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	: 26/08/2023
Student ID	: 20222165 / W1985549
Student First Name	: SAHAN
Student Surname	: DHARMARATHNE

"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 : THANTRIGE SAHAN VIMUKTHI DHARMARATHNE

Student ID : 20222165

Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1.	Food Queue Initialized Correctly After program starts, 100 or VFQ.	Displays 'empty' for all Queues. ********** * Cashiers * ********** X X X X X X X X X X	Displays 'empty' for all Queues. ********** * Cashiers * ********** X X X X X X X X X X	Pass
2.	View empty queues, 101 or ACQ. Ex:- In this case all the cashiers are like following. ********** * Cashiers * ********** O O X O O X X X X X	The empty queue slots are printing like following. Queue 1: -, - Queue 2: -, -, 3 Queue 3: 1, 2, 3, 4, 5	The empty queue slots are printing like following. Queue 1: -, - Queue 2: -, -, 3 Queue 3: 1, 2, 3, 4, 5	Pass

3.	Adding a customer to a queue, 102 or ACQ. (As a example here in the results, adding a customer with giving the customers first name, customers second name, customers ID and customer required pizza amount. O – Occupied X – Not Occupied	********** * Cashiers * ********** O X X O X X X X X X	********** * Cashiers * ********** O X X O X X X X X X	Pass
4.	Removing a customer from a specific location, 103 or RCQ. (As a example here in the results, Removing the customer from the second queue in position 3.)	********** * Cashiers * ********** O O O O O O X O O O	********** * Cashiers * ********** O O O O O O X O O O	Pass
5.	Removing a served customer from relevant queue,104 or PCQ. (As a example here in the results, Removing the served customer from 1st queue and 3 rd queue.)	********** * Cashiers * ********** O O O X O O O O O X	********** * Cashiers * ********** O O O X O O O O O X	Pass

6.	View customers sorted in alphabetical order,105 or VCS. (As a example if the added two customers first and second name pairs are respectively Amal, Mahesh and Kamal, Vimukthi.)	Show the customers in alphabetical order. ex:- Amal Mahesh Kamal Vimukthi	Show the customers in alphabetical order. ex:- Amal Mahesh Kamal Vimukthi	Pass
7.	Store program data in file, 106 or SPD.	We can store the customer details like customers first and second names, customers ID, customers required pizza amount, customers positions.	We can store the customer details like customers first and second names, customers ID, customers required pizza amount, customers positions.	Pass
8.	Load program data from file, 107 or LPD.	We can load the customer details from the stored file, like customers first and second names, customers ID, customer required pizza amount, customer position.	We can load the customer details from the stored file, like customers first and second names, customers ID, customer required pizza amount, customer position.	Pass
9.	View remaining pizza stock, 108 or STK.	When enter 108 or STK this method will give remaining pizza stock. And also give a warn message when the pizza stock reached up to 20 pizzas. Ex:-When customer added and he	When enter 108 or STK this method will give remaining pizza stock. And also give a warn message when the pizza stock reached up to 20 pizzas. Ex:-When customer added and he	Pass

		required 10 pizzas, This method show the remaining pizza stock is 90. (Because the pizza stock has only 100 pizzas.)	required 10 pizzas, This method show the remaining pizza stock is 90. (Because the pizza stock has only 100 pizzas.)	
10.	Add pizzas to stock, 109 or AFS.	When you enter 109 or AFS this method will allows user to add pizzas to the main pizza stock. This will print a message with total pizza stock. Ex:- If you add 10 pizzas when the remaining pizza stock is 50.The printed message will be like this "The pizza stock after adding pizzas : 60 "	When you enter 109 or AFS this method will allows user to add pizzas to the main pizza stock. This will print a message with total pizza stock. Ex:- If you add 10 pizzas when the remaining pizza stock is 50.The printed message will be like this "The pizza stock after adding pizzas : 60 "	Pass
11.	Income of each queue, 110 or IFQ.	User can view the income of each queue. Ex:- Each customer has required 10 pizzas and all cashiers are full with customers.	User can view the income of each queue. Ex:- Each customer has required 10 pizzas and all cashiers are full with customers.	Pass

		********** * Cashiers * ********** O O O O O O O O O O Cashier Position 01 : LKR 27000 Cashier Position 02 : LKR 40500 Cashier Position 03 : LKR 67500	********** * Cashiers * ********** O O O O O O O O O O Cashier Position 01 : LKR 27000 Cashier Position 02 : LKR 40500 Cashier Position 03 : LKR 67500	
12.	Exit the program, 999 or EXT.	Program will give message, "You Have Exit The Program and it is stopped!!!"	Program will give message, "You Have Exit The Program and it is stopped!!!"	Pass

Discussion

01. Test case 1:-

Food Queue Initialized Correctly, this test case ensure that the food queue and all the cashiers are initialized correctly. In displays empty for all queues. It shows all the positions that customers can add. From that we can verify the cashiers and queue structure is setup correctly.

