A logo of a globe with a yellow ring around it

Description automatically generated

**INDIVIDUAL ASSIGNMENT**

**TECHNOLOGY PARK MALAYSIA**

**CT074-3-2-CCP**

**CONCURRENT PROGRAMMING**

**APD2F2305CS(DA)**

**HAND OUT DATE: 24th NOVEMBER 2023**

**HAND IN DATE: 23rd FEBRUARY 2024**

**WEIGHTAGE: 20%**

**LECTURER NAME: Dr. KUAN YIK JUNN**

**STUDENT NAME: SOO JIUN GUAN**

**TP NUMBER: TP068687**

**Table of Contents**

[1.0 Introduction 3](#_Toc159106526)

[1.1 Concurrency Concepts Used 3](#_Toc159106527)

[1.2 Assumptions 4](#_Toc159106528)

[2.0 Basic Requirements Met 5](#_Toc159106529)

[3.0 Additional Requirements Met 9](#_Toc159106530)

[4.0 Requirements Not Met 16](#_Toc159106531)

[5.0 Conclusion 17](#_Toc159106532)

[6.0 References 18](#_Toc159106533)

# Introduction

## Concurrency Concepts Used

Concurrency in programming terms generally refers to multiple tasks executed in a synchronized manner through context switching and completed during an overlapping timeframe by controlling the usage of shared resources (What Is Concurrent Programming?, n.d.). In the simulation of the GoGo Coffee Cafe, I explore the application of concurrency concepts in Java to model a real-world scenario where multiple entities operate independently and concurrently. A few concurrency concepts were utilized to simulate the real-life scenario of order fulfillment in GoGo Coffee Cafe, including:

**Threads:** The simulation utilizes Java threads to execute the concurrent activities of baristas and customers. Each thread represents an independent entity with its own behavior and operates simultaneously with others (Threads and Concurrency - Operating System Notes, n.d.). In the context of this case study, threads are used to reflect real-world concurrency as customers enter, then place orders, as well as consume their drinks, while baristas prepare orders.

**Thread Communication:** Threads communicate with each other using mechanisms like wait, notify, and notifyAll to coordinate their activities (Jaiswal, 2023). These classic concurrency mechanisms are employed to coordinate between customers waiting for their orders and baristas notifying them when orders are ready.

**Synchronization:** Synchronization mechanisms such as semaphores and locks are critical for managing exclusive access to the shared resources such as the espresso machine, milk frothing machine, and juice tap. The lock mechanism is used for actions like updating the state of tables or processing the order queue. While the semaphore mechanism ensures that only one thread (barista) can access a specific resource at a time (Youzentech Technologies, 2023), which is essential for avoiding conflicts and ensuring the correct preparation of orders.

**Atomic Variables:** The use of AtomicInteger for variables like total sales and active customer count provides a thread-safe way to perform atomic operations, crucial for maintaining accurate counts in a concurrent environment without interruption from other threads (Combat Shared-Mutability Using Atomic Variables, n.d.-b).

**Concurrent Data Structures:** The adoption of ConcurrentLinkedQueue for managing orders allows for thread-safe operations without the overhead of explicit synchronization for each enqueue or dequeue action, facilitating efficient order management.

## Assumptions

Several assumptions are made in the case study to model the operations of the GoGo Coffee Cafe:

1. Customers will leave immediately if the cafe is too full or if there is no suitable seating that aligns with their preferences for preferring private tables or being willing to share with others.
2. The customer places the order online and then the system will randomly awaken one of the sleeping baristas to take the order.
3. The distribution of drink orders is assumed to be 70% cappuccinos, 20% espressos, and 10% juices to offer variety in customer preferences and resource usage.
4. The machine (espresso machine, milk frothing machine, and juice tap) can only be used by one barista at a time.

# Basic Requirements Met

1. Baristas sleep when no customers are waiting and are randomly awakened to take orders as soon as a new customer arrives, then go back to sleep when there are no customers waiting. A screenshot of a computer code

   Description automatically generated

The run() method of the Barista class contains the main execution logic for barista threads. Within the while loop, barista threads continuously check if the cafe is open, if there are any orders in the queue, or if there are any active customers. If none of these conditions are met, indicating that no customers are waiting, then the barista thread enters a waiting state by calling wait(). The orderLock object serves as a synchronization point to ensure that barista threads wait until notified of new orders.

