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	: 20222152/ w1985731
Student First Name	: Vinura
Student Surname	: Imesh

"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 : Vinura Imesh
Student ID : 20222152

Task 1 - Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 100 or VFQ	********** * Cashiers * ********** X X X X X X X X X X	********** * Cashiers * ********** X X X X X X X X X X	Pass
2	Show empty Queues 101 or VEQ when only queue 1 is empty	********** * Cashiers * ********** X X	********** * Cashiers * ********** X X	pass
3	Show empty Queues 101 or VEQ when queue 1 and 3 are empty	********** * Cashiers * ********** X X X X X X X	********** * Cashiers * ********** X X X X X X X	pass
4	Show empty Queues : 101 or VEQ when queue 1 and 3 are empty (not fully empty some customers are there)	********** * Cashiers * ********** O O X O X X X	********** * Cashiers * ********** O O X O X X X	pass
5	Add customer “Jane” to Queue 2 102 or ACQ Enter Queue: 2 Enter Name: Jane	********** * Cashiers * ********** X O X X X X X X X X	********** * Cashiers * ********** X O X X X X X X X X	pass
6	Validating the burger count	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	pass

7	Remove a customer from a queue (Specific location) : 103 or RCQ	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	pass
8	If there are no customer in that location	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	pass
9	Validating the Queue number	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	pass
10	Validating the Which One	Which one (1,2,....):0 Invalid Input	Which one (1,2,....):0 Invalid Input	pass
11	Removed a served Customer: 104 or PCQ(When type the queue number it remove the first customer and other customers comes up)	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X X	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X X	pass

12	View Customers Sorted in alphabetical order :105 or VCS	Show the customers in alphabetical order	Show the customers in alphabetical order	pass
13	Store Program Data into file : 106 or SPD	Store the queue details into the file	Store the queue details into the file	pass
14	Load Program Data from file : 107 or LPD	Lord data from that file and show in the console	Lord data from that file and show in the console	pass
15	View Remaining burgers Stock : 108 or STK	Enter what you want to do: 108 40 burgers are remaining	Enter what you want to do: 108 40 burgers are remaining	pass
16	Add burgers to Stock : 109 or AFS	Enter what you want to do: 109 Burgers Added	Enter what you want to do: 109 Burgers Added	pass
17	If burgers not reached minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	pass

18	When all the queues are full	Queues are Full please wait	Queues are Full please wait	pass
19	Exit the Program : 999 or EXT:	Exit the program	Exit the program	pass

Task 1 - Discussion

Test Case 1: Food Queue Initialized Correctly, this test case ensures that the food queue is initialized correctly after the program starts. It covers the initial setup and verifies if the cashiers and queue structure are displayed correctly.

Test Case 2: Show empty Queues, this test case ensures that the program correctly displays the empty queue and doesn't show the queues which are full filled with customers.
Test Case 3: Show empty Queues when queues 1 and 3 are empty, this test case extends the previous case by checking the scenario when multiple queues are empty. It verifies if the program correctly handles and displays multiple empty queues.
Test Case 4: Show empty Queues when queues 1 and 3 are not fully empty, this test case examines a situation where the queues are not fully empty, but some customers remain. It checks if the program accurately represents the queues with remaining customers.
Test Case 5: Add customer "Jane" to Queue 2, This test case validates the ability to add a customer named "Jane" to a specific queue. It ensures that the program correctly adds the customer to the desired queue.
Test Case 6: Validating the burger count, this test case focuses on validating user input for the number of burgers required. It checks if the program handles invalid input, such as non-integer values, and prompts the user to enter a valid integer.
Test Case 7: Remove a customer from a queue (Specific location), This test case tests the removal of a customer from a specific queue and position. It verifies if the program correctly removes the customer and adjusts the queue accordingly.
Test Case 8: If there are no customers in that location, this test case covers the scenario where there are no customers to remove from the specified location. It ensures that the program handles this case appropriately and provides the correct feedback.
Test Case 9: Validating the Queue number, this test case checks if the program validates the user input for the queue number correctly. It verifies that the program handles invalid queue numbers and provides appropriate feedback.
Test Case 10: Validating the Which One, this test case examines the validation of user input for the "Which one" parameter. It ensures that the program handles invalid inputs and prompts the user for a valid option.
Test Case 11: Removed a served Customer, this test case tests the removal of a served customer from a queue. It ensures that the program removes the correct customer from the specified queue and adjusts the remaining customers accordingly.
Test Case 12: View Customers Sorted in alphabetical order, this test case validates the functionality of displaying customers in alphabetical order. It verifies if the program correctly sorts and displays the customers in alphabetical order.
Test Case 13: Store Program Data into file, this test case tests the storage of program data into a file. It ensures that the program correctly saves the queue details into the specified file.
Test Case 14: Load Program Data from file, this test case examines the loading of program data from a file. It ensures that the program successfully loads the data from the file and displays it in the console.
Test Case 15: View Remaining burgers Stock, this test case checks the functionality to view the remaining stock of burgers. It verifies if the program accurately displays the number of remaining burgers.
Test Case 16: Add burgers to Stock, this test case validates the ability to add burgers to the stock. It ensures that the program correctly adds the burgers and updates the stock count accordingly.
Test Case 17: If burgers not reached minimum level, this test case covers the scenario when the number of burgers is below the minimum level. It ensures that the program provides the appropriate feedback when the stock is insufficient.
Test Case 18: This test case check weather queues are full and print a massege to the user.
Test Case 19: Exit the Program, this test case verifies the functionality of exiting the program. It ensures that the program correctly terminates when the exit command is given.
Overall, the provided test cases cover a range of scenarios and inputs, including queue initialization, customer addition and removal, input validation, data storage and retrieval, stock management, and program termination. This helps ensure that various aspects of your program are thoroughly tested.

