UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Department of Computing

Software Development II Coursework Report

Module	: 4COSC010C.3: Software Development II (2022)
Module Leader	: Mr. Deshan Sumanathilaka
Date of submission	: 17.07.2023
Student ID	: <20223160> / <w1986657>
Student First Name	: Raleena
Student Surname	: Fernando

"I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely my own. Any work from other authors is duly referenced and acknowledged."

Name : K.R Christien Fernando.

Student ID : 20223160

Test Cases

No	Test Case		Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After the program starts, 100 or VFQ	-	Displays 'empty' for all queues.	Displays 'empty' for all Queues.	Pass
2.	Enter 101 to display the empty queues	**2.1**	Display Empty queues: Cashier-1 Cashier-2 Cashier-3	Display Empty queues: Cashier-1 Cashier-2 Cashier-3	Pass
		2.2	Display Empty queues: Cashier-2 Cashier-3	Display Empty queues: Cashier-2 Cashier-3	Pass
		2.3	Display Empty queues: Cashier-3	Display Empty queues: Cashier-3	Pass
		2.4	No empty queues found	No empty queues found	Pass
3.	Add customer to Queue: Option 102	**3.1** Enter the queue number(1/2/3): 1	Selected queue number:1	Selected queue number:1	Pass

		3.2 Enter the customer Name: Mary	Customer name: Mary 1 Mary Customer added to the queue successfully	Customer name: Mary 1 Mary Customer added to the queue successfully	Pass
		3.3 Enter the queue number(1/2/3): 4	Invalid input! Queue number must be between 1 and 3. Enter the queue number (1 / 2 / 3):	Invalid input! Queue number must be between 1 and 3. Enter the queue number (1 / 2 / 3):	Pass
		3.4 Enter the queue number (1 / 2 / 3): 1	Queue 1 is full. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	Queue 1 is full. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	Pass
		3.5 Enter the queue number (1 / 2 / 3): a	Invalid input! Please enter a valid integer.	Invalid input! Please enter a valid integer.	Pass
		3.6 Enter the customer Name: Mark	Customer name: Mark 1 Mark Customer added to the queue successfully	Customer name: Mark 1 Mark Customer added to the queue successfully	Pass
		3.7 Enter the customer Name:	Invalid input! Customer name cannot be empty	Invalid input! Customer name cannot be empty	Pass
		3.8	No queues are available at the moment	No queues are available at the moment	Pass

		3.9	Insufficient burger stock. Cannot add customer.	Insufficient burger stock. Cannot add customer.	Pass
		3.10	Warning: Available Burger Stock Reached the minimum warning limit of 10 Please add burgers	Warning: Available Burger Stock Reached the minimum warning limit of 10 Please add burgers	Pass
		3.11 Enter the first name:	Invalid input! First name cannot be empty.	Invalid input! First name cannot be empty.	Pass
		3.12 Enter the first name: John	first name: John	first name: John	Pass
		3.13 Enter the second name: Flex	second name: Flex	second name: Flex	Pass
		3.14 Enter the number of required burgers :5	Entered burger count: 5 Customer added to the queue successfully. Queue Number: 1 Customer Name : John Flex	Entered burger count: 5 Customer added to the queue successfully. Queue Number: 1 Customer Name : John Flex	Pass
		3.15 Enter the number of required burgers :45	Warning: Available Burger Stock Reached the minimum warning limit of 10 Please add burgers	Warning: Available Burger Stock Reached the minimum warning limit of 10 Please add burgers	Pass

		3.16 Enter the second name:	Invalid input! Second name cannot be empty.	Invalid input! Second name cannot be empty.	Pass
		3.17 Enter the number of required burgers:5	Not enough burgers in the stock. Add more burgers	Not enough burgers in the stock. Add more burgers	Pass
		3.18 Enter the number of required burgers 0	Burgers required should be greater than zero Enter the number of required burgers	Burgers required should be greater than zero Enter the number of required burgers	Pass
		3.19 Enter the number of required burgers 5	No queues are available at the moment. Customer added to the waiting queue. Customer Name : John Flex	No queues are available at the moment. Customer added to the waiting queue. Customer Name : John Flex	Pass
4.	Remove customer from queue: Option number 103	**4.1** Enter the queue number (1 / 2 / 3):4	Invalid input! Queue number must be between 1 and 3.	Invalid input! Queue number must be between 1 and 3.	Pass
		4.2 Enter the queue number (1 / 2 / 3):vg	Invalid input! Please enter a valid integer	Invalid input! Please enter a valid integer	Pass
		4.3 Enter the queue number (1 / 2 / 3): 1	Queue 1 is empty. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	Queue 1 is empty. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	Pass

		4.4.	All queues are empty	All queues are empty	Pass
		4.5 Enter the queue number (1 / 2 / 3): 2	Selected queue number: 2	Selected queue number: 2	Pass
		4.6 Enter the queue position(index) [0-2]0	Customer removed from the queue successfully. Queue Number: 2 | Queue Index: 0	Customer removed from the queue successfully. Queue Number: 2 | Queue Index: 0	Pass
		4.7 Enter the queue position(index) [0-2]1	Selected location is empty	Customer removed from the queue successfully. Queue Number: 2 | Queue Index: 1	Fail
		4.8 Enter the queue position(index) [0-2]3	Invalid input! Queue index must be between 0 and 1 Enter the queue position(index) [0-1]	Invalid input! Queue index must be between 0 and 1 Enter the queue position(index) [0-1]	Pass
		4.9 Enter the queue position(index) [0-2]i	Invalid input! Please enter a valid integer	Invalid input! Please enter a valid integer	Pass
		4.10 Enter the queue position(index) [0-2]2	Selected location is empty	Selected location is empty	Pass

5.	Remove a served customer: Option number 104	**5.1** Enter the queue number (1 / 2 / 3): 1	Selected queue number: 1 Served Customer removed from the queue successfully. Queue Number: 1	Selected queue number: 1 Served Customer removed from the queue successfully. Queue Number: 1	Pass
		5.2 Enter the queue number (1 / 2 / 3): 4	Invalid input! Queue number must be between 1 and 3. Enter the queue number (1 / 2 / 3):	Invalid input! Queue number must be between 1 and 3. Enter the queue number (1 / 2 / 3):	Pass
		5.3 Enter the queue number (1 / 2 / 3): 1	Queue 1 is empty. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	Queue 1 is empty. Enter Different Queue Number Enter the queue number (1 / 2 / 3):	
		5.4	All queues are empty	All queues are empty	Pass
		5.5	Served Customer removed from the queue successfully. Queue Number: 1	Served Customer removed from the queue successfully. Queue Number: 1	Pass
		5.6 Enter the queue number (1 / 2 / 3): 1	Next customer in the waiting queue added successfully. Customer Name : Chris Mark Queue Number: 1	Next customer in the waiting queue added successfully. Customer Name : Chris Mark Queue Number: 1	Pass
6.					

7.	Store Program Data into file:Option number 106	**7.1**	Program data stored successfully.	Program data stored successfully.	Pass
8.	Load Program Data from file : Option number 107	**8.1**	Program data loaded successfully.	Program data loaded successfully.	Pass
9.	View Remaining burgers Stock: Option number 108	**9.1**	Remaining burgers stock: 35 Burgers stock on Hold: 10	Remaining burgers stock: 35 Burgers stock on Hold: 10	Pass
10.	Add burgers to Stock: Option number 109	**10.1** Enter the number of burgers to add: 20	Burgers added to the stock. New stock: 20	Burgers added to the stock. New stock: 20	Pass
		10.2 Enter the number of burgers to add: 100	Burger Stock is exceeding the maximum limit of 50 Enter the number of burgers to add:	Burger Stock is exceeding the maximum limit of 50 Enter the number of burgers to add:	Pass
		10.3 Enter the number of burgers to add: ab	Invalid input! Please enter a valid integer. Enter the number of burgers to add:	Invalid input! Please enter a valid integer. Enter the number of burgers to add:	Pass
11.	View Stock: Option number 110	**11.1**	Income from queue 01: 3250 Income from queue 02: 3250 Income from queue 03: 13000 Total Income : 19500	Income from queue 01: 3250 Income from queue 02: 3250 Income from queue 03: 13000 Total Income : 19500	Pass

12.	Input burger stock	**12.1**	Burger Stock is exceeding the maximum limit of “ MAX_Burgers	Burger Stock is exceeding the maximum limit of “ MAX_Burgers	Pass
		12.2	Invalid input! Please enter a valid integer.	Invalid input! Please enter a valid integer.	Pass
13.	Input Queue Index	**13.1**	"Enter the queue position(index) [0-"+(queue.size()-1)+"]: "	"Enter the queue position(index) [0-"+(queue.size()-1)+"]: "	Pass
		13.2	"Invalid input! Queue index for this queue must be between 0 and "+(queue.size()-1)"	"Invalid input! Queue index for this queue must be between 0 and "+(queue.size()-1)"	Pass
		13.3	"Invalid input! Please enter a valid integer."	"Invalid input! Please enter a valid integer."	Pass
14.	Input Name	**14.1**	“Enter the “ + option + “name”	“Enter the “ + option + “name”	Pass

Discussion

<<Discussion of how you chose your test cases to ensure that your tests cover all aspects of your program

---

Test case No 1.1—Input: User enters option 100 to display all queues. In the beginning, all the queues must have to be empty.