02. Test case 2:-

This test case showing all the queues which have at least one empty slot. And also it indicating the empty slot while printing the empty queues. The cashiers filled with maximum customers are not shown any empty slots. There for this case ensures that the view all empty queues is setup as expected.

03. Test case 3:-

This will be add a customer to the queue that has the minimum length at the appointed time. First the program will asks users first name, then asks the second name, then asks the customers ID, then asks the pizza amount that customer required. After that the customer will add to the queue that has a minimum length. If all the main three cashiers are full of customers then the next customer will be added to a waiting list cashier.

04. Test case 4:-

This method ensure that a customer can be removed from a queue in a specific location. In this case the method prompt to the user to enter the cashier number and also prompt to the user to enter the relevent position of the cashier number. After that the customer will remove from that specific location and the other customers in that queue will comes up. In this case when removing customers pizza count will not be reduced. (When removing customers from the queue, if there was any customer in waiting list cashier they will automatically come to the empty queue slot.)

05. Test case 5:-

In this test case the program will prompt to user to enter the queue number that user want to remove the served customer. When we give the queue number it will remove the first customer from the cashier and other customers will comes up. In this case when removing the customers pizza count will be reduced.

(When removing customers from the queue, if there was any customer in waiting list cashier they will automatically come to the empty queue slot.)

06. Test case 6:-

This method will display the added all customers in alphabetical order. It verifies the program is sorting the customers names correctly.

07. Test case 7:-

In this test case the program will write the all customer details like customers first name, customers second name, customers ID, customers required pizza amount, customers queue position in a file. This method ensure that all added customer details are store correctly in a file. It verifies the method is working correctly.

08. Test case 8:-

In this test case the program will load and print all customer details like customers first name, customers second name, customers ID, customers required pizza amount, customers queue position. This method ensure that all added customer details are store correctly in a file. And also this method ensure that all added customer details are loaded correctly to the program. It verifies the method is working correctly.

09. Test case 9:-

This test case verifies that when customers are adding to the queues the pizza stock is being updating with the required pizza amount of customers. And also the program is warning when the pizza stock is reached up to 20 pizzas. This ensures that this method is working correctly.

10. Test case 10:-

This test case ensures that when the program give a warning message that the pizza stock is reached up to 20 pizzas, the pizza stock should be updated by adding more pizzas. We can use this method to add more pizzas to the main pizza stock. After adding pizzas the main pizza stock will be updated by the new added pizzas. That verifies this method is working correctly.

11. Test case 11:-

In this test case it can calculate the relevant queue's incomes. This test case verifies the method is working correctly because the calculations of each cashiers are correct.

12. Test case 12:-

This test case ensures that when user want to end the program, it will stopped the program by entering 999 or EXT.

Code :

