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	: **20221671 / w1999483**
Student First Name	: Hansaja
Student Surname	: Hapuarachchi

"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 : Hansaja Hapuarachchi

Student ID : 20221671

Test Cases

	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
1	**Food Queue Initialized Correctly After the program starts,** 100 or VFQ	Displays X in all the queue slots.	Displays X in all the queue slots.	Pass
2	**View all Empty Queues.** 101 or VEQ	Display, Queue 1 Queue 2 Queue 3	Display, Queue 1 Queue 2 Queue 3	Pass

3	**Add customer "senuri" to Queue 2** 102 or ACQ Enter Queue: 2 Enter Name: senuri	Display 'Customer Senuri added to the queue successfully.	Display "Customer Senuri added to the queue successfully.	Pass
4	**Remove a customer from Queue 1** 103 or RCQ Enter the cashier number (1, 2, or 3):	Display Customer "Customer Name" removed from the queue.	Display Customer "Customer Name" removed from the queue.	Pass
5	**Remove a Served customer** 104 or PCQ	Display "Customer (Name) served and removed from queue"	Display "Customer (Name) served and removed from queue"	Pass
6	**View Customers Sorted in alphabetical order.** 105 or VCS	Display all customer names in alphabetical order.	Display all customer names in alphabetical order. Example: "Customers Sorted in alphabetical order:" Binuri Mindula Hansaja Perera Kaviru Ransara Sahan Madhawa	Pass

7	**Store Program Data into file.** 106 or SPD	Create a text file and write Customer Names per queue and the remaining amount of Burgers	Created a text file and write Customer Names per queue and the remaining amount of Burgers	Pass
8	**Load Program Data from the file**. 107 or LPD	Read the data from the text file created on option 6 and load data from it.	Read the data from the text file created on option 6 and load data from it.	Pass

9	**View Remaining Burger Stock.** 108 or STK	Display "Remaining burger stock: " and show a warning message if stock is running low	Display "Remaining burger stock: " and shows a message when burger stock reached 10	Pass
10	**Add burgers to Stock.** 109 or AFS	Display "Enter the number of burgers to add: "	Display "Enter the number of burgers to add: "	Pass
11	**Exit the Program.** 999 or EXT	Display "Exiting the program" Exit from the program.	Display "Exiting the program..." Process finished with exit code 0	Pass
12	**Add Customer To a Queue (Task 2)** 102 or ACQ Enter First Name Enter Last Name Number of Burgers required	Gets customer's first name, last name, and number of burgers required. Adds customer to the queue with minimum length.	Gets customer's first name, last name, and number of burgers required. Adds customer to the queue with minimum length.	Pass

13	**Print the Income of each queue (Task 2)** 110 or IFQ	Calculate and display the income of each queue based on the number of burgers required by the customers and the burger price.	Calculates and prints the income of each queue separately. Example: Income of Each Queue: Queue 1: Rs.3250.00 Queue 2: Rs.6500.00 Queue 3: Rs.3900.00	Pass
14	**Adding customers to a waiting list after all three queues are filled. (Task 3)** ACQ or 102	Adds new customers to a waiting list after all 3 queues are filled.	Adds new customers to a waiting list after all queues are full and displays the message "All queues are full! Customer added to the waiting list."	Pass
15	**Adds a waiting list customer to the queue after removing a Served customer** 104 or PCQ	Added a waiting list customer to the queue after removing a Served customer	Added a waiting list customer to the queue after removing a Served customer	Pass

Discussion

When choosing test cases to cover all aspects of my program, I considered the different scenarios that can occur.

Viewing Queues: This helps verify that the program displays the queues correctly, including their status (occupied or empty).

Adding and Removing Customers: I included test cases to cover adding a customer to a queue and removing a customer from a queue. This ensures that the program correctly handles customer addition and removal.

Sorting Customers: I included a test case to view customers sorted in alphabetical order. This ensures that the program correctly sorts the customers without using a library sort routine, as per the coursework description.

Burger Stock Management: I included test cases to add burgers to the stock and view the remaining burger stock. These test cases cover the functionality of managing the burger stock and verifying its accuracy.

Storing and Loading Program Data: I included test cases to store program data into a file and load program data from a file. This ensures that the program can successfully save and retrieve its state, including the burger stock and queue data.

By including these test cases, I aimed to cover the major functionalities of the program and ensure that it behaves as expected in various scenarios. This helps identify any issues or bugs related to customer management, queue operations, sorting, and stock management.