Test case No 2.1, 2.2, 2.3— Input: User enters option 101 to display all empty queues. Empty queues will display according to the added customers.

Test case No 3.1— Input: The user enters option 102 to add customers to the queues and selects the queue number as the user's choice.

Test case No 3.2— Input: The user enters option 102 to add customers to the queues and selects the queue number as the user's choice. After the queue selection, the user must enter the customer's name.

Test case No 3.3— Input: The user enters option 102 to add customers to the queues and selects the queue number as the user's choice. Whenever the user inputs an invalid integer there will be an error message. This test case checks if the program handles an invalid queue number input and prompts the user to enter a valid queue number.

Test case No 3.4— Input: The user enters option 102 to add customers to the queues and selects a full queue number. The program displays an error message. This test case checks if the program handles the scenario where the selected queue is full and prompts the user to select a different queue.

Test case No 3.5— Input: The user enters option 102 to add customers to the queues and selects the queue number as A. The program displays an error message. This test case checks if the program handles an invalid queue number input and prompts the user to enter a valid integer.

Test case No 3.6— Input: The user enters option 102 to add customers to the queues and enters the customer name. This test case checks if the program correctly adds a customer to the selected queue and displays a success message.

Test case No 3.7, 3.16— Input: The user enters option 102 to add customers to the queues without entering a customer name. This test case checks if the program handles the scenario where the user tries to add a customer without providing a name and prompts them to enter a valid name.

Test case No 3.8— Input: The user enters option 102 to add customers to the queue but all queues are full. This test case checks if the program handles the scenario where all queues are full and inform the user that no queues are available.

Test case No 3.9— Input: The user enters option 102 to add customers to the queue but there is insufficient burger stock. This test case checks if the program handles the scenario where there is not enough burger stock to add a customer to the queue and informs the user about the insufficient stock.

Test case No 3.10— Input: The user enters option 102 to add customers to the queue, and the available burger stock is near the warning limit. This test case checks if the program alerts the user when the available burger stock reaches the minimum warning limit and prompts them to add more burgers.

Test case No 3.11— Input: The user enters option 102 to add customers to a queue and does not enter the first name. This test case checks if the program handles the scenario where the user does not enter a valid first name and prompts them to enter a valid input.

Test case No 3.12, 3.13— Input: The user enters option 102 to add customers to a queue and enter the first name and second name. This test case checks if the program correctly accepts and displays the first name and the second name of the customer.

Test case No 3.14— Input: The user enters option 102 to add customers to a queue and enters the number of required burgers. This test case checks if the program adds a customer to the selected queue with the specified number of required burgers and displays the success message along with the queue number and customer name.

Test case No 3.15— Input: The user enters option 102 to add customers to a queue and enters a high number of required burgers. This test case checks if the program warns the user when the available burger stock reaches the minimum warning limit and prompts them to add more burgers.

Test case No 3.17— Input: The user enters option 102 to add customers to a queue and enters the number of required burgers, but there is insufficient burger stock. This test case checks if the program handles the scenario where there is not enough burger stock to fulfill the customer’s order and prompts the user to add more burgers.

Test case No 3.18— Input: The user enters option 102 to add customers to a queue and enters the number of required burgers as 0. This test case checks if the program handles the scenario where the user enters 0 as the number of required burgers and prompts them to enter a valid quantity.

Test case No 3.19—This test case,the user is prompted to enter the number of required burgers, which is set to 5. However, at that moment, there are no available queues in the Foodies Food Center. This means that all the Food queues are full. Instead of adding the customer to a specific Food queue, the program adds the customer to the Waiting List queue since there are no available queues at the moment. This ensures that the customer is still added to the system and can be served as soon as a Food queue becomes available.

Test case No 4.1— Input: The user enters option 103 to remove a customer from a queue and enters an invalid queue number. This test case checks if the program handles invalid queue number input when removing a customer from a queue and prompts the user to enter a valid queue number.

Test case No 4.2— Input: The user enters option 103 to remove a customer from a queue and enters an invalid queue number. This test case checks if the program handles invalid queue number input when removing a customer from a queue and prompts the user to enter a valid integer.

Test case No 4.3— Input: The user enters option 103 to remove a customer from a queue and selects an empty queue number. This test case checks if the program handles the scenario where a selected queue is empty and prompts the user to select a different queue.

Test case No 4.4— This test case is important for the program to provide accurate information about the state of the queues so the user can select an appropriate option.

Test case No 4.5— Input: The user enters option 103 to remove a customer from a queue and selects a non-empty queue, and enters an invalid queue position. This test case checks if the program handels an invalid queue index input when removing a customer from a queue and prompts the user to enter a valid index.

Test case No 4.6— Input: The user enters option 103 to remove a customer from a queue and selects a non-empty queue, and enters a valid queue position. This test case checks if the program correctly removes a customer from the selected queue at the specified position and displays a success message along with the queue number.

Test case No 4.7— Input: The user enters option 103 to remove a customer from a queue and selects a non-empty queue, and enters a queue position that is empty. This test case handles the scenario where the selected queue position is empty and notifies the user about it.

Test case No 4.8— Input: The user enters option 103 to remove a customer from a queue and enters an invalid queue index for a queue with two elements. This test case checks if the program handles invalid queue index input when removing a customer from a queue and prompts the user to enter a valid index.

Test case No 5.1— Input: The user enters option 104 to remove a served customer from a queue and selects a valid queue number. This test case checks if the program correctly removes a served customer from the selected queue and displays a success message.

Test case No 5.2— Input: The user enters option 104 to remove a served customer from a queue and selects an invalid queue number. This test case checks if the program handles an invalid queue number input when removing a served customer from a queue and prompts the user to enter a valid queue number.

Test case No 5.3— Input: The user enters option 104 to remove a served customer from a queue and selects an empty queue. This test case checks if the program handles the scenario where the select queue is empty and prompts the user to select a different queue.

Test case No 5.4—This test case checks whether all the queues are empty. The expected outcome is that all queues should be empty.

Test case No 5.5—This test case is to maintain the integrity of the queue system and ensure that customers are removed from the queue once they have been served. This allows the Foodies Food Center to efficiently serve customers and manage the queues effectively.

Test case No 5.6— This test case ensures that when a served customer is removed from a Food queue, the next customer in the Waiting List queue is automatically moved to the Food queue, ensuring a continuous flow of customers being served.

Test case No7.1—This test case verifies whether the program data can be successfully stored in a file. The expected outcome is that the program data is stored without any errors.

Test case No8.1— This test case checks whether the program data can be successfully loaded from a file. The expected outcome is that the program data is loaded without any errors.

Test case No9.1— This test case displays the remaining burger stock and the burgers on hold. The expected outcome is to see the correct values for the remaining burger stock and the burgers on hold.

Test case No10.1— This test case allows the user to add a specific number of burgers to the stock. The expected outcome is that the burgers are added to the stock, and the new stock value is displayed.

Test case No10.2— This test case checks what happens when the user tries to add more burgers than the maximum limit allows. The expected outcome is to receive a message indicating that the burger stock is exceeding the maximum limit.

Test case No10.3— This test case checks the input validation when the user enters invalid input (non-integer value) for adding burgers to the stock. The expected outcome is to receive a message indicating that the input is invalid.

