lang.Thread"
Thread.Create #6		0	main	SleepingBarber.main	Name="pool-1-thread-6" Class="java.lang.Thread"
Thread.Create #7		0	main	SleepingBarber.main	Name="pool-1-thread-7" Class="java.lang.Thread"
Thread.Create #8		0	main	SleepingBarber.main	Name="pool-1-thread-8" Class="java.lang.Thread"
Thread.Create #9		0	main	SleepingBarber.main	Name="pool-1-thread-9" Class="java.lang.Thread"
Thread.Create #10		0	main	SleepingBarber.main	Name="pool-1-thread-10" Class="java.lang.Thread"
Thread.Create #11		0	main	SleepingBarber.main	Name="pool-1-thread-11" Class="java.lang.Thread"
Thread.Create #12		0	main	SleepingBarber.main	Name="pool-1-thread-12" Class="java.lang.Thread"
Thread.Create #13		0	main	SleepingBarber.main	Name="pool-1-thread-13" Class="java.lang.Thread"
Thread.Create #14		0	main	SleepingBarber.main	Name="pool-1-thread-14" Class="java.lang.Thread"
Thread.Create #15		0	main	SleepingBarber.main	Name="pool-1-thread-15" Class="java.lang.Thread"
Thread.Create #16		0	main	SleepingBarber.main	Name="pool-1-thread-16" Class="java.lang.Thread"
Thread.Create #17		0	main	SleepingBarber.main	Name="pool-1-thread-17" Class="java.lang.Thread"
Thread.Create #18		0	main	SleepingBarber.main	Name="pool-1-thread-18" Class="java.lang.Thread"
Thread.Create #19		0	main	SleepingBarber.main	Name="pool-1-thread-19" Class="java.lang.Thread"
Thread.Create #20		0	main	SleepingBarber.main	Name="pool-1-thread-20" Class="java.lang.Thread"
Thread.Create #21		0	main	SleepingBarber.main	Name="Thread-0" Class="java.lang.Thread"
Thread.Create #22		0	AWT-Windows	Thread.run	Name="AWT-Shutdown" Class="java.lang.Thread"

Threads created by executor class for solving the problem

- A description of where a solution to the sleeping barber(s) problem is used in practice.

Following are the areas where the solution to sleeping barber problem can be applied:

Online bus/air ticket booking system: Multiple users try to book available seats on the buses or flights. Since requests come from different users the system needs to handle multiple threads at same time. When a user picks a seat, it is locked for him for certain time till he makes payment to confirm the seat. If the seat is not paid for within given time, it is released from lock and made available to other users. If the ticket is sold, then remaining users will get a message that seat isn't available to book. In this way, the system works with multiple threads for booking different seats in different buses, handling payments and updating available and non-available seats.

In AI systems like Robots and self-driving cars: In the case of AI, engineers who build real-time systems for robots or self-driving cars, they have undoubtedly have to address the sleeping barber problems of deadlock or resource starvation since multiple processes are involved and therefore they should design their systems accordingly so as to ensure proper process synchronization and inter-process communication.

References

1. <https://orajavasolutions.wordpress.com/2014/05/03/sleeping-barber-problem/>
2. <https://github.com/lennonno/The-Sleeping-Barber-Problem>
3. <http://njavatech.blogspot.com/2014/03/sleeping-barber-problem-using-reentrant.html>
4. <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html>
5. <https://www.codejava.net/java-core/concurrency/java-linkedblockingqueue-example>
6. <https://www.geeksforgeeks.org/linkedblockingqueue-class-in-java/>
7. <https://examples.javacodegeeks.com/corejava/util/concurrent/linkedblockingqueue/java-util-concurrent-linkedblockingqueue-example/>