Code:

Main Method:

```
import java.io.FileWriter;

import java.io.FileReader;

import java.io.IOException;

import java.io.BufferedReader;

import java.util.Scanner;


public class Shop {

    private FoodQueue[] queues;

    private FoodQueue waitingList;

    private int remainingBurgers;

    private int burgerPrice;

    private int income;


    public Shop() {

        queues = new FoodQueue[3];

        queues[0] = new FoodQueue(2);

        queues[1] = new FoodQueue(3);
```

```
        queues[2] = new FoodQueue(5);
        waitingList = new FoodQueue(10);
        remainingBurgers = 50;
        burgerPrice = 650;
        income = 0;
    }

    //Displaying the menu
    private void displayMenu() {
        System.out.println("\n\nOptions Menu:");
        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 (Do not use
library sort routine)");
        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("110 or IFQ: Print income of each queue");
    System.out.println("999 or EXT: Exit the Program");
}

//Displaying all queues
private void viewAllQueues() {
    System.out.println("\n");
    System.out.println("******************");
    System.out.println("*   Cashiers    *");
    System.out.println("******************\n");

    for (FoodQueue queue : queues) {
        queue.displayQueue();
    }
    System.out.println("\nX - Not Occupied\tO - Occupied");
}

//Displaying empty queues
```

```
private void viewEmptyQueues() {
    boolean emptyQueues = true;

    for (FoodQueue queue : queues) {
        if (!queue.isEmpty()) {
            emptyQueues = false;
            break;
        }
    }

    if (emptyQueues) {
        System.out.println("All queues are empty.");
    } else {
        System.out.println(" Empty Queues.");
        for (FoodQueue queue : queues) {
            if (queue.isEmpty()) {
                System.out.println("Queue " + queue.getQueueNumber());
            }
        }
```

```
    }
}

private int nextQueueSlot = 0; // To keep track of the next available slot for each queue

//Customers Adding
private void addCustomerToQueue() {
    if (waitingList.isFull()) {
        System.out.println("Waiting List is full. Customer cannot be added.");
        return;
    }

    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the first name of the customer: ");
    String firstName = scanner.nextLine();
    System.out.print("Enter the last name of the customer: ");
    String lastName = scanner.nextLine();

    int burgersRequired = 0;
```

```
boolean validBurgers = false;
while (!validBurgers) {
    System.out.print("Enter the number of burgers required: ");
    if (scanner.hasNextInt()) {
        burgersRequired = scanner.nextInt();
        validBurgers = true;
    } else {
        System.out.println("Invalid input. Please enter a valid integer.");
        scanner.nextLine(); // Consume invalid input
    }
}

Customer customer = new Customer(firstName, lastName, burgersRequired);

FoodQueue targetQueue = null;
for (int i = 0; i < queues.length; i++) {
    int queueIndex = (nextQueueSlot + i) % queues.length;
    FoodQueue currentQueue = queues[queueIndex];
    if (!currentQueue.isFull()) {
```

```
            targetQueue = currentQueue;

            nextQueueSlot = (queueIndex + 1) % queues.length; // Update nextQueueSlot for the next customer

            break;

        }

    }



    if (targetQueue != null) {

        targetQueue.addCustomer(customer);

        remainingBurgers -= burgersRequired;

        if (remainingBurgers <= 10) {

            System.out.println("...Warning... Remaining burgers stock is low (" + remainingBurgers + " burgers left).");

        }

        System.out.println("Customer " + customer.getFullName() + " added to Queue " + targetQueue.getQueueNumber());

    } else {

        waitingList.addCustomer(customer);

        System.out.println("All queues are full. Customer added to Waiting List.");

    }

  }
```

```
    //Customers removing
    private void removeCustomerFromQueue() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the queue number: ");
        int queueNumber = 0;
        boolean validQueue = false;
        while (!validQueue) {
            if (scanner.hasNextInt()) {
                queueNumber = scanner.nextInt();
                if (queueNumber >= 1 && queueNumber <= 3) {
                    validQueue = true;
                } else {
                    System.out.println("Invalid queue number. Please enter a valid queue number (1-
3).");
                    System.out.print("Enter the queue number: ");
                }
```

```
            } else {
                System.out.println("Invalid input. Please enter a valid integer.");
                System.out.print("Enter the queue number: ");
                scanner.nextLine(); // Consume invalid input
            }
        }
        scanner.nextLine(); // Consume newline character

        FoodQueue queue = getQueue(queueNumber);
        if (queue != null) {
            int position = 0;
            boolean validPosition = false;
            while (!validPosition) {
                System.out.print("Enter the position of the customer to remove (1-" +
queue.getQueueSize() + "): ");
                if (scanner.hasNextInt()) {
                    position = scanner.nextInt();
                    if (position >= 1 && position <= queue.getQueueSize()) {
                        validPosition = true;
                    } else {
```

```
                System.out.println("Invalid position. Please enter a valid position (1-" +
queue.getQueueSize() + ").");

                }

            } else {

                System.out.println("Invalid input. Please enter a valid integer.");

                scanner.nextLine(); //Consume invalid input

            }

        }

        scanner.nextLine(); //Consume newline character



        Customer removedCustomer = queue.removeCustomer(position);

        if (removedCustomer != null) {

            System.out.println("Customer " + removedCustomer.getFullName() + " removed from
Queue " + queueNumber);



            if (!waitingList.isEmpty()) {

                Customer nextCustomer = waitingList.removeCustomer(1);

                queue.addCustomer(nextCustomer);

                System.out.println("Next customer in Waiting List added to Queue " +
queueNumber);

            }
```

```
            } else {
                System.out.println("Invalid position. Customer removal failed.");
            }
        } else {
            System.out.println("Invalid queue number. Customer removal failed.");
        }
    }


    //Served customers removing