Test case No11— This test case displays the income from each queue and the total income. The expected outcome is to see the correct income values for each queue and the total income.

Test case No12.1— In the test case "Burger Stock is exceeding the maximum limit of MAX_Burgers," the program checks if the input for adding burgers to the stock exceeds the maximum limit defined by MAX_Burgers. If the input exceeds the limit, the program should display a message indicating that the burger stock is exceeding the maximum limit.

Test case No12.2— In the test case "Invalid input! Please enter a valid integer," the program checks if the user enters an invalid input when prompted to enter the number of burgers to add to the stock. The program expects the user to enter a valid integer. If the user enters an invalid input, such as a non-integer value, the program should display an error message indicating that the input is invalid.

Test case No13.1— In the test case "Enter the queue position(index) [0-(queue.size()-1)]:," the program prompts the user to enter the queue position or index. The program expects the user to enter a valid integer within the range of 0 to the size of the queue minus 1. This test case checks if the prompt is displayed correctly.

Test case No13.2— In the test case "Invalid input! Queue index for this queue must be between 0 and (queue.size()-1)," the program checks if the user enters an invalid input for the queue index. The program expects the user to enter a valid integer within the range of 0 to the size of the queue minus 1. If the user enters an invalid input, the program should display an error message indicating the valid range of the queue index.

Test case No13.3— In the test case "Invalid input! Please enter a valid integer," the program checks if the user enters an invalid input when prompted to enter the queue index. The program expects the user to enter a valid integer. If the user enters an invalid input, such as a non-integer value, the program should display an error message indicating that the input is invalid.

Test case No14.1— In the test case "Enter the option + name," the program prompts the user to enter a name based on a specified option. This test case checks if the prompt is displayed correctly, with the option dynamically added to the message. For example, if the option is "customer," the prompt should ask the user to enter the customer's name.

>>

Code :

