SWE 4302:Object-Oriented Concepts II Lab
Multithreading in Java - Restaurant Simulation

Namisa Najah Raisa 210042112

November 2023

1 Customer class

There are fields and constructor.

The run() method Displays a message indicating the customer's presence in the restaurant and requests table assignment from the receptionist.Increments customersServedCount to track the number of customers served.Checks if the number of customers served exceeds the customersServed limit. If it does, the customer releases the table using the receptionist and stops ordering.Simulates placing an order by randomly selecting 1 to 3 dishes and adding them to the order queue.Waits for the order queue to become empty, simulating the customer waiting for food to be served. Prints a message indicating that the customer finished dining.

The getRandomDish() method Generates a random dish from a predefined list of dishes (Burger, Pizza, Pasta).

2 Chef class

There are fields and constructor.

The run() method Utilizes an infinite loop to continuously check for orders in the orderQueue.Within the synchronized block, checks if the orderQueue is not empty.If an order exists (dish != null), the chef begins preparing the dish.Displays a message indicating which dish the chef is preparing ("Chef [chefID] is preparing [dish]").Simulates cooking time using Thread.sleep() to pause execution for a random duration between 1 to 5 seconds.After cooking, the dish is added to the cookedFoodQueue.

3 Waiter class

There are fields and constructor.

The run() method Uses an infinite loop to continuously check for cooked dishes in the cookedFoodQueue.Within the synchronized block, checks if the cookedFoodQueue is not empty.If a dish exists (dish != null), the waiter starts serving the dish.Displays a message indicating which dish the waiter is serving ("Waiter [waiterID] is serving [dish]").Simulates serving time using Thread.sleep() to pause execution for a random duration between 1 to 4 seconds.

4 Receptionist class

There are fields and constructor.

The assignTable() method Uses synchronization to ensure thread safety in accessing and modifying shared data (tablesAvailable).When a customer arrives and requests a table, the method is called.Enters a while loop to check if there are available tables (tablesAvailable $>$ 0).If no tables are available, the thread waits (wait()) until a table becomes available.Once a table is available, the customer is assigned a table by decrementing the tablesAvailable.Prints a message indicating that the customer has been assigned a table.Notifies other waiting threads (chef/waiter) that a table has been assigned (notify()).

The releaseTable() method Uses synchronization to ensure thread safety.When a customer finishes dining and releases the table, this method is called.Increments the tablesAvailable to indicate that a table has become available.Notifies the receptionist that a table is available (notify()).

The synchronized methods ensure that only one thread at a time can execute these methods to avoid race conditions.The assignTable() method checks if there are available tables. If not, it waits until a table is released. When a table is released via releaseTable(), it notifies waiting threads that a table is available again.

5 Main(RestaurantSimulation) class

Initializes variables for the number of tables (tables), chefs (chefs), waiters (waiters), total customers in the restaurant (totalCustomers), and the number of customers to be served before stopping the simulation (customersServed).

Creates two queues: orderQueue for placing orders and cookedFoodQueue for cooked dishes. Initializes a Receptionist instance, passing the number of tables available. Customer, Chef, and Waiter Initialization: Creates a list of Customer

instances based on the totalCustomers variable. Each customer is initialized with references to the Receptionist, orderQueue, and the customersServed limit.

Creates a list of Chef instances based on the chefs variable. Each chef is initialized with references to the orderQueue and cookedFoodQueue. Creates a list of Waiter instances based on the waiters variable. Each waiter is initialized with a reference to the cookedFoodQueue.

Starts the threads for customers, chefs, and waiters by invoking the start() method on each object.

Waits for all customer threads to finish dining by using join() on each customer thread.This part ensures that the main thread waits until all customers are served and finish their dining before moving forward.

After all customers finish dining (indicated by the completion of their threads), the simulation prints "Simulation complete. All threads finished." This class orchestrates the initialization of the simulation, starts the threads representing customers, chefs, and waiters, and waits for all customers to complete their dining experience before ending the simulation.

_____X_____