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 (2023)
Module Leader	: TG Deshan K Sumanthilaka
Date of submission	: 07/17/2023
Student ID	: IIT No - 20221680/ UOW No - w1999517
Student First Name	: Nethmi
Student Surname	:Wijewikrama

"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 : W.L.N Umaya

Student ID : 20221680

Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After the program starts, 100 or VFQ	Displays "view all queues" for all queues.	Display "view all queues"	Pass
2	View all Empty queues initialized correctly After the program starts,101 or VEQ	Displays "view all empty queues", for all queues	Displays "view all empty queues",	pass
3	Add customer "Jane" to Queue 2 102 or ACQ Enter Your choice:102. Enter Name: Jane Enter Preferred Cashier (1,2,3):2	Display "Jane added to the cashier 2 successfully".	Display "Customer Jane added to the cashier 2 successfully".	Pass
	Enter your choice:102. Enter Name: Peter Enter Preferred Cashier (1,2,3) :1	Display "Peter added to cashier 1 successfully".	Display "Peter added to cashier 1 successfully".	Pass
	Enter your choice:102. Enter Name: John Enter Preferred Cashier (1,2,3) :3	Display "John added to the cashier 3 successfully"	Display "John added to the cashier 3 successfully"	Pass
4	Remove customer "Jane" from cashier 2. 103 or RCQ Enter cashier number (1,2,3): 2. Enter customer index to remove (0,1,2,3,4) :0	Display "customer Jane removed from cashier 2"	Display "customer Jane removed from cashier 2"	pass
.5.	Remove Served customer from cashier 104 or PCQ.	Display "customer removed from cashier "	Display "customer removed from cashier "	pass
6	View customers sorted in alphabetical order. 105 or VCS	Display "customer sorted in alphabetical order, John.	Display "customer sorted in alphabetical order, John.	pass

7	Store program Data into a file. 106 or SPD	Display “Programme Data stored successfully”.	Display “Programme Data stored successfully”.	Pass
8	Load program data from file 107 or LPD	Display “---Loading Data ---" cashier 3- John	Display “---Loading Data ---" cashier 3- John	Pass
9	View Remaining burgers stock 108 or STK.	Display “Remaining burger stock 45”	Display “Remaining burger stock 45”	Pass
10	Add Burgers to stock 109 or AFS. Enter the number of burgers to add: 15.	Display “15 burgers added to the stock”	Display “15 burgers added to the stock”	Pass
11	Exit the program 999 or EXT	Display “Exiting Burger_Food_Center program...”	Display “Exiting Burger_Food_Center program...”	Pass
		”		

Test Case (Task 2 Class Version)	Expected Result	Actual Result	Pass/Fail
After the program starts, 100 or VFQ	Displays "view all queues" for all queues	Displays "view all queues"	*pass*
After the program starts, 101 or VEQ	Displays "view all empty queues" for all queues	Displays "view all empty queues	**pass**
Add customer "Jane" cashier 2 102 or ACQ Enter Your choice:102. Enter Customer First Name: Jane Enter Customer Second Name: Dior Enter the number of burgers required:4	Display "Jane Dior added to cashier 2 successfully"	Display "Jane Dior added to cashier 2 successfully"	pass
Add customer "Peter" to cashier 1 Enter your choice:102. Enter Customer Name: Peter Enter Customer Second Name: Stem Enter the number of burgers required:5	Display "Peter stem added to cashier1 1 successfully"	Display "Peter Stem added to cashier 1 successfully"	pass
Add customer "John" to cashier 3 Enter your choice:102. Enter Customer Name: John	Display "John Steve added to cashier 3 successfully"	Display "John Steve added to cashier 3 successfully"	pass

Enter Customer Second Name: Steve Enter the number of burgers required:11			
Remove customer “Peter Stem” from cashier 2. 103 or RCQ Enter cashier number (1,2,3): 2. Enter customer index to remove (0,1,2,3) :0	Display "Customer Peter Stem removed from Cashier 2"	Display "Customer Jane removed from Cashier 2"	pass
Remove served customer from any queue. 104 or PCQ.	Display "Customer removed from cashier"	Display "Customer removed from cashier"	pass
View customers sorted in alphabetical order. 105 or VCS	Display "All Customers Sorted in alphabetical order"	Display "All Customers Sorted in alphabetical order"	pass
Store program data in a file. 106 or SPD	Display "Manager data stored successfully"	Display "Manager data stored successfully"	pass
Load program data from a file. 107 or LPD	Display "---Loading Data---" and loaded program data	Display "---Loading Data---" and loaded program data	pass
View remaining burger stock. 108 or STK.	Display "Remaining burgers stock: [Stock Count]"	Display "Remaining burgers stock: [Stock Count]"	pass

Add burgers to stock. 109 or AFS.	Display "Added [Number of Burgers] burgers to the stock	Display "Added [Number of Burgers] burgers to the stock"	pass
Display waiting queue.	Display "Waiting queue: [Empty or Customer List]"	Display "Waiting queue: [Empty or Customer List]"	pass
Exit the program 999 or EXT	Display “Existing food centre program	Display “Existing Burger Food centre…Thanks for your visit…Come Again	pass

Discussion

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

Here's a breakdown of the main components and functionality of the program:

1. **Array Part (Task 1)**

Constants:

1. Stock_of_Burgers: This constant represents the initial stock of burgers in the burger food centre. Its value is set to 50, indicating that there are initially 50 burgers available.

2. Burgers_per_Client: This constant represents the number of burgers served per client. Its value is set to 5, indicating that each client is served 5 burgers.

Variables:

1. cashier1, cashier2, cashier3: These variables are arrays that represent the queues for Cashier 1, Cashier 2, and Cashier 3, respectively. Each array holds the names of the customers in the corresponding cashier's queue.