<<paste your code>>

Task_1

```
package task_1;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class FoodiesFoodCenter {

    private static String[] cashier1 = new String[2];

    private static String[] cashier2 = new String[3];

    private static String[] cashier3 = new String[5];

    private static final int MAX_BURGERS = 50;

    static final int BURGER_STOCK_WARNING_LIMIT=10;

    private static final int MAX_BURGERS_PER_CUSTOMER = 5;

    private static int burgerStock = MAX_BURGERS;

    private static int burgersOnHold = 0;

    public static void main(String[] args) {
        Arrays.fill(cashier1,"empty");
        Arrays.fill(cashier2,"empty");
        Arrays.fill(cashier3,"empty");

        String option;
        Scanner scanner = new Scanner(System.in);
        do {
            displayMenu();
            option =  scanner.nextLine();;
            // Consume newline character
            switch (option) {
                case "100":
                    viewAllQues();
                    break;
                case "101":
                    viewEmptyQueues();
                    break;
                case "102":
                    addCustomerToTheQueue();
                    break;
```

```
                case "103":
                    removeCustomerFromQueue();
                    break;
                case "104":
                    removeServedCustomerFromQueue();
                    break;
                case "105":
//                      viewSortedCustomers();
                    break;
                case "106":
                    storeProgramData();
                    break;
                case "107":
                    loadProgramData();
                    break;
                case "108":
                    viewBurgerStock();
                    break;
                case "109":
                    addBurgersToStock();
                    break;
                case "999":
                    System.out.println("Exiting the program. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        } while (!option.equals("999"));
    }

    private static void displayMenu() {
        System.out.println("");
        System.out.println("========== Foodies Fave Food Center ==========");
        System.out.println("100 or VFQ: View all Queues");
        System.out.println("101 or VEQ: View all Empty Queues");
        System.out.println("102 or ACQ: Add customer to a Queue");
        System.out.println("103 or RCQ: Remove a customer from a Queue (From
a specific location)");
        System.out.println("104 or PCQ: Remove a served customer");
        System.out.println("105 or VCS: View Customers Sorted in alphabetical
order");
        System.out.println("106 or SPD: Store Program Data into file");
        System.out.println("107 or LPD: Load Program Data from file");
        System.out.println("108 or STK: View Remaining burgers Stock");
        System.out.println("109 or AFS: Add burgers to Stock");
        System.out.println("999 or EXT: Exit the Program");
        System.out.println("============================================");
        System.out.print("Enter an option: ");
    }

    // ------------------- 100 viewAllQues ------------------------------

    private static void viewAllQues(){
        System.out.println("*****************");
        System.out.println("*   Cashiers    *");
        System.out.println("*****************");
```

```
        for (int i = 0; i < cashier3.length; i++) {
            System.out.print(" ");
            if(i<cashier1.length){
                printQueueElement(cashier1[i]);
            }else{
                System.out.print("  -  ");
            }
            if(i<cashier2.length){
                printQueueElement(cashier2[i]);
            }else{
                System.out.print("  -  ");
            }
            if(i<cashier3.length){
                printQueueElement(cashier3[i]);
            }
            System.out.println("      ");

        }
        System.out.println("X-Not Occupied   O-Occupied");

    }

    // ------------------- 101 viewEmptyQueues --------------------------

    private static void viewEmptyQueues() {
        List<String> emptyQueues = getEmptyQueues();
        if (emptyQueues.isEmpty()) {
            System.out.println("No empty queues found.");//test case no 2.4
        } else {
            System.out.println("Empty queues:");//test case no 2.1, 2.2, 2.3
            {
                for (String queueName : emptyQueues) {
                    System.out.println(queueName);
                }
            }
        }
    }

    // ------------------- 102 addCustomerToTheQueue ---------------------

    private static void addCustomerToTheQueue() {
        if (!isAnyQueueAvailable()){
            System.out.println("No queues are available at the
moment");//test case no 3.8
        } else {
            if ((burgerStock-burgersOnHold) >= MAX_BURGERS_PER_CUSTOMER) {
                int queueNumber = inputQueueNumber("ADD");
                String customerName = inputCustomerName();
                int queueIndex = getSmallestQueueIndex(queueNumber);
                String [] queue = getQueueByQueueNumber(queueNumber);
                queue[queueIndex] = customerName;

                burgersOnHold += MAX_BURGERS_PER_CUSTOMER;

                System.out.println("Customer added to the queue
successfully.");//test case no 3.2
                System.out.println("Queue Number: "+ queueNumber);
```

```
                System.out.println("Customer Name : "+ customerName);

                if((burgerStock-burgersOnHold) <=
BURGER_STOCK_WARNING_LIMIT){
                    System.out.println("Warning: Available Burger Stock
Reached the minimum warning limit of "+BURGER_STOCK_WARNING_LIMIT);//test
case no 3.10
                    System.out.println("Please add burgers");
                }
            } else {
                System.out.println("Insufficient burger stock. Cannot add
customer.");//test case no 3.9
            }
        }

    }

    // ------------------- 103 removeCustomerFromQueue -------------------

    private static void removeCustomerFromQueue(){
        if(areAllQueuesEmpty()){
            System.out.println("All queues are empty");
        }else {
            int queueNumber = inputQueueNumber("REMOVE");
            int queueIndex = inputQueueIndex(queueNumber);
            shiftElementsToLeft(queueNumber, queueIndex);
            burgersOnHold -= MAX_BURGERS_PER_CUSTOMER;
            System.out.println("Customer removed from the queue
successfully.");//test case no 4.4
            System.out.println("Queue Number: "+ queueNumber + " | Queue
Index: "+queueIndex);;
        }
    }

    // ------------------- 104 removeServedCustomerFromQueue -------------

    private static void removeServedCustomerFromQueue(){
        if(areAllQueuesEmpty()){
            System.out.println("All queues are empty");//test case no5.4
        }else {
            int queueNumber = inputQueueNumber("REMOVE");
            int queueIndex = 0;
            shiftElementsToLeft(queueNumber, queueIndex);
            burgersOnHold -= MAX_BURGERS_PER_CUSTOMER;
            burgerStock -= MAX_BURGERS_PER_CUSTOMER;
            System.out.println("Served Customer removed from the queue
successfully.");//test case no 5.1
            System.out.println("Queue Number: "+ queueNumber);
        }
    }

    // ------------------- //TODO - 105 viewSortedCustomers --------------


    // ------------------- 106 storeProgramData --------------------------
    private static void storeProgramData(){
        try{
```

```
            BufferedWriter writer= new BufferedWriter(new
FileWriter("program_data.txt"));

            writer.write(Arrays.toString(cashier1));
            writer.newLine();
            writer.write(Arrays.toString(cashier2));
            writer.newLine();
            writer.write(Arrays.toString(cashier3));
            writer.close();

            System.out.println("Program data stored successfully.");//test
case no 7.1
        } catch (IOException e){
            System.out.println("An error occurred while storing the program
data");
        }
    }

    // ------------------- 107 loadProgramData -------------------------

    private static void loadProgramData(){
        try{
            BufferedReader reader = new BufferedReader(new
FileReader("program_data.txt"));

            String line;
            if((line = reader.readLine()) != null){
                cashier1 = parseQueue(line);
            }

            if((line = reader.readLine()) != null){
                cashier2 = parseQueue(line);
            }

            if((line = reader.readLine()) != null){
                cashier3 = parseQueue(line);
            }
            reader.close();

            System.out.println("Program data loaded successfully.");//test
case no 8.1
        } catch (IOException e){
            System.out.println("An error occurred while loading the program
data.");
        }
    }

    private static String[] parseQueue(String line){
        line = line.trim();
        line = line.substring(1, line.length() -1);
        String [] elements = line.split(", ");
        int length = elements.length;
        String[] queue = new String[length];
        for (int i = 0; i< length; i++){
            queue[i] = elements[i];
        }
        return queue;
```

```
    }


    // ------------------- viewBurgerStock -----------------------------

    private static void viewBurgerStock(){
        System.out.println("Remaining burgers stock: " + burgerStock);//test
case no 9.1
        System.out.println("Burgers stock on Hold: " + burgersOnHold);//test
case no 9.1
    }

    // ------------------- addBurgersToStock -----------------------------

    private static void addBurgersToStock() {
        int burgersToAdd = inputBurgerStock();
        burgerStock += burgersToAdd;
        System.out.println("Burgers added to the stock. New stock: " +
burgerStock); //test case no 10.1
    }

    // ------------------- user inputs ----------------------------------
    private static int inputQueueIndex(int queueNumber){
        String[] queue = getQueueByQueueNumber(queueNumber);
        Scanner scanner = new Scanner(System.in);
        int queueIndex = -1;
        boolean isValidInput = false;

        do {
            System.out.print("Enter the queue position(index) [0-
"+(queue.length-1)+"]");
            if (scanner.hasNextInt()) {
                queueIndex = scanner.nextInt();
                if (queueIndex >= 0 && queueIndex <= (queue.length-1)) {
                    if(queue[queueIndex].equals("empty")){
                        System.out.println("Selected location is
empty");//test case no 4.10
                    }else {
                        isValidInput = true;
                    }
                }else {
                    System.out.println("Invalid input! Queue index must be
between 0 and "+(queue.length-1));//test case 4.8
                }
            }else{
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 4.9
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        }while (!isValidInput);
        return queueIndex;
    }

    private static int inputQueueNumber(String option){
        Scanner scanner = new Scanner(System.in);
        int queueNumber = -1;
```

```
        boolean isValidInput = false;
        do {
            System.out.print("Enter the queue number (1 / 2 / 3): ");//test
case 4.5
            if (scanner.hasNextInt()) {
                queueNumber = scanner.nextInt();
                if (queueNumber >= 1 && queueNumber <= 3) {
                    if (option.equals("ADD")){
                        if (getSmallestQueueIndex(queueNumber) != -1){
                            isValidInput = true;
                        }else {
                            System.out.println("Queue " + queueNumber+ " is
full. Enter Different Queue Number" ); // test case no 4.4
                        }
                    }else if (option.equals("REMOVE")){
                        if (getSmallestQueueIndex(queueNumber) != 0){
                            isValidInput = true;
                        }else {
                            System.out.println("Queue " + queueNumber+ " is
empty. Enter Different Queue Number" ); //test case no 4.3
                        }
                    }
                } else {
                    System.out.println("Invalid input! Queue number must be
between 1 and 3."); //test case 4.1
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");//test case 4.2
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        } while (!isValidInput);

        System.out.println("Selected queue number: " + queueNumber);
        return queueNumber;
    }

    private static String inputCustomerName(){
        Scanner scanner = new Scanner(System.in);
        String customerName;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the customer name: ");//test case no 3.6
            customerName = scanner.nextLine();

            if (customerName.trim().isEmpty()) {
                System.out.println("Invalid input! Customer name cannot be
empty.");//test case no 3.7
            } else {
                isValidInput = true;
            }
        } while (!isValidInput);

        System.out.println("Customer name: " + customerName);
```

```
        return customerName;

    }

    private static int inputBurgerStock(){
        Scanner scanner = new Scanner(System.in);
        int burgersToAdd = 0;
        boolean isValidInput = false;
        do {
            System.out.print("Enter the number of burgers to add: ");//test
case no 10.1
            if (scanner.hasNextInt()) {
                burgersToAdd = scanner.nextInt();
                if(burgerStock + burgersToAdd <= MAX_BURGERS){
                    isValidInput =true;
                }else {
                    System.out.println("Burger Stock is exceeding the maximum
limit of "+MAX_BURGERS);//test case no 10.2
                }
            }else {
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 10.3
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }

        }while (!isValidInput);

        return burgersToAdd;
    }

    // ------------------- common methods --------------------------------

    private static void printQueueElement(String element){
        if(element.equals("empty")){
            System.out.print("  X  "); // X-Not Occupied
        }else{
            System.out.print("  O  "); // O-Occupied
        }
    }

    private static int getSmallestQueueIndex(int queueNumber){
        int index = -1;
        String[] queue = getQueueByQueueNumber(queueNumber);
        for (int i = 0; i < queue.length; i++) {
            if(queue[i].equals("empty")){
                index = i;
                break;
            }
        }
        return index;
    }

    private static String[] getQueueByQueueNumber(int queueNumber){
        String[] queue = new String[5];
        switch (queueNumber){
```

```
            case 1 : {
                queue = cashier1;
                break;
            }
            case 2 :{
                queue = cashier2;
                break;
            }
            case 3 :{
                queue = cashier3;
                break;
            }
        }
        return queue;
    }

    private static List<String> getEmptyQueues() {
        List<String> emptyQueueNames = new ArrayList<>();

        if(cashier1[0].equals("empty")){
            emptyQueueNames.add("Cashier-1");
        }
        if(cashier2[0].equals("empty")){
            emptyQueueNames.add("Cashier-2");
        }
        if(cashier3[0].equals("empty")){
            emptyQueueNames.add("Cashier-3");
        }

        return  emptyQueueNames;

    }

    private static boolean isAnyQueueAvailable(){
        return (getSmallestQueueIndex(1) != -1) ||
                (getSmallestQueueIndex(2) != -1) ||
                (getSmallestQueueIndex(3) != -1);
    }

    private static boolean areAllQueuesEmpty(){
        return (getSmallestQueueIndex(1) == 0) &&
                (getSmallestQueueIndex(2) == 0) &&
                (getSmallestQueueIndex(3) == 0);
    }

    private static void shiftElementsToLeft(int queueNumber, int startIndex)
{
        String[] queue = getQueueByQueueNumber(queueNumber);
        for (int i = startIndex; i < queue.length - 1; i++) {
            queue[i] = queue[i + 1];
        }
        queue[queue.length - 1] = "empty";
    }
}
```