A close up of a text

Description automatically generated

When a new order is placed and added to the orderQueue, a barista thread is notified via orderLock.notifyAll() within the Customer class, causing the thread to wake up and process the order accordingly.

1. Baristas require specific machines to prepare drinks:

espresso and milk frothing machines for a RM9 Cappuccino,

espresso machine for a RM6 Espresso,

juice tap for a RM7 Juice.

 A screenshot of a computer code

Description automatically generated A screenshot of a computer code

Description automatically generated A computer screen shot of text

Description automatically generated

Semaphores are used here to control access to shared resources such as the espresso machine, milk frothing machine, and juice tap. They are initialized with a single permit to indicate that only one thread can acquire each machine at a time. For the preparation, the prepareOrder() method first determines which type of drink to prepare based on the orderType of the Order object. Moving forward, the barista would utilize different machine to craft each drink, with each drink preparation method (makeCappuccino(), makeEspresso(), makeJuice()) acquires the required machines by calling the acquireMachine() method, which uses the Semaphore.acquire() to ensure exclusive access. After the drink is prepared, the releaseMachine() method is called to release the semaphore permit using Semaphore.release(), allowing other baristas to use the machine. By using these functions according to the specific ingredients for preparing drinks. They could ensure that only one barista can use a machine at a time, preventing resource conflicts and ensuring proper resource utilization.

1. Customers pay upon ordering and collect their drinks when ready.

A computer code with text

Description automatically generated with medium confidence

The Order class contains fields for customerName and orderType, and a boolean flag isOrderReady to track if the order has been prepared. The orderReady() method is synchronized to ensure thread safety when modifying the isOrderReady flag. It sets isOrderReady to true and notifies any threads waiting on this object. Additionally, the waitForOrder() method is also synchronized to prevent concurrent access issues. It makes the customer thread wait (wait()) until its order is ready (isOrderReady becomes true), at which point it will notify the customer thread using notify() to collect the drink.

1. Customers leave immediately if more than 5 people are waiting, otherwise the customer enters and waits.



Atomic variable is utilized here to monitor the accurate counts for the customers of each activity in the cafe.

.

A computer screen shot of a computer code

Description automatically generated

tryEnterCafe() method attempts to enter the cafe and checks if the cafe is too crowded. A synchronized block on the orderLock object is utilized within this method to ensure thread safety when reading and updating the number of lost, active and waiting customers. If the cafe is not too full, it increments the number of waiting customers (customer who attempt to order or is waiting his/her order) and the number of active customers (customers who are in the cafe), then returns true (indicating that a new customer has entered the cafe). Otherwise, it increment the number of customers lost and returns false (indicating that the customer has left).

# Additional Requirements Met

1. Customers are monitored their arrival order, ensuring the longest-waiting person is served next.

To ensure the customer who has been waiting the longest is always served next, concurrent data structure (ConcurrentLinkedQueue)is used for the order queue. The FIFO (First-In-First-Out) nature of this queue ensures that orders are processed in the order they were added.

1. Customers who are standing keep track of their order in line, so the person who has been waiting the longest gets the next available seat.



Same with first additional requirement, ConcurrentLinkedQueue is implemented to manage the queue of customers waiting for tables.

A screenshot of a computer code

Description automatically generated

In the function above, add() method of concurrent data structure will add the customer threads into a queue to ensure that they get their seats by ordering. If the customer finds the seat, poll() method will help go to the next customer by moving the "pointer" to the next element in the queue after removing the current head. When the customer cannot find the seat within a specified time, it will utilize remove() to remove the customer from the concurrent linked queue.

1. Some customers prefer not to share a table with strangers, opting to wait for a private table for a period before they are willing to share a table with strangers. After a period of standing, some waiting customers may get tired and leave the cafe.



A Boolean flag prefersPrivate is used to randomly determine the customers’ seating preferences.