Code :

```
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.Scanner;
public class QueueManagementSysyem {

    static boolean RunProgramme = true;
    static int BurgerStock = 50;

    static String[] cashier1 = {"x","x"}; //queue1
    static String[] cashier2 = {"x","x","x"}; //queue2
    static String[] cashier3 = {"x","x","x","x","x"}; //queue3


    static String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names

    //customer positions for write in file
    static int Position1 = 0;
    static int Position2 = 0;
    static int Position3 = 0;


    //method to view all queues
    static void ViewAllQueues() {
        //print queues using print formating
        System.out.println("\n");
        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");
        System.out.printf("%2s %5s %6s
%n",cashier1[0],cashier2[0],cashier3[0]);
        System.out.printf("%2s %5s %6s
%n",cashier1[1],cashier2[1],cashier3[1]);
        System.out.printf("%8s %6s %n",cashier2[2],cashier3[2]);
        System.out.printf("%15s %n",cashier3[3]);
        System.out.printf("%15s %n",cashier3[4]);
        System.out.println("o-Occupied x-Notoccupied");
        System.out.println("\n");
    }

    //method to view empty queues
    static void ViewEmptyQueues() {
        boolean Queue1 = false;
        boolean Queue2 = false;
        boolean Queue3 = false;

        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (String Element : cashier1) {
```

```
            if (Element.equals("x")) {
                Queue1 = true;
                break;
            }
        }

        for (String Element : cashier2) {
            if (Element.equals("x")) {
                Queue2 = true;
                break;
            }
        }

        for (String Element : cashier3) {
            if (Element.equals("x")) {
                Queue3 = true;
                break;
            }
        }

        if(Queue1 && Queue2 && Queue3) {
            System.out.printf("%2s %5s %6s
%n",cashier1[0],cashier2[0],cashier3[0]);
            System.out.printf("%2s %5s %6s
%n",cashier1[1],cashier2[1],cashier3[1]);
            System.out.printf("%8s %6s %n",cashier2[2],cashier3[2]);
            System.out.printf("%15s %n",cashier3[3]);
        } else if (Queue1 && Queue2) {
            System.out.printf("%2s %5s %n",cashier1[0],cashier2[0]);
            System.out.printf("%2s %5s %n",cashier1[1],cashier2[1]);
            System.out.printf("%8s %n",cashier2[2]);
        } else if (Queue1 && Queue3){
            System.out.printf("%2s %12s %n",cashier1[0],cashier3[0]);
            System.out.printf("%2s %12s %n",cashier1[1],cashier3[1]);
            System.out.printf("%15s %n",cashier3[2]);
            System.out.printf("%15s %n",cashier3[3]);
            System.out.printf("%15s %n",cashier3[4]);
        } else if (Queue2 && Queue3) {
            System.out.printf("%8s %6s %n",cashier2[0],cashier3[0]);
            System.out.printf("%8s %6s %n",cashier2[1],cashier3[1]);
            System.out.printf("%8s %6s %n",cashier2[2],cashier3[2]);
            System.out.printf("%15s %n",cashier3[3]);
            System.out.printf("%15s %n",cashier3[4]);
        } else if (Queue1) {
            System.out.printf("%2s %n",cashier1[0]);
            System.out.printf("%2s %n",cashier1[1]);
        } else if (Queue2) {
            System.out.printf("%8s %n",cashier2[0]);
            System.out.printf("%8s %n",cashier2[1]);
            System.out.printf("%8s %n",cashier2[2]);
        }else {
            System.out.printf("%15s %n",cashier3[0]);
            System.out.printf("%15s %n",cashier3[1]);
            System.out.printf("%15s %n",cashier3[2]);
            System.out.printf("%15s %n",cashier3[3]);
            System.out.printf("%15s %n",cashier3[4]);
        }
```

```
        System.out.println("o-Occupied x-Notoccupied");
    }


    // use in AddCustomer method,RemoveCustomer and RemoveServedCustomer to
check how many customers in the queue
    /*
    * param(Cashier) = use to check which cashier in AddCostomer method*/
    static int CountCustomer(String[] Cashier) {
        int Count = 0;

        for (String element : Cashier) {
            if (element.equals("o")) {
                Count++;

            }
        }
        return Count;
    }

    //Method for add customer names into 2D array
    /*
    * param 1(QueueNumber) = checking which queue in AddCustomer method
    * param 2(Name) = Get customer name
    */
    static void AddCustomerName(int QueueNumber ,String Name ){

        if(QueueNumber == 1){
            CustomerLine[0][Position1] = Name;
            Position1++;
        }else if(QueueNumber == 2){
                CustomerLine[1][Position2] = Name;
                Position2++;
        }else {
            CustomerLine[2][Position3] = Name;
            Position3++;
        }

    }

    //method to add customer
    static void AddCustomer(){


        Scanner input1 = new Scanner(System.in);

        System.out.print("Enter the customer's name :");
        String Name = input1.next();

        try {


            System.out.print("Which queue you want to add to(Enter 1,2 or 3)
:");
            int WhichQueue = input1.nextInt();
```

```
            int CashierFull;

            if (0 < WhichQueue && WhichQueue < 4) {

                if (WhichQueue == 1) {
                    CashierFull = CountCustomer(cashier1);

                    if (CashierFull < 2) {
                        AddCustomerName(1, Name);

                        for (int i = 0; i < cashier1.length; i++) {
                            if (cashier1[i].equals("x")) {
                                cashier1[i] = "o";
                                break;
                            }
                        }
                    } else {
                        System.out.println("\n");
                        System.out.println("Queue is Full Please wait!");
                        System.out.println("\n");
                    }

                } else if (WhichQueue == 2) {
                    CashierFull = CountCustomer(cashier2);

                    if (CashierFull < 3) {
                        AddCustomerName(2, Name);

                        for (int i = 0; i < cashier2.length; i++) {
                            if (cashier2[i].equals("x")) {
                                cashier2[i] = "o";
                                break;
                            }
                        }
                    } else {
                        System.out.println("\n");
                        System.out.println("Queue is Full Please wait!");
                        System.out.println("\n");
                    }

                } else {
                    CashierFull = CountCustomer(cashier3);

                    if (CashierFull < 5) {
                        AddCustomerName(3, Name);

                        for (int i = 0; i < cashier3.length; i++) {
                            if (cashier3[i].equals("x")) {
                                cashier3[i] = "o";
                                break;
                            }
                        }
                    } else {
                        System.out.println("\n");
                        System.out.println("Queue is Full Please wait!");
                        System.out.println("\n");
                    }
```

```
                }
            } else {
                System.out.println("Wrong Input");
            }

        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }

        ViewAllQueues();

    }

    // method to remove a customer in a specific location
    static void RemoveCustomer(){

        try {
            Scanner input2 = new Scanner(System.in);
            int WhichQueue;
            int WhichOne;
            // looping until get correct input
            do {

                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input2.nextInt();

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);


            if (WhichQueue == 1) {
                Position1 -= 1;
                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt();

                    if (WhichOne < 1 || WhichOne > cashier1.length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > cashier1.length);


                if (cashier1[WhichOne - 1].equals("o")) {
                    for (int i = WhichOne - 1; i < 1; i++) {
                        cashier1[i] = cashier1[i + 1];
                        CustomerLine[0][i] = CustomerLine[0][i + 1];

                    }
                    cashier1[1] = "x";
                    CustomerLine[0][1] = null;
```

```
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }


            } else if (WhichQueue == 2) {
                Position2 -= 1;

                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt();

                    if (WhichOne < 1 || WhichOne > cashier2.length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > cashier2.length);

                if (cashier2[WhichOne - 1].equals("o")) {

                    for (int i = WhichOne - 1; i < 2; i++) {
                        cashier2[i] = cashier2[i + 1];
                        CustomerLine[1][i] = CustomerLine[1][i + 1];

                    }
                    cashier2[2] = "x";
                    CustomerLine[1][2] = null;

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else {
                Position3 -= 1;

                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt();

                    if (WhichOne < 1 || WhichOne > cashier3.length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > cashier3.length);

                if (cashier3[WhichOne - 1].equals("o")) {

                    for (int i = WhichOne - 1; i < 4; i++) {
                        cashier3[i] = cashier3[i + 1];
                        CustomerLine[2][i] = CustomerLine[2][i + 1];

                    }
                    cashier3[4] = "x";
```

```
                    CustomerLine[2][4] = null;

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }
        ViewAllQueues();
    }


    //Method for remove served customer
    static void RemoveServedCustomer(){
        try {
            BurgerStock -= 5; // reduse burger stock by 5
            int WhichQueue;

            Scanner input3 = new Scanner(System.in);

            // looping until get correct input
            do {

                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input3.nextInt();

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);

            int CustomerCount;

            if (WhichQueue == 1) {
                Position1 -= 1;
                CustomerCount = CountCustomer(cashier1);

                if (CustomerCount != 0) {

                    for (int i = 0; i < 1; i++) {
                        cashier1[i] = cashier1[i + 1];
                        CustomerLine[0][i] = CustomerLine[0][i + 1];

                    }
                    cashier1[1] = "x";
                    CustomerLine[0][1] = null;

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else if (WhichQueue == 2) {
```

```
                Position2 -= 1;
                CustomerCount = CountCustomer(cashier2);

                if (CustomerCount != 0) {

                    for (int i = 0; i < 2; i++) {
                        cashier2[i] = cashier2[i + 1];
                        CustomerLine[1][i] = CustomerLine[1][i + 1];

                    }
                    cashier2[2] = "x";
                    CustomerLine[1][2] = null;

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else {
                Position3 -= 1;
                CustomerCount = CountCustomer(cashier3);

                if (CustomerCount != 0) {

                    for (int i = 0; i < 4; i++) {
                        cashier3[i] = cashier3[i + 1];
                        CustomerLine[2][i] = CustomerLine[2][i + 1];

                    }
                    cashier3[4] = "x";
                    CustomerLine[2][4] = null;

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }
        ViewAllQueues();
    }


    //Method to view customers in alpebatical order
    static void ViewCustomers(){


        try {

            // Flatten the 2D array into a 1D array
            String[] flattenedArray = Arrays.stream(CustomerLine)
                    .flatMap(Arrays::stream)
                    .toArray(String[]::new);

            // Bubble sort algorithm to Sort customers in alphabetical order
            int n = flattenedArray.length;
```

```
            boolean swapped;

            for (int i = 0; i < n - 1; i++) {
                swapped = false;
                for (int j = 0; j < n - i - 1; j++) {
                    if (flattenedArray[j] != null && flattenedArray[j + 1] !=
null && flattenedArray[j].compareTo(flattenedArray[j + 1]) > 0) {
                        // Swap array[j] and array[j+1]
                        String temp = flattenedArray[j];
                        flattenedArray[j] = flattenedArray[j + 1];
                        flattenedArray[j + 1] = temp;
                        swapped = true;
                    }
                }

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

            for (String item : flattenedArray) {
                if (item != null) {
                    System.out.println(item);
                }
            }

        }catch (NullPointerException e){
            System.out.println("No customers to show");
        }
    }

    //Method to add data to QueueManagementDetails.txt
    static void AddDataToFile(){
        try {
            File customerDetails = new File("QueueManagementDetails.txt");
            FileWriter Details = new FileWriter(customerDetails);

            Details.write("---------------------Queue Management Details-----
----------------");
            Details.write("\n");

            for(int i=0; i<CustomerLine.length; i++){
                for(int j = 0; j<CustomerLine[i].length; j++){
                    if (CustomerLine[i][j] != null) {
                        Details.write(CustomerLine[i][j]);
                        Details.write(" Position :" + "Queue " + (i+1) + "
Number " + (j+1));
                        Details.write("\n");
                    }

                }
            }

            Details.close();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        System.out.println("Data stored successfully");
    }

    //Method to read data from QueueManagementDetails.txt
    static void ReadFileData(){

        try {

            File customerDetails = new File("QueueManagementDetails.txt");

            Scanner ReadDetails = new Scanner(customerDetails);

            while (ReadDetails.hasNextLine()) {
                String data = ReadDetails.nextLine();
                System.out.println(data);
            }
            ReadDetails.close();

        }catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }


    }


    //Method to view burger count
    static void ViewBurgerCount(){

        System.out.println(BurgerStock + " burgers are remaining");

    }

    //Method to add burgers
    static void AddBurgers(){
        // checking burger count reach minimum level
        if (BurgerStock == 10) {
            BurgerStock += 40;
            System.out.println("Burgers Added");
        }else {
            System.out.println("The number of Burgers has not reachrd the
minimum level");
        }

    }

    //Method to stop execution
    static void Exit(){
        RunProgramme = false;
    }

    //Method for select item from the menue
    static void Menu(){
```

```
        Scanner input = new Scanner(System.in);

        System.out.print("Enter what you want to do: ");
        String task = input.next().toLowerCase();

        //Enhanced swich case to call all methods
        switch (task) {
            case "100", "vfq" -> ViewAllQueues();
            case "101", "veq" -> ViewEmptyQueues();
            case "102", "acq" -> AddCustomer();
            case "103", "rcq" -> RemoveCustomer();
            case "104", "pcq" -> RemoveServedCustomer();
            case "105", "vcs" -> ViewCustomers();
            case "106", "spd" -> AddDataToFile();
            case "107", "lpd" -> ReadFileData();
            case "108", "stk" -> ViewBurgerCount();
            case "109", "afs" -> AddBurgers();
            case "999", "ext" -> Exit();
            default -> System.out.println("Wrong Input");
        }


    }



    public static void main(String[] args) {

        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];



        System.out.println("\n");
        System.out.println("------------------------------Foodies Fave Queue
Management System.-------------------------------");
        System.out.println("\n");

        while (RunProgramme) {

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

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 a
Queue.                ||");
```

```
            System.out.println("||  104 or PCQ: Remove a served customer.
||");
            System.out.println("||  105 or VCS: View Customers Sorted in
alphabetical order.  ||");
            System.out.println("||  106 or SPD: Store Program Data into file.
||");
            System.out.println("||  107 or LPD: Load Program Data from file.
||");
            System.out.println("||  108 or STK: View Remaining burgers Stock.
||");
            System.out.println("||  109 or AFS: Add burgers to Stock.
||");
            System.out.println("||  999 or EXT: Exit the Program.
||");

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

            System.out.println("\n");


            //Print warning massege when burger count reach 10
            if(BurgerStock == 10){
                System.out.println("******-WARNING-*****");
                System.out.println("* Low burger stock *");
                System.out.println("********************");
            }


            Menu();

        }
    }
}
```

Task 2 - Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 100 or VFQ	********** * Cashiers * ********** X X X X X X X X X X	********** * Cashiers * ********** X X X X X X X X X X	Pass
2	Show empty Queues 101 or VEQ when only queue 1 is empty	********** * Cashiers * ********** X X	********** * Cashiers * ********** X X	pass
3	Show empty Queues 101 or VEQ when queue 1 and 3 are empty	********** * Cashiers * ********** X X X X X X X	********** * Cashiers * ********** X X X X X X X	pass
4	Show empty Queues : 101 or VEQ when queue 1 and 3 are empty (not fully empty some customers are there)	********** * Cashiers * ********** O O X O X X X	********** * Cashiers * ********** O O X O X X X	pass
5	Add customer “Jane” in to shortest queue 102 or ACQ Enter First Name: Jane Enter Last Name: Harris	********** * Cashiers * ********** O O X X X X X X X X	********** * Cashiers * ********** O O X X X X X X X X	pass

6	Validating the burger count	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	pass
7	Remove a customer from a queue (Specific location) : 103 or RCQ	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	pass
8	If there are no customer in that location	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	pass
9	Validating the Queue number	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	pass
10	Validating the Which One	Which one (1,2,....):0 Invalid Input	Which one (1,2,....):0 Invalid Input	pass
11	Removed a served Customer: 104 or PCQ(When type the queue number it	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	pass

	remove the first customer and other customers comes up)	X	X	
12	View Customers Sorted in alphabetical order :105 or VCS	Show the customers in alphabetical order	Show the customers in alphabetical order	pass
13	Store Program Data into file : 106 or SPD	Store the queue details into the file	Store the queue details into the file	pass
14	Load Program Data from file : 107 or LPD	Lord data from that file and show in the console	Lord data from that file and show in the console	pass
15	View Remaining burgers Stock : 108 or STK	Enter what you want to do: 108 40 burgers are remaining	Enter what you want to do: 108 40 burgers are remaining	pass
16	Add burgers to Stock : 109 or AFS	Enter what you want to do: 109 Burgers Added	Enter what you want to do: 109 Burgers Added	pass

17	If burgers not reached minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	pass
18	When all the queues are full	Queues are Full please wait	Queues are Full please wait	pass
19	View Income of Each Queue : 110 or IFQ	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	pass
20	Exit the Program : 999 or EXT:	Exit the program	Exit the program	pass

Task 2 - Discussion

Test Case 1: Food Queue Initialized Correctly, this test case ensures that the food queue is initialized correctly after the program starts. It covers the initial setup and verifies if the cashiers and queue structure are displayed correctly.