2. Burgers: This variable represents the remaining stock of burgers in the burger food centre. Its initial value is set to the value of Stock_of_Burgers constant (50). The value of Burgers decreases as customers are served and increases when burgers are added to the stock.

Methods:

1. main: This method is the entry point of the program. It contains the main execution logic and handles user input to perform various operations on the queues and stock.

2. ViewAllQueues: This method displays the contents of all the cashier queues. It iterates over each queue array (cashier1, cashier2, cashier3), converts it into a symbolic representation using Queue_with_Symbols method, and then prints the contents.

3. isEmptyQueue: This method checks if a given queue is empty by iterating over the elements of the queue array and returning false if any customer name is found. If no customers are found, it returns true.

4. ViewEmptyQueues: This method displays the cashier queues that are empty. It calls isEmptyQueue method for each queue array (cashier1, cashier2, cashier3) and prints the queue number if it is found to be empty.

5. Queue_with_Symbols: This method adds symbols ("O" for occupied and "X" for empty) to represent the state of a queue. It takes a queue array as input, iterates over its elements, and assigns the corresponding symbol based on whether the element is null or not. It returns a new array with the symbols.

6. storeProgramData: This method stores the program data (queue contents) into a file named "program_data.txt". It uses a FileWriter to write the data for each queue into the file.

7. LoadData: This method loads the program data (queue contents) from a file named "Burger_Food_Center_data.txt". It uses a FileReader and BufferedReader to read the data line by line and prints it on the console.

8. AddCustomer: This method allows the user to add a customer to a specific cashier queue. It takes customer's name and preferred cashier number as input. It validates the inputs and assigns the selected cashier array based on the cashier number. It then finds an available index in the selected cashier's queue using getAvailableIndex method and adds the customer name to that index.

9. containsNumericValues: This method checks if a given string contains numeric values. It iterates over each character of the string and checks if it is a digit. If any digit is found, it returns true; otherwise, it returns false.

10. getAvailableIndex: This method finds the first available index in a queue array by iterating over its elements. It returns the index if an available slot (null element) is found; otherwise,it returns -1.

11. RemoveCustomer: This method allows the user to remove a customer from a specific cashier queue. It takes the cashier number as input and prompts the user to enter the customer index to remove. It checks the validity of inputs and removes the customer from the specified index if it exists.

12. RemoveServedCustomer: This method removes a served customer from any cashier queue. It checks if any of the cashier queues (cashier1, cashier2, cashier3) are not empty and removes the first customer from the first non-empty queue using shiftQueueLeft method. It also reduces the burger stock by the number of burgers served.

13. reduceBurgerStock: This method reduces the stock of burgers by a specified amount. It takes the amount as input and subtracts it from the Burgers variable.

14. shiftQueueLeft: This method shifts all elements of a queue array from one position to the left. It starts from the first index and assigns the value of the next index to the current index, effectively shifting all elements to the left. The last element is set to null.

15. ViewCustomersSorted: This method displays the customers from all cashier queues sorted in alphabetical order. It creates a new array (allCustomers) to hold all customers, retrieves customers from each cashier queue, stores them in allCustomers, and then sorts allCustomers using sortCustomers method. Finally, it prints the sorted customers on the console.

16. sortCustomers: This method sorts an array of customers in alphabetical order using the bubble sort algorithm. It compares adjacent elements and swaps them if they are in the wrong order. The sorting process continues until the array is completely sorted.

17. ViewRemainingStock: This method displays the remaining stock of burgers (Burgers) on the console.

18. AddBurgersToStock: This method allows the user to add burgers to the stock. It takes the number of burgers to add as input and adds the specified number to the Burgers variable, effectively increasing the stock.

2. Class Version (Task 2)

(Main Java)

Constants:

1. Stock_of_Burgers: This constant represents the total number of burgers available in stock. It is set to 50 in this case, indicating that initially, there are 50 burgers in stock.

2. cashier1_CAPACITY, cashier2_CAPACITY, cashier3_CAPACITY: These constants represent the maximum capacity of each cashier queue. They define the maximum number of customers that each cashier queue can hold at a given time.

3. Burger_price: This constant represents the price of a burger. It is set to 650.00 in this case.

Variables:

1. cashier1, cashier2, cashier3: These variables represent instances of the FoodQueue class and represent the three cashier queues available in the Burger Food Center. Customers are added to and removed from these queues.

2. BurgersStock: This variable represents the current stock of burgers available. It is initially set to the value of Stock_of_Burgers (50). The value of BurgersStock is updated whenever a customer is served or burgers are added to the stock.

3. Waiting_list: This variable represents an instance of the CircularQueue<Customer> class. It serves as a waiting list for customers when all cashier queues are full. Customers in the

4. waiting list will be served when a cashier queue becomes available.

Methods:

1. Main (String[] args): This is the main method of the program. It contains the main logic of the program, including the menu system and handling user input.

2. viewAllQueues(): This method displays the state of all cashier queues, showing the number of customers in each queue.

3. viewEmptyQueues(): This method displays the cashier queues that are currently empty. It checks each queue and prints the cashier number if it is empty.

4. storeProgramData (): This method stores the program data (the state of each cashier queue) into a file named "Manager_data.txt".

5. loadData (): This method loads the program data from the "Manager_data.txt" file and displays it on the console.

6. addCustomer (Scanner scanner): This method adds a customer to one of the cashier queues. It prompts the user to enter the customer's first name, last name, and the number of burgers required. It then adds the customer to the shortest available queue or the waiting list if all queues are full.

7. isStringOnly (String input): This method checks if a given string consists of only letters. It is used to validate the first and last names entered by the user.