SnackKingQueueManagementSystem (Main Method)

```
//Thantrige Sahan Vimukthi Dharmarathne - 20222165 - W1985549.

import java.util.Scanner;


public class SnackKingQueueManagementSystem {

    //Creating the Scanner object for the programs all inputs.
    public static Scanner input = new Scanner(System.in);


    public static void main(String[] args) {

        FoodQueue foodQueue = new FoodQueue();//Creating the object from the
Food queue class.
        foodQueue.queue();

        boolean Exit_Code = true;// Getting a boolean variable for code exit.
        //Getting a while loop for repetition of the program console menu.
        while (Exit_Code) {
            // Printing all the console menu for the operator.
            System.out.println(" ");
            System.out.println(" ");

System.out.println("=======================================CONSOLE
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.");
            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 pizza Stock.");
            System.out.println("109 or AFS: Add pizza to Stock.");
            System.out.println("110 or IFQ: Print the income of each
queue.");
            System.out.println("999 or EXT: Exit the Program.");
            System.out.println(" ");

System.out.println("============================================================
==================================");

            //Getting the operators input for the User's Option.
            System.out.print("Enter Your Menu Option : ");
            String Users_Option = input.next();
```

```
            //Use the switch case for all options in the console menu.
            //And also calling the methods in the Food Queue class by above
created class object.
            switch (Users_Option) {
                case "100":
                case "VFQ":
                    foodQueue.view_all_Queues();
                    break;

                case "101":
                case "VEQ":
                    foodQueue.view_all_Empty_Queues();
                    break;

                case "102":
                case "ACQ":
                    foodQueue.adding_customer_to_A_Queue();
                    break;

                case "103":
                case "RCQ":
                    foodQueue.removing_A_Customer_From_Queue();
                    break;

                case "104":
                case "PCQ":
                    foodQueue.removing_A_Served_Customer();
                    break;

                case "105":
                case "VCS":
                    foodQueue.view_customers_Sorted();
                    break;

                case "106":
                case "SPD":
                    foodQueue.program_Data_Storing();
                    break;

                case "107":
                case "LPD":
                    foodQueue.program_Data_Loading();
                    break;

                case "108":
                case "STK":
                    foodQueue.view_remaining_Pizza_Stock();
                    break;

                case "109":
                case "AFS":
                    foodQueue.add_Pizza_Stock();
                    break;

                case "110":
                case "IFQ":
                    foodQueue.print_Income_Of_Queue();
```

```
                    break;

                case "999":
                case "EXT":
                    Exit_Code = false;
                    foodQueue.Exit_Program();
                    break;

                default:
                    System.out.println("Invalid option entered!Try
again.");//Error message, if the user enter invalid console menu option.
            }

System.out.println("==========================================================
===================================");
        }
    }
}
```

Food Queue Class

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Objects;
import java.util.Scanner;

public class FoodQueue {

    static Customer[][] PositionOfCashier = new Customer[3][];//Create a 2d
array.
    static Customer[] WaitingList_Queue = new Customer[5];//Create a 1d array
for the waiting queue.

    public void queue() { //Making 3 customer arrays.
        PositionOfCashier[0] = new Customer[2];
        PositionOfCashier[1] = new Customer[3];
        PositionOfCashier[2] = new Customer[5];
    }

    //Initializing three integer variables for each cashier positions.
    static int count_of_Cashier_Position_01 = 0;
    static int count_of_Cashier_Position_02 = 0;
    static int count_of_Cashier_Position_03 = 0;

    //Getting two integer variables for pizza stock and sold pizza stock.
    static int Pizza_Stock = 100;
    static int Customer_RequiredPizzaAmount = 0;

    public static Scanner input = new Scanner(System.in);//Creating the
Scanner object for the programs all inputs.

    //View all cashier queues when operator give the users option input as
100 or VFQ.
    public static void view_all_Queues() {
        System.out.println("*****************");
        System.out.println("*    Cashiers   *");
        System.out.println("*****************");

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < PositionOfCashier.length; j++) {
                if (i < PositionOfCashier[j].length) {
                    if (PositionOfCashier[j][i] == null) {
                        System.out.print(" X    ");
                    } else {
                        System.out.print(" O    ");
                    }
                } else {
                    System.out.print("      ");
                }
            }
            System.out.println();
```

```
        }
    }

    //View all empty cashier queues when operator give the users option input
as 101 or VEQ.
    public static void view_all_Empty_Queues() {
        boolean Cashierline_01 = true;
        boolean Cashierline_02 = true;
        boolean Cashierline_03 = true;

        //Using print format for print the empty cashiers.
        if (Cashierline_01) {
            System.out.print("Queue 1: ");
            System.out.printf("%s\t", PositionOfCashier[0][0] != null ? "-,"
: "1,");
            System.out.printf("%s %n", PositionOfCashier[0][1] != null ? "-"
: "2");
        }

        if (Cashierline_02) {
            System.out.print("Queue 2: ");
            System.out.printf("%s\t", PositionOfCashier[1][0] != null ? "-,"
: "1,");
            System.out.printf("%s\t", PositionOfCashier[1][1] != null ? "-,"
: "2,");
            System.out.printf("%s %n", PositionOfCashier[1][2] != null ? "-"
: "3");
        }

        if (Cashierline_03) {
            System.out.print("Queue 3: ");
            System.out.printf("%s\t", PositionOfCashier[2][0] != null ? "-,"
: "1,");
            System.out.printf("%s\t", PositionOfCashier[2][1] != null ? "-,"
: "2,");
            System.out.printf("%s\t", PositionOfCashier[2][2] != null ? "-,"
: "3,");
            System.out.printf("%s\t", PositionOfCashier[2][3] != null ? "-,"
: "4,");
            System.out.printf("%s %n", PositionOfCashier[2][4] != null ? "-"
: "5");
        }
        else {
            System.out.println("There are no empty queues.");
        }
    }

    private static int LengthOfQueue (int queue) {//Making the method to find
the length of queues.

