Java Threads - Sleeping Barbers Problem

Description of the design of the program:

The sleeping barbers problem is solved using synchronization (synchronized methods and synchronized statements) and Linkedlist class of Java. Multi-Threaded and Multi-Core solution is also implemented in Java. Please find below the design of each class of the program:

Class SleepingBarbersProblem (Main Class):

- Input is taken from the user to define the number of seconds that the Barber-shop will be functioning. Error handling is also done for the same to accept numbers greater than zero. Barber-shop will be closed after the user defined time.
- Input is taken from the user to define the number of barbers available in the shop. Error handling is also done for the same to accept numbers greater than zero.
- Using Executors Interface and Thread Pools, number of Barber threads are created as per user defined input to execute on different cores on a multicore machine.
- A thread named CustomerCreator is triggered to generate customers randomly for the barber shop.
- A timer is set to run the Barber-shop for the user defined time. Once the timer is set-off,
    - CustomerCreator thread is stopped from generating new customers.
    - ExecutorService is stopped using shutdown() method allowing current tasks to complete but not taking new tasks to execute. As the barber threads function on a infinite loop of a while condition, the barber threads are stopped using a volatile Boolean flag.

      Note: A barber thread might continue executing the cutHair() method of class Barbershop by waiting on the customerqueue after initiating wait(). As the customer generator is stopped, this might lead to deadlock. To solve this problem, awaitTermination method of ExecutorService is used to wait sometime for the thread to complete and then initiate shutdownNow() to stop all actively executing tasks. Using the isTerminated() method, it is made sure that all the tasks finished execution and the barber-shop can be closed now [1].
    - salon.leave() is called to ask the customers who are sitting in the waiting room to leave as the shop is closing.

Class Barber (Thread):

- Every Barber thread generated by the ExecutorService executes the cutHair() method of class Barbershop where all the functions of the barber is defined. A volatile Boolean flag is defined and set to control the functioning of the infinite loop for executing cutHair() method of class Barbershop [2].
- A Final AtomicInteger variable named idGenerator is used to set names for the barbers.

Class CustomerCreator (Thread):

- This thread generates Customer Threads at random intervals using Math.random() and sleep method with mean ‘m’ and standard deviation ‘sd’. Note, here multiple threads of customers can be created at one time if the random function generates 0 because multiple customers may approach the shop at one time but they have to enter sequentially as there is only one entrance door, which is handled in the customer class.

- The time for each customer's arrival is set using setInTime() and inTime [3]. The name of each customer thread is set using getID().
- A volatile Boolean flag is defined and set to control the functioning of the infinite loop for generating customer threads.

Class Customer (Thread):

- Methods for generating name and time of arrival for each customer is defined in this class. A Final AtomicInteger variable named idGenerator is used to set names for the customers.
- Each customer tries to execute the haircut() method which is both private to the class and synchronised. This method is made private so that only members of the Customer class can access them [4]. This method is made synchronised to control the access of multiple customer to the shared resource i.e haircut() [5]. There is one entrance door and one customer can enter the shop at one time in sequential order to get a barber or a waiting chair. As all the customers generated at random will try to access the haircut() method simultaneously which allows the thread to enter the shop, mutual exclusion is achieved by making haircut() a synchronised method. Inside haircut(), newcustomer() method is called where all the functions of a customer is defined.

Class Barbershop:

- Input is taken from the user to define the number of waiting chairs available in the shop. Error handling is also done for the same to accept numbers greater than zero.
- To manage sequential arrival and execution of customer threads, a linked list object named "customerqueue" is created [6].
- salon.leave() method : This method notifies the customers who are waiting in the waiting room for a haircut that the shop is closing and the barber won't serve any new customers now. Therefore, the waiting customers leave the shop.
- newcustomer() method :
  - Synchronized statement (for customers) is applied on the object "customerqueue" so that new customer threads can use the linked list object one at a time [7].
    The customer leaves the shop if he finds all the waiting chairs filled i.e the barbers are busy and there is no waiting chair to sit and wait. Otherwise, the customer thread is added to the "customerqueue" using the add() method [6].
    If there are no free barbers, but there are chairs available in the waiting room, the customer waits in the waiting room. It is done using the "freebarber" variable which increments or decrements based on the availability of the barbers.
    If the size of "customerqueue" gets to 1 from 0, it means that all barbers were sleeping till now. Hence, a notify call is generated on the object "customerqueue" for the first barber thread to wake up who called wait on this object.
- cuthair() method :
  - Synchronized statement (for barbers) is applied on the object "customerqueue" so that barber threads can use the linked list object one at a time [7].
    If the "customerqueue" is empty, the barber goes to sleep and generate a wait call on the object "customerqueue". Once a notify call is generated on "customerqueue", the barber thread waiting for it picks up the customer using the remove() method [7].