8. getShortestQueue (): This method determines and returns the cashier queue with the fewest customers. It checks the size of each queue and returns the queue with the smallest size.

9. removeCustomer (Scanner scanner): This method removes a customer from a specific cashier queue based on user input. It prompts the user to enter the cashier number and the index of the customer to remove. It then removes the customer from the specified queue if the inputs are valid.

10. removeServedCustomer (): This method removes a served customer from the cashier queues. It checks each cashier queue in order (cashier 1, cashier 2, cashier 3) and removes the first customer in each queue. If a customer is removed, the burger stock is reduced accordingly. If there are customers in the waiting list, the next customer is added to the shortest queue if available.

11. reduceBurgerStock (int amount): This method reduces the burger stock by a specified amount. It is called when a customer is served, and the amount of burgers required by the customer is subtracted from the stock.

12. viewCustomersSorted (): This method displays all customers sorted in alphabetical order based on their full names. It collects all customers from the cashier queues and the waiting list, sorts them using the sortCustomers method, and then displays their full names on the console.

13. sortCustomers (ArrayList<Customer> customers): This method sorts a list of customers in alphabetical order based on their full names. It utilizes the Collections.sort method and a custom Comparator implementation to perform the sorting.

14. viewRemainingStock (): This method displays the remaining stock of burgers available.

15. addBurgersToStock (Scanner scanner): This method adds burgers to the stock based on user input. It prompts the user to enter the number of burgers to add and updates the stock accordingly.

16. printIncome (): This method calculates and prints the income generated by each cashier queue. It calculates the income by multiplying the number of customers in each queue by the Burger_price constant.

Classes:

1. Customer: This class represents a customer in the Burger Food Center. It has attributes such as first name, last name, and the number of burgers required. Each customer object represents an individual customer.

2. FoodQueue: This class represents a cashier queue in the Burger Food Center. It is implemented using an ArrayList. It has methods to add customers, remove customers, get the current size of the queue, and calculate the income generated by the queue.

3. CircularQueue<T>: This class represents a circular queue data structure. It is a generic class that can be used to create a queue of any type. It is utilized for the waiting list in this program.

(FoodQueue)

Variables:

1. customers: An ArrayList<Customer> that stores the customers in the queue.
2. capacity: An integer representing the maximum capacity of the queue.

Methods:

1. getSize(): Returns the current size of the queue, which is the number of customers in it.
2. getCapacity(): Returns the maximum capacity of the queue.
3. isEmpty(): Checks if the queue is empty by verifying if the customers list is empty.
4. isFull(): Checks if the queue is full by comparing the size of the customers list with the capacity.
5. getCustomers(): Returns the ArrayList of customers in the queue.
6. getCashierNumber(): Returns the cashier number associated with the queue (1, 2, or 3) by comparing the queue instance with the corresponding instances in the Manager class.
7. addCustomer(Customer customer): Adds a customer to the queue if it is not already full.
8. removeCustomer(int index): Removes the customer at the specified index from the queue if the index is valid.
9. isValidIndex(int index): Checks if the provided index is within the valid range of the customers list.
10. getIncome(double burgerPrice): Calculates and returns the total income generated by the customers in the queue based on the given burger price.
11. getDataString(String prefix): Generates a formatted string representation of the customer data in the queue, including their full names and the number of burgers required.

12. toString(): Overrides the toString() method to provide a string representation of the queue's occupancy status. It uses 'O' to represent an occupied position in the queue and 'X' to represent an unoccupied position.

(Customer)

Variables:

1. **firstName** and **lastName**: These are private **String** variables that store the customer's first name and last name, respectively. They are used to hold the customer's personal information.

2. **burgersRequired**: This is a private integer variable that represents the number of burgers required by the customer. It stores the quantity of burgers the customer wants to order.

3. **Customer (String firstName, String lastName, int burgersRequired)**: This is a constructor that initializes a **Customer** object with the provided first name, last name, and the number of burgers required. It assigns the given values to the corresponding instance variables.

4. **getFullName ()**: This is a public method that returns the full name of the customer. It concatenates the **firstName** and **lastName** variables together with a space in between and returns the resulting full name as a **String**.

5. **getBurgersRequired()**: This is a public method that returns the number of burgers required by the customer. It simply returns the value stored in the **burgersRequired** variable.

Code :

- **Array part(Task1)**

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.Arrays;
import java.io.BufferedReader;
import java.io.FileReader;

public class Burger_Food_Center {
    //Constants
    private static final int Stock_of_Burgers= 50;
    private static final int Burgers_per_Client = 5;

    //Arrays for Queues
    private static String[] cashier1 = new String[2];
    private static String[] cashier2 = new String[3];
    private static String[] cashier3 = new String[5];

    private static int Burgers = Stock_of_Burgers;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        //Starting the while Loop
        while (true) {
            //giving menu option
            System.out.println("****  Burger_Food_Center  ****");
            System.out.println("-----------------------------------------
");
            System.out.println("Please select an option");
            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
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 (Do not use library sort routine)");
            System.out.println("106 or SPD: Store Program Data into files");
            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 your choice: ");



            //get input choice from user
```

```
            String input = scanner.nextLine().trim();

            switch (input) {
                case "100":
                case "VFQ":
                    ViewAllQueues();
                    break;
                case "101":
                case "VEQ":
                    ViewEmptyQueues();
                    break;
                case "102":
                case "ACQ":
                    AddCustomer(scanner);
                    break;
                case "103":
                case "RCQ":
                    RemoveCustomer(scanner);
                    break;
                case "104":
                case "PCQ":
                    RemoveServedCustomer();
                    break;
                case "105":
                case "VCS":
                    ViewCustomersSorted();
                    break;
                case "106":
                case "SPD":
                    storeProgramData();
                    break;
                case "107":
                case "LPD":
                    LoadData();
                    break;
                case "108":
                case "STK":
                    ViewRemainingStock();
                    break;
                case "109":
                case "AFS":
                    AddBurgersToStock(scanner);
                    break;
                case "999":
                case "EXT":
                    System.out.println("Exiting Burger_Food_Center
program...");
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
            }
        }
    }

