CSE214

HOMEWORK 4 - SPRING 2025

HOMEWORK 4 - due Tuesday, March 25th no later than 7:00PM

REMINDERS:

- **Submit your files only via CodeGrade on the Content page in Brightspace. Access CodeGrade by clicking "HW4 - CODEGRADE SUBMISSION LINK" for your submission.**
- **Use of a package is optional. If you wish to use it, make sure to name it "hw4" (all in lowercase). Otherwise, you will lose points.**
- **Due to the random nature of the results in this assignment, the automated code grading features will be disabled. However, you are still required to submit your code to CodeGrade, where it will be evaluated manually.**
- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **You may use any Java API class that you wish.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

Welcome to the CSE 214 Diner! In this assignment, we will be simulating the serving of customers at different restaurants. The objective of the simulation is to mirror a real-life analysis on how different variables affect trends in restaurant efficiency and profit. Efficiency is represented in broad terms by the amount of time it takes to serve a customer and the number of customers lost due to an inability to provide good service. We will be simulating multiple restaurants at the same time; there are more details on this below.

BACKGROUND

Firstly, a customer will enter one of the restaurants. The probability of a customer entering a restaurant will be determined by user input. Note that a maximum of *four customers per restaurant* can enter a restaurant during a simulation unit. If there is an empty spot present at the same restaurant that the customer is waiting in, the customer will be seated at that restaurant. If the restaurant is full, then the customer will leave.

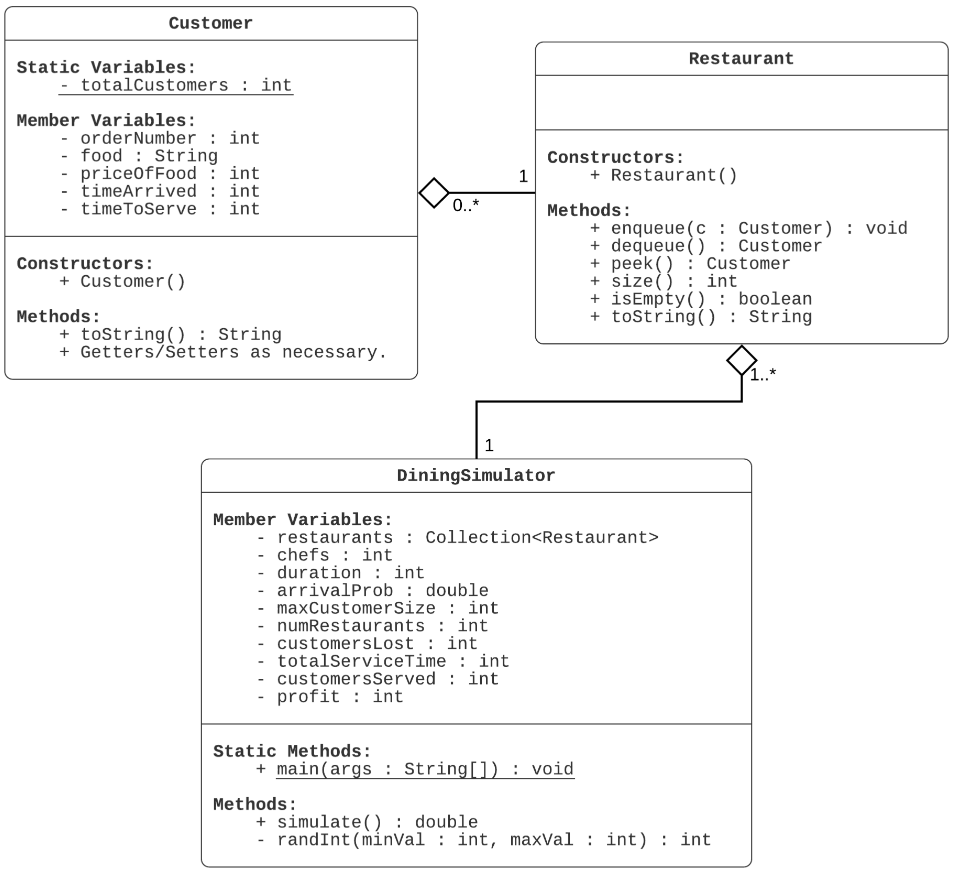
When the customer is finally seated at a restaurant, they will choose the food that they want at random. There are five options: a cheeseburger, steak, grilled cheese, chicken tenders, and chicken wings. Each food has an equal probability of being chosen by the customer. Steak takes on average 30 minutes to make, and costs 25 dollars. Chicken wings takes on average 30 minutes to make, and costs 20 dollars. A cheeseburger takes on average 25 minutes to make, and costs 15 dollars. Chicken Tenders takes on average 25 minutes to make, and costs 10 dollars. Grilled Cheese takes on average 15 minutes to make, and costs 10 dollars.