If the customerqueue is not empty, the barber thread picks up the first customer of the queue using the remove() method [6].
Note: If the barber threads, ExecutorService and customer generator thread are stopped, there is a possible situation where a barber thread is waiting on the "customerqueue" and keeps on doing so as the "customerqueue" is empty. This might lead to deadlock. To solve this problem, shutdownNow() is triggered after sometime which will cause an InterruptedException and make the thread to stop and return.

- Once a barber thread gets a customer thread, it cuts hair for a random duration. Note, the random number generated is added with '1' as the number generated could be 0 and a barber cannot finish cutting hair in 0 seconds.
- Once a barber thread finishes cutting a customer thread, it holds the door and wait for the customer to leave for a random duration. Note, the random number generated is added with '1' as the number generated could be 0 and a barber cannot finish holding the door for the customer to leave in 0 seconds.

Efficiency:

- Customer: All the new customers generated will access the haircut() method and enter the shop one at a time as synchronized method is used for the same. The customers after entering the shop will be listed in the "customerqueue" as per their arrival order. As the object "customerqueue" is a linked list, the customers are added and offered to barbers in FIFO (First In First Out) manner using add() and remove() methods respectively.
- Barber: The barber takes customers from the "customerqueue" object. The object "customerqueue" is synchronised so that barbers can access the queue one at a time. While one barber is accessing the queue for taking a customer, the other barbers wait. When a barber is done accessing the queue, the next barber gets to access the queue.

Correctness:

- **Safety Properties:**
  - **Mutual Exclusion:** In the code, the shared methods or objects or resources are
    1) the method haircut() that allows customers to enter the barber shop. There is only one entrance and one customer can enter at one time. To avoid multiple customer threads to use the method at the same time, synchronised method is used.
    2) The object "customerqueue" which is used by customer thread to add to the queue and used by the barber threads to delete from the queue. Synchronised block is used on the object to maintain proper interleaving of sequences of actions.
    Thus, mutual exclusion is achieved.
  - **Absence of deadlock:** The only time wait-and-notify occurs is when the "customerqueue" is empty. As "customerqueue" is a synchronised block, only one barber can hold it and call wait() on it only if the "customerqueue" is empty [8]. The notify() method is called as soon as the first customer thread access the synchronised block of "customerqueue" which allows only one customer to access the queue at one time. Thus, there is an absence of deadlock.
    Note: Once the Barber-shop started closing, the customer threads will stop generating and there might be a situation where a barber thread is waiting on the "customerqueue" leading to deadlock. This type of case is handled using shutdownNow() method and InterruptedException as mentioned in the description of the classes.

- **Liveness Properties:**
  - **Fairness:** Out of the four ways for specifying fairness, FIFO method is used here by implementing a linked list for treating the customers fairly. As for the barber threads, the outputs of many runs showed that they follow linear waiting when it comes to fairness.
  - **Absence of starvation:** Starvation problem is solved by utilizing a queue (linkedlist) where customers are added as they arrive, so that barber can serve them all on a first come first served basis [9]. Also, as mentioned that the output runs have shown to follow linear waiting in terms of barber threads, therefore there will be absence of starvation for the barber threads also.

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

- **ATMs inside a Bank:** Suppose there are M ATMs inside a bank which can address one customer at one time. If there are no customers, the machines stay idle. Customers arrive in the bank at random intervals and use the machines. If all the machines are occupied, the customers stand in the queue for their turn and access the machines sequentially. If the waiting area inside the bank is filled, the customer leaves.
- **Self-Checkout machines at shops:** Suppose there are M self-checkout machines in a shop which can address one customer at one time. If there are no customers, the machines will stay idle. Customers arrive at random intervals and use the machines. If all the M machines are occupied, the customers stand in a queue and proceed sequentially after the machines are freed. If a customer finds the queue to be too long or standing area is filled, the customer leaves and uses the checkout counter managed by employee.

The program is triggered multiple times and the output is checked. Positive results were achieved all the time confirming an efficient solution to the sleeping barbers problem.

Ideas, methods, techniques and different ways to approach the problem are known from the below references but no code is directly copied / pasted in the program. The ideas and approaches known are moulded and changed as per requirement and expertise to satisfy the problem statement. Also, some references are added to uphold the justifications for fairness and correctness properties of the program.

Reference :

1. https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ExecutorService.html#shutdown
2. https://javaconceptoftheday.com/how-to-stop-a-thread-in-java/
3. https://github.com/Java-aid/Interview-Preparations/blob/master/InterviewsPreparation/src/main/java/com/javaaid/ip/app_dynamics/MeetingScheduler.java
4. https://beginnersbook.com/2013/05/java-access-modifiers/
5. https://www.javatpoint.com/synchronization-in-java
6. https://www.geeksforgeeks.org/linked-list-in-java/
7. https://www.programcreek.com/2009/02/notify-and-wait-example/
8. http://math.hws.edu/bridgeman/courses/331/f05/handouts/barber.html
9. https://en.wikipedia.org/wiki/Sleeping_barber_problem