Test Case 2: Show empty Queues, this test case ensures that the program correctly displays the empty queue and doesn't show the queues which are full filled with customers.
Test Case 3: Show empty Queues when queues 1 and 3 are empty, this test case extends the previous case by checking the scenario when multiple queues are empty. It verifies if the program correctly handles and displays multiple empty queues.
Test Case 4: Show empty Queues when queues 1 and 3 are not fully empty, this test case examines a situation where the queues are not fully empty, but some customers remain. It checks if the program accurately represents the queues with remaining customers.
Test Case 5: Add customer "Jane harris" to the shortest queue, this test case validates the ability to add a customer named "Jane" to the shortest queue. It ensures that the program correctly adds the customer to the shortest queue.
Test Case 6: Validating the burger count, this test case focuses on validating user input for the number of burgers required. It checks if the program handles invalid input, such as non-integer values, and prompts the user to enter a valid integer.
Test Case 7: Remove a customer from a queue (Specific location), This test case tests the removal of a customer from a specific queue and position. It verifies if the program correctly removes the customer and adjusts the queue accordingly.
Test Case 8: If there are no customers in that location, this test case covers the scenario where there are no customers to remove from the specified location. It ensures that the program handles this case appropriately and provides the correct feedback.
Test Case 9: Validating the Queue number, this test case checks if the program validates the user input for the queue number correctly. It verifies that the program handles invalid queue numbers and provides appropriate feedback.
Test Case 10: Validating the Which One, this test case examines the validation of user input for the "Which one" parameter. It ensures that the program handles invalid inputs and prompts the user for a valid option.
Test Case 11: Removed a served Customer, this test case tests the removal of a served customer from a queue. It ensures that the program removes the correct customer from the specified queue and adjusts the remaining customers accordingly.
Test Case 12: View Customers Sorted in alphabetical order, this test case validates the functionality of displaying customers in alphabetical order. It verifies if the program correctly sorts and displays the customers in alphabetical order.
Test Case 13: Store Program Data into file, this test case tests the storage of program data into a file. It ensures that the program correctly saves the queue details into the specified file.
Test Case 14: Load Program Data from file, this test case examines the loading of program data from a file. It ensures that the program successfully loads the data from the file and displays it in the console.
Test Case 15: View Remaining burgers Stock, this test case checks the functionality to view the remaining stock of burgers. It verifies if the program accurately displays the number of remaining burgers.
Test Case 16: Add burgers to Stock, this test case validates the ability to add burgers to the stock. It ensures that the program correctly adds the burgers and updates the stock count accordingly.
Test Case 17: If burgers not reached minimum level, this test case covers the scenario when the number of burgers is below the minimum level. It ensures that the program provides the appropriate feedback when the stock is insufficient.
Test Case 18: This test case verifies the functionality of viewing the income of each queue. It ensures that the program correctly displays the income of each queue and calculates the total income.
Test Case 19: This test case check weather queues are full and print a massege to the user.
Test Case 20: Exit the Program, this test case verifies the functionality of exiting the program. It ensures that the program correctly terminates when the exit command is given.
Overall, the provided test cases cover a range of scenarios and inputs, including queue initialization, customer addition and removal, input validation, data storage and retrieval, stock management, and program termination. This helps ensure that various aspects of your program are thoroughly tested.

```
import java.util.Scanner;
public class QueueManagementSystem {

    public static void main(String[] args) {

        System.out.println("\n");
        System.out.println("-------------------------------Foodies Fave Queue
Management System.--------------------------------");
        System.out.println("\n");

        FoodQueue myObj = new FoodQueue(); //creating object to call methods
in FoodQueue class
        myObj.Queues(); //call the method to define the queue length

        while (true) {
            System.out.println("\n");


System.out.println("||=========================================================
=====||");
            System.out.println("||                                MENU
```

```
||");

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 a
Queue.                ||");
            System.out.println("||  104 or PCQ: Remove a served customer.
||");
            System.out.println("||  105 or VCS: View Customers Sorted in
alphabetical order.  ||");
            System.out.println("||  106 or SPD: Store Program Data into file.
||");
            System.out.println("||  107 or LPD: Load Program Data from file.
||");
            System.out.println("||  108 or STK: View Remaining burgers Stock.
||");
            System.out.println("||  109 or AFS: Add burgers to Stock.
||");
            System.out.println("||  110 or IFQ: View Income of Each Queue.
||");
            System.out.println("||  999 or EXT: Exit the Program.
||");

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

            System.out.println("\n");

            myObj.PrintWarningMessage(); //print the warning message burger
is low

            Scanner input = new Scanner(System.in);

            System.out.print("Enter what you want to do: ");
            String task = input.next().toLowerCase();

            //Enhanced switch case to call all methods
            switch (task) {
                case "100", "vfq" -> myObj.ViewAllQueues();
                case "101", "veq" -> myObj.ViewEmptyQueues();
                case "102", "acq" -> myObj.AddCustomer();
                case "103", "rcq" -> myObj. RemoveCustomer();
                case "104", "pcq" -> myObj.RemoveServedCustomer();
                case "105", "vcs" -> myObj.ViewCustomers();
                case "106", "spd" -> myObj.AddDataToFile();
                case "107", "lpd" -> myObj.ReadFileData();
                case "108", "stk" -> myObj.ViewBurgerCount();
                case "109", "afs" -> myObj.AddBurgers();
                case "110", "ifq" -> myObj.CalculateTheIncome();
                case "999", "ext" -> System.exit(0);
                default -> System.out.println("Wrong Input");
```

```
            }


        }



    }
}
```

```
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.Scanner;
public class FoodQueue {

    static Customer[][] Cashier = new Customer[3][]; //2D array for all the
cashiers
    static int BurgerStock = 50; //define the burger stock
    static int SoldBurgers = 0; //counting the sold burgers

    static int Queue1BurgerCount = 0; //counting burger count for cashier 1
    static int Queue2BurgerCount = 0; //counting burger count for cashier 2
    static int Queue3BurgerCount = 0; //counting burger count for cashier 3

    //define the size for queues
    public void Queues(){
        Cashier[0] = new Customer[2];
        Cashier[1] = new Customer[3];
        Cashier[2] = new Customer[5];

    }




    //method to view all queues
    public static void ViewAllQueues() {
        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (int i = 0; i < 5; i++) { //check the customer
            for (int j = 0; j < Cashier.length; j++) { // check the cashier
                if (i < Cashier[j].length) {
                    if (Cashier[j][i] == null) {
                        System.out.print(" X    ");
                    } else {
                        System.out.print(" O    ");
                    }
                } else {
```

```
                    System.out.print("      ");
                }
            }
            System.out.println();
        }
        System.out.println("0-Occupied X-NotOccupied");
    }

    //method for view empty queues
    public void ViewEmptyQueues(){
        // boolean variables for check the cashiers contains null values
        boolean ContainsNullLine1 = false;
        boolean ContainsNullLine2 = false;
        boolean ContainsNullLine3 = false;

        System.out.println("***************");
        System.out.println("*   Cashiers   *");
        System.out.println("***************");

        for (Customer element : Cashier[0]) {
            if (element == null) {
                ContainsNullLine1 = true;
                break;
            }
        }

        for (Customer element : Cashier[1]) {
            if (element == null) {
                ContainsNullLine2 = true;
                break;
            }
        }

        for (Customer element : Cashier[2]) {
            if (element == null) {
                ContainsNullLine3 = true;
                break;
            }
        }

        if(ContainsNullLine1 && ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%2s %5s %6s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x", Cashier[2][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %6s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%8s %6s %n", Cashier[1][2] != null ? "o" :
"x", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine2) {
            System.out.printf("%2s %5s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x");
            System.out.printf("%8s %n", Cashier[1][2] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine3){
            System.out.printf("%2s %12s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[2][0] != null ? "o" : "x");
```

```
            System.out.printf("%2s %12s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][4] != null ? "o" : "x");
        } else if (ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%8s %6s %n",Cashier[1][0] != null ? "o" :
"x",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][1]!= null ? "o" :
"x",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][2]!= null ? "o" :
"x",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        } else if (ContainsNullLine1) {
            System.out.printf("%2s %n",Cashier[0][0]!= null ? "o" : "x");
            System.out.printf("%2s %n",Cashier[0][1]!= null ? "o" : "x");
        } else if (ContainsNullLine2) {
            System.out.printf("%8s %n",Cashier[1][0]!= null ? "o" : "x");
            System.out.printf("%8s %n",Cashier[1][1]!= null ? "o" : "x");
            System.out.printf("%8s %n",Cashier[1][2]!= null ? "o" : "x");
        }else {
            System.out.printf("%15s %n",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        }
        System.out.println("0-Occupied X-NotOccupied");
    }

    //method for check the queue length
    /*
     * param1 Queue - use to get the cashier number for checking
     */
    public static int CheckQueueLength(int Queue){
        int Counter = 0; //variable for get the customer count

        if(Queue == 1){
            for (int i = 0; i < 2; i++){
                if(Cashier[0][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 2){
            for (int i = 0; i < 3; i++){
                if(Cashier[1][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 3){
            for (int i = 0; i < 5; i++){
                if(Cashier[2][i] != null){
```

```
                    Counter += 1;
                }
            }
        }

        return Counter;
    }

    //method to find the smallest queue
    public static int FindTheSmallestQueue() {
        int SmallestQueue =0;
        int Queue1Length = CheckQueueLength(1); //checking the queue length
using CheckQueueLength method according to the queue
        int Queue2Length = CheckQueueLength(2);
        int Queue3Length = CheckQueueLength(3);

        if(Queue1Length !=2 || Queue2Length != 3 || Queue3Length != 5) { //
check the queues are full or not
            if (Queue1Length <= Queue2Length && Queue1Length <= Queue3Length
&& Queue1Length != 2) {
                SmallestQueue = 1;
            } else if (Queue2Length <= Queue1Length && Queue2Length <=
Queue3Length && Queue2Length != 3) {
                SmallestQueue = 2;
            } else {
                SmallestQueue = 3;
            }
        }else {
            System.out.println("Queues are Full please wait");
        }

        return SmallestQueue;
    }

    //method for add customers
    public static void AddCustomer(){

        Scanner input = new Scanner(System.in);

        System.out.print("Enter Customer's First name : ");
        String FirstName = input.next(); // get the customers first name by
user

        System.out.print("Enter Customer's Last name : ");
        String LastName = input.next(); // get the customers last name by
user

        // validating the burger count as an integer
        int BurgersRequired = 0;
        boolean ValidInput = false;
        while (!ValidInput) {
            System.out.print("Enter how many Burgers required: ");
            if (input.hasNextInt()) {
                BurgersRequired = input.nextInt(); // get the customers
required burger count by user
                ValidInput = true;
            } else {
```

```
                System.out.println("Wrong input. Please enter a valid
integer.");
                input.next(); // Clear the invalid input from the scanner
            }
        }

        Customer Person = new Customer(FirstName, LastName, BurgersRequired);
//creating object to save customer names

        int SmallestQueue = FindTheSmallestQueue(); //run
FindTheSmallestQueue method and find the smallest queue

        if (SmallestQueue == 1) {
            Queue1BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[0].length; i++) {
                if (Cashier[0][i] == null) {
                    Cashier[0][i] = Person; //adding the customer details
into queue 1
                    break;
                }
            }
        } else if (SmallestQueue == 2) {
            Queue2BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[1].length; i++) {
                if (Cashier[1][i] == null) {
                    Cashier[1][i] = Person; //adding the customer details
into queue 2
                    break;
                }
            }
        } else {
            Queue3BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[2].length; i++) {
                if (Cashier[2][i] == null) {
                    Cashier[2][i] = Person; //adding the customer details
into queue 3
                    break;
                }
            }
        }


        ViewAllQueues(); //run ViewAllQueues method to show customer is added

    }

    // method to remove a customer in a specific location
    public static void RemoveCustomer(){
        try {

            Scanner input2 = new Scanner(System.in);
            int WhichQueue;
            int WhichOne;
            // looping until get correct input
            do {

                System.out.print("Enter the Queue to remove the customer (1,2
```

```
or 3):");
                WhichQueue = input2.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);


            if (WhichQueue == 1) {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[0].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[0].length);

                if (Cashier[0][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[0][1] = null; //set last index to null
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }


            } else if (WhichQueue == 2) {
                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[1].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[1].length);

                if (Cashier[1][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
```

```
elements of cashier

                    }
                    Cashier[1][2] = null; //set last index to null
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }


            } else {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[2].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[2].length);

                if (Cashier[2][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[2][4] = null; //set last index to null
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }

        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
    }


    //Method for remove served customer
    static void RemoveServedCustomer(){
        try {
            int WhichQueue;
            int RequiredBurgerCount;

            Scanner input3 = new Scanner(System.in);

            do {
```

```
                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input3.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);

            int CustomerCount;

            if (WhichQueue == 1) {
                RequiredBurgerCount = Cashier[0][0].getBurgersRequired();
                SoldBurgers += RequiredBurgerCount;
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(1);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[0][1] = null; //set last index to null
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else if (WhichQueue == 2) {
                RequiredBurgerCount = Cashier[1][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(2);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[1][2] = null; //set last index to null
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            } else {
                RequiredBurgerCount = Cashier[2][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(3);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[2][4] = null; //set last index to null
```

```
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }
        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
    }


    //Method to view customers in alphabetical order
    static void ViewCustomers(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName();
                }
            }
        }


        try {

            // Flatten the 2D array into a 1D array
            String[] flattenedArray = Arrays.stream(CustomerLine)
                    .flatMap(Arrays::stream)
                    .toArray(String[]::new);

            // Bubble sort algorithm to Sort customers in alphabetical order
            int n = flattenedArray.length;
            boolean swapped;

            for (int i = 0; i < n - 1; i++) {
                swapped = false;
                for (int j = 0; j < n - i - 1; j++) {
                    if (flattenedArray[j] != null && flattenedArray[j + 1] !=
null && flattenedArray[j].compareTo(flattenedArray[j + 1]) > 0) {
                        // Swap array[j] and array[j+1]
                        String temp = flattenedArray[j];
                        flattenedArray[j] = flattenedArray[j + 1];
                        flattenedArray[j + 1] = temp;
                        swapped = true;
```

```
                    }
                }

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

            for (String item : flattenedArray) {
                if (item != null) {
                    System.out.println(item);
                }
            }

        }catch (NullPointerException e){
            System.out.println("No customers to show");
        }
    }

    //method for add data to a file
    static void AddDataToFile(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName() + " " + Cashier[i][j].getBurgersRequired() + "
Burgers Required ";
                }
            }
        }

        try {
            File customerDetails = new File("QueueManagementDetails.txt");
            FileWriter Details = new FileWriter(customerDetails);

            Details.write("--------------------Queue Management Details-----
----------------");
            Details.write("\n");

            for(int i=0; i<CustomerLine.length; i++){
                for(int j = 0; j<CustomerLine[i].length; j++){
                    if (CustomerLine[i][j] != null) {
                        Details.write(CustomerLine[i][j]);
                        Details.write(" Position :" + "Queue " + (i+1) + "
```

```
Number " + (j+1));
                        Details.write("\n");
                    }

                }
            }

            Details.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Data stored successfully");
    }


    //Method to read data from QueueManagementDetails.txt
    static void ReadFileData(){

        try {

            File customerDetails = new File("QueueManagementDetails.txt");

            Scanner ReadDetails = new Scanner(customerDetails);

            while (ReadDetails.hasNextLine()) {
                String data = ReadDetails.nextLine();
                System.out.println(data);
            }
            ReadDetails.close();

        }catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }


    }


    //Method to view burger count
    static void ViewBurgerCount(){

        System.out.println(BurgerStock + " burgers are remaining");

    }


    static void AddBurgers(){
        // checking burger count reach minimum level
        if (BurgerStock == 10) {
            BurgerStock += 40;
            System.out.println("Burgers Added");
        }else {
            System.out.println("The number of Burgers has not reached the
minimum level");
        }
```

```
    }

    //method for calculate the income
    static void CalculateTheIncome(){
        int Queue1Income = Queue1BurgerCount * 650;
        int Queue2Income = Queue2BurgerCount * 650;
        int Queue3Income = Queue3BurgerCount * 650;

        System.out.println("╔═══════════════════════════════════╗");
        System.out.println("│              Income               │");
        System.out.println("├───────────────────────────────────╢");
        System.out.println("│ Queue 1 : " + Queue1Income);
        System.out.println("│ Queue 2 : " + Queue2Income);
        System.out.println("│ Queue 2 : " + Queue3Income);
        System.out.println("│ Total   : " +
(Queue1Income+Queue2Income+Queue3Income));
        System.out.println("╚═══════════════════════════════════╝");
    }

    //method for print the warning message
    static void PrintWarningMessage(){
        //Print warning message when burger count reach 10
        if(BurgerStock == 10){
            System.out.println("******-WARNING-*****");
            System.out.println("* Low burger stock *");
            System.out.println("********************");
        }
    }


}
```

```
public class 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;} //return Customers first
name using this method

    public String getLastName() {return LastName;} //return Customers last
name using this method

    public int getBurgersRequired() {return BurgersRequired;} // returning
the customer required burger count
```

```
}
```