Task_2

```
package task_2;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class FoodiesFoodCenter {

    private static FoodQueue cashier1 = new FoodQueue(1);
    private static FoodQueue cashier2 = new FoodQueue(2);
    private static FoodQueue cashier3 = new FoodQueue(3);
    private static final int[] MAX_CUSTOMERS = {2, 3, 5};

    private static final int MAX_BURGERS = 50;

    static final int BURGER_STOCK_WARNING_LIMIT=10;

    private static final int MAX_BURGERS_PER_CUSTOMER = 5;

    private static int burgerStock = MAX_BURGERS;

    private static int burgersOnHold = 0;

    public static void main(String[] args) {

        String option;
        Scanner scanner = new Scanner(System.in);
        do {
            displayMenu();
            option =  scanner.nextLine();;
            // Consume newline character
            switch (option) {
                case "100":
                    viewAllQues();
                    break;
                case "101":
                    viewEmptyQueues();
                    break;
                case "102":
                    addCustomerToTheQueue();
                    break;
                case "103":
                    removeCustomerFromQueue();
                    break;
                case "104":
                    removeServedCustomerFromQueue();
                    break;
                case "105":
//                    viewSortedCustomers();
                    break;
                case "106":
```

```
                    storeProgramData();
                    break;
                case "107":
                    loadProgramData();
                    break;
                case "108":
                    viewBurgerStock();
                    break;
                case "109":
                    addBurgersToStock();
                    break;
                case "110":
                    viewIncome();
                    break;
                case "999":
                    System.out.println("Exiting the program. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        } while (!option.equals("999"));
    }

    private static void displayMenu() {
        System.out.println("");
        System.out.println("========== Foodies Fave Food Center ==========");
        System.out.println("100: View all Queues");
        System.out.println("101: View all Empty Queues");
        System.out.println("102: Add customer to a Queue");
        System.out.println("103: Remove a customer from a Queue (From a
specific location)");
        System.out.println("104: Remove a served customer");
        System.out.println("105: View Customers Sorted in alphabetical
order");
        System.out.println("106: Store Program Data into file");
        System.out.println("107: Load Program Data from file");
        System.out.println("108: View Remaining burgers Stock");
        System.out.println("109: Add burgers to Stock");
        System.out.println("110: View income");
        System.out.println("999: Exit the Program");
        System.out.println("=============================================");
        System.out.print("Enter an option: ");
    }

    // ------------------- 100 viewAllQues -----------------------------

    private static void viewAllQues(){
        System.out.println("*****************");
        System.out.println("*   Cashiers    *");
        System.out.println("*****************");

        for (int i = 0; i < MAX_CUSTOMERS[2]; i++) {
            System.out.print(" ");
            if(i<MAX_CUSTOMERS[0]){
                if((i+1) <= cashier1.size()){
                    printQueueElement("not_empty");
                }else {
```

```
                printQueueElement("empty");
            }
        }else{
            System.out.print("  -  ");
        }

        if(i<MAX_CUSTOMERS[1]){
            if((i+1) <= cashier2.size()){
                printQueueElement("not_empty");
            }else {
                printQueueElement("empty");
            }
        }else{
            System.out.print("  -  ");
        }


        if((i+1) <= cashier3.size()){
            printQueueElement("not_empty");
        }else {
            printQueueElement("empty");
        }

        System.out.println("      ");
    }
    System.out.println("X-Not Occupied   O-Occupied");

}

// ------------------- 101 viewEmptyQueues --------------------------

private static void viewEmptyQueues() {
    List<String> emptyQueues = getEmptyQueues();
    if (emptyQueues.isEmpty()) {
        System.out.println("No empty queues found.");
    } else {
        System.out.println("Empty queues:");
        {
            for (String queueName : emptyQueues) {
                System.out.println(queueName);
            }
        }
    }
}

// ------------------- 102 addCustomerToTheQueue ---------------------

private static void addCustomerToTheQueue() {
    if (!isAnyQueueAvailable()){
        System.out.println("No queues are available at the moment");
    } else {
        String firstName = inputName("first");//test case 3.11, 3.12
        String secondName = inputName("second");//test case 3.13
        int burgersRequired = inputBurgers(); //test case 3.14

        if ((burgerStock-burgersOnHold) >= burgersRequired) {
            Customer customer = new Customer(firstName, secondName,
```

```
burgersRequired);
                FoodQueue foodQueue = getNextQueue();
                foodQueue.addCustomer(customer);

                burgersOnHold += burgersRequired;

                System.out.println("Customer added to the queue
successfully.");//test case 3.14
                System.out.println("Queue Number: "+
foodQueue.getIndex());//test case 3.14
                System.out.println("Customer Name : "+
customer.getFullName());//test case 3.14

                if((burgerStock-burgersOnHold) <=
BURGER_STOCK_WARNING_LIMIT){
                    System.out.println("Warning: Available Burger Stock
Reached the minimum warning limit of "+BURGER_STOCK_WARNING_LIMIT);//test
case 3.15
                    System.out.println("Please add burgers");//test case 3.15
                }
            } else {
                System.out.println("Insufficient burger stock. Cannot add
customer.");//test case 3.9
            }
        }

    }

    // ------------------- 103 removeCustomerFromQueue -------------------

    private static void removeCustomerFromQueue(){
        if(areAllQueuesEmpty()){
            System.out.println("All queues are empty");//test case no 4.4
        }else {
            int queueNumber = inputQueueNumber("REMOVE");
            FoodQueue foodQueue = getQueueByQueueNumber(queueNumber);
            int queueIndex = inputQueueIndex(queueNumber);
            burgersOnHold -=
foodQueue.getCustomer(queueIndex).getBurgersRequired();
            foodQueue.removeCustomer(queueIndex);
            System.out.println("Customer removed from the queue
successfully.");//test case no 4.6
            System.out.println("Queue Number: "+ queueNumber + " | Queue
Index: "+queueIndex);;//test case no 4.6
        }
    }

    // ------------------- 104 removeServedCustomerFromQueue -------------
    private static void removeServedCustomerFromQueue(){
        if(areAllQueuesEmpty()){
            System.out.println("All queues are empty");
        }else {
            int queueNumber = inputQueueNumber("REMOVE");
            FoodQueue foodQueue = getQueueByQueueNumber(queueNumber);
            int queueIndex = 0;
            int burgersSold =
foodQueue.getCustomer(queueIndex).getBurgersRequired();
```

```
            burgersOnHold -= burgersSold;
            burgerStock -= burgersSold;
            foodQueue.addSoldBurgers(burgersSold);
            foodQueue.removeCustomer(queueIndex);
            System.out.println("Served Customer removed from the queue
successfully.");//test case no 5.1
            System.out.println("Queue Number: "+ queueNumber);//test case no
5.1
        }
    }

    // ------------------- //TODO - 105 viewSortedCustomers --------------


    // ------------------- 106 storeProgramData --------------------------

    private static void storeProgramData(){
        try{
            BufferedWriter writer= new BufferedWriter(new
FileWriter("program_data_02.txt"));

            writer.write(cashier1.toString());
            writer.newLine();
            writer.write(cashier2.toString());
            writer.newLine();
            writer.write(cashier3.toString());
            writer.close();

            System.out.println("Program data stored successfully.");//test
case no 7.1
        } catch (IOException e){
            System.out.println("An error occurred while storing the program
data");
        }
    }

    // ------------------- 107 loadProgramData --------------------------

    private static void loadProgramData(){
        try{
            BufferedReader reader = new BufferedReader(new
FileReader("program_data_02.txt"));

            String line;
            if((line = reader.readLine()) != null){
                cashier1 = parseQueue(line);
            }

            if((line = reader.readLine()) != null){
                cashier2 = parseQueue(line);
            }

            if((line = reader.readLine()) != null){
                cashier3 = parseQueue(line);
            }
            reader.close();
```

```
            System.out.println("Program data loaded successfully.");//test
case no 8.1
        } catch (IOException e){