A screen shot of a computer code

Description automatically generated

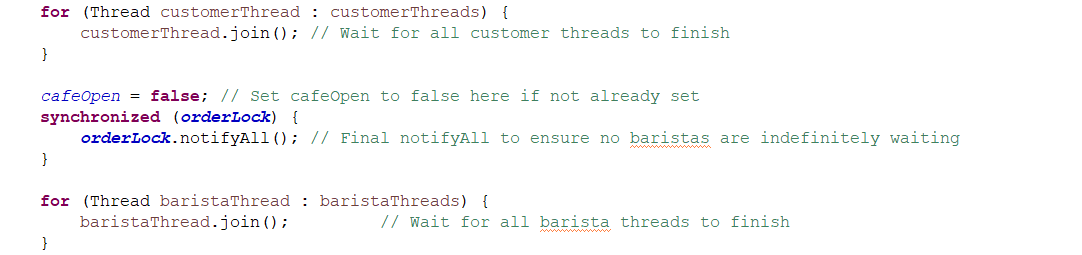
Customers' seating preferences are managed with a synchronized method in the Table class, which checks if the table state matches the customer's preference (private or shared) and whether there's an available seat. If returns true, it means the customer has successfully found a table according to their preference. Otherwise, there is no suitable seating for the customers.

A screenshot of a computer code

Description automatically generated

In this function within Customer class, it continues to run as long as the finding seat time is less than a maximum waiting time and the available table is found. Within the loop, there is a synchronized block that locks on the tables array to ensure that no two threads (customers) can modify the state of the tables at the same time, which could lead to incorrect behavior such as two customers who prefer private table sitting at the same table simultaneously. Inside the synchronized block, it iterates over all the tables. If the function is still working, meaning the customer didn’t find their preferred seat and the customer will be asked to line up. If the customer who prefer a private table was waiting for a period but still no available seat, he/she is willing to share table with others. After attempting to find a table and waiting available seat for a maximum waiting time, the customer will decide to leave due to tiredness.

1. Closing cafe after all customers have left and all baristas are sleeping.



Using join() on threads ensures that the main thread waits for all customer and barista threads to complete their execution before closing the cafe. This synchronization mechanism ensures a clean shutdown process.

1. Drink preparation and drinking progress are visible.

A screenshot of a computer program

Description automatically generated

The prepareOrder() method in the Barista class simulates the steps involved in preparing the drink and provides feedback on the preparation and completion steps to show the progress of drink preparation.

A screenshot of a computer code

Description automatically generated

The simulateCustomerActivity() method models a cafe customer's experience, encapsulating the entire lifecycle of a customer's visit, which is from deciding on an order to leaving the cafe. And also print statements are shown for each step of customers’ activity.

1. When baristas attempt to acquire machines, their availability should be clearly stated, including which machines are used by which barista. A computer screen shot of text

   Description automatically generated

The availability of machine is checked by availablePermit() of semaphore and synchronization mechanism used to ensure that the status printed (available or in use) is accurate at the moment. When a barista thread wants to access the specific, it calls acquire() on the semaphore. If there are permits available, acquire() decrements the permit count and allows the barista thread to access the machine. Otherwise, the acquire() method blocks the calling barista thread until a permit is released by another barista thread. After that, print statements provide clarity on which barista is using which machine at any given time. Additionally, print statements also will be shown after the machine is finished using by the barista.

1. Time delays added for each event 

Time delays are added to various parts of the code by making the thread sleep for a specified or random amount of time to simulate real-life scenarios such as the time it takes for customers to arrival, for customers’ order decision making, for baristas to prepare drinks, for customers to enjoy their drinks, and so on.

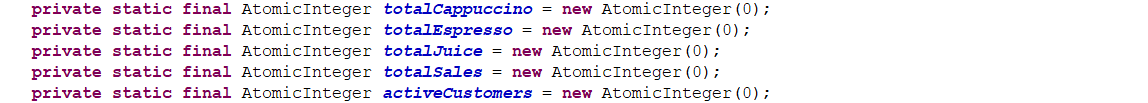
1. 70% of customers order Cappuccino, 20% order Espresso, and 10% order Juice.

A computer screen with a number and text

Description automatically generated with medium confidence

The distribution of drink orders uses random selection based on predefined percentages (70% for Cappuccino, 20% for Espresso, 10% for Juice) to ensure that orders align with the specified distribution.