Task 3 - Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 100 or VFQ	********** * Cashiers * ********** X X X X X X X X X X	********** * Cashiers * ********** X X X X X X X X X X	Pass
2	Show empty Queues 101 or VEQ when only queue 1 is empty	********** * Cashiers * ********** X X	********** * Cashiers * ********** X X	pass
3	Show empty Queues 101 or VEQ when queue 1 and 3 are empty	********** * Cashiers * ********** X X X X X X X	********** * Cashiers * ********** X X X X X X X	pass
4	Show empty Queues : 101 or VEQ when queue 1 and 3 are empty (not fully empty some customers are there)	********** * Cashiers * ********** O O X O X X X	********** * Cashiers * ********** O O X O X X X	pass
5	Add customer “Jane” in to shortest queue 102 or ACQ Enter First Name: Jane Enter Last Name: Harris	********** * Cashiers * ********** O O X X X X X X X X	********** * Cashiers * ********** O O X X X X X X X X	pass

6	Validating the burger count	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	pass
7	Remove a customer from a queue (Specific location) : 103 or RCQ	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	pass
8	If there are no customer in that location	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	pass
9	Validating the Queue number	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	pass
10	Validating the Which One	Which one (1,2,....):0 Invalid Input	Which one (1,2,....):0 Invalid Input	pass
11	Removed a served Customer: 104 or PCQ(When type the queue number it	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	pass

	remove the first customer and other customers comes up)	X	X	
12	View Customers Sorted in alphabetical order :105 or VCS	Show the customers in alphabetical order	Show the customers in alphabetical order	pass
13	Store Program Data into file : 106 or SPD	Store the queue details into the file	Store the queue details into the file	pass
14	Load Program Data from file : 107 or LPD	Lord data from that file and show in the console	Lord data from that file and show in the console	pass
15	View Remaining burgers Stock : 108 or STK	Enter what you want to do: 108 40 burgers are remaining	Enter what you want to do: 108 40 burgers are remaining	pass
16	Add burgers to Stock : 109 or AFS	Enter what you want to do: 109 Burgers Added	Enter what you want to do: 109 Burgers Added	pass

17	If burgers not reached minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	pass
18	When all the queues are full	Queues are Full please wait	Queues are Full please wait	pass
19	View Income of Each Queue : 110 or IFQ	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	pass
20	Customers added to the Waiting queue if the queues are full	Waiting list is full please wait	Waiting list is full please wait	pass
21	Exit the Program : 999 or EXT:	Exit the program	Exit the program	pass

Task 3 - Discussion

Test Case 1: Food Queue Initialized Correctly, this test case ensures that the food queue is initialized correctly after the program starts. It covers the initial setup and verifies if the cashiers and queue structure are displayed correctly.

Test Case 2: Show empty Queues, this test case ensures that the program correctly displays the empty queue and doesn't show the queues which are full filled with customers.
Test Case 3: Show empty Queues when queues 1 and 3 are empty, this test case extends the previous case by checking the scenario when multiple queues are empty. It verifies if the program correctly handles and displays multiple empty queues.
Test Case 4: Show empty Queues when queues 1 and 3 are not fully empty, this test case examines a situation where the queues are not fully empty, but some customers remain. It checks if the program accurately represents the queues with remaining customers.
Test Case 5: Add customer "Jane harris" to the shortest queue, this test case validates the ability to add a customer named "Jane" to the shortest queue. It ensures that the program correctly adds the customer to the shortest queue.
Test Case 6: Validating the burger count, this test case focuses on validating user input for the number of burgers required. It checks if the program handles invalid input, such as non-integer values, and prompts the user to enter a valid integer.
Test Case 7: Remove a customer from a queue (Specific location), This test case tests the removal of a customer from a specific queue and position. It verifies if the program correctly removes the customer and adjusts the queue accordingly.
Test Case 8: If there are no customers in that location, this test case covers the scenario where there are no customers to remove from the specified location. It ensures that the program handles this case appropriately and provides the correct feedback.
Test Case 9: Validating the Queue number, this test case checks if the program validates the user input for the queue number correctly. It verifies that the program handles invalid queue numbers and provides appropriate feedback.
Test Case 10: Validating the Which One, this test case examines the validation of user input for the "Which one" parameter. It ensures that the program handles invalid inputs and prompts the user for a valid option.
Test Case 11: Removed a served Customer, this test case tests the removal of a served customer from a queue. It ensures that the program removes the correct customer from the specified queue and adjusts the remaining customers accordingly.
Test Case 12: View Customers Sorted in alphabetical order, this test case validates the functionality of displaying customers in alphabetical order. It verifies if the program correctly sorts and displays the customers in alphabetical order.
Test Case 13: Store Program Data into file, this test case tests the storage of program data into a file. It ensures that the program correctly saves the queue details into the specified file.
Test Case 14: Load Program Data from file, this test case examines the loading of program data from a file. It ensures that the program successfully loads the data from the file and displays it in the console.
Test Case 15: View Remaining burgers Stock, this test case checks the functionality to view the remaining stock of burgers. It verifies if the program accurately displays the number of remaining burgers.
Test Case 16: Add burgers to Stock, this test case validates the ability to add burgers to the stock. It ensures that the program correctly adds the burgers and updates the stock count accordingly.
Test Case 17: If burgers not reached minimum level, this test case covers the scenario when the number of burgers is below the minimum level. It ensures that the program provides the appropriate feedback when the stock is insufficient.
Test Case 18: This test case verifies the functionality of viewing the income of each queue. It ensures that the program correctly displays the income of each queue and calculates the total income.
Test Case 19: This test case check weather queues are full and print a massege to the user.
Test Case 20: This test case check weather waiting queue is full and print a massege.
Test Case 21: Exit the Program, this test case verifies the functionality of exiting the program. It ensures that the program correctly terminates when the exit command is given.
Overall, the provided test cases cover a range of scenarios and inputs, including queue initialization, customer addition and removal, input validation, data storage and retrieval, stock management, and program termination. This helps ensure that various aspects of your program are thoroughly tested.

```
import java.util.Scanner;
public class QueueManagementSystem {

    public static void main(String[] args) {

        System.out.println("\n");
        System.out.println("------------------------------Foodies Fave Queue
Management System.------------------------------");
        System.out.println("\n");

        FoodQueue myObj = new FoodQueue();
        myObj.Queues();

        while (true) {
            System.out.println("\n");


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

```
            System.out.println("||                                    MENU
||");

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 a
Queue.                ||");
            System.out.println("||  104 or PCQ: Remove a served customer.
||");
            System.out.println("||  105 or VCS: View Customers Sorted in
alphabetical order.  ||");
            System.out.println("||  106 or SPD: Store Program Data into file.
||");
            System.out.println("||  107 or LPD: Load Program Data from file.
||");
            System.out.println("||  108 or STK: View Remaining burgers Stock.
||");
            System.out.println("||  109 or AFS: Add burgers to Stock.
||");
            System.out.println("||  110 or IFQ: View Income of Each Queue.
||");
            System.out.println("||  111 or DWQ: Display Waiting Queue.
||");
            System.out.println("||  999 or EXT: Exit the Program.
||");

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

            System.out.println("\n");

            myObj.PrintWarningMessage();

            Scanner input = new Scanner(System.in);

            System.out.print("Enter what you want to do: ");
            String task = input.next().toLowerCase();

            //Enhanced swich case to call all methods
            switch (task) {
                case "100", "vfq" -> myObj.ViewAllQueues();
                case "101", "veq" -> myObj.ViewEmptyQueues();
                case "102", "acq" -> myObj.AddCustomer();
                case "103", "rcq" -> myObj. RemoveCustomer();
                case "104", "pcq" -> myObj.RemoveServedCustomer();
                case "105", "vcs" -> myObj.ViewCustomers();
                case "106", "spd" -> myObj.AddDataToFile();
                case "107", "lpd" -> myObj.ReadFileData();
                case "108", "stk" -> myObj.ViewBurgerCount();
                case "109", "afs" -> myObj.AddBurgers();
                case "110", "ifq" -> myObj.CalculateTheIncome();
```

```
                case "111", "dwq" -> myObj.DisplayWaitingQueue();
                case "999", "ext" -> System.exit(0);
                default -> System.out.println("Wrong Input");
            }


        }



    }
}
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;
public class FoodQueue {

    static Customer[][] Cashier = new Customer[3][]; //2D array for all the
cashiers

    static Customer[] WaitingQueue = new Customer[5]; // Making Arraylist for
waiting Queue
    static int FirstFlag = 0; //define the first one of the array
    static int LastFlag = 0; //define the first last of the array
    static int BurgerStock = 50; //define the burger stock
    static int SoldBurgers = 0; //counting the sold burgers

    static int Queue1BurgerCount = 0; //counting burger count for cashier 1
    static int Queue2BurgerCount = 0; //counting burger count for cashier 2
    static int Queue3BurgerCount = 0; //counting burger count for cashier 3


    //define the size for queues
    public void Queues(){
        Cashier[0] = new Customer[2];
        Cashier[1] = new Customer[3];
        Cashier[2] = new Customer[5];

    }




    //method to view all queues
    public static void ViewAllQueues() {
        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (int i = 0; i < 5; i++) { //check the customer
```

```
            for (int j = 0; j < Cashier.length; j++) { // check the cashier
                if (i < Cashier[j].length) {
                    if (Cashier[j][i] == null) {
                        System.out.print(" X    ");
                    } else {
                        System.out.print(" O    ");
                    }
                } else {
                    System.out.print("      ");
                }
            }
            System.out.println();
        }
        System.out.println("0-Occupied X-Notoccupied");
    }

    //method for view empty queues
    public void ViewEmptyQueues(){
        // boolean variables for check the cashiers contains null values
        boolean ContainsNullLine1 = false;
        boolean ContainsNullLine2 = false;
        boolean ContainsNullLine3 = false;

        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (Customer element : Cashier[0]) {
            if (element == null) {
                ContainsNullLine1 = true;
                break;
            }
        }

        for (Customer element : Cashier[1]) {
            if (element == null) {
                ContainsNullLine2 = true;
                break;
            }
        }

        for (Customer element : Cashier[2]) {
            if (element == null) {
                ContainsNullLine3 = true;
                break;
            }
        }

        if(ContainsNullLine1 && ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%2s %5s %6s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x", Cashier[2][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %6s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%8s %6s %n", Cashier[1][2] != null ? "o" :
"x", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine2) {
```