    //View all queues
    private static void ViewAllQueues() {
```

```
        System.out.println("*****************");
        System.out.println("*   Cashiers    *");
        System.out.println("*****************");
        System.out.println("Cashier 1: " +
Arrays.toString(Queue_with_Symbols(cashier1)));
        System.out.println("Cashier 2: " +
Arrays.toString(Queue_with_Symbols(cashier2)));
        System.out.println("Cashier 3: " +
Arrays.toString(Queue_with_Symbols(cashier3)));
    }

    // Check if a queue is empty
    private static boolean isEmptyQueue(String[] queue) {
        for (String customer : queue) {
            if (customer != null) {
                return false;
            }
        }
        return true;
    }

    // View empty queues
    private static void ViewEmptyQueues() {
        System.out.println("Empty Queues:");
        if (isEmptyQueue(cashier1)) {
            System.out.println("Cashier 1");
        }
        if (isEmptyQueue(cashier2)) {
            System.out.println("Cashier 2");
        }
        if (isEmptyQueue(cashier3)) {
            System.out.println("Cashier 3");
        }
    }

    // Add symbols to represent the queue
    private static String[] Queue_with_Symbols(String[] queue) {
        String[] queueWithSymbols = new String[queue.length];
        for (int i = 0; i < queue.length; i++) {
            if (queue[i] == null) {
                queueWithSymbols[i] = "X";
            } else {
                queueWithSymbols[i] = "O";
            }
        }
        return queueWithSymbols;
    }
    // Store program data to a file
    private static void storeProgramData() {
        try {
            FileWriter writer = new FileWriter("program_data.txt");

            // Write the data for QUEUE1
            for (String customer : cashier1) {
                if (customer != null) {
                    writer.write("Queue1:" + customer + "\n");
                }
```

```
            }

            // Write the data for QUEUE2
            for (String customer : cashier2) {
                if (customer != null) {
                    writer.write("Queue2:" + customer + "\n");
                }
            }

            // Write the data for QUEUE3
            for (String customer : cashier3) {
                if (customer != null) {
                    writer.write("Queue3:" + customer + "\n");
                }
            }

            writer.close();
            System.out.println("Program data stored successfully.");
        } catch (IOException e) {
            System.out.println("Error storing program data: " +
e.getMessage());
        }
    }

    private static void LoadData() {
        System.out.println("---Loading Data---");
        try {
            FileReader fileReader = new
FileReader("Burger_Food_Center_data.txt");
            BufferedReader bufferedReader = new BufferedReader(fileReader);
            String line;

            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }

            bufferedReader.close();
            System.out.println("Data Loaded Successfully!");
        } catch (IOException e) {
            System.out.println("ERROR while loading data");
        }
    }



    // Add a customer to a cashier queue
    private static void AddCustomer(Scanner scanner) {
        System.out.print("Enter client name: ");
        String customerName = scanner.nextLine();
        if (customerName == null || customerName.isEmpty() ||
customerName.trim().isEmpty()) {
            System.out.println("Error: Client name cannot be empty.");
            return;
        }

        if (containsNumericValues(customerName)) {
            System.out.println("Error: Client name cannot contain numeric
```

```
values.");
            return;
        }

        System.out.print("Enter preferred cashier (1, 2, or 3): ");
        int preferredCashier = scanner.nextInt();
        scanner.nextLine();

        String[] selectedCashier =null;

        if (preferredCashier == 1) { // Assign the selected cashier array
based on user input
            selectedCashier = cashier1;
        } else if (preferredCashier == 2) {
            selectedCashier = cashier2;
        } else if (preferredCashier == 3) {
            selectedCashier = cashier3;
        } else {
            System.out.println("Invalid cashier choice. Customer not
added.");
            return;
        }

        int availableIndex = getAvailableIndex(selectedCashier);

        if (availableIndex != -1) {
            selectedCashier[availableIndex] = customerName; // Add the
customer to the selected cashier array

            System.out.println("Customer " + customerName + " added to
Cashier " + preferredCashier + ".");
        } else {
            System.out.println("Cashier " + preferredCashier + " is full.
Customer not added.");
        }

        if (Stock_of_Burgers <= 10) {
            System.out.println("Warning: Low stock! Remaining burgers: " +
Stock_of_Burgers);
        }
        if (Stock_of_Burgers == 0) {
            System.out.println("Burger stock is empty...");
        }
    }

    private static boolean containsNumericValues(String text) {
        for (char c : text.toCharArray()) {
            if (Character.isDigit(c)) { // Check if the customer name
contains numeric values
                return true;
            }
        }
        return false;
    }
    private static int getAvailableIndex(String[] queue) {
        for (int i = 0; i < queue.length; i++) {
            //Check if the element at the current index is null
```

```
            if (queue[i] == null) {
                return i; //Return the index if an available slot is found
            }
        }
        return -1;// Return -1 if no available slot is found
    }

    private static void RemoveCustomer(Scanner scanner) {
        // Prompt the user to enter the cashier number
        System.out.print("Enter cashier number (1, 2, or 3): ");
        int cashierNumber = scanner.nextInt();
        scanner.nextLine();

        String[] selectedCashier = null;
        // Determine the selected cashier based on the cashier number

        switch (cashierNumber) {
            case 1:
                selectedCashier = cashier1;
                break;
            case 2:
                selectedCashier = cashier2;
                break;
            case 3:
                selectedCashier = cashier3;
                break;
            default:
                System.out.println("Invalid cashier number. Customer not
removed.");
                return;
        }