The definition of "average" on the menu is defined as a restaurant with 3 chefs. The number of chefs for each restaurant will remain constant throughout each restaurant, and this number will be generated by user input. If there are more than 3 chefs, the amount of time it takes to make the food decreases by 5 minutes for every extra chef, up to a realistic maximum of 10 more minutes of efficiency. If there are less than 3 chefs, each missing chef will increase the average by 5 minutes. You should always have at least one chef in the kitchen.

Once the customer has received their food, they will take 15 minutes to eat. Please note that one simulation unit elapses 5 minutes of time. When they are done eating, they will pay their bill and leave the restaurant. You should keep track of the amount of money gained at the end of the day and the average time of each order. You should also keep track of the number of customers that ran out of patience and left the restaurant because there was no space.

UML

The UML Diagram for all the classes specified below is as follows:



SPECIFICATIONS

Implement the simulator given these guidelines. The goal of the simulation is to determine how parameters can change Restaurant efficiency. You will be able to analyze the trends through experiment repetition. You will be graded on how closely you follow the trends.

1. `Customer`

This class represents a customer that will dine at your restaurant.

- `public Customer()`
  You may also include an additional constructor with parameters.
- Getters / Setters for all member variables, as necessary.
- `public String toString()`
  Returns a `String` representation of this `Customer` object. This should be in the format `[#orderNumber, food, timeToServe min.]`, as shown in the Sample Output below. To shorten the I/O, your food should be abbreviated by initial, i.e. "S" for steak, "CW" for chicken wings, "CT" for chicken tenders, etc.
- `private static int totalCustomers`
  The total number of customers that have showed up to the chain. This number is incremented by one every time a customer enters one of the restaurants.
- `private int orderNumber`
  Every time a customer enters one of the restaurants, they will be assigned a number. This will be automatically determined using `totalCustomers`.
- `private String food`
  This String is a representation of the food that a customer has ordered.
- `int priceOfFood`
  The price of the food that will be added to the restaurant's profit once the customer pays the bill.
- `int timeArrived`
  The simulation unit when the customer arrived at the restaurant.
- `int timeToServe`
  The time it will take to serve the customer before the customer leaves the restaurant in minutes, dictated by the guidelines set out above regarding which food the customer ordered. For every simulation unit, this number will decrease. Once it reaches zero, it means that the customer has finished their food and that they are going to leave at the beginning of the next simulation unit.

2. `Restaurant`

This class will function as a Queue. The queue will take Customer objects and seat them. You may extend a Java API class to simplify this definition.

- `public Restaurant()`
  You may also include an additional constructor with parameters.
- `public void enqueue(Customer c)`
  Adds a new customer `c` to the restaurant.
- `public Customer dequeue()`
  Removes and returns the first `Customer` in the restaurant.
- `public Customer peek()`
  Returns, but does not remove, the first `Customer` in the restaurant.
- `public int size()`
  Returns the number of `Customers` in the restaurant.
- `public boolean isEmpty()`
  Returns `true` if there are no `Customers` in the restaurant, `false` otherwise.
- `public String toString()`
  Returns a String representation of this `Restaurant` object. See Sample I/O for more details.

3. `DiningSimulator`

This class will contain the main method which will be used to run the simulation. The following parameters should be tracked:

- `Collection<Restaurant> restaurants`
  The collection of `Restaurant` objects, which `Customers` may enter. This will be used to seat customers and to feed them. You may use any data structure that you want (ArrayList, LinkedList, etc.).
- `private int chefs`
  The number of chefs preset at each restaurant.
- `private int duration`
  The number of simulation units to perform.
- `private double arrivalProb`
  The probability ([0.0, 1.0]) of a new customer arriving.
- `private int maxCustomerSize`
  The maximum number of customers that can be seated at each restaurant.
- `private int numRestaurants`
  The number of restaurants to simulate.
- `private int customersLost`
  The number of customers which were not able to be seated and left.
- `private int totalServiceTime`
  The sum total of the time each customer has spent at a restaurant.
- `private int customersServed`
  The number of customers which were properly served.
- `private int profit`
  The amount of money gained at the end of the simulation.