        int count = 0;//Initialise an integer count variable.

        if (queue == 1) {
            for (int j = 0; j < 2; j++) {
                if (PositionOfCashier[0][j] != null) {
                    count += 1;
                }
```

```
            }
        }
        if (queue == 2) {
            for (int j = 0; j < 3; j++) {
                if (PositionOfCashier[1][j] != null) {
                    count += 1;
                }
            }
        }
        if (queue == 3) {
            for (int j = 0; j < 5; j++) {
                if (PositionOfCashier[2][j] != null) {
                    count += 1;
                }
            }
        }
        if (queue == 4) {
            for(int i = 0; i < 5; i++) {
                if (WaitingList_Queue[i] != null) {
                    count += 1;
                }
            }
        }
        return count;
    }

    private int findingTheSmallestCashier() { //Making the method for finding
the smallest cashier.
        int SmallQueue = 0;
        int length_of_cashier_01 = LengthOfQueue(1);
        int length_of_cashier_02 = LengthOfQueue(2);
        int length_of_cashier_03 = LengthOfQueue(3);

        if (length_of_cashier_01 != 2 || length_of_cashier_02 != 3 ||
length_of_cashier_03 != 5) {
            if (length_of_cashier_01 != 2) {
                SmallQueue = 1;
            } else if (length_of_cashier_02 != 3) {

                SmallQueue = 2;
            } else {
                SmallQueue = 3;
            }
        } else {
            System.out.println("Please wait some time queues are
full.");//Error message when the queues are full.
        }
        return SmallQueue;
    }

    //Adding a customer to the Cashier Queues when operator give the users
option input as 102 or ACQ.
    public void adding_customer_to_A_Queue() {

        System.out.print("Please enter your first name : ");//Getting the
input for the customers first name.
        String FirstName = input.next();
```

```
        System.out.print("Please enter your second name : ");//Getting the
input for the customers second name.
        String SecondName = input.next();

        System.out.print("Please enter your ID : ");//Getting the input for
the customers ID number.
        int Identity = input.nextInt();


        boolean Input = false;//Initializing a boolean variable.
        while (!Input) {
            System.out.print("Enter the Pizza amount that you want : ");//Get
the input for customer required pizzas.
            if (input.hasNextInt()) {
                Customer_RequiredPizzaAmount = input.nextInt();
                Input = true;
            } else {
                System.out.println("Invalid integer input.Please enter a
valid number of integer.");
                input.next(); // Invalid input is clearing from the scanner.
            }
            Pizza_Stock -= Customer_RequiredPizzaAmount;

            if(Pizza_Stock <= 20){// Printing warning message when pizza
stock reached up t0 20 pizzas.
                System.out.println("Warning: The stock is getting low. The
MAXIMUM PIZZA STOCK reaches up to 20 PIZZAS.");
            }
        }

        //Creating the object for the constructor in the Customer class.
        Customer Customer_details = new Customer(FirstName, SecondName,
Customer_RequiredPizzaAmount,Identity);

        if (LengthOfQueue(1) == 2 && LengthOfQueue(2) == 3 &&
LengthOfQueue(3) == 5) {
            if (LengthOfQueue(4) != 5) {
                for (int i = 0; i < WaitingList_Queue.length; i++) {
                    if (WaitingList_Queue[i] == null) {
                        WaitingList_Queue[i] = Customer_details;
                        System.out.println("The new customer has been added
to the waiting list cashier.");
                        break;
                    }
                }
            }
            else {
                System.out.println("Sorry!!! Waiting List cashier is also
full.Please wait some time.");
            }
        }

        int small_queue = findingTheSmallestCashier();

        if (small_queue == 1) {
            count_of_Cashier_Position_01 += Customer_RequiredPizzaAmount;
```