    private void removeServedCustomer() {
        for (FoodQueue queue : queues) {
            if (!queue.isEmpty()) {
                Customer servedCustomer = queue.removeCustomer(1);
                int burgersServed = servedCustomer.getBurgersRequired();
                int customerIncome = burgersServed * burgerPrice;
                income += customerIncome;
                System.out.println("Customer " + servedCustomer.getFullName() + " served from
Queue " + queue.getQueueNumber() + " (Income: $" + customerIncome + ")");
                return; // Serve only the first customer found and exit the method
```

```
        }
    }
    System.out.println("No customers to serve in any queue.");
}

//Sorting customers
private void viewCustomersSorted() {
    Customer[] allCustomers = new Customer[getTotalCustomers()];
    int index = 0;

    for (FoodQueue queue : queues) {
        Customer[] queueCustomers = queue.getAllCustomers();
        System.arraycopy(queueCustomers, 0, allCustomers, index, queue.getQueueSize());
        index += queue.getQueueSize();
    }

    if (index > 0) {
        //Sort the customers alphabetically
        sortCustomers(allCustomers);
```

```
        System.out.println("Customers Sorted in Alphabetical Order:");
        for (Customer customer : allCustomers) {
            System.out.println(customer.getFullName());
        }
    } else {
        System.out.println("No customers found.");
    }
}

//Store data
private void storeProgramData() {
    try {
        FileWriter writer = new FileWriter("program_data.txt");
        for (FoodQueue queue : queues) {
            Customer[] customers = queue.getAllCustomers();
            writer.write("Queue " + queue.getQueueNumber() + ":\n");
            for (int i = 0; i < queue.getQueueSize(); i++) {
                Customer customer = customers[i];
```

```
                writer.write(customer.getFullName() + " - Burgers: " +
customer.getBurgersRequired() + "\n");

            }

            writer.write("\n");

        }

        writer.write("Remaining Burgers: " + remainingBurgers + "\n");

        writer.close();

        System.out.println("Program data successfully stored into file.");

    } catch (IOException e) {

        System.out.println("An error occurred while storing program data.");

        e.printStackTrace();

    }

}



//Loading data

private void loadProgramData() {

    try {

        BufferedReader reader = new BufferedReader(new FileReader("program_data.txt"));

        String line;

        while ((line = reader.readLine()) != null) {
```

```
            if (!line.isEmpty()) {
                System.out.println(line);
            }
        }
        reader.close();
    } catch (IOException e) {
        System.out.println("An error occurred while loading program data.");
        e.printStackTrace();
    }
}

//Remaining stocks
private void viewRemainingStock() {
    System.out.println("Remaining Burgers Stock: " + remainingBurgers);
}

//Adding burgers
private void addBurgersToStock() {
    Scanner scanner = new Scanner(System.in);
```

```
    int burgersToAdd = 0;
    boolean validBurgers = false;
    while (!validBurgers) {
        System.out.print("Enter the number of burgers to add: ");
        if (scanner.hasNextInt()) {
            burgersToAdd = scanner.nextInt();
            validBurgers = true;
        } else {
            System.out.println("Invalid input. Please enter a valid integer.");
            scanner.nextLine(); //Consume invalid input
        }
    }
    scanner.nextLine(); //Consume newline character


    remainingBurgers += burgersToAdd;
    System.out.println(burgersToAdd + " burgers added to stock.");
}
```

```
//Printing income
private void printIncomeOfEachQueue() {
    System.out.println("Income of Each Queue:");
    for (FoodQueue queue : queues) {
        int totalIncome = 0;
        Customer[] customers = queue.getAllCustomers();
        for (Customer customer : customers) {
            int burgersRequired = customer.getBurgersRequired();
            totalIncome += burgersRequired * burgerPrice;
        }
        System.out.println("Queue " + queue.getQueueNumber() + ": Rs." + totalIncome+".00");
    }
    System.out.println("Served customer income: Rs." + income+".00");
}



//Exiting the programme
private void exitProgram() {
    System.out.println("Exiting the program...");
```

```
    System.exit(0);
}

private FoodQueue getQueue(int queueNumber) {
    for (FoodQueue queue : queues) {
        if (queue.getQueueNumber() == queueNumber) {
            return queue;
        }
    }
    return null;
}

private int getTotalCustomers() {
    int totalCustomers = 0;
    for (FoodQueue queue : queues) {
        totalCustomers += queue.getQueueSize();
    }
```

```
    return totalCustomers;
}

private void sortCustomers(Customer[] customers) {
    for (int i = 0; i < customers.length - 1; i++) {
        for (int j = i + 1; j < customers.length; j++) {
            if (customers[i].compareTo(customers[j]) > 0) {
                Customer temp = customers[i];
                customers[i] = customers[j];
                customers[j] = temp;
            }
        }
    }
}

//Switch cases
public void start() {
    Scanner scanner = new Scanner(System.in);
```

```
while (true) {
    displayMenu();
    System.out.print("\nEnter your choice: ");
    String choice = scanner.nextLine().toUpperCase();

    switch (choice) {
        case "100":
        case "VFQ":
            viewAllQueues();
            break;
        case "101":
        case "VEQ":
            viewEmptyQueues();
            break;
        case "102":
        case "ACQ":
            addCustomerToQueue();
            break;
        case "103":
```

```
case "RCQ":
    removeCustomerFromQueue();
    break;
case "104":
case "PCQ":
    removeServedCustomer();
    break;
case "105":
case "VCS":
    viewCustomersSorted();
    break;
case "106":
case "SPD":
    storeProgramData();
    break;
case "107":
case "LPD":
    loadProgramData();
    break;
```

```
        case "108":
        case "STK":
            viewRemainingStock();
            break;
        case "109":
        case "AFS":
            addBurgersToStock();
            break;
        case "110":
        case "IFQ":
            printIncomeOfEachQueue();
            break;
        case "999":
        case "EXT":
            exitProgram();
            break;
        default:
            System.out.println("Invalid choice. Please try again.");
    }
```

```
        }
    }


    //Main method
    public static void main(String[] args) {
        Shop shop = new Shop();
        shop.start();
    }
}
```