In addition, you should also implement the following methods:

- `public double simulate(/*arguments/*)`
  Runs the simulator. You should calculate and return the average time the customer spent at the restaurant.
- `private int randInt(int minVal, int maxVal)`
  This is a helper method that can be used to generate a random number between `minVal` and `maxVal`, inclusively. Returns the randomly generated number.
- `public static void main(String[] args)`
  This is the main method, and prompts the user for the input required to perform the simulation. The simulator will run and then output the results. You should prompt the user whether or not another simulation should be performed.

4. You may include additional methods, variables, or classes as necessary or as you find convenient.

NOTES

- **Note that there does exist a `Queue` type in the Java API framework.**
- **Further note that it is an *interface*, not a class. This means if you choose to utilize the standalone Java API `Queue` type, it must be *implement*ed instead of *extend*ed.**
- **Remember that interfaces themselves provide no implementation: it is up to the class which implements that interface to supply the implementation. By choosing to implement an interface, all methods declared in the interface *must* be defined in the class in order for it to be a *concrete class*.**
- **Remember the differences between an object's *apparent type* and its *actual type*, and think about how polymorphism can make your implementation simpler.**
- **For more information about the Java API Queue interface (as well as a list of Java API classes which already `implements Queue` that you can choose to extend, instead), refer to the documentation.**