```
            for (int j = 0; j < PositionOfCashier[0].length; j++) {
                if (PositionOfCashier[0][j] == null) {
                    PositionOfCashier[0][j] = Customer_details;
                    break;
                }
            }
        } else if (small_queue == 2) {
            count_of_Cashier_Position_02 += Customer_RequiredPizzaAmount;
            for (int j = 0; j < PositionOfCashier[1].length; j++) {
                if (PositionOfCashier[1][j] == null) {
                    PositionOfCashier[1][j] = Customer_details;
                    break;
                }
            }
        } else if (small_queue == 3)  {
            count_of_Cashier_Position_03 += Customer_RequiredPizzaAmount;
            for (int j = 0; j < PositionOfCashier[2].length; j++) {
                if (PositionOfCashier[2][j] == null) {
                    PositionOfCashier[2][j] = Customer_details;
                    break;
                }
            }
        }
    }

    //A Customer is removing in a specific location from a queue when
operator give the users option input as 103 or RCQ.
    public static void removing_A_Customer_From_Queue() {
        try { //Using try catch block for the error handling.
            int Cashier_Number;
            int Customer_IndexPosition;

            // Looping till the correct input is inputting for the cashier
number.
            do {
                System.out.print("Enter the cashier number that you want to
remove (1, 2, or 3) :");
                Cashier_Number = input.nextInt();

                if (Cashier_Number > 3 || Cashier_Number < 1) {
                    System.out.println(" ");
                    System.out.println("*** Invalid input for cashier
number.");
                    System.out.println(" ");
                }
            } while (Cashier_Number > 3 || Cashier_Number < 1);

            int selectedQueueIndex = Cashier_Number - 1;

            // Looping till the correct input is inputting for the position
number.
            do {
                System.out.print("Enter the index that you want to remove the
customer :");
                Customer_IndexPosition = input.nextInt();

                if (Customer_IndexPosition < 1 || Customer_IndexPosition >
```

```
PositionOfCashier[selectedQueueIndex].length) {
                    System.out.println(" ");
                    System.out.println("There are no customers to remove.");
                    System.out.println(" ");
                }
            } while (Customer_IndexPosition < 1 || Customer_IndexPosition >
PositionOfCashier[selectedQueueIndex].length);

            if (PositionOfCashier[selectedQueueIndex][Customer_IndexPosition
- 1] != null) {
                for (int i = Customer_IndexPosition - 1; i <
PositionOfCashier[selectedQueueIndex].length - 1; i++) {
                    PositionOfCashier[selectedQueueIndex][i] =
PositionOfCashier[selectedQueueIndex][i + 1];
                }

PositionOfCashier[selectedQueueIndex][PositionOfCashier[selectedQueueIndex].l
ength - 1] = null;
                System.out.println("*** Customer removed.");
                if (WaitingList_Queue[0] != null) {
                    PositionOfCashier[selectedQueueIndex]
[PositionOfCashier[selectedQueueIndex].length-1] = WaitingList_Queue[0];
                    for (int j = 0; j < WaitingList_Queue.length-1; j++) {
                        WaitingList_Queue[j] = WaitingList_Queue[j + 1];
                    }
                    WaitingList_Queue[WaitingList_Queue.length-1] = null;
                }
                Pizza_Stock += Customer_RequiredPizzaAmount;
            } else {
                System.out.println(" ");
                System.out.println("There are no customers to remove.");
                System.out.println(" ");
                System.out.println("\n");
            }

        } catch (InputMismatchException e) {
            System.out.println(" ");
            System.out.println("Invalid input for cashier number.");
            System.out.println(" ");
        }
    }

    //A served customer is removing when operator give the users option input
as 104 or PCQ.
    public static void removing_A_Served_Customer() {
        try { //Using try catch block for the error handling.
            int Cashier_Number;

            // Looping till correct input is inputting for the queue number
            do {
                System.out.print("Enter the cashier number that you want to
remove (1, 2, or 3) :");
                Cashier_Number = input.nextInt();

                if (Cashier_Number > 3 || Cashier_Number < 1) {
                    System.out.println(" ");
                    System.out.println("Invalid input for cashier number.");
```

```
                    System.out.println(" ");
                }
            } while (Cashier_Number > 3 || Cashier_Number < 1);

            int queueIndex = Cashier_Number - 1;