            System.out.println("An error occurred while loading the program
data.");
        }
    }

    private static FoodQueue parseQueue(String line){
        FoodQueue queue = new FoodQueue();
        line = line.trim();
        line = line.substring(1, line.length() -1);
        String [] customers = line.split(";");
        for (String el: customers ) {
            String[] customerString = el.split(",");
            Customer customer = new Customer(customerString[0],
customerString[1], Integer.parseInt(customerString[2]));
            queue.addCustomer(customer);
        }

        return queue;
    }

    // ------------------- 108 viewBurgerStock ------------------------------
-
    private static void viewBurgerStock(){
        System.out.println("Remaining burgers stock: " + burgerStock);//test
case no 9.1
        System.out.println("Burgers stock on Hold: " + burgersOnHold);//test
case no 9.1
    }

    // ------------------- 109 addBurgersToStock ----------------------------
-

    private static void addBurgersToStock() {
        int burgersToAdd = inputBurgerStock();
        burgerStock += burgersToAdd;
        System.out.println("Burgers added to the stock. New stock: " +
burgerStock);//test case no 10.1
    }

    // ------------------- 110 viewIncome -------------------------------

    private static void viewIncome(){
        int burgerPrice= 650;
        int q1Income = cashier1.getSoldBurgers()*burgerPrice;
        int q2Income = cashier2.getSoldBurgers()*burgerPrice;
        int q3Income = cashier3.getSoldBurgers()*burgerPrice;
        System.out.println( "Income from queue 01: "+ q1Income);//test case
11.1
        System.out.println( "Income from queue 02: "+ q2Income);//test case
11.1
        System.out.println( "Income from queue 03: "+ q3Income);//test case
11.1
        System.out.println( "Total Income : "+
(q1Income+q2Income+q3Income));//test case 11.1
```

```
    }

    // ------------------- user inputs ----------------------------------

    private static String inputName(String option){
        Scanner scanner = new Scanner(System.in);
        String name;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the "+option+" name: ");//test case no
14.1
            name = scanner.nextLine();

            if (name.trim().isEmpty() || name.trim().contains(" ")
                    || name.trim().contains(",") || name.trim().contains(";")
|| name.trim().contains("@")
            ) {
                System.out.println("Invalid input! "+option+" name cannot be
empty and have space characters ");//test case 3.11, 3.12
            } else {
                isValidInput = true;
            }
        } while (!isValidInput);

        System.out.println(option+" name: " + name);

        return name;

    }

    private static int inputBurgers(){
        Scanner scanner = new Scanner(System.in);
        int burgerCount = 0;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the number of required burgers: ");//test
case 3.17
            if (scanner.hasNextInt()) {
                burgerCount = scanner.nextInt();
                if(burgerCount==0){
                    System.out.print("Burgers required should be greater than
zero. ");//test case 3.18
                } else if (burgerCount > 0 && burgerCount <= (burgerStock-
burgersOnHold)) {
                    isValidInput = true;
                } else {
                    System.out.println("Not enough burgers in the stock. Add
more burgers ");//test case 3.17
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
```

```
        } while (!isValidInput);

        System.out.println("Entered burger count: " + burgerCount);
        return burgerCount;
    }

    private static int inputQueueNumber(String option){
        Scanner scanner = new Scanner(System.in);
        int queueNumber = -1;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the queue number (1 / 2 / 3): ");//test
case 3.1
            if (scanner.hasNextInt()) {
                queueNumber = scanner.nextInt();
                if (queueNumber >= 1 && queueNumber <= 3) {
                    if (option.equals("ADD")){
                        if (getSmallestQueueIndex(queueNumber) != -1){
                            isValidInput = true;
                        }else {
                            System.out.println("Queue " + queueNumber+ " is
full. Enter Different Queue Number: " );//test case 3.4
                        }
                    }else if (option.equals("REMOVE")){
                        if (getSmallestQueueIndex(queueNumber) != 0){
                            isValidInput = true;
                        }else {
                            System.out.println("Queue " + queueNumber+ " is
empty. Enter Different Queue Number" );
                        }
                    }
                } else {
                    System.out.println("Invalid input! Queue number must be
between 1 and 3.");
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        } while (!isValidInput);

        System.out.println("Selected queue number: " + queueNumber);
        return queueNumber;
    }

    private static int inputQueueIndex(int queueNumber){
        FoodQueue queue = getQueueByQueueNumber(queueNumber);
        Scanner scanner = new Scanner(System.in);
        int queueIndex = -1;
        boolean isValidInput = false;

        do {
            System.out.print("Enter the queue position(index) [0-
"+(queue.size()-1)+"]: ");//test case no 13.1
```

```
            if (scanner.hasNextInt()) {
                queueIndex = scanner.nextInt();
                if (queueIndex >= 0 && queueIndex <= (queue.size()-1)) {
                    isValidInput = true;
                }else {
                    System.out.println("Invalid input! Queue index for this
queue must be between 0 and "+(queue.size()-1));//test case no 13.2
                }
            }else{
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 13.3
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        }while (!isValidInput);
        return queueIndex;
    }

    private static int inputBurgerStock(){
        Scanner scanner = new Scanner(System.in);
        int burgersToAdd = 0;
        boolean isValidInput = false;
        do {
            System.out.print("Enter the number of burgers to add: ");
            if (scanner.hasNextInt()) {
                burgersToAdd = scanner.nextInt();
                if(burgerStock + burgersToAdd <= MAX_BURGERS){
                    isValidInput =true;
                }else {
                    System.out.println("Burger Stock is exceeding the maximum
limit of "+MAX_BURGERS);//test case no 12.1
                }
            }else {
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 12.2
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }

        }while (!isValidInput);

        return burgersToAdd;
    }

    // ------------------- common methods --------------------------------

    private static FoodQueue getNextQueue(){
        FoodQueue foodQueue = null;
        int cashier1RemainingLength = MAX_CUSTOMERS[0]-cashier1.size();
        int cashier2RemainingLength = MAX_CUSTOMERS[1]-cashier2.size();
        int cashier3RemainingLength = MAX_CUSTOMERS[2]-cashier3.size();

        if(cashier1RemainingLength >0 &&  cashier1.size() <= cashier2.size()
&& cashier1.size() <= cashier3.size()){
            foodQueue = cashier1;
        } else if (cashier2RemainingLength>0 && cashier2.size() <=
cashier3.size()){
```

```
            foodQueue = cashier2;
        } else if (cashier3RemainingLength>0 ) {
            foodQueue = cashier3;
        }

        return foodQueue;
    }

    private static void printQueueElement(String element){
        if(element.equals("empty")){
            System.out.print("  X  "); // X-Not Occupied
        }else{
            System.out.print("  O  "); // O-Occupied
        }
    }

    private static List<String> getEmptyQueues() {
        List<String> emptyQueueNames = new ArrayList<>();

        if(cashier1.size()==0){
            emptyQueueNames.add("Cashier-1");
        }
        if(cashier2.size()==0){
            emptyQueueNames.add("Cashier-2");
        }
        if(cashier3.size()==0){
            emptyQueueNames.add("Cashier-3");
        }

        return  emptyQueueNames;

    }

    private static int getSmallestQueueIndex(int queueNumber){
        int index = 0;
        FoodQueue queue = getQueueByQueueNumber(queueNumber);
        if(queue.size()>0){
            index= queue.size();
        }
        return index;
    }

    private static FoodQueue getQueueByQueueNumber(int queueNumber){
        FoodQueue queue = null;
        switch (queueNumber){
            case 1 : {
                queue = cashier1;
                break;
            }
            case 2 :{
                queue = cashier2;
                break;
            }
            case 3 :{
                queue = cashier3;
                break;
            }
```

```
        }
        return queue;
    }

    private static boolean isAnyQueueAvailable(){
        return (cashier1.size() < MAX_CUSTOMERS[0]) ||
                (cashier2.size() < MAX_CUSTOMERS[1])  ||
                (cashier3.size() < MAX_CUSTOMERS[2]) ;
    }

    private static boolean areAllQueuesEmpty(){
        return (getSmallestQueueIndex(1) == 0) &&
                (getSmallestQueueIndex(2) == 0) &&
                (getSmallestQueueIndex(3) == 0);
    }

}
```