        // Check if the selected cashier's queue is empty
        if (isEmptyQueue(selectedCashier)) {
            System.out.println("Cashier " + cashierNumber + " is already
empty. No customers to remove.");
        } else {

            // Prompt the user to enter the customer index to remove
            System.out.print("Enter customer index to remove (0 to " +
(selectedCashier.length - 1) + "): ");
            int customerIndex = scanner.nextInt();
            scanner.nextLine();

            // Remove the customer at the specified index if it's valid
            if (customerIndex >= 0 && customerIndex < selectedCashier.length
&& selectedCashier[customerIndex] != null) {
                String removedCustomer = selectedCashier[customerIndex];
                selectedCashier[customerIndex] = null;
                System.out.println("Customer " + removedCustomer + " removed
from Cashier " + cashierNumber + ".");
            } else {
                System.out.println("Invalid customer index. Customer not
removed.");
            }
        }
    }
```

```
    private static void RemoveServedCustomer() {
        // Check if the cashier's queues are not empty
        if (!isEmptyQueue(cashier1)) {
            String removedCustomer = cashier1[0];
            shiftQueueLeft(cashier1);
            reduceBurgerStock(Burgers_per_Client);
            System.out.println("Customer " + removedCustomer + " removed from
Cashier 1.");
        } else if (!isEmptyQueue(cashier2)) {
            String removedCustomer = cashier2[0];
            shiftQueueLeft(cashier2);
            reduceBurgerStock(Burgers_per_Client);
            System.out.println("Customer " + removedCustomer + " removed from
Cashier 2.");
        } else if (!isEmptyQueue(cashier3)) {
            String removedCustomer = cashier3[0];
            shiftQueueLeft(cashier3);
            reduceBurgerStock(Burgers_per_Client);
            System.out.println("Customer " + removedCustomer + " removed from
Cashier 3.");
        } else {
            System.out.println("All cashiers are empty. No customers to
remove.");
        }
    }

    private static void reduceBurgerStock(int amount) {
        Burgers -= amount;
    }
    private static void shiftQueueLeft(String[] queue) {
        // Shift all elements of the queue one position to the left
        for (int i = 0; i < queue.length - 1; i++) {
            queue[i] = queue[i + 1];
        }
        // Set the last element to null
        queue[queue.length - 1] = null;
    }

    private static void ViewCustomersSorted() {
        // Create an array to hold all customers from all cashiers' queues

        String[] allCustomers = new String[cashier1.length + cashier2.length
+ cashier3.length];
        int index = 0;


        // Retrieve customers from cashiers and add them to the array
        for (String customer : cashier1) {
            if (customer != null) {
                allCustomers[index++] = customer;
            }
        }
        for (String customer : cashier2) {
            if (customer != null) {
                allCustomers[index++] = customer;
            }
```

```
        }
        for (String customer : cashier3) {
            if (customer != null) {
                allCustomers[index++] = customer;
            }
        }
        // Check if there are any customers in the array
        if (index == 0) {
            System.out.println("No customers in the queues.");
        } else {

            // Sort the customers array in alphabetical order
            sortCustomers(allCustomers, index);
            System.out.println("Customers Sorted in alphabetical order:");
            for (int i = 0; i < index; i++) {
                System.out.println(allCustomers[i]);
            }
        }
    }

    private static void sortCustomers(String[] customers, int size) {

        // Sort the customers array using bubble sort algorithm
        for (int i = 0; i < size - 1; i++) {
            for (int j = 0; j < size - i - 1; j++) {

                // Compare adjacent elements and swap if necessary
                if (customers[j].compareTo(customers[j + 1]) > 0) {
                    String temp = customers[j];
                    customers[j] = customers[j + 1];
                    customers[j + 1] = temp;
                }
            }
        }
    }

    private static void ViewRemainingStock() {
        // Display the remaining stock of burgers
        System.out.println("Remaining burgers stock: " + Burgers);
    }

    private static void AddBurgersToStock(Scanner scanner) {

        // Prompt the user to enter the number of burgers to add
        System.out.print("Enter the number of burgers to add: ");
        int burgersToAdd = scanner.nextInt();
        scanner.nextLine();
        // Add the specified number of burgers to the stock
        Burgers += burgersToAdd;
        System.out.println("Added " + burgersToAdd + " burgers to the stock.
New stock: " + Burgers);
    }
```

(Class version)

i. Manager main

```
      import java.util.Scanner;
import java.util.ArrayList;
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Collections;
import java.util.Comparator;

public class Manager {
    // Constants
    private static final int  Stock_of_Burgers = 50; // Total number of
burgers in stock
    private static final int cashier1_CAPACITY = 2; // Capacity of Queue 1
    private static final int cashier2_CAPACITY = 3; // Capacity of Queue 2
    private static final int cashier3_CAPACITY = 5; // Capacity of Queue 3
    private static final double Burger_price = 650.00;// Price of a burger

    // Create three food queues for cashiers
    public static final FoodQueue cashier1 = new
FoodQueue(cashier1_CAPACITY);
    public static final FoodQueue cashier2 = new
FoodQueue(cashier2_CAPACITY);
    public static final FoodQueue cashier3 = new
FoodQueue(cashier3_CAPACITY);


    // Initialize the stock of burgers
    private static int BurgersStock =  Stock_of_Burgers;
    // Waiting List
    private static final CircularQueue<Customer> Waiting_list = new
CircularQueue<>(cashier1_CAPACITY + cashier2_CAPACITY + cashier3_CAPACITY);
    // The main method of the program
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            // Print the menu and options
            System.out.println("-----------------------------------------
");
            System.out.println("*****  Burger_Food_Center  *****");
            System.out.println("-----------------------------------------
");
            System.out.println("PLEASE SELECT AN OPTION");
            System.out.println("************************");
```