            // Checking the selected queue is there are customers.
            if (PositionOfCashier[queueIndex].length == 0) {
                System.out.println("There are no customers to remove.");
            } else {
                int customerCount = LengthOfQueue(Cashier_Number);

                // Shifting customers in the cashier.
                if (customerCount > 0) {
                    for (int i = 0; i < PositionOfCashier[queueIndex].length
- 1; i++) {
                        PositionOfCashier[queueIndex][i] =
PositionOfCashier[queueIndex][i + 1];
                    }

PositionOfCashier[queueIndex][PositionOfCashier[queueIndex].length - 1] =
null;
                } else {
                    System.out.println("There are no customers to remove.");
                }
            }

            if (WaitingList_Queue[0] != null) {
                PositionOfCashier[queueIndex]
[PositionOfCashier[queueIndex].length-1] = WaitingList_Queue[0];
                for (int j = 0; j < WaitingList_Queue.length-1; j++) {
                    WaitingList_Queue[j] = WaitingList_Queue[j + 1];
                }
                WaitingList_Queue[WaitingList_Queue.length-1] = null;
            }
            else {
                System.out.println(" Customer is removed.");
            }
        } catch (InputMismatchException e) {
            System.out.println(" ");
            System.out.println("Invalid input");
            System.out.println(" ");
        }
    }

    //Showing the customers in alphabetical order that is entered to cashiers
when operator give the users option input as 105 or VCS.
    public static void view_customers_Sorted() {
        String[][] Cashier_lineOfCustomer = new String[3][]; // Using a 2d
array for storing customer's names.

        // Set sizes for 2d array elements to store customer's names.
        Cashier_lineOfCustomer[0] = new String[2];
        Cashier_lineOfCustomer[1] = new String[3];
        Cashier_lineOfCustomer[2] = new String[5];

        // Getting data from Cashier and stored it in Cashier_lineOfCustomer
```

```
array.
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < PositionOfCashier[i].length; j++) {
                if (PositionOfCashier[i][j] != null) {
                    Cashier_lineOfCustomer[i][j] =
PositionOfCashier[i][j].getFirstName() + " " +
PositionOfCashier[i][j].getSecondName();
                }
            }
        }

        try {
            // Flatting 1d array from the 2d array.
            String[] Flatten_array = Arrays.stream(Cashier_lineOfCustomer)
                    .flatMap(Arrays::stream)
                    .filter(Objects::nonNull)
                    .toArray(String[]::new);

            // Bubble sort method is using to sort customers in alphabetical
order.
            int n = Flatten_array.length;
            boolean Swapped;

            for (int i = 0; i < n - 1; i++) {
                Swapped = false;
                for (int j = 0; j < n - i - 1; j++) {
                    if (Flatten_array[j].compareTo(Flatten_array[j + 1]) > 0)
{
                        // Swapped Flatten_array[j] and Flatten_array[j+1]
                        String temp = Flatten_array[j];
                        Flatten_array[j] = Flatten_array[j + 1];
                        Flatten_array[j + 1] = temp;
                        Swapped = true;
                    }
                }

                // If there are no two elements were swapped in the inner
loop, the array is already sorted in alphabetical order.
                if (!Swapped) {
                    break;
                }
            }

            for (String element : Flatten_array) {
                System.out.println(element);
            }
        } catch (NullPointerException e) {
            System.out.println("There are no customers to show.");
        }
    }


    //Creating a file named MyData.txt and write the Data in the MyData.txt
file and save it when operator give the users option input as 106 or SPD.
    public static void program_Data_Storing() {
        String[][] Cashier_lineOfCustomer = new String[3][]; // 2d array for
storing customer names.
```

```
        // Set sizes for 2d array elements to store customer's names.
        Cashier_lineOfCustomer[0] = new String[2];
        Cashier_lineOfCustomer[1] = new String[3];
        Cashier_lineOfCustomer[2] = new String[5];

        // Get data from Cashier and store it in Cashier_lineOfCustomer
array.
        for (int j = 0; j < 3; j++) {
            for (int i = 0; i < PositionOfCashier[j].length; i++) {
                if (PositionOfCashier[j][i] != null) {
                    Cashier_lineOfCustomer[j][i] =
PositionOfCashier[j][i].getFirstName() + " "
                            + PositionOfCashier[j][i].getSecondName() + "
Customer Is Requiring "
                            + PositionOfCashier[j][i].getPizza_AMOUNT() + "
Pizzas."
                            + " | ID : " + PositionOfCashier[j][i].getID() +
" |";
                }
            }
        }

        try { //Using try catch block to handle errors.
            File detail_file = new File("MyData.txt");
            FileWriter DetailsFile = new FileWriter(detail_file);