Task_3

```
package task_3;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class FoodiesFoodCenter {

    private static FoodQueue cashier1 = new FoodQueue(1);
    private static FoodQueue cashier2 = new FoodQueue(2);
    private static FoodQueue cashier3 = new FoodQueue(3);
    private static FoodQueue waitingQueue = new FoodQueue(4);
    private static final int[] MAX_CUSTOMERS = {2, 3, 5};

    private static final int MAX_BURGERS = 50;

    static final int BURGER_STOCK_WARNING_LIMIT = 10;

//    private static final int MAX_BURGERS_PER_CUSTOMER = 5;

    private static int burgerStock = MAX_BURGERS;

    private static int burgersOnHold = 0;

    public static void main(String[] args) {

        String option;
        Scanner scanner = new Scanner(System.in);
        do {
            displayMenu();
            option = scanner.nextLine();
            ;
            // Consume newline character
            switch (option) {
                case "100":
                    viewAllQues();
                    break;
                case "101":
                    viewEmptyQueues();
                    break;
                case "102":
                    addCustomerToTheQueue();
                    break;
                case "103":
                    removeCustomerFromQueue();
                    break;
                case "104":
                    removeServedCustomerFromQueue();
                    break;
                case "105":
//                      viewSortedCustomers();
```

```
                    break;
                case "106":
                    storeProgramData();
                    break;
                case "107":
                    loadProgramData();
                    break;
                case "108":
                    viewBurgerStock();
                    break;
                case "109":
                    addBurgersToStock();
                    break;
                case "110":
                    viewIncome();
                    break;
                case "999":
                    System.out.println("Exiting the program. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        } while (!option.equals("999"));
    }

    private static void displayMenu() {
        System.out.println("");
        System.out.println("========== Foodies Fave Food Center ==========");
        System.out.println("100: View all Queues");
        System.out.println("101: View all Empty Queues");
        System.out.println("102: Add customer to a Queue");
        System.out.println("103: Remove a customer from a Queue (From a
specific location)");
        System.out.println("104: Remove a served customer");
        System.out.println("105: View Customers Sorted in alphabetical
order");
        System.out.println("106: Store Program Data into file");
        System.out.println("107: Load Program Data from file");
        System.out.println("108: View Remaining burgers Stock");
        System.out.println("109: Add burgers to Stock");
        System.out.println("110: View income");
        System.out.println("999: Exit the Program");
        System.out.println("============================================");
        System.out.print("Enter an option: ");
    }

    // ------------------ 100 viewAllQues ----------------------------

    private static void viewAllQues() {
        System.out.println("*****************");
        System.out.println("*   Cashiers    *");
        System.out.println("*****************");

        for (int i = 0; i < MAX_CUSTOMERS[2]; i++) {
            System.out.print(" ");
            if (i < MAX_CUSTOMERS[0]) {
                if ((i + 1) <= cashier1.size()) {
```

```
                    printQueueElement("not_empty");
                } else {
                    printQueueElement("empty");
                }
            } else {
                System.out.print("  -  ");
            }

            if (i < MAX_CUSTOMERS[1]) {
                if ((i + 1) <= cashier2.size()) {
                    printQueueElement("not_empty");
                } else {
                    printQueueElement("empty");
                }
            } else {
                System.out.print("  -  ");
            }


            if ((i + 1) <= cashier3.size()) {
                printQueueElement("not_empty");
            } else {
                printQueueElement("empty");
            }

            System.out.println("      ");
        }
        System.out.println("X-Not Occupied   O-Occupied");

    }

    // ------------------- 101 viewEmptyQueues ---------------------------

    private static void viewEmptyQueues() {
        List<String> emptyQueues = getEmptyQueues();
        if (emptyQueues.isEmpty()) {
            System.out.println("No empty queues found.");
        } else {
            System.out.println("Empty queues:");
            {
                for (String queueName : emptyQueues) {
                    System.out.println(queueName);
                }
            }
        }
    }

    // ------------------- 102 addCustomerToTheQueue ---------------------

    private static void addCustomerToTheQueue() {
        String firstName = inputName("first");//test case 3.11, 3.12
        String secondName = inputName("second");//test case 3.13
        int burgersRequired = inputBurgers();

        if ((burgerStock - burgersOnHold) >= burgersRequired) {
            Customer customer = new Customer(firstName, secondName,
burgersRequired);
```

```
            FoodQueue foodQueue = getNextQueue();
            foodQueue.addCustomer(customer);

            burgersOnHold += burgersRequired;
            if (!isAnyQueueAvailable()) {
                System.out.println("*****************");
                System.out.println("No queues are available at the
moment");//test case 3.19
                System.out.println("Customer added to the waiting
queue.");//test case 3.19
                System.out.println("Customer Name : " +
customer.getFullName());//test case 3.19
            } else {
                System.out.println("*****************");
                System.out.println("Customer added to the queue
successfully.");//test case 3.14
                System.out.println("Queue Number: " +
foodQueue.getIndex());//test case 3.14
                System.out.println("Customer Name : " +
customer.getFullName());//test case 3.14
            }

            if ((burgerStock - burgersOnHold) <= BURGER_STOCK_WARNING_LIMIT)
{
                System.out.println("*****************");
                System.out.println("Warning: Available Burger Stock
Reached/pass the minimum warning limit of " +
BURGER_STOCK_WARNING_LIMIT);//test case 3.15
                System.out.println("Please add burgers");//test case 3.15
            }
        } else {
            System.out.println("Insufficient burger stock. Cannot add
customer.");//test case 3.9
        }



    }

    // ------------------- 103 removeCustomerFromQueue -------------------

    private static void removeCustomerFromQueue() {
        if (areAllQueuesEmpty()) {
            System.out.println("All queues are empty");//test case no 4.4
        } else {
            int queueNumber = inputQueueNumber("REMOVE");
            FoodQueue foodQueue = getQueueByQueueNumber(queueNumber);
            int queueIndex = inputQueueIndex(queueNumber);
            burgersOnHold -=
foodQueue.getCustomer(queueIndex).getBurgersRequired();
            foodQueue.removeCustomer(queueIndex);
            System.out.println("Customer removed from the queue
successfully.");//test case no 4.6
            System.out.println("Queue Number: " + queueNumber + " | Queue
Index: " + queueIndex);//test case no 4.6
            ;
        }
```

```
    }

    // ------------------- 104 removeServedCustomerFromQueue -------------
    private static void removeServedCustomerFromQueue() {
        if (areAllQueuesEmpty()) {
            System.out.println("All queues are empty");
        } else {
            int queueNumber = inputQueueNumber("REMOVE");
            FoodQueue foodQueue = getQueueByQueueNumber(queueNumber);
            int queueIndex = 0;
            int burgersSold =
foodQueue.getCustomer(queueIndex).getBurgersRequired();
            burgersOnHold -= burgersSold;
            burgerStock -= burgersSold;
            foodQueue.addSoldBurgers(burgersSold);
            if(waitingQueue.size()>0){
                foodQueue.removeCustomer(queueIndex);
                System.out.println("Served Customer removed from the queue
successfully.");//test case no 5.1
                System.out.println("Queue Number: " + queueNumber);//test
case no 5.1

                Customer nextCustomer = waitingQueue.getCustomer(0);
                foodQueue.addCustomer(nextCustomer);
                waitingQueue.removeCustomer(nextCustomer);
                System.out.println("*****************");
                System.out.println("Next customer in the waiting queue  added
successfully.");//test case no 5.6
                System.out.println("Customer Name : " +
nextCustomer.getFullName());//test case no 5.6
                System.out.println("Queue Number: " + queueNumber);//test
case no 5.6
            }else {
                foodQueue.removeCustomer(queueIndex);
                System.out.println("Served Customer removed from the queue
successfully.");//test case no 5.5
                System.out.println("Queue Number: " + queueNumber);//test
case no 5.5
            }

        }
    }

    // ------------------- //TODO - 105 viewSortedCustomers --------------


    // ------------------- 106 storeProgramData --------------------------

    private static void storeProgramData() {
        try {
            BufferedWriter writer = new BufferedWriter(new
FileWriter("program_data_02.txt"));

            writer.write(cashier1.toString());
            writer.newLine();
            writer.write(cashier2.toString());
            writer.newLine();
```

```
            writer.write(cashier3.toString());
            writer.newLine();
            writer.write(waitingQueue.toString());
            writer.close();

            System.out.println("Program data stored successfully.");//test
case no 7.1
        } catch (IOException e) {
            System.out.println("An error occurred while storing the program
data");
        }
    }

    // ------------------- 107 loadProgramData --------------------------

    private static void loadProgramData() {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("program_data_02.txt"));

