**Declaration:**

A (Java program and Word file) submitted to Dublin City University, School of Computing for module CA670 Concurrent Programming, 2019/2020.

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

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.

By signing this form or by submitting this material online I confirm that 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. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml)

Name: Joseph Cherian

Date: 18/03/2020

**A description of the design of the program**

4 different java classes are used here under the same package name sleeping\_barber12 and same project name.

1. **Barber\_12.java**

Sleeping barber problem is an inter process communication between 2 processes **barber** and the **customers**. The Barber\_12 class inherits the **Runnable** interface and threads are used in this method. The implementation of the thread takes place by allocating a customer in the top of the queue for the barber if he is asleep.

**callAndShave()** method is used to allocate a customer to the queue waiting if necessary for another thread to receive it. The time taken for a barber to shave the particular customer is allocated randomly using the **nextGaussian()** function. It returns random numbers with a mean of 0 and a standard deviation of 1\*15.

Once the thread activity is interrupted as all the customers’ hair has been cut the **exception is** caught using the **catch** block. The Barber\_12 constructor is used to wait until a customer is available for the barber.

2. **Customer\_12.java**

Customer\_12 class inherits the **Runnable** interface and threads are used in this method. To update the id for every customer AtomicInteger class is used. To generate the random numbers with a mean of 0 and a standard deviation of 1\*15 nextGaussian() function is used.

A SynchronousQueue is used in which each insert operation in the queue for the barber must wait for a corresponding remove operation in the waiting customer queue by another thread and vice versa. As the customer enters the waiting room his **id** is incremented.

The implementation of the thread takes place by allocating the new customer to a seat. The **waitToBeCalledAndShaved()** function removes and retrieves the customer from the waiting queue to be called for shaving. The time taken to be called and shaved is also calculated using the nextGaussian() function.

**shaved1** is a Boolean variable which is returned if the customer’s hair has been shaved. String toString() function returns a string object for the customer id.

3. **Execution.java**

The execution class is the class containing the main method. Under java.util.concurrent package the executor interfaces:

**a. Executor:** To launch new tasks

**b. ExecutorService:** Adds features that help manage tasks’ lifecycleare used.

A fixed thread pool is used to create a thread pool that reuses a fixed number of threads. An object of the Waiting class is created and 10 customers are allocated in the waiting queue. Runnable tasks are submitted for execution using the **submit** method. The customer then waits for the barber to be ready. Then we generate the customers according to the requirement in the barber shop using the **generate** method in the **Stream** class. We also provide a limit for the number of customers using the **limit** keyword.

Until all the customers who are in the waiting queue have not shaved their hair the thread sleeps for 1 second. Once all the customers’ hair has been shaved the control comes out of the **while** loop and the appropriate message is printed.

Then all the actively executing tasks are shutdown and the termination is awaited using the awaitTermination method depending on the status of the thread .

4. **Waiting.java**

A bounded blocking queue is used to order elements in FIFO using the **ArrayBlockingQueue** class.

The **nextCustomer()** method is used to remove the topmost element of the queue(customer who has waited for the longest time) and allocate the person to a free barber.

The **takeASeat()** is used to allocate a new customer to the tail of the queue(the empty seat in the waiting room).

The **Waiting** constructor is used to allocate new customers to the waiting queue(waiting room) upto the capacity of seats in the waiting room.

**Justifications as to the correctness and fairness properties of the program**

Safety properties:

1.Mutual Exclusion

. ThreadPool is used to enhance the application performance. Thread pools run multiple tasks simultaneously, and they prevent the time and memory overhead incur during thread creation which is an expensive operation. (Logicbig, n.d.)ThreadPoolExecutor separates the task creation and its execution. With ThreadPoolExecutor, you only have to implement the Runnable objects and send them to the executor. It is responsible for their execution, instantiation, and running with necessary threads. Hence mutual exclusion is followed as fixed number of threads are reused to perform multiple tasks.

2. Absence of deadlock

. Thread pools avoid running out of resources (processors, cores, memory etc) by avoiding unlimited thread creation at a time. After a certain number of threads are created, they typically put the extra threads in a waiting queue till a thread is available for a new task. Hence the program can respond to any signal. (Logicbig, n.d.)

Liveness properties:

1.Absence of starvation

. As compared to creating thread per task, thread pools avoid running out of resources (processors, cores, memory etc) by avoiding unlimited thread creation at a time. After creating certain number of threads, they put the extra tasks in a waiting queue till a thread is available for a new task. (Logicbig, n.d.)Therefore, information sent is always delivered.

2. Fairness

. Contention is resolves using **FIFO** were if a process makes a request, it will be granted before any other process that makes a later request.

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

Used in the allocation of multiple processors for various tasks in a computer.

In computer science, the **sleeping barber problem** is an inter-process communication and synchronization problem between multiple operating system processes. The problem is analogous to that of keeping a barber working when there are customers, resting when there are none and doing so in an orderly manner. The barber and his customers represent the mentioned processes. (Wikipedia, n.d.)

# References

Anon., n.d. *logicbig.* [Online]   
Available at: https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/thread-pools.html

Anon., n.d. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/Sleeping\_barber\_problem