            DetailsFile.write("SNACK KING QUEUE MANAGEMENT SYSTEM\n");

            for (int i = 0; i < Cashier_lineOfCustomer.length; i++) {
                for (int j = 0; j < Cashier_lineOfCustomer[i].length; j++) {
                    if (Cashier_lineOfCustomer[i][j] != null) {
                        DetailsFile.write(Cashier_lineOfCustomer[i][j]);
                        DetailsFile.write(" Customer Position : Cashier " +
(i + 1)+ "," + " Pose " + (j + 1));
                        DetailsFile.write("\n");
                    }
                }
            }

            DetailsFile.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Your customers detail file is wrote
successfully.");//When the file written successfully this message will be
printed.
    }

    //Read the data in MyData.txt file and print it as the output when
operator give the users option input as 107 or LPD.
    public static void program_Data_Loading() {
        try {
            File detail_file = new File("MyData.txt");
            Scanner Details_reading = new Scanner(detail_file);
```

```
            while (Details_reading.hasNextLine()) {
                String data = Details_reading.nextLine();
                System.out.println(data);
            }
            Details_reading.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error is detected.");
            e.printStackTrace();
        }
    }

    //Showing the remaining pizza stock when operator give the users option
input as 108 or STK.
    public static void view_remaining_Pizza_Stock() {
        System.out.println("The remaining pizza stock is : " + Pizza_Stock);
//Remaining pizza stock is printed.
    }



    //Adding pizzas to the pizza stock when operator give the users option
input as 109 or AFS.
    public static void add_Pizza_Stock() {
        //Get the pizza count as an input that operator wants to update the
pizza stock.
        System.out.print("Enter the number of pizza that you want to add to
the pizza stock : ");
        int Adding_Pizzas = input.nextInt();
        if (Adding_Pizzas <= 0) {
            System.out.println("Invalid quantity.Please enter a valid
quantity.");
        }
        else {
            Pizza_Stock += Adding_Pizzas; //Update the pizza stock with the
pizza count that have been added by operator.
            int Remaining_PizzaStock1 = Pizza_Stock;
            System.out.println("The pizza stock after adding burgers : " +
Remaining_PizzaStock1);
        }
    }

    //Printing the income of each queue when operator give the users option
input as 110 or IFQ.
    public static void print_Income_Of_Queue() { //Calculate the income of
each queue.
        int[] Incomes = new int[3];
        Incomes[0] = count_of_Cashier_Position_01 * 1350;//A one pizza price
is LKR 1350.
        Incomes[1] = count_of_Cashier_Position_02 * 1350;
        Incomes[2] = count_of_Cashier_Position_03 * 1350;

        System.out.println("INCOME FROM PIZZA");


        for (int i = 0; i < Incomes.length; i++) {
            System.out.println(" Cashier Position 0" + (i + 1) + " : LKR " +
Incomes[i]);
```

```
        }
    }

    //Exit the program when operator give the users option input as 999 or
EXT.
    public static void Exit_Program() {
        System.out.println("You Have Exit The Program and it is stopped!!!");
    }
}
```

Customer Class

```
public class Customer{
    private String firstName; //Initialize a string variable for customers
first name.
    private String secondName; //Initialize a string variable for customers
second name.
    private int PIZZA_AMOUNT; //Initialize an integer variable for customers
required pizza amount.

    private int ID;//Initialize an integer variable for customers ID.

    // Creating the constructor.
    public Customer(String firstname, String second_name, int pizza_amount,
int identity) { // Using "this" keyword to assume variables to arguments.
        this.firstName = firstname;
        this.secondName = second_name;
        this.PIZZA_AMOUNT = pizza_amount;
        this.ID = identity;
    }

    public String getFirstName() {
        return firstName;
    } //Creating the method to get the customer's first name.

    public String getSecondName() {  //Creating the method to get the
customer's second name.
        return secondName;
    } //Creating the method to get the customer's second name.

    public int getPizza_AMOUNT() { return PIZZA_AMOUNT;} //Creating the
method to get the customer's required pizza amount.

    public int getID() {return ID;} //Creating the method to get the
customer's ID.
}
```

<<END>>