            String line;
            if ((line = reader.readLine()) != null) {
                cashier1 = parseQueue(line);
            }

            if ((line = reader.readLine()) != null) {
                cashier2 = parseQueue(line);
            }

            if ((line = reader.readLine()) != null) {
                cashier3 = parseQueue(line);
            }
            if ((line = reader.readLine()) != null) {
                waitingQueue = parseQueue(line);
            }
            reader.close();

            System.out.println("Program data loaded successfully.");//test
case no 8.1
        } catch (IOException e) {
            System.out.println("An error occurred while loading the program
data.");
        }
    }

    private static FoodQueue parseQueue(String line) {
        FoodQueue queue = new FoodQueue();
        line = line.trim();
//        line = line.substring(1, line.length() - 1);
        String[] customers = line.split(";");
        for (String el : customers) {
            String[] customerString = el.split(",");
            Customer customer = new Customer(customerString[0],
customerString[1], Integer.parseInt(customerString[2]));
            queue.addCustomer(customer);
        }
```

```
        return queue;
    }

    // ------------------- 108 viewBurgerStock ------------------------------
-
    private static void viewBurgerStock() {
        System.out.println("Remaining burgers stock: " + burgerStock);//test
case no 9.1
        System.out.println("Burgers stock on Hold: " + burgersOnHold);//test
case no 9.1
    }

    // ------------------- 109 addBurgersToStock ----------------------------
-

    private static void addBurgersToStock() {
        int burgersToAdd = inputBurgerStock();
        burgerStock += burgersToAdd;
        System.out.println("Burgers added to the stock. New stock: " +
burgerStock);//test case no 10.1
    }

    // ------------------- 110 viewIncome -------------------------------

    private static void viewIncome() {
        int burgerPrice = 650;
        int q1Income = cashier1.getSoldBurgers() * burgerPrice;
        int q2Income = cashier2.getSoldBurgers() * burgerPrice;
        int q3Income = cashier3.getSoldBurgers() * burgerPrice;
        System.out.println("Income from queue 01: " + q1Income);//test case
11.
        System.out.println("Income from queue 02: " + q2Income);//test case
11.
        System.out.println("Income from queue 03: " + q3Income);//test case
11.
        System.out.println("Total Income : " + (q1Income + q2Income +
q3Income));//test case 11.
    }

    // ------------------- user inputs ----------------------------------

    private static String inputName(String option) {
        Scanner scanner = new Scanner(System.in);
        String name;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the " + option + " name: ");//test case
no 14.1
            name = scanner.nextLine();

            if (name.trim().isEmpty() || name.trim().contains(" ")
                    || name.trim().contains(",") || name.trim().contains(";")
|| name.trim().contains("@")
            ) {
                System.out.println("Invalid input! " + option + " name cannot
be empty and have space/special characters");//test case 3.11, 3.12
```

```
            } else {
                isValidInput = true;
            }
        } while (!isValidInput);

        System.out.println(option + " name: " + name);

        return name;

    }

    private static int inputBurgers() {
        Scanner scanner = new Scanner(System.in);
        int burgerCount = 0;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the number of required burgers: ");//test
case 3.17
            if (scanner.hasNextInt()) {
                burgerCount = scanner.nextInt();
                if (burgerCount == 0) {
                    System.out.print("Burgers required should be greater than
zero");//test case 3.18
                } else if (burgerCount > 0 && burgerCount <= (burgerStock -
burgersOnHold)) {
                    isValidInput = true;
                } else {
                    System.out.println("Not enough burgers in the stock. Add
more burgers");//test case 3.17
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        } while (!isValidInput);

        System.out.println("Entered burger count: " + burgerCount);
        return burgerCount;
    }

    private static int inputQueueNumber(String option) {
        Scanner scanner = new Scanner(System.in);
        int queueNumber = -1;

        boolean isValidInput = false;
        do {
            System.out.print("Enter the queue number (1 / 2 / 3): ");//test
case 3.1
            if (scanner.hasNextInt()) {
                queueNumber = scanner.nextInt();
                if (queueNumber >= 1 && queueNumber <= 3) {
                    if (option.equals("ADD")) {
                        if (getSmallestQueueIndex(queueNumber) != -1) {
                            isValidInput = true;
```

```
                        } else {
                            System.out.println("Queue " + queueNumber + " is
full. Enter Different Queue Number");//test case 3.4
                        }
                    } else if (option.equals("REMOVE")) {
                        if (getSmallestQueueIndex(queueNumber) != 0) {
                            isValidInput = true;
                        } else {
                            System.out.println("Queue " + queueNumber + " is
empty. Enter Different Queue Number");
                        }
                    }
                } else {
                    System.out.println("Invalid input! Queue number must be
between 1 and 3.");
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        } while (!isValidInput);

        System.out.println("Selected queue number: " + queueNumber);
        return queueNumber;
    }

    private static int inputQueueIndex(int queueNumber) {
        FoodQueue queue = getQueueByQueueNumber(queueNumber);
        Scanner scanner = new Scanner(System.in);
        int queueIndex = -1;
        boolean isValidInput = false;

        do {
            System.out.print("Enter the queue position(index) [0-" +
(queue.size() - 1) + "]");//test case no 13.1
            if (scanner.hasNextInt()) {
                queueIndex = scanner.nextInt();
                if (queueIndex >= 0 && queueIndex <= (queue.size() - 1)) {
                    isValidInput = true;
                } else {
                    System.out.println("Invalid input! Queue index for this
queue must be between 0 and " + (queue.size() - 1));//test case no 13.2
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 13.3
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }
        } while (!isValidInput);
        return queueIndex;
    }

    private static int inputBurgerStock() {
        Scanner scanner = new Scanner(System.in);
```

```
        int burgersToAdd = 0;
        boolean isValidInput = false;
        do {
            System.out.print("Enter the number of burgers to add: ");
            if (scanner.hasNextInt()) {
                burgersToAdd = scanner.nextInt();
                if (burgerStock + burgersToAdd <= MAX_BURGERS) {
                    isValidInput = true;
                } else {
                    System.out.println("Burger Stock is exceeding the maximum
limit of " + MAX_BURGERS);//test case no 12.1
                }
            } else {
                System.out.println("Invalid input! Please enter a valid
integer.");//test case no 12.2
                scanner.next(); // Clear the invalid input from the scanner
buffer
            }

        } while (!isValidInput);

        return burgersToAdd;
    }

    // ------------------- common methods --------------------------------

    private static FoodQueue getNextQueue() {
        FoodQueue foodQueue = null;

        if (!isAnyQueueAvailable()) {
            foodQueue = waitingQueue;
        }else {
            int cashier1RemainingLength = MAX_CUSTOMERS[0] - cashier1.size();
            int cashier2RemainingLength = MAX_CUSTOMERS[1] - cashier2.size();
            int cashier3RemainingLength = MAX_CUSTOMERS[2] - cashier3.size();

            if (cashier1RemainingLength > 0 && cashier1.size() <=
cashier2.size() && cashier1.size() <= cashier3.size()) {
                foodQueue = cashier1;
            } else if (cashier2RemainingLength > 0 && cashier2.size() <=
cashier3.size()) {
                foodQueue = cashier2;
            } else if (cashier3RemainingLength > 0) {
                foodQueue = cashier3;
            }
        }


        return foodQueue;
    }

    private static void printQueueElement(String element) {
        if (element.equals("empty")) {
            System.out.print("  X  "); // X-Not Occupied
        } else {
            System.out.print("  O  "); // O-Occupied
        }
```

```
    }

    private static List<String> getEmptyQueues() {
        List<String> emptyQueueNames = new ArrayList<>();

        if (cashier1.size() == 0) {
            emptyQueueNames.add("Cashier-1");
        }
        if (cashier2.size() == 0) {
            emptyQueueNames.add("Cashier-2");
        }
        if (cashier3.size() == 0) {
            emptyQueueNames.add("Cashier-3");
        }

        return emptyQueueNames;

    }

    private static int getSmallestQueueIndex(int queueNumber) {
        int index = 0;
        FoodQueue queue = getQueueByQueueNumber(queueNumber);
        if (queue.size() > 0) {
            index = queue.size();
        }
        return index;
    }

    private static FoodQueue getQueueByQueueNumber(int queueNumber) {
        FoodQueue queue = null;
        switch (queueNumber) {
            case 1: {
                queue = cashier1;
                break;
            }
            case 2: {
                queue = cashier2;
                break;
            }
            case 3: {
                queue = cashier3;
                break;
            }
        }
        return queue;
    }

    private static boolean isAnyQueueAvailable() {
        return (cashier1.size() < MAX_CUSTOMERS[0]) ||
                (cashier2.size() < MAX_CUSTOMERS[1]) ||
                (cashier3.size() < MAX_CUSTOMERS[2]);
    }

    private static boolean areAllQueuesEmpty() {
        return (cashier1.isEmpty() && cashier2.isEmpty() &&
cashier3.isEmpty());
    }
```

```
}
```

<<Note : Do not use screenshots or images for the report.>>

<<END>>