```
            System.out.printf("%2s %5s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x");
            System.out.printf("%8s %n", Cashier[1][2] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine3){
            System.out.printf("%2s %12s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[2][0] != null ? "o" : "x");
            System.out.printf("%2s %12s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][4] != null ? "o" : "x");
        } else if (ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%8s %6s %n",Cashier[1][0] != null ? "o" :
"x",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][1]!= null ? "o" :
"x",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][2]!= null ? "o" :
"x",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        } else if (ContainsNullLine1) {
            System.out.printf("%2s %n",Cashier[0][0]!= null ? "o" : "x");
            System.out.printf("%2s %n",Cashier[0][1]!= null ? "o" : "x");
        } else if (ContainsNullLine2) {
            System.out.printf("%8s %n",Cashier[1][0]!= null ? "o" : "x");
            System.out.printf("%8s %n",Cashier[1][1]!= null ? "o" : "x");
            System.out.printf("%8s %n",Cashier[1][2]!= null ? "o" : "x");
        }else {
            System.out.printf("%15s %n",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        }
        System.out.println("0-Occupied X-Notoccupied");
    }

    //method for check the queue length
    /*
    * param1 Queue - use to get the cashier number for checking
    */
    public static int CheckQueueLength(int Queue){
        int Counter = 0; //variable for get the customer count

        if(Queue == 1){
            for (int i = 0; i < 2; i++){
                if(Cashier[0][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 2){
            for (int i = 0; i < 3; i++){
                if(Cashier[1][i] != null){
```

```
                    Counter += 1;
                }
            }
        }

        if(Queue == 3){
            for (int i = 0; i < 5; i++){
                if(Cashier[2][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 4){  //Checking customer count in WaitingQueue
            for (int i = 0; i < 5; i++){
                if(WaitingQueue[i] != null){
                    Counter += 1;
                }
            }
        }

        return Counter;
    }

    //method to find the smallest queue
    public static int FindTheSmallestQueue() {
        int SmallestQueue =0;
        int Queue1Length = CheckQueueLength(1); //checking the queue length
using CheckQueueLength method according to the queue
        int Queue2Length = CheckQueueLength(2);
        int Queue3Length = CheckQueueLength(3);

        if(Queue1Length !=2 || Queue2Length != 3 || Queue3Length != 5) { //
check the queues are full or not
            if (Queue1Length <= Queue2Length && Queue1Length <= Queue3Length
&& Queue1Length != 2) {
                SmallestQueue = 1;
            } else if (Queue2Length <= Queue1Length && Queue2Length <=
Queue3Length && Queue2Length != 3) {
                SmallestQueue = 2;
            } else {
                SmallestQueue = 3;
            }
        }else {
            System.out.println("Queues are Full please wait");
        }

        return SmallestQueue;
    }

    //method for add customers
    public static void AddCustomer(){

        Scanner input = new Scanner(System.in);

        System.out.print("Enter Customer's First name : ");
        String FirstName = input.next(); // get the customers first name by
```

```
user


        System.out.print("Enter Customer's Last name : ");
        String LastName = input.next();// get the customers last name by user

        // validating the burger count as an integer
        int BurgersRequired = 0;
        boolean ValidInput = false;
        while (!ValidInput) {
            System.out.print("Enter how many Burgers required: ");
            if (input.hasNextInt()) {
                BurgersRequired = input.nextInt(); // get the customers
required burger count by user
                ValidInput = true;
            } else {
                System.out.println("Wrong input. Please enter a valid
integer.");
                input.next(); // Clear the invalid input from the scanner
            }
        }

        Customer Person = new Customer(FirstName, LastName, BurgersRequired);
//creating object to save customer names

        if (CheckQueueLength(1) == 2 && CheckQueueLength(2) == 3 &&
CheckQueueLength(3) == 5){ // check the queues are full
            if (CheckQueueLength(4) != 5) {

                if (WaitingQueue[LastFlag] == null) {
                        WaitingQueue[LastFlag] = Person; //add customer to
the waiting queue
                        System.out.println("\nCustomer Added to Waiting
list\n");
                        if (LastFlag != 4){
                            LastFlag += 1;
                        }else {
                            LastFlag = 0;
                        }

                }

            }else {
                System.out.println("Waiting list is full Please wait");
            }
        }

        int SmallestQueue = FindTheSmallestQueue(); //run
FindTheSmallestQueue method and find the smallest queue

        if (SmallestQueue == 1) {
            Queue1BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[0].length; i++) {
                if (Cashier[0][i] == null) {
                    Cashier[0][i] = Person; //adding the customer details
into queue 1
                    break;
```

```
                }
            }
        } else if (SmallestQueue == 2) {
            Queue2BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[1].length; i++) {
                if (Cashier[1][i] == null) {
                    Cashier[1][i] = Person; //adding the customer details
into queue 2
                    break;
                }
            }
        } else {
            Queue3BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[2].length; i++) {
                if (Cashier[2][i] == null) {
                    Cashier[2][i] = Person; //adding the customer details
into queue 3
                    break;
                }
            }
        }


        ViewAllQueues(); //run ViewAllQueues method to show customer is added

    }

    // method to remove a customer in a specific location
    public static void RemoveCustomer(){
        try {

            Scanner input2 = new Scanner(System.in);
            int WhichQueue;
            int WhichOne;
            // looping until get correct input
            do {

                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input2.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);


            if (WhichQueue == 1) {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove
```

```
                    if (WhichOne < 1 || WhichOne > Cashier[0].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[0].length);

                if (Cashier[0][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[0][1] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }


            } else if (WhichQueue == 2) {
                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[1].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[1].length);

                if (Cashier[1][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[1][2] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
```

```
                    System.out.println("\n");
                }


            } else {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[2].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[2].length);

                if (Cashier[2][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[2][4] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }

        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
    }


    //Method for remove served customer
    static void RemoveServedCustomer(){
        try {
            int WhichQueue;
            int RequiredBurgerCount;

            Scanner input3 = new Scanner(System.in);

            do {
```

```
                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input3.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);

            int CustomerCount;

            if (WhichQueue == 1) {
                RequiredBurgerCount = Cashier[0][0].getBurgersRequired();
                SoldBurgers += RequiredBurgerCount;
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(1);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[0][1] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else if (WhichQueue == 2) {
                RequiredBurgerCount = Cashier[1][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(2);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[1][2] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
```

```
                }
            } else {
                RequiredBurgerCount = Cashier[2][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(3);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[2][4] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }
        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
    }


    //Method to view customers in alpebatical order
    static void ViewCustomers(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName();
                }
            }
        }


        try {

            // Flatten the 2D array into a 1D array
```

```
            String[] flattenedArray = Arrays.stream(CustomerLine)
                    .flatMap(Arrays::stream)
                    .toArray(String[]::new);

            // Bubble sort algorithm to Sort customers in alphabetical order
            int n = flattenedArray.length;
            boolean swapped;

            for (int i = 0; i < n - 1; i++) {
                swapped = false;
                for (int j = 0; j < n - i - 1; j++) {
                    if (flattenedArray[j] != null && flattenedArray[j + 1] !=
null && flattenedArray[j].compareTo(flattenedArray[j + 1]) > 0) {
                        // Swap array[j] and array[j+1]
                        String temp = flattenedArray[j];
                        flattenedArray[j] = flattenedArray[j + 1];
                        flattenedArray[j + 1] = temp;
                        swapped = true;
                    }
                }

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

            for (String item : flattenedArray) {
                if (item != null) {
                    System.out.println(item);
                }
            }

        }catch (NullPointerException e){
            System.out.println("No customers to show");
        }
    }

    //method for add data to a file
    static void AddDataToFile(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName() + " " + Cashier[i][j].getBurgersRequired() + "
```

```
Burgers Required ";
                }
            }
        }

        try {
            File customerDetails = new File("QueueManagementDetails.txt");
            FileWriter Details = new FileWriter(customerDetails);

            Details.write("---------------------Queue Management Details-----
----------------");
            Details.write("\n");

            for(int i=0; i<CustomerLine.length; i++){
                for(int j = 0; j<CustomerLine[i].length; j++){
                    if (CustomerLine[i][j] != null) {
                        Details.write(CustomerLine[i][j]);
                        Details.write(" Position :" + "Queue " + (i+1) + "
Number " + (j+1));
                        Details.write("\n");
                    }

                }
            }

            Details.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Data stored successfully");
    }


    //Method to read data from QueueManagementDetails.txt
    static void ReadFileData(){

        try {

            File customerDetails = new File("QueueManagementDetails.txt");

            Scanner ReadDetails = new Scanner(customerDetails);

            while (ReadDetails.hasNextLine()) {
                String data = ReadDetails.nextLine();
                System.out.println(data);
            }
            ReadDetails.close();

        }catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }


    }
```

```
    //Method to view burger count
    static void ViewBurgerCount(){

        System.out.println(BurgerStock + " burgers are remaining");

    }


    static void AddBurgers(){
        // checking burger count reach minimum level
        if (BurgerStock == 10) {
            BurgerStock += 40;
            System.out.println("Burgers Added");
        }else {
            System.out.println("The number of Burgers has not reachrd the
minimum level");
        }

    }

    //method for calculate the income
    static void CalculateTheIncome(){
        int Queue1Income = Queue1BurgerCount * 650;
        int Queue2Income = Queue2BurgerCount * 650;
        int Queue3Income = Queue3BurgerCount * 650;

        System.out.println("╔═══════════════════════════════════════╗");
        System.out.println("║               Income                ╒");
        System.out.println("╟───────────────────────────────────────╢");
        System.out.println("╟ Queue 1 : " + Queue1Income);
        System.out.println("╟ Queue 2 : " + Queue2Income);
        System.out.println("╟ Queue 2 : " + Queue3Income);
        System.out.println("╟ Total   : " +
(Queue1Income+Queue2Income+Queue3Income));
        System.out.println("╚═══════════════════════════════════════╝");
    }

    //method for print waiting queue First flag to last flag
    /*
    * param1 String[] Array - to get waiting queue array in to the method
    * param2 int FirstElement - to define the first element
     */
    public static void PrintCircularQueue(String[] Array, int FirstElement) {

        int index = FirstElement % Array.length;
        int j = 1;
        for (int i = index; i < index + Array.length; i++) {
            if(Array[i % Array.length] != null) {
                System.out.println(j + "." + Array[i % Array.length]);
                j += 1;
            }
        }

    }

    //method for display waiting queue
    static void DisplayWaitingQueue(){
```

```
        String[] WaitingQueueArray = new String[5];

        //Get data from WaitingQueue and store it in WaitingQueueArray
        for (int i = 0; i < 5; i++) {
            if (WaitingQueue[i] != null) {
                WaitingQueueArray[i] = WaitingQueue[i].getFirstName() + " " +
WaitingQueue[i].getLastName();
            }
        }

        PrintCircularQueue(WaitingQueueArray,FirstFlag);
    }

    //method for print the warning message
    static void PrintWarningMessage(){
        //Print warning massege when burger count reach 10
        if(BurgerStock == 10){
            System.out.println("******-WARNING-*****");
            System.out.println("* Low burger stock *");
            System.out.println("********************");
        }
    }