```
            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 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 (Do not use library sort routine)");
            System.out.println("106 or SPD: Store Program Data into files");
            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("110 or IFQ: Print income of each queue.");
            System.out.println("999 or EXT: Exit the Program.");
            System.out.println("----------------------------------------
");
            System.out.print("Enter your choice: ");

            // Read the user's input
            String input = scanner.nextLine();

            input = input.trim().toLowerCase();
            // Process the user's choice
            switch (input) {
                case "100":
                case "vfq":
                    viewAllQueues();
                    break;
                case "101":
                case "veq":
                    viewEmptyQueues();
                    break;
                case "102":
                case "acq":
                    addCustomer(scanner);
                    break;
                case "103":
                case "rcq":
                    removeCustomer(scanner);
                    break;
                case "104":
                case "pcq":
                    removeServedCustomer();
                    break;
                case "105":
                case "vcs":
                    viewCustomersSorted();
                    break;
                case "106":
                case "spd":
                    storeProgramData();
                    break;
                case "107":
                case "lpd":
                    loadData();
                    break;
                case "108":
```

```
                case "stk":
                    viewRemainingStock();
                    break;
                case "109":
                case "afs":
                    addBurgersToStock(scanner);
                    break;
                case "110":
                case "ifq":
                    printIncome();
                    break;
                case "999":
                case "ext":
                    System.out.println("Exiting Burger_Food_Center ...");
                    System.out.println("Thanks for your visit");
                    System.out.println("Come Again");
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
            }
        }
    }

    // View all queues
    private static void viewAllQueues() {
        System.out.println();
        System.out.println("*****************");
        System.out.println("*   Cashiers    *");
        System.out.println("*****************");

        System.out.println("Cashier 1: " + cashier1);
        System.out.println("Cashier 2: " + cashier2);
        System.out.println("Cashier 3: " + cashier3);
        System.out.println();
        System.out.println("O-Occupied  X- Not Occupied");
    }

    // View empty queues
    private static void viewEmptyQueues() {
        System.out.println("Empty Cashiers:");
        if (cashier1.isEmpty()) {
            System.out.println("Cashier 1");
        }
        if (cashier2.isEmpty()) {
            System.out.println("Cashier 2");
        }
        if (cashier3.isEmpty()) {
            System.out.println("Cashier 3");
        }
    }

    // Store program data
    private static void storeProgramData() {
        try {
            FileWriter writer = new FileWriter("Manager_data.txt");
            writer.write(cashier1.getDataString("Cashier1:"));
```

```
            writer.write(cashier2.getDataString("Cashier2:"));
            writer.write(cashier3.getDataString("Cashier3:"));
            writer.close();
            System.out.println("Manager data stored successfully.");
        } catch (IOException e) {
            System.out.println("Error storing Manager data: " +
e.getMessage());
        }
    }

    // Load program data
    private static void loadData() {
        System.out.println("---Loading Data---");
        try {
            FileReader fileReader = new FileReader("Manager_data.txt");
            BufferedReader bufferedReader = new BufferedReader(fileReader);
            String line;
            // Read and display each line of the program data
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }

            bufferedReader.close();
            System.out.println(" Manager Data Loaded Successfully!");
        } catch (IOException e) {
            System.out.println("ERROR while loading Manager data");
        }
    }

    // Add customer to a queue
    private static void addCustomer(Scanner scanner) {
        System.out.print("Enter customer first name: ");
        String firstName = scanner.nextLine();

        // Validate first name
        while (!isStringOnly(firstName)) {
            System.out.println("Invalid input. Please enter a valid string
for the first name.");
            System.out.print("Enter customer first name: ");
            firstName = scanner.nextLine();
        }

        System.out.print("Enter customer last name: ");
        String lastName = scanner.nextLine();

        // Validate last name
        while (!isStringOnly(lastName)) {
            System.out.println("Invalid input. Please enter a valid string
for the last name.");
            System.out.print("Enter customer last name: ");
            lastName = scanner.nextLine();
        }

        int burgersRequired;
        while (true) {
            System.out.print("Enter the number of burgers required: ");
            String burgersInput = scanner.nextLine();
```

```
            try {
                burgersRequired = Integer.parseInt(burgersInput);
                break;
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid
integer.");
            }
        }
        // Create a customer object with the provided information
        Customer customer = new Customer(firstName, lastName,
burgersRequired);
        FoodQueue shortestQueue = getShortestQueue();

        if (shortestQueue != null && shortestQueue.getSize() <
shortestQueue.getCapacity())
        {
            // Add the customer to the shortest queue
            shortestQueue.addCustomer(customer);
            System.out.println("Customer " + customer.getFullName() + " added
to Cashier " + shortestQueue.getCashierNumber() + ".");
        } else {
            if (!Waiting_list.isFull()) {
                // Add the customer to the waiting list if all cashiers are
full
                Waiting_list.enqueue(customer);

                System.out.println("All cashiers are full. Added customer " +
customer.getFullName() + " to the waiting list.");
            } else {
                System.out.println("All cashiers and waiting list are full.
Customer not added.");
            }
        }
        // Check burger stock
        if (BurgersStock <= 10) {
            System.out.println("Warning: Low stock! Remaining burgers: " +
BurgersStock);
        }
        if (BurgersStock == 0) {
            System.out.println("Burger stock is empty...");
        }
    }

    // Check if a string consists of only letters
    private static boolean isStringOnly(String input) {
        for (char c : input.toCharArray()) {
            if (!Character.isLetter(c)) {
                return false;
            }
        }
        return true;
    }