SAMPLE INPUT/OUTPUT

```
// Comments in green, input in red, output in black.
// Your output should follow this format. Do not expect the numbers to be the same.

Starting simulator...

Enter the number of restaurants: 3
Enter the maximum number of customers a restaurant can serve: 5
Enter the arrival probability of a customer: 0.3
Enter the number of chefs: 5 // Note the number of chefs
Enter the number of simulation units: 10

Time: 1
Customer #1 has entered Restaurant 1.
Customer #2 has entered Restaurant 3.
Customer #1 has been seated with order "Chicken Tenders".
Customer #2 has been seated with order "Steak".
R1: {[#1, CT, 30 min.]} // CT normally takes 25 min, we have 2 extra chefs (-10 min), customer takes 15 min to eat
R2: {}
R3: {[#2, S, 35 min.]}

Time: 2
Customer #3 has entered Restaurant 1.
Customer #4 has entered Restaurant 2.
Customer #5 has entered Restaurant 3.
Customer #3 has been seated with order "Grilled Cheese".
Customer #4 has been seated with order "Cheeseburger".
Customer #5 has been seated with order "Steak".
R1: {[#1, CT, 25 min.], [#3, GC, 20 min.]}
R2: {[#4, C, 30 min.]}
R3: {[#2, S, 30 min.], [#5, S, 35 min.]}

Time: 3
Customer #6 has entered Restaurant 2.
Customer #7 has entered Restaurant 3.
Customer #6 has been seated with order "Chicken Tenders".
Customer #7 has been seated with order "Chicken Wing".
R1: {[#1, CT, 20 min.], [#3, GC, 15 min.]}
R2: {[#4, C, 25 min.]}, [#6, CT, 30 min.]}
R3: {[#2, S, 25 min.], [#5, S, 30 min.], [#7, CW, 35 min.]}

Time: 4
Customer #8 has entered Restaurant 1.
Customer #9 has entered Restaurant 1.
Customer #10 has entered Restaurant 1.
Customer #11 has entered Restaurant 3.
Customer #8 has been seated with the order "Cheeseburger".
Customer #9 has been seated with the order "Steak".
Customer #10 has been seated with the order "Cheeseburger".
Customer #11 has been seated with the order "Chicken Tenders".
R1: {[#1, CT, 15 min.], [#3, GC, 10 min.], [#8, C, 30 min.], [#9, S, 35 min.], [#10, C, 30 min.]
R2: {[#4, C, 20 min.]}, [#6, CT, 25 min.]}
R3: {[#2, S, 20 min.], [#5, S, 25 min.], [#7, CW, 30 min.], [#11, CT, 30 min.]}

Time: 5
Customer #12 has entered Restaurant 1.
Customer #13 has entered Restaurant 3.
Customer #12 cannot be seated! They have left the restaurant.
Customer #13 has been seated with the order "Grilled Cheese."
R1: {[#1, CT, 10 min.], [#3, GC, 5 min.], [#8, C, 25 min.], [#9, S, 30 min.], [#10, C, 25 min.]
R2: {[#4, C, 15 min.]}, [#6, CT, 20 min.]}
R3: {[#2, S, 15 min.], [#5, S, 20 min.], [#7, CW, 25 min.], [#11, CT, 25 min.], [#13, GC, 20 min.]}

Time: 6
Customer #3 has enjoyed their food! $10 profit.
Customer #14 has entered Restaurant 2.
Customer #15 has entered Restaurant 2.
Customer #16 has entered Restaurant 3..
Customer #14 has been seated with the order "Chicken Wings".
Customer #15 has been seated with the order "Chicken Wings".
Customer #16 cannot be seated! They have left the restaurant.
R1: {[#1, CT, 5 min.], [#8, C, 20 min.], [#9, S, 25 min.], [#10, C, 20 min.]}
R2: {[#4, C, 10 min.]}, [#6, CT, 15 min.], [#14, CW, 35 min.], [#15, CW, 35 min.]}
```

```
R3: {[#2, S, 10 min.], [#5, S, 15 min.], [#7, CW, 20 min.], [#11, CT, 20 min.], [#13, GC, 15 min.]}

Time: 7
Customer #1 has enjoyed their food! $10 profit.
Customer #17 has entered Restaurant 1.
Customer #18 has entered Restaurant 2.
Customer #17 has been seated with the order "Cheeseburger".
Customer #18 has been seated with the order "Grilled Cheese".
R1: {[#8, C, 15 min.], [#9, S, 20 min.], [#10, C, 15 min.], [#17, C, 30 min.]}
R2: {[#4, C, 5 min.]}, [#6, CT, 10 min.], [#14, CW, 30 min.], [#15, CW, 30 min.], [#18, GC, 20 min.]}
R3: {[#2, S, 5 min.], [#5, S, 10 min.], [#7, CW, 15 min.], [#11, CT, 15 min.], [#13, GC, 10 min.]}

Time: 8
Customer #4 has enjoyed their food! $15 profit.
Customer #2 has enjoyed their food! $25 profit.
Customer #19 has entered Restaurant 1.
Customer #19 has been seated with the order "Steak".
R1: {[#8, C, 10 min.], [#9, S, 15 min.], [#10, C, 10 min.], [#17, C, 25 min.], [#19, S, 35 min.] }
R2: {[#6, CT, 5 min.], [#14, CW, 25 min.], [#15, CW, 25 min.], [#18, GC, 15 min.]}
R3: {[#5, S, 5 min.], [#7, CW, 10 min.], [#11, CT, 10 min.], [#13, GC, 5 min.]}

Time: 9
Customer #6 has enjoyed their food! $10 profit.
Customer #5 has enjoyed their food! $25 profit.
Customer #13 has enjoyed their food! $10 profit.
Customer #20 has entered Restaurant 1.
Customer #21 has entered Restaurant 2.
Customer #22 has entered Restaurant 3.
Customer #20 cannot be seated! They have left the restaurant.
Customer #21 has been seated with the order "Chicken Wings".
Customer #22 has been seated with the order "Chicken Wings".
R1: {[#8, C, 5 min.], [#9, S, 10 min.], [#10, C, 5 min.], [#17, C, 20 min.], [#19, S, 30 min.]}
R2: {[#14, CW, 20 min.], [#15, CW, 20 min.], [#18, GC, 15 min.], [#21, CW, 35 min.]}
R3: {[#7, CW, 5 min.], [#11, CT, 5 min.], [#22, CW, 35 min.]}

Time: 10
Customer #8 has enjoyed their food! $15 profit.
Customer #10 has enjoyed their food! $15 profit.
Customer #7 has enjoyed their food! $20 profit.
Customer #11 has enjoyed their food! $10 profit.
Customer #23 has entered Restaurant 3.
Customer #23 has been seated with the order "Cheeseburger".
R1: {[#9, S, 5 min.], [#17, C, 15 min.], [#19, S, 25 min.]}
R2: {[#14, CW, 15 min.], [#15, CW, 15 min.], [#18, GC, 10 min.], [#21, CW, 30 min.]}
R3: {[#22, CW, 30 min.], [#23, C, 30 min.]}

Simulation ending...

// This is the resultant output. You will be graded on the trends through simulations with various parameters. You can find the requirements on the grading sheet for this assignment.
Total customer time: 325 minutes     // This is the total amount of time spent in restaurants by customers who have received their orders, paid, and left.
Total customers served: 11            // The total number of customers who received their orders, paid, and left.
Average customer time lapse: 29.55 minutes per order
Total Profit: $165
Customers that left: 3

Do you want to try another simulation? (y/n): n

Program terminating normally...
```