FoodQueue Class:

```
public class FoodQueue {
    private int queueNumber ;
    private static int queueNumberCounter = 1;

    private Customer[] customers;
    private int maxSize;
    private int size;

    public FoodQueue(int maxSize) {
        this.queueNumber = generateQueueNumber();
        this.maxSize = maxSize;
        this.size = 0;
        this.customers = new Customer[maxSize];
    }

    public int getQueueNumber() {
        return queueNumber;
    }
```

```
public int getQueueSize() {
    return size;
}

public int getMaxSize() {
    return maxSize;
}

public boolean isFull() {
    return size == maxSize;
}

public boolean isEmpty() {
    return size == 0;
}

//Displaying queue
public void displayQueue() {
    System.out.print("Queue " + queueNumber + ": ");
```

```
    for (int i = 0; i < size; i++) {
        System.out.print("O ");
    }
    for (int i = size; i < maxSize; i++) {
        System.out.print("X ");
    }
    System.out.println();
}

// Adding customers
public void addCustomer(Customer customer) {
    if (size < maxSize) {
        customers[size] = customer;
        size++;
    }
}

//Removing customers
public Customer removeCustomer(int position) {
    if (position >= 1 && position <= size) {
```

```
        Customer removedCustomer = customers[position - 1];
        for (int i = position - 1; i < size - 1; i++) {
            customers[i] = customers[i + 1];
        }
        size--;
        customers[size] = null;
        return removedCustomer;
    }
    return null;
}

//Getting customers
public Customer[] getAllCustomers() {
    Customer[] allCustomers = new Customer[size];
    System.arraycopy(customers, 0, allCustomers, 0, size);
    return allCustomers;
}
//generateQueueNumber
```

```
    //getting queue numbers
    private int generateQueueNumber() {
        int generatedQueueNumber = queueNumberCounter;
        queueNumberCounter = (queueNumberCounter % 3) + 1;
        return generatedQueueNumber;
    }
}
```

Customer Class:

```
public class Customer implements Comparable<Customer> {
    private String firstName;
    private String lastName;
    private int burgersRequired;

    public Customer(String firstName, String lastName, int burgersRequired) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.burgersRequired = burgersRequired;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
```

```
    public int getBurgersRequired() {
        return burgersRequired;
    }

    public String getFullName() {
        return firstName + " " + lastName;
    }

    @Override
    public int compareTo(Customer otherCustomer) {
        return this.getFullName().compareToIgnoreCase(otherCustomer.getFullName());
    }
}
```