    // Get the shortest queue among all cashiers
    private static FoodQueue getShortestQueue() {
        FoodQueue shortestQueue = null;
```

```
        // Check if QUEUE1 is shorter than its capacity
        if (cashier1.getSize() < cashier1.getCapacity()) {
            shortestQueue = cashier1;
        }
        // Check if QUEUE2 is shorter than its capacity and shorter than the
current shortest queue
        if (cashier2.getSize() < cashier2.getCapacity() && (shortestQueue ==
null || cashier2.getSize() < shortestQueue.getSize())) {
            shortestQueue = cashier2;
        }
        // Check if QUEUE3 is shorter than its capacity and shorter than the
current shortest queue
        if (cashier3.getSize() < cashier3.getCapacity() && (shortestQueue ==
null || cashier3.getSize() < shortestQueue.getSize())) {
            shortestQueue = cashier3;
        }

        return shortestQueue;
    }

    // Remove a customer from a specific cashier queue
    private static void removeCustomer(Scanner scanner) {
        System.out.print("Enter cashier number (1, 2, or 3): ");
        int cashierNumber = scanner.nextInt();
        scanner.nextLine();

        // Determine the selected cashier based on the user input
        FoodQueue selectedCashier = null;
        switch (cashierNumber) {
            case 1:
                selectedCashier = cashier1;
                break;
            case 2:
                selectedCashier = cashier2;
                break;
            case 3:
                selectedCashier = cashier3;
                break;
            default:
                System.out.println("Invalid cashier number. Customer not
removed.");
                return;
        }

        if (selectedCashier.isEmpty()) {
            // Check if the selected cashier is already empty
            System.out.println("Cashier " + cashierNumber + " is already
empty. No customers to remove.");
        } else {
            System.out.print("Enter customer index to remove (0 to " +
(selectedCashier.getSize() - 1) + "): ");
            int customerIndex = scanner.nextInt();
            scanner.nextLine();

            if (selectedCashier.isValidIndex(customerIndex)) {
                // Check if the entered customer index is valid for the
```

```
selected cashier
                Customer removedCustomer =
selectedCashier.removeCustomer(customerIndex);
                System.out.println("Customer " +
removedCustomer.getFullName() + " removed from Cashier " + cashierNumber +
".");
            } else {
                // The entered customer index is invalid
                System.out.println("Invalid customer index. Customer not
removed.");
            }
        }
    }

    // Remove a served customer
    private static void removeServedCustomer() {
        if (!cashier1.isEmpty()) {
            // Remove customer from Cashier 1
            Customer removedCustomer = cashier1.removeCustomer(0);
            // Reduce burger stock
            reduceBurgerStock(removedCustomer.getBurgersRequired());
            System.out.println("Customer " + removedCustomer.getFullName() +
" removed from Cashier 1.");
        } else if (!cashier2.isEmpty()) {
            // Remove customer from Cashier 2
            Customer removedCustomer = cashier2.removeCustomer(0);
            // Reduce burger stock
            reduceBurgerStock(removedCustomer.getBurgersRequired());
            System.out.println("Customer " + removedCustomer.getFullName() +
" removed from Cashier 2.");
        } else if (!cashier3.isEmpty()) {
            // Remove customer from Cashier 3
            Customer removedCustomer = cashier3.removeCustomer(0);
            // Reduce burger stock
            reduceBurgerStock(removedCustomer.getBurgersRequired());
            System.out.println("Customer " + removedCustomer.getFullName() +
" removed from Cashier 3.");
        } else {
            // No customers in the cashier queues
            System.out.println("All cashiers and waiting list are empty. No
customers to remove.");
            return;
        }

        // Check if there are customers in the waiting list
        if (!Waiting_list.isEmpty()) {
            // Get the next customer from the waiting list
            Customer nextCustomer = Waiting_list.dequeue();
            // Get the shortest cashier queue
            FoodQueue shortestQueue = getShortestQueue();
            if (shortestQueue != null) {
                // Add the next customer to the shortest queue
                shortestQueue.addCustomer(nextCustomer);
                System.out.println("Next customer from the waiting list, " +
nextCustomer.getFullName() + ", added to Cashier " +
shortestQueue.getCashierNumber() + ".");
            } else {
```

```
                System.out.println("All cashiers are full. Next customer from
the waiting list not added.");
            }

            // Reduce burger stock
            reduceBurgerStock(nextCustomer.getBurgersRequired());
            System.out.println("Customer " + nextCustomer.getFullName() + "
removed from the waiting list.");
        }

        // Check burger stock
        if (BurgersStock <= 10) {
            System.out.println("Warning: Low stock! Remaining burgers: " +
BurgersStock);
        }
        if (BurgersStock == 0) {
            System.out.println("Burger stock is empty...");
        }
    }

    // Reduce the burger stock by a given amount
    private static void reduceBurgerStock(int amount) {
        if (BurgersStock >= amount) {
            BurgersStock -= amount;
        } else {
            System.out.println("Cannot remove a customer. No burgers in
stock.");
        }
    }

    // View all customers sorted in alphabetical order
    private static void viewCustomersSorted() {
        ArrayList<Customer> allCustomers = new ArrayList<>();

        allCustomers.addAll(cashier1.getCustomers());
        allCustomers.addAll(cashier2.getCustomers());
        allCustomers.addAll(cashier3.getCustomers());

        if (allCustomers.isEmpty()) {
            // Check if the list of customers is empty
            System.out.println("Cashier is empty.");
        } else {
            sortCustomers(allCustomers);
            System.out.println("All Customers Sorted in alphabetical
order:");
            // Display the sorted customers' names
            for (Customer customer : allCustomers) {
                System.out.println(customer.getFullName());
            }
        }
    }