}
```

```
public class 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;}




}
```

Task 4 - Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 100 or VFQ	********** * Cashiers * ********** X X X X X X X X X X	********** * Cashiers * ********** X X X X X X X X X X	Pass
2	Show empty Queues 101 or VEQ when only queue 1 is empty	********** * Cashiers * ********** X X	********** * Cashiers * ********** X X	pass
3	Show empty Queues 101 or VEQ when queue 1 and 3 are empty	********** * Cashiers * ********** X X X X X X X	********** * Cashiers * ********** X X X X X X X	pass
4	Show empty Queues : 101 or VEQ when queue 1 and 3 are empty (not fully empty some customers are there)	********** * Cashiers * ********** O O X O X X X	********** * Cashiers * ********** O O X O X X X	pass
5	Add customer “Jane” in to shortest queue 102 or ACQ Enter First Name: Jane Enter Last Name: Harris	********** * Cashiers * ********** O O X X X X X X X X	********** * Cashiers * ********** O O X X X X X X X X	pass

6	Validating the burger count	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	Enter how many Burgers required: ww Wrong input. Please enter a valid integer. Enter how many Burgers required:	pass
7	Remove a customer from a queue (Specific location) : 103 or RCQ	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	Enter the Queue to remove the customer (1,2 or 3):2 Which one (1,2,....):2 ********** * Cashiers * ********** O O O O X O X O O O	pass
8	If there are no customer in that location	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	Enter the Queue to remove the customer (1,2 or 3):3 Which one (1,2,....):3 No customer to remove	pass
9	Validating the Queue number	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	Enter the Queue to remove the customer (1,2 or 3):4 Invalid Input	pass
10	Validating the Which One	Which one (1,2,....):0 Invalid Input	Which one (1,2,....):0 Invalid Input	pass
11	Removed a served Customer: 104 or PCQ(When type the queue number it	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	Enter the Queue to remove the customer (1,2 or 3):2 ********** * Cashiers * ********** O O O X X O X X X	pass

	remove the first customer and other customers comes up)	X	X	
12	View Customers Sorted in alphabetical order :105 or VCS	Show the customers in alphabetical order	Show the customers in alphabetical order	pass
13	Store Program Data into file : 106 or SPD	Store the queue details into the file	Store the queue details into the file	pass
14	Load Program Data from file : 107 or LPD	Lord data from that file and show in the console	Lord data from that file and show in the console	pass
15	View Remaining burgers Stock : 108 or STK	Enter what you want to do: 108 40 burgers are remaining	Enter what you want to do: 108 40 burgers are remaining	pass
16	Add burgers to Stock : 109 or AFS	Enter what you want to do: 109 Burgers Added	Enter what you want to do: 109 Burgers Added	pass

17	If burgers not reached minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	Enter what you want to do: 109 The number of Burgers has not reached the minimum level	pass
18	When all the queues are full	Queues are Full please wait	Queues are Full please wait	pass
19	View Income of Each Queue : 110 or IFQ	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	Income Queue 1 : 0 Queue 2 : 0 Queue 2 : 0 Total : 0	pass
20	Customers added to the Waiting queue if the queues are full	Waiting list is full please wait	Waiting list is full please wait	pass
21	Graphical User Interface : 112 or GUI	Open the java fx application	Open the java fx application	pass
22	Press “ Check The Current Status” button	Show all the queue status and income of the queues	Show all the queue status and income of the queues	pass

23	Search in GUI(when there was two customers named peter harris ,peter samuel and enter peter)	peter harris - 1 Burgers Required (Position :Queue 1 Number 1) peter samual - 2 Burgers Required (Position :Queue 2 Number 1)	peter harris - 1 Burgers Required (Position :Queue 1 Number 1) peter samual - 2 Burgers Required (Position :Queue 2 Number 1)	pass
23	Search in GUI(when there was two customers named peter harris ,peter samuel and enter peter samual)	peter samual - 2 Burgers Required (Position :Queue 2 Number 1)	peter samual - 2 Burgers Required (Position :Queue 2 Number 1)	pass
24	Exit the Program : 999 or EXT:	Exit the program	Exit the program	pass

Task 4 - Discussion

Test Case 1: Food Queue Initialized Correctly, this test case ensures that the food queue is initialized correctly after the program starts. It covers the initial setup and verifies if the cashiers and queue structure are displayed correctly.

Test Case 2: Show empty Queues, this test case ensures that the program correctly displays the empty queue and doesn't show the queues which are full filled with customers.

Test Case 3: Show empty Queues when queues 1 and 3 are empty, this test case extends the previous case by checking the scenario when multiple queues are empty. It verifies if the program correctly handles and displays multiple empty queues.
Test Case 4: Show empty Queues when queues 1 and 3 are not fully empty, this test case examines a situation where the queues are not fully empty, but some customers remain. It checks if the program accurately represents the queues with remaining customers.
Test Case 5: Add customer "Jane harris" to the shortest queue, this test case validates the ability to add a customer named "Jane" to the shortest queue. It ensures that the program correctly adds the customer to the shortest queue.
Test Case 6: Validating the burger count, this test case focuses on validating user input for the number of burgers required. It checks if the program handles invalid input, such as non-integer values, and prompts the user to enter a valid integer.
Test Case 7: Remove a customer from a queue (Specific location), This test case tests the removal of a customer from a specific queue and position. It verifies if the program correctly removes the customer and adjusts the queue accordingly.
Test Case 8: If there are no customers in that location, this test case covers the scenario where there are no customers to remove from the specified location. It ensures that the program handles this case appropriately and provides the correct feedback.
Test Case 9: Validating the Queue number, this test case checks if the program validates the user input for the queue number correctly. It verifies that the program handles invalid queue numbers and provides appropriate feedback.
Test Case 10: Validating the Which One, this test case examines the validation of user input for the "Which one" parameter. It ensures that the program handles invalid inputs and prompts the user for a valid option.
Test Case 11: Removed a served Customer, this test case tests the removal of a served customer from a queue. It ensures that the program removes the correct customer from the specified queue and adjusts the remaining customers accordingly.
Test Case 12: View Customers Sorted in alphabetical order, this test case validates the functionality of displaying customers in alphabetical order. It verifies if the program correctly sorts and displays the customers in alphabetical order.
Test Case 13: Store Program Data into file, this test case tests the storage of program data into a file. It ensures that the program correctly saves the queue details into the specified file.
Test Case 14: Load Program Data from file, this test case examines the loading of program data from a file. It ensures that the program successfully loads the data from the file and displays it in the console.
Test Case 15: View Remaining burgers Stock, this test case checks the functionality to view the remaining stock of burgers. It verifies if the program accurately displays the number of remaining burgers.
Test Case 16: Add burgers to Stock, this test case validates the ability to add burgers to the stock. It ensures that the program correctly adds the burgers and updates the stock count accordingly.
Test Case 17: If burgers not reached minimum level, this test case covers the scenario when the number of burgers is below the minimum level. It ensures that the program provides the appropriate feedback when the stock is insufficient.
Test Case 18: This test case verifies the functionality of viewing the income of each queue. It ensures that the program correctly displays the income of each queue and calculates the total income.
Test Case 19: This test case check weather queues are full and print a massege to the user.
Test Case 20: This test case check weather waiting queue is full and print a massege.
Test Case 21: Graphical User Interface, This test case ensures that the program's graphical user interface (GUI) opens successfully. It verifies the program's ability to launch the JavaFX application and present the GUI to the user.
Test Case 22: Press "Check The Current Status" button, This test case tests the functionality of the "Check The Current Status" button in the GUI. It ensures that when the button is pressed, the program displays the current status of the queues and their respective incomes.
Test Case 23: Search in GUI, This test case ensures that the program correctly handles the exit functionality. It verifies that when the user enters the exit command, the program terminates successfully.
Test Case 21: Exit the Program, this test case verifies the functionality of exiting the program. It ensures that the program correctly terminates when the exit command is given.

Overall, the provided test cases cover a range of scenarios and inputs, including queue initialization, customer addition and removal, input validation, data storage and retrieval, stock management, and program termination. This helps ensure that various aspects of your program are thoroughly tested.

```
package com.example.part4;

import java.util.Scanner;
public class QueueManagementSystem {

    public static void main(String[] args) {

        System.out.println("\n");
        System.out.println("------------------------------Foodies Fave Queue
Management System.------------------------------");
        System.out.println("\n");

        FoodQueue myObj = new FoodQueue();
        myObj.Queues();

        QueueManagementController myObj2 = new QueueManagementController();

        while (true) {
            System.out.println("\n");