1. Reporting total number of each drink and sales for the day



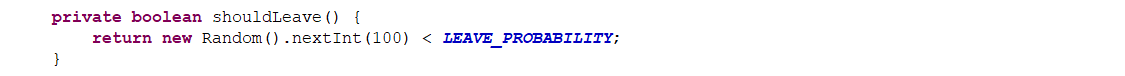
Atomic Variables are implemented here for counting the total number of drinks sold (totalCappuccino, totalEspresso, totalJuice) and accumulating the total sales (totalSales). These variables ensure that updates to the counts and sales are atomic and visible across threads. A screenshot of a computer program

Description automatically generated

Synchronization on orderLock is used to ensure that the final wake-up notification to barista threads is done safely, preventing any thread from missing the notification and being stuck indefinitely. After that, it waits for all customer threads to finish their execution using join() method of the Thread class. This ensures that all customer orders are processed before closing the cafe. After all customers finish their consumption, it marks the cafe as closed to prevent new orders from being accepted. And also use orderLock.notifyAll() to ensure that any asleep barista threads are notified to stop waiting and join() to wait for all barista threads to finish their jobs. Then, reporting the total sales and the number of each drink sold which utilize the atomic variables for thread-safe accurate counting.

# Requirements Not Met

1. The customers queue up to place their orders if all baristas are busy.

There is an assumption made at the start of the assignment, which is that order placements occur online. This means the customers do not need to queue up for order placements, even though all baristas are busy, so this requirement would not meet. However, I modified this requirement so that “customer threads have the option to either wait or not if notice that all baristas are occupied”. 

The shouldLeave() method generates a random number between 0 and 99 and compares it to a LEAVE\_PROBABILITY constant to determine whether the customer is willing to wait when all baristas are busy. If the generated number is less than the LEAVE\_PROBABILITY, the method returns true, indicating the customer will leave. Otherwise, return false.

A screenshot of a computer code

Description automatically generated

The checkBaristaAvailable() method is synchronized, ensuring that it handles one thread at a time to avoid concurrency issues. Within this method, it checks if any baristas are available by calling the availableBaristas.get() method. If all baristas are unavailable, it randomly decides whether the customer will wait or leave. If the customer decides to leave, it decrements the number of waiting customers as well as increments the number of customers lost.

# Conclusion

The GoGo Coffee Cafe simulation, utilizing concurrency concepts in Java, showcases a comprehensive model of a bustling cafe environment, emphasizing the importance of thread management, synchronization, and concurrent data structures. By simulating the intricate interactions between customers and baristas and the careful handling of shared resources, the assignment not only demonstrates the practical application of concurrent programming but also highlights the challenges and considerations in replicating real-world dynamics. From this assignment, I realized the significance of concurrency in software development, and willing to conduct a further detailed exploration of how parallel processes can be orchestrated to enhance operational efficiency and user experience in complex systems in the future.

# References

*Combat Shared-Mutability using atomic variables*. (n.d.). OpenClassrooms. https://openclassrooms.com/en/courses/5684021-scale-up-your-code-with-java-concurrency/6600431-combat-shared-mutability-using-atomic-variables

Jaiswal, S. (2023, December 1). Inter-Thread Communication in Java: A Deep Dive into wait(), notify(), and notifyAll(). *Medium*. https://medium.com/@satyendra.jaiswal/inter-thread-communication-in-java-a-deep-dive-into-wait-notify-and-notifyall-efb8a8c29958#:~:text=The%20wait()%20%2C%20notify()%20%2C%20and%20notifyAll()%20methods%20provide,%2C%20efficient%2C%20and%20scalable%20applications.

*Threads and Concurrency - Operating system notes*. (n.d.). https://applied-programming.github.io/Operating-Systems-Notes/3-Threads-and-Concurrency/

*What is concurrent programming?* (n.d.). Educative. https://www.educative.io/answers/what-is-concurrent-programming

Youzentech Technologies. (2023, May 27). *Understanding semaphores: synchronization and resource management in concurrent programming*. https://www.linkedin.com/pulse/understanding-semaphores-synchronization-resource/