|  |  |  |
| --- | --- | --- |
| **Item** | **Answer** | |
| **Section** | G1 | |
| **Name(s)** | Adeline Chin Wen Jie (Adeline)  Gokarn Mallika Nitin (Mallika) | |
| **Brief description of project** | Our project is a restaurant simulator. The owner of a newly opened restaurant - Gluttons Bay Restaurant (fictional) is interested in looking at the difference between his company and the restaurant Subway, in terms of how much time it takes to service a certain number of customers.  Subway has a very linear process and attends to one customer fully at a time before moving onto the next customer. In reality Subway employs one person to execute work of both waiter and chef. For the sake of comparison, we have split the role of the waiter into waiter and chef. [See Figure1 Appendix A] Gluttons Bay on the other hand is a Vegan 5 course restaurant that consists of Soup, Salad, Appetiser, Main Course and Dessert. Gluttons Bay provides seating as well, thereby requiring multiple customers to be serviced at a time. [See Figure2 Appendix A]  In order to compare Subway and Gluttons Bay, a few assumptions have been made, they are as follows:   1. The food preparation overall, takes the same amount of time for chefs of both Gluttons Bay and Subway. [See Figure3 Appendix B] 2. The time taken by each customer on average overall, to decide on their food and thereafter let the waiter place the order is the same between both Gluttons Bay and Subway. [See Figure4 Appendix B] 3. The time taken by each waiter on average overall, to serve customers their food is the same between both Gluttons Bay and Subway. [See Figure5 Appendix B] 4. The time taken by each customer on average overall, to eat the food served has been *assumed* to be the same between both Gluttons Bay and Subway. *Additionally, we are looking at a high class Subway that provides seating and functions like a normal restaurant apart from the fact that they profess and follow self-service.* [See Figure6 Appendix B] 5. We are assuming that both Subway and Gluttons Bay have only one efficiently working waiter or chef at a time, that is to say, that to maximise efficiency and keep the time taken on preparation, serving etc. constant, the employees are employed in shifts that result in only 1 waiter and 1 chef being on duty at any given point in time. 6. As Gluttons Bay’s previous data shows that we cater to 15 customers on average per day, we will be using that as the upper-bound for our comparison.   One caveat of this comparison is that, on comparing the Single-threaded version and Multi-threaded version, there is no way to take into account the effect of the print statements and the time taken in switching between threads for multi-threaded. | |
| **Justification for multi-threading** | As shown in the process above, Gluttons Bay caters to multiple customers at a time and has the technology and ability to keep track of multiple orders thereby letting a single worker concentrate on doing the job dictated as the technology tells him to do it. This means that the waiter can wait on a customer and immediately after placing the order either wait on the next customer or serve the next order to the respective customer therefore requiring concurrency. In subway on the other hand, a single waiter, can only remember and prepare a single order for a single customer at a given point in time, requiring single threaded programming. | |
| **Transactional integrity** | To implement concurrency in Gluttons Bay, we made use of BlockingQueue interface with implementations of ArrayBlockingQueue and LinkedBlockingDeque so as to ensure that the first order given to each actor (waiter, chef) is the one acted on. All the methods accessing the four queues are written with keeping integrity in mind given that more than one thread can otherwise be accessing and editing the queue.  One very important race condition that had to be met was on the LinkedBlockingDeque queue called customersServed. When a customer is served, by the waiter, the customer needs to ensure that the dish served is corresponding to what he/she ordered. This required the customer thread to both check the Dish, and return it to the queue if it is not meant for him/her. This required synchronization on the queue to ensure that no other customer thread is accessing the queue to remove the next dish on it.  We believe that all possible race conditions have been taken care of. | |
| **Performance** | The Waiter thread has to both serve customers and take orders from customers. Both functions cannot take place concurrently by a single Waiter Thread. This means that at some point the Waiter thread needs to switch between the serving and the taking of orders and back respectively. For this purpose, we made use of the [**offer**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html) e, long timeout, **[TimeUnit](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html" \o "enum in java.util.concurrent)** unit) and [**poll**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html) unit) methods that made the Waiter thread wait for an update in the queue, that is the next order to be served/ taken for exactly 1 second each before switching to check the other queue.  [Waiter.java lines 30 and 57] | |
| **Evidence of exploration** | 1. BlockingQueue 2. ExecutorPool | 1. Thread killing 2. CountDownLatch |
| **Innovation** | Our project allows the owner of Gluttons Bay to evaluate whether the “sit-down” service model as compared to the “self-service” model would work more efficiently for his own company. After seeing the results, he feels the technology he has invested in, accompanied with the ambience and restaurant values well justify the time spent by his employees as compared to Subway’s. Additionally, the owner was happy to know that we have provided code that by commenting and uncommenting certain sections, the Gluttons Bay model would provide a more real life simulation based on the variance of time taken to prepare different courses etc. The sections of code that need uncommenting are on lines 36 and 44, 61 and 69 of Waiter.java, and 37 and 44 of Chef.java. The subsequent pieces of code that need to be commented out are on lines 45 and 51, 70 and 76 in Waiter.java, and 46 and 51 in Chef.java. | |
| **Adherence to coding conventions & good practices** | We used the following site as a reference on what the best practices when writing multi-threading applications are: [**link**](https://www.javaworld.com/article/2078679/java-concurrency/java-concurrency-modern-threading-for-not-quite-beginners.html)  The main ones we implemented include:   1. Thread management and Executor Thread pools - by re-using threads through newCachedThreadPool in the Executor Service 2. Runnables vs. Callables – used both in different implementations although we did not throw any error when using Callable | |
| **References/ Acknowledgement** | * [Project](https://github.com/heidtJJ/Multi-Thread-Restaurant) written by GitHub user heidtJJ (Jared Heidt) as our starting point and built on the concept (folder titled reference). * StopWatch class written by Prof Kevin and modified by Prof Mok * Links in exploration document | |

# Appendix A

# Appendix B