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

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 a
Queue.                ||");
            System.out.println("||  104 or PCQ: Remove a served customer.
||");
            System.out.println("||  105 or VCS: View Customers Sorted in
alphabetical order.  ||");
            System.out.println("||  106 or SPD: Store Program Data into file.
||");
            System.out.println("||  107 or LPD: Load Program Data from file.
||");
            System.out.println("||  108 or STK: View Remaining burgers Stock.
||");
            System.out.println("||  109 or AFS: Add burgers to Stock.
||");
            System.out.println("||  110 or IFQ: View Income of Each Queue.
||");
            System.out.println("||  111 or DWQ: Display Waiting Queue.
||");
            System.out.println("||  112 or GUI: Graphical User Interface.
||");
            System.out.println("||  999 or EXT: Exit the Program.
||");
```

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

            System.out.println("\n");

            myObj.PrintWarningMessage();

            Scanner input = new Scanner(System.in);

            System.out.print("Enter what you want to do: ");
            String task = input.next().toLowerCase();

            //Enhanced swich case to call all methods
            switch (task) {
                case "100", "vfq" -> myObj.ViewAllQueues();
                case "101", "veq" -> myObj.ViewEmptyQueues();
                case "102", "acq" -> myObj.AddCustomer();
                case "103", "rcq" -> myObj. RemoveCustomer();
                case "104", "pcq" -> myObj.RemoveServedCustomer();
                case "105", "vcs" -> myObj.ViewCustomers();
                case "106", "spd" -> myObj.AddDataToFile();
                case "107", "lpd" -> myObj.ReadFileData();
                case "108", "stk" -> myObj.ViewBurgerCount();
                case "109", "afs" -> myObj.AddBurgers();
                case "110", "ifq" -> myObj.CalculateTheIncome();
                case "111", "dwq" -> myObj.DisplayWaitingQueue();
                case "112", "gui" -> myObj2.Prompt();
                case "999", "ext" -> System.exit(0);
                default -> System.out.println("Wrong Input");
            }


        }



    }
}
```

```
package com.example.part4;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;
public class FoodQueue {

    static Customer[][] Cashier = new Customer[3][]; //2D array for all the
cashiers

    static Customer[] WaitingQueue = new Customer[5]; // Making Arraylist for
```

```
waiting Queue
    static int FirstFlag = 0; //define the first one of the array
    static int LastFlag = 0; //define the first last of the array
    static int BurgerStock = 50; //define the burger stock
    static int SoldBurgers = 0; //counting the sold burgers

    static int Queue1BurgerCount = 0; //counting burger count for cashier 1
    static int Queue2BurgerCount = 0; //counting burger count for cashier 2
    static int Queue3BurgerCount = 0; //counting burger count for cashier 3


    //define the size for queues
    public void Queues(){
        Cashier[0] = new Customer[2];
        Cashier[1] = new Customer[3];
        Cashier[2] = new Customer[5];

    }




    //method to view all queues
    public static void ViewAllQueues() {
        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (int i = 0; i < 5; i++) { //check the customer
            for (int j = 0; j < Cashier.length; j++) { // check the cashier
                if (i < Cashier[j].length) {
                    if (Cashier[j][i] == null) {
                        System.out.print(" X    ");
                    } else {
                        System.out.print(" O    ");
                    }
                } else {
                    System.out.print("      ");
                }
            }
            System.out.println();
        }
        System.out.println("0-Occupied X-Notoccupied");
    }

    //method for view empty queues
    public void ViewEmptyQueues(){
        // boolean variables for check the cashiers contains null values
        boolean ContainsNullLine1 = false;
        boolean ContainsNullLine2 = false;
        boolean ContainsNullLine3 = false;

        System.out.println("****************");
        System.out.println("*   Cashiers   *");
        System.out.println("****************");

        for (Customer element : Cashier[0]) {
```

```
            if (element == null) {
                ContainsNullLine1 = true;
                break;
            }
        }

        for (Customer element : Cashier[1]) {
            if (element == null) {
                ContainsNullLine2 = true;
                break;
            }
        }

        for (Customer element : Cashier[2]) {
            if (element == null) {
                ContainsNullLine3 = true;
                break;
            }
        }

        if(ContainsNullLine1 && ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%2s %5s %6s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x", Cashier[2][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %6s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%8s %6s %n", Cashier[1][2] != null ? "o" :
"x", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine2) {
            System.out.printf("%2s %5s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[1][0] != null ? "o" : "x");
            System.out.printf("%2s %5s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[1][1] != null ? "o" : "x");
            System.out.printf("%8s %n", Cashier[1][2] != null ? "o" : "x");
        } else if (ContainsNullLine1 && ContainsNullLine3){
            System.out.printf("%2s %12s %n", Cashier[0][0] != null ? "o" :
"x", Cashier[2][0] != null ? "o" : "x");
            System.out.printf("%2s %12s %n", Cashier[0][1] != null ? "o" :
"x", Cashier[2][1] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][2] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][3] != null ? "o" : "x");
            System.out.printf("%15s %n", Cashier[2][4] != null ? "o" : "x");
        } else if (ContainsNullLine2 && ContainsNullLine3) {
            System.out.printf("%8s %6s %n",Cashier[1][0] != null ? "o" :
"x",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][1]!= null ? "o" :
"x",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%8s %6s %n",Cashier[1][2]!= null ? "o" :
"x",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        } else if (ContainsNullLine1) {
            System.out.printf("%2s %n",Cashier[0][0]!= null ? "o" : "x");
            System.out.printf("%2s %n",Cashier[0][1]!= null ? "o" : "x");
        } else if (ContainsNullLine2) {
            System.out.printf("%8s %n",Cashier[1][0]!= null ? "o" : "x");
            System.out.printf("%8s %n",Cashier[1][1]!= null ? "o" : "x");
```

```
            System.out.printf("%8s %n",Cashier[1][2]!= null ? "o" : "x");
        }else {
            System.out.printf("%15s %n",Cashier[2][0]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][1]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][2]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][3]!= null ? "o" : "x");
            System.out.printf("%15s %n",Cashier[2][4]!= null ? "o" : "x");
        }
        System.out.println("0-Occupied X-Notoccupied");
    }

    //method for check the queue length
    /*
    * param1 Queue - use to get the cashier number for checking
    */
    public static int CheckQueueLength(int Queue){
        int Counter = 0; //variable for get the customer count

        if(Queue == 1){
            for (int i = 0; i < 2; i++){
                if(Cashier[0][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 2){
            for (int i = 0; i < 3; i++){
                if(Cashier[1][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 3){
            for (int i = 0; i < 5; i++){
                if(Cashier[2][i] != null){
                    Counter += 1;
                }
            }
        }

        if(Queue == 4){  //Checking customer count in WaitingQueue
            for (int i = 0; i < 5; i++){
                if(WaitingQueue[i] != null){
                    Counter += 1;
                }
            }
        }

        return Counter;
    }

    //method to find the smallest queue
    public static int FindTheSmallestQueue() {
        int SmallestQueue =0;
        int Queue1Length = CheckQueueLength(1); //checking the queue length
```

```
using CheckQueueLength method according to the queue
        int Queue2Length = CheckQueueLength(2);
        int Queue3Length = CheckQueueLength(3);

        if(Queue1Length !=2 || Queue2Length != 3 || Queue3Length != 5) { //
check the queues are full or not
            if (Queue1Length <= Queue2Length && Queue1Length <= Queue3Length
&& Queue1Length != 2) {
                SmallestQueue = 1;
            } else if (Queue2Length <= Queue1Length && Queue2Length <=
Queue3Length && Queue2Length != 3) {
                SmallestQueue = 2;
            } else {
                SmallestQueue = 3;
            }
        }else {
            System.out.println("Queues are Full please wait");
        }

        return SmallestQueue;
    }

    //method for add customers
    public static void AddCustomer(){

        Scanner input = new Scanner(System.in);

        System.out.print("Enter Customer's First name : ");
        String FirstName = input.next(); // get the customers first name by
user


        System.out.print("Enter Customer's Last name : ");
        String LastName = input.next();// get the customers last name by user

        // validating the burger count as an integer
        int BurgersRequired = 0;
        boolean ValidInput = false;
        while (!ValidInput) {
            System.out.print("Enter how many Burgers required: ");
            if (input.hasNextInt()) {
                BurgersRequired = input.nextInt(); // get the customers
required burger count by user
                ValidInput = true;
            } else {
                System.out.println("Wrong input. Please enter a valid
integer.");
                input.next(); // Clear the invalid input from the scanner
            }
        }

        Customer Person = new Customer(FirstName, LastName, BurgersRequired);
//creating object to save customer names

        if (CheckQueueLength(1) == 2 && CheckQueueLength(2) == 3 &&
CheckQueueLength(3) == 5){ // check the queues are full
            if (CheckQueueLength(4) != 5) {
```

```
                if (WaitingQueue[LastFlag] == null) {
                        WaitingQueue[LastFlag] = Person; //add customer to
the waiting queue
                        System.out.println("\nCustomer Added to Waiting
list\n");
                        if (LastFlag != 4){
                            LastFlag += 1;
                        }else {
                            LastFlag = 0;
                        }

                }

            }else {
                System.out.println("Waiting list is full Please wait");
            }
        }

        int SmallestQueue = FindTheSmallestQueue(); //run
FindTheSmallestQueue method and find the smallest queue

        if (SmallestQueue == 1) {
            Queue1BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[0].length; i++) {
                if (Cashier[0][i] == null) {
                    Cashier[0][i] = Person; //adding the customer details
into queue 1
                    break;
                }
            }
        } else if (SmallestQueue == 2) {
            Queue2BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[1].length; i++) {
                if (Cashier[1][i] == null) {
                    Cashier[1][i] = Person; //adding the customer details
into queue 2
                    break;
                }
            }
        } else {
            Queue3BurgerCount += BurgersRequired;
            for (int i = 0; i < Cashier[2].length; i++) {
                if (Cashier[2][i] == null) {
                    Cashier[2][i] = Person; //adding the customer details
into queue 3
                    break;
                }
            }
        }


        ViewAllQueues(); //run ViewAllQueues method to show customer is added

    }

    // method to remove a customer in a specific location
```

```
    public static void RemoveCustomer(){
        try {

            Scanner input2 = new Scanner(System.in);
            int WhichQueue;
            int WhichOne;
            // looping until get correct input
            do {

                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input2.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);


            if (WhichQueue == 1) {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[0].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[0].length);

                if (Cashier[0][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[0][1] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }


            } else if (WhichQueue == 2) {
```

```
                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[1].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[1].length);

                if (Cashier[1][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[1][2] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }


            } else {

                // looping until get correct input
                do {

                    System.out.print("Which one (1,2,....):");
                    WhichOne = input2.nextInt(); //asking which customer want
to remove

                    if (WhichOne < 1 || WhichOne > Cashier[2].length) {
                        System.out.println("Invalid Input");
                    }

                } while (WhichOne < 1 || WhichOne > Cashier[2].length);

                if (Cashier[2][WhichOne - 1] != null) {
                    for (int i = WhichOne - 1; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier

                    }
                    Cashier[2][4] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
```

```
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                }else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                    System.out.println("\n");
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }

        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
    }


    //Method for remove served customer
    static void RemoveServedCustomer(){
        try {
            int WhichQueue;
            int RequiredBurgerCount;

            Scanner input3 = new Scanner(System.in);

            do {

                System.out.print("Enter the Queue to remove the customer (1,2
or 3):");
                WhichQueue = input3.nextInt(); //asking queue to remove
customer

                if (WhichQueue > 3 || WhichQueue < 1) {
                    System.out.println("Invalid Input");
                }

            } while (WhichQueue > 3 || WhichQueue < 1);

            int CustomerCount;

            if (WhichQueue == 1) {
                RequiredBurgerCount = Cashier[0][0].getBurgersRequired();
                SoldBurgers += RequiredBurgerCount;
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(1);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 1; i++) {
                        Cashier[0][i] = Cashier[0][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[0][1] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
```

```
                    }else {
                        FirstFlag = 0;
                    }

                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }

            } else if (WhichQueue == 2) {
                RequiredBurgerCount = Cashier[1][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(2);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 2; i++) {
                        Cashier[1][i] = Cashier[1][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[1][2] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            } else {
                RequiredBurgerCount = Cashier[2][0].getBurgersRequired();
                BurgerStock = BurgerStock - RequiredBurgerCount;
                CustomerCount = CheckQueueLength(3);

                if (CustomerCount != 0) {
                    for (int i = 0; i < 4; i++) {
                        Cashier[2][i] = Cashier[2][i + 1]; //shifting the
elements of cashier
                    }
                    Cashier[2][4] = WaitingQueue[FirstFlag];
                    WaitingQueue[FirstFlag] = null;
                    if (FirstFlag != 4) {
                        FirstFlag += 1;
                    }else {
                        FirstFlag = 0;
                    }
                } else {
                    System.out.println("No customer to remove"); //If there
are no customers in the queue
                }
            }
        }catch (InputMismatchException e){
            System.out.println("Wrong input");
        }
        ViewAllQueues(); //run ViewAllQueues method to show customer was
removed
```

```
    }


    //Method to view customers in alpebatical order
    static void ViewCustomers(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName();
                }
            }
        }


        try {

            // Flatten the 2D array into a 1D array
            String[] flattenedArray = Arrays.stream(CustomerLine)
                    .flatMap(Arrays::stream)
                    .toArray(String[]::new);

            // Bubble sort algorithm to Sort customers in alphabetical order
            int n = flattenedArray.length;
            boolean swapped;

            for (int i = 0; i < n - 1; i++) {
                swapped = false;
                for (int j = 0; j < n - i - 1; j++) {
                    if (flattenedArray[j] != null && flattenedArray[j + 1] !=
null && flattenedArray[j].compareTo(flattenedArray[j + 1]) > 0) {
                        // Swap array[j] and array[j+1]
                        String temp = flattenedArray[j];
                        flattenedArray[j] = flattenedArray[j + 1];