    // Sort the customers in alphabetical order
    private static void sortCustomers(ArrayList<Customer> customers) {
        Collections.sort(customers, new Comparator<Customer>() {
                    public int compare(Customer c1, Customer c2) {
                        // Compare the full names of two customers
```

```
                        return c1.getFullName().compareTo(c2.getFullName());
                    }
                }
        );
    }

    // View the remaining burger stock
    private static void viewRemainingStock() {
        System.out.println("Remaining burgers stock: " + BurgersStock);
    }

    // Add burgers to the stock
    private static void addBurgersToStock(Scanner scanner) {
        System.out.print("Enter the number of burgers to add: ");
        int burgersToAdd = scanner.nextInt();
        scanner.nextLine();

        BurgersStock += burgersToAdd;
        System.out.println("Added " + burgersToAdd + " burgers to the
stock.");
        System.out.println("New stock: " + BurgersStock);
    }

    // Print the income of each queue
    private static void printIncome() {
        double income1 = cashier1.getIncome(Burger_price);
        double income2 = cashier2.getIncome(Burger_price);
        double income3 = cashier3.getIncome(Burger_price);

        System.out.println("Income of each queue:");
        System.out.println("Cashier 1: " + income1);
        System.out.println("Cashier 2: " + income2);
        System.out.println("Cashier 3: " + income3);
    }
}

// Circular Queue implementation
class CircularQueue<T> {
    private Object[] elements;
    private int front;
    private int rear;
    private int capacity;
    private int size;

    public CircularQueue(int capacity) {
        this.capacity = capacity;
        this.elements = new Object[capacity];
        this.front = -1;
        this.rear = -1;
        this.size = 0;
    }

    // Check if the queue is empty
    public boolean isEmpty() {
        return size == 0;
    }
```

```
    // Check if the queue is full
    public boolean isFull() {
        return size == capacity;
    }

    // Get the current size of the queue
    public int getSize() {
        return size;
    }

    // Enqueue an item into the queue
    public void enqueue(T item) {
        if (isFull()) {
            throw new IllegalStateException("CircularQueue is full.");
        }

        if (isEmpty()) {
            front = 0;
        }

        rear = (rear + 1) % capacity;
        elements[rear] = item;
        size++;
    }

    // Dequeue an item from the queue
    public T dequeue() {
        if (isEmpty()) {
            return null;
        }

        T item = (T) elements[front];
        elements[front] = null;

        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % capacity;
        }

        size--;
        return item;
    }
}
```

ii. **Customer class**

```
public class Customer {
    private String firstName; // Variable to store the customer's first name
    private String lastName; // Variable to store the customer's last name
    private int burgersRequired; // Variable to store the number of burgers
required by the customer

    // Constructor to initialize the Customer object with the provided
details
    public Customer(String firstName, String lastName, int burgersRequired) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.burgersRequired = burgersRequired;
    }

    // Method to get the full name of the customer
    public String getFullName() {
        return firstName + " " + lastName;
    }

    // Method to get the number of burgers required by the customer
    public int getBurgersRequired() {
        return burgersRequired;
    }
}
```

ii. FoodQueue class

```
import java.util.ArrayList;

class FoodQueue {
    private ArrayList<Customer> customers; // ArrayList to store the
customers in the queue
    private int capacity; // Maximum capacity of the queue

    // Constructor to initialize the FoodQueue object with the provided
capacity
    public FoodQueue(int capacity) {
        this.capacity = capacity;
        this.customers = new ArrayList<>(capacity);
    }

    // Method to get the current size of the queue (number of customers)
    public int getSize() {
        return customers.size();
    }

    // Method to get the capacity of the queue
    public int getCapacity() {
```

```
        return capacity;
    }

    // Method to check if the queue is empty
    public boolean isEmpty() {
        return customers.isEmpty();
    }

    // Method to check if the queue is full
    public boolean isFull() {
        return customers.size() == capacity;
    }

    // Method to get the list of customers in the queue
    public ArrayList<Customer> getCustomers() {
        return customers;
    }

    // Method to get the cashier number associated with the queue
    public int getCashierNumber() {
        if (this == Manager.cashier1) {
            return 1;
        } else if (this == Manager.cashier2) {
            return 2;
        } else if (this == Manager.cashier3) {
            return 3;
        } else {
            return -1;
        }
    }

    // Method to add a customer to the queue if it is not full
    public void addCustomer(Customer customer) {
        if (!isFull()) {
            customers.add(customer);
        }
    }

    // Method to remove a customer from the queue at the specified index
    public Customer removeCustomer(int index) {
        if (isValidIndex(index)) {
            return customers.remove(index);
        }
        return null;
    }

    // Method to check if the provided index is valid within the range of the
queue
    public boolean isValidIndex(int index) {
        return index >= 0 && index < customers.size();
    }

    // Method to calculate the total income generated by the customers in the
queue based on the burger price
    public double getIncome(double burgerPrice) {
        double income = 0.0;
        for (Customer customer : customers) {
```

```
            income += customer.getBurgersRequired() * burgerPrice;
        }
        return income;
    }

    // Method to get a formatted string representation of the customer data
in the queue
    public String getDataString(String prefix) {
        StringBuilder sb = new StringBuilder();
        for (Customer customer : customers) {

sb.append(prefix).append(customer.getFullName()).append(":").append(customer.
getBurgersRequired()).append("\n");
        }
        return sb.toString();
    }

    // Override the toString() method to get a string representation of the
queue's occupancy status
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < capacity; i++) {
            if (i < customers.size()) {
                sb.append("O"); // O represents an occupied position in the
queue
            } else {
                sb.append("X"); // X represents an unoccupied position in the
queue
            }
            if (i != capacity - 1) {
                sb.append(" ");
            }
        }
        return sb.toString();
    }
}
```

<<END>>