                        flattenedArray[j + 1] = temp;
                        swapped = true;
                    }
                }

                // If no two elements were swapped in the inner loop, the
array is already sorted
                if (!swapped) {
                    break;
                }
            }
```

```
            for (String item : flattenedArray) {
                if (item != null) {
                    System.out.println(item);
                }
            }

        }catch (NullPointerException e){
            System.out.println("No customers to show");
        }
    }

    //method for add data to a file
    static void AddDataToFile(){

        String[][]  CustomerLine = new String[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new String[2];
        CustomerLine[1] = new String[3];
        CustomerLine[2] = new String[5];

        //Get data from Cashier and store it in CustomerLine Array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < Cashier[i].length; j++) {
                if (Cashier[i][j] != null) {
                    CustomerLine[i][j] = Cashier[i][j].getFirstName() + " " +
Cashier[i][j].getLastName() + " " + Cashier[i][j].getBurgersRequired() + "
Burgers Required ";
                }
            }
        }

        try {
            File customerDetails = new File("QueueManagementDetails.txt");
            FileWriter Details = new FileWriter(customerDetails);

            Details.write("---------------------Queue Management Details-----
----------------");
            Details.write("\n");

            for(int i=0; i<CustomerLine.length; i++){
                for(int j = 0; j<CustomerLine[i].length; j++){
                    if (CustomerLine[i][j] != null) {
                        Details.write(CustomerLine[i][j]);
                        Details.write(" Position :" + "Queue " + (i+1) + "
Number " + (j+1));
                        Details.write("\n");
                    }

                }
            }

            Details.close();
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
        System.out.println("Data stored successfully");
    }


    //Method to read data from QueueManagementDetails.txt
    static void ReadFileData(){

        try {

            File customerDetails = new File("QueueManagementDetails.txt");

            Scanner ReadDetails = new Scanner(customerDetails);

            while (ReadDetails.hasNextLine()) {
                String data = ReadDetails.nextLine();
                System.out.println(data);
            }
            ReadDetails.close();

        }catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }


    }


    //Method to view burger count
    static void ViewBurgerCount(){

        System.out.println(BurgerStock + " burgers are remaining");

    }


    static void AddBurgers(){
        // checking burger count reach minimum level
        if (BurgerStock == 10) {
            BurgerStock += 40;
            System.out.println("Burgers Added");
        }else {
            System.out.println("The number of Burgers has not reachrd the
minimum level");
        }

    }

    //method for calculate the income
    static void CalculateTheIncome(){
        int Queue1Income = Queue1BurgerCount * 650;
        int Queue2Income = Queue2BurgerCount * 650;
        int Queue3Income = Queue3BurgerCount * 650;

        System.out.println("╔═══════════════════════════════════╗");
        System.out.println("║                Income                ╕");
```

```
        System.out.println("╟──────────────────────────────────────────╢");
        System.out.println("╟ Queue 1 : " + Queue1Income);
        System.out.println("╟ Queue 2 : " + Queue2Income);
        System.out.println("╟ Queue 2 : " + Queue3Income);
        System.out.println("╟ Total   : " +
(Queue1Income+Queue2Income+Queue3Income));
        System.out.println("╚══════════════════════════════════════════╝");
    }

    //method for print waiting queue First flag to last flag
    /*
    * param1 String[] Array - to get waiting queue array in to the method
    * param2 int FirstElement - to define the first element
     */
    public static void PrintCircularQueue(String[] Array, int FirstElement) {

        int index = FirstElement % Array.length;
        int j = 1;
        for (int i = index; i < index + Array.length; i++) {
            if(Array[i % Array.length] != null) {
                System.out.println(j + "." + Array[i % Array.length]);
                j += 1;
            }
        }

    }

    //method for display waiting queue
    static void DisplayWaitingQueue(){

        String[] WaitingQueueArray = new String[5];

        //Get data from WaitingQueue and store it in WaitingQueueArray
        for (int i = 0; i < 5; i++) {
            if (WaitingQueue[i] != null) {
                WaitingQueueArray[i] = WaitingQueue[i].getFirstName() + " " +
WaitingQueue[i].getLastName();
            }
        }

        PrintCircularQueue(WaitingQueueArray,FirstFlag);
    }

    //method for print the warning message
    static void PrintWarningMessage(){
        //Print warning massege when burger count reach 10
        if(BurgerStock == 10){
            System.out.println("******-WARNING-*****");
            System.out.println("* Low burger stock *");
            System.out.println("********************");
        }
    }


}
```

```java
package com.example.part4;

public class 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;}




}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="793.0" prefWidth="601.0" style="-fx-
background-color: #000000;" xmlns="http://javafx.com/javafx/19"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.part4.QueueManagementController">
   <children>
      <Button layoutX="222.0" layoutY="419.0" mnemonicParsing="false"
onAction="#AddDetails" style="-fx-background-color: #9b72e0;" text="Check The
Current Status" textFill="WHITE" />
      <Label fx:id="C1P1" layoutX="84.0" layoutY="198.0" textFill="WHITE" />
      <Label fx:id="C1P2" layoutX="84.0" layoutY="232.0" textFill="WHITE" />
      <Label fx:id="C2P1" layoutX="213.0" layoutY="198.0" textFill="WHITE" />
      <Label fx:id="C2P2" layoutX="213.0" layoutY="233.0" textFill="WHITE" />
      <Label fx:id="C2P3" layoutX="212.0" layoutY="267.0" textFill="WHITE" />
      <Label fx:id="C3P1" layoutX="349.0" layoutY="198.0" textFill="WHITE" />
      <Label fx:id="C3P2" layoutX="352.0" layoutY="233.0" textFill="WHITE" />
      <Label fx:id="C3P3" layoutX="352.0" layoutY="267.0" textFill="WHITE" />
```

```
      <Label fx:id="C3P4" layoutX="352.0" layoutY="303.0" textFill="WHITE" />
      <Label fx:id="C3P5" layoutX="352.0" layoutY="343.0" textFill="WHITE" />
      <Label fx:id="W1" layoutX="479.0" layoutY="198.0" textFill="WHITE" />
      <Label fx:id="W2" layoutX="479.0" layoutY="233.0" textFill="WHITE" />
      <Label fx:id="W3" layoutX="479.0" layoutY="267.0" textFill="WHITE" />
      <Label fx:id="W4" layoutX="479.0" layoutY="303.0" textFill="WHITE" />
      <Label fx:id="W5" layoutX="479.0" layoutY="343.0" textFill="WHITE" />
      <Label layoutX="62.0" layoutY="164.0" text="Cashier 1" textFill="WHITE"
/>
      <Label layoutX="190.0" layoutY="164.0" text="Cashier 2"
textFill="WHITE" />
      <Label layoutX="326.0" layoutY="164.0" text="Cashier 3"
textFill="WHITE" />
      <Label layoutX="450.0" layoutY="164.0" text="wainting List"
textFill="WHITE" />
      <Label layoutX="123.0" layoutY="24.0" text="Foodies Fave Queue
Management System" textFill="WHITE">
         <font>
            <Font name="BM HANNA Air OTF" size="20.0" />
         </font></Label>
      <TextField fx:id="search" layoutX="190.0" layoutY="503.0"
promptText="Enter the name" />
      <Label layoutX="17.0" layoutY="508.0" text="Search Customer Position"
textFill="WHITE" />
      <Button layoutX="364.0" layoutY="503.0" mnemonicParsing="false"
onAction="#SearchDetails" style="-fx-background-color: #afe072;"
text="Search" textFill="WHITE" />
      <TextArea fx:id="ShowSearch" layoutX="14.0" layoutY="573.0"
prefHeight="200.0" prefWidth="349.0" />
      <Label layoutX="191.0" layoutY="457.0" prefHeight="17.0"
prefWidth="219.0" text="Click the button to check current status"
textFill="WHITE" />
      <Label layoutX="464.0" layoutY="561.0" prefHeight="25.0"
prefWidth="44.0" text="Income" textFill="WHITE" />
      <Label fx:id="Q1I" layoutX="525.0" layoutY="600.0" textFill="WHITE" />
      <Label fx:id="Q2I" layoutX="525.0" layoutY="636.0" textFill="WHITE" />
      <Label fx:id="Q3I" layoutX="525.0" layoutY="676.0" textFill="WHITE" />
      <Label layoutX="417.0" layoutY="600.0" text="Queue 1" textFill="WHITE"
/>
      <Label layoutX="416.0" layoutY="636.0" text="Queue 2" textFill="WHITE"
/>
      <Label layoutX="416.0" layoutY="676.0" text="Queue 3" textFill="WHITE"
/>
      <Button layoutX="550.0" layoutY="747.0" mnemonicParsing="false"
onAction="#Exit" style="-fx-background-color: #eb2626;" text="Exit"
textFill="WHITE" />
      <Label fx:id="total" layoutX="525.0" layoutY="711.0" textFill="WHITE"
/>
      <Label layoutX="504.0" layoutY="693.0" text="--------------"
textFill="WHITE" />
      <Label layoutX="504.0" layoutY="719.0" text="--------------"
textFill="WHITE" />
      <ImageView fitHeight="76.0" fitWidth="62.0" layoutX="53.0"
layoutY="88.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@cashier.png" />
         </image>
```

```
      </ImageView>
      <ImageView fitHeight="85.0" fitWidth="62.0" layoutX="185.0"
layoutY="88.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@cashier.png" />
         </image>
      </ImageView>
      <ImageView fitHeight="62.0" fitWidth="68.0" layoutX="318.0"
layoutY="88.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@cashier.png" />
         </image>
      </ImageView>
      <ImageView fitHeight="62.0" fitWidth="52.0" layoutX="456.0"
layoutY="93.0" pickOnBounds="true" preserveRatio="true">
         <image>
            <Image url="@queue.png" />
         </image>
      </ImageView>
   </children>
</Pane>
```

```
package com.example.part4;

import javafx.application.Application;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.TextArea;

import java.util.ArrayList;
import java.util.Arrays;

public class QueueManagementController {
    @FXML
    private Label C1P1;

    @FXML
    private Label C1P2;

    @FXML
    private Label C2P1;

    @FXML
    private Label C2P2;

    @FXML
    private Label C2P3;

    @FXML
    private Label C3P1;

    @FXML
    private Label C3P2;
```

```
    @FXML
    private Label C3P3;

    @FXML
    private Label C3P4;

    @FXML
    private Label C3P5;

    @FXML
    private Label W1;

    @FXML
    private Label W2;

    @FXML
    private Label W3;

    @FXML
    private Label W4;

    @FXML
    private Label W5;

    @FXML
    private Label Q1I;

    @FXML
    private Label Q2I;

    @FXML
    private Label Q3I;

    @FXML
    private Label total;

    @FXML
    private TextField search;

    @FXML
    private TextArea ShowSearch;

    public void Prompt() {
        Application.launch(QueueManagementApplication.class);
    }



    static FoodQueue obj = new FoodQueue();


    @FXML
    protected void AddDetails(){

        for (int i = 0; i < 4; i++) {
            if (i == 0) {
```

```
                if (obj.Cashier[0][0] == null) {
                    C1P1.setText("X");
                } else {
                    C1P1.setText("0 - " + obj.Cashier[0][0].getFirstName() +
" " + obj.Cashier[0][0].getLastName());
                }

                if (obj.Cashier[0][1] == null) {
                    C1P2.setText("X");
                } else {
                    C1P2.setText("0 - " + obj.Cashier[0][1].getFirstName() +
" " + obj.Cashier[0][1].getLastName());
                }


            }else if (i == 1) {
                if (obj.Cashier[1][0] == null) {
                    C2P1.setText("X");
                } else {
                    C2P1.setText("0 - " + obj.Cashier[1][0].getFirstName() +
" " + obj.Cashier[1][0].getLastName());
                }

                if (obj.Cashier[1][1] == null) {
                    C2P2.setText("X");
                } else {
                    C2P2.setText("0 - " + obj.Cashier[1][1].getFirstName() +
" " + obj.Cashier[1][1].getLastName());
                }

                if (obj.Cashier[1][2] == null) {
                    C2P3.setText("X");
                } else {
                    C2P3.setText("0 - " + obj.Cashier[1][2].getFirstName() +
" " + obj.Cashier[1][2].getLastName());
                }

            }else if(i == 2){
                if (obj.Cashier[2][0] == null) {
                    C3P1.setText("X");
                } else {
                    C3P1.setText("0 - " + obj.Cashier[2][0].getFirstName() +
" " + obj.Cashier[2][0].getLastName());
                }

                if (obj.Cashier[2][1] == null) {
                    C3P2.setText("X");
                } else {
                    C3P2.setText("0 - " + obj.Cashier[2][1].getFirstName() +
" " + obj.Cashier[2][1].getLastName());
                }

                if (obj.Cashier[2][2] == null) {
                    C3P3.setText("X");
                } else {
                    C3P3.setText("0 - " + obj.Cashier[2][2].getFirstName() +
" " + obj.Cashier[2][2].getLastName());
```

```
                }

                if (obj.Cashier[2][3] == null) {
                    C3P4.setText("X");
                } else {
                    C3P4.setText("0 - " + obj.Cashier[2][3].getFirstName() +
" " + obj.Cashier[2][3].getLastName());
                }

                if (obj.Cashier[2][4] == null) {
                    C3P5.setText("X");
                } else {
                    C3P5.setText("0 - " + obj.Cashier[2][4].getFirstName() +
" " + obj.Cashier[2][4].getLastName());
                }

            }else {
                if (obj.WaitingQueue[0] == null) {
                    W1.setText("X");
                } else {
                    W1.setText("0 - " + obj.WaitingQueue[0].getFirstName() +
" " + obj.WaitingQueue[0].getLastName());
                }

                if (obj.WaitingQueue[1] == null) {
                    W2.setText("X");
                } else {
                    W2.setText("0 - " + obj.WaitingQueue[1].getFirstName() +
" " + obj.WaitingQueue[1].getLastName());
                }

                if (obj.WaitingQueue[2] == null) {
                    W3.setText("X");
                } else {
                    W3.setText("0 - " + obj.WaitingQueue[2].getFirstName() +
" " + obj.WaitingQueue[2].getLastName());
                }

                if (obj.WaitingQueue[3] == null) {
                    W4.setText("X");
                } else {
                    W4.setText("0 - " + obj.WaitingQueue[3].getFirstName() +
" " + obj.WaitingQueue[3].getLastName());
                }

                if (obj.WaitingQueue[4] == null) {
                    W5.setText("X");
                } else {
                    W5.setText("0 - " + obj.WaitingQueue[4].getFirstName() +
" " + obj.WaitingQueue[4].getLastName());
                }
            }
        }

        //Calculating the Income
        int Queue1Income = obj.Queue1BurgerCount * 650;
        int Queue2Income = obj.Queue2BurgerCount * 650;
```

```
        int Queue3Income = obj.Queue3BurgerCount * 650;
        int Total = (obj.Queue1BurgerCount + obj.Queue2BurgerCount +
obj.Queue3BurgerCount) * 650;

        Q1I.setText("\u00a3" + String.valueOf(Queue1Income));
        Q2I.setText("\u00a3" + String.valueOf(Queue2Income));
        Q3I.setText("\u00a3" + String.valueOf(Queue3Income));
        total.setText("\u00a3" + String.valueOf(Total));
    }

    @FXML
    protected void SearchDetails(){
        Customer[][]  CustomerLine = new Customer[3][]; // 2D array for store
customer names


        // set size for 2D array element which stored customer names
        CustomerLine[0] = new Customer[2];
        CustomerLine[1] = new Customer[3];
        CustomerLine[2] = new Customer[5];

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < obj.Cashier[i].length; j++) {
                if (obj.Cashier[i][j] != null) {
                    CustomerLine[i][j] = obj.Cashier[i][j];
                }
            }
        }

        ArrayList<String> MachingNames = new ArrayList<String>();

        String SearchString = search.getText();


        for (int i = 0; i < CustomerLine.length; i++) {
            for (int j = 0; j < CustomerLine[i].length; j++) {
                if (CustomerLine[i][j] != null &&
(CustomerLine[i][j].getFirstName().equals(SearchString) ||
(CustomerLine[i][j].getFirstName() + " " +
CustomerLine[i][j].getLastName()).equals(SearchString))) {
                    MachingNames.add(obj.Cashier[i][j].getFirstName() + " " +
obj.Cashier[i][j].getLastName() + " - " +
obj.Cashier[i][j].getBurgersRequired() + " Burgers Required " + "(Position :"
+ "Queue " + (i+1) + " Number " + (j+1) + ")");

                }
            }

        }


        if (!MachingNames.isEmpty()) {
            StringBuilder stringBuilder = new StringBuilder();

            for (String name : MachingNames) {
                stringBuilder.append(name).append("\n");
            }
```

```
            ShowSearch.setText(stringBuilder.toString());
        } else {
            ShowSearch.setText("No matching customers found.");
        }
    }

    @FXML
    protected void Exit(){
        System.exit(0);
    }

}
```

```
package com.example.part4;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class QueueManagementApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(QueueManagementApplication.class.getResource("QueueManagement-
view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 600, 800);
        stage.setTitle("Foodies Fave Queue Management System");
        stage.setScene(scene);
        stage.show();

    }



    public static void main(String[] args) {
        launch();
    }
}
```

<END>