Informatics Institute of Technology

Department of Computing

Software Development II Coursework Report

Module	: 4COSC010C.3: Software Development II (2022)
Module Leader	: Mr. TG Deshan Koshala Sumanathilaka
Date of submission	: 17/07/23
Student ID	: <20222357> / <W1989400>
Student First Name	: surruthisha
Student Surname	: sundareswaran

Name	: Surruthisha sundareswaran
Student ID	: 20222357

Test Cases

Array version

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 101 or VEQ	Displays 'empty' for all queues.	Displays Queue 1 is empty Queue 2 is empty Queue 3 is empty	Pass
2	Add customer "Jane" to Queue 2 102 or ACQ Enter cashier number: 2 Enter Name: Jane	Display "Queue is full"	Display: Customer jane added to the queue.	Fail
3.	Add customer "Izzy" to Queue 1 102 or ACQ Enter cashier number: 1 Enter Name: Izzy	Display 'Izzy added to the queue 1 successfully"	Customer Izzy added to the queue.	pass
.4.	Add customer "Alice" to Queue 1 102 or ACQ Enter cashier number: 1 Enter Name: Alice	Display 'Alice added to the queue 1 successfully"	Customer Alice added to the queue	pass
.5	Add customer to Queue 1 102 or ACQ Enter cashier number: 1	Display "queue is full."	Queue is full customer cannot be added.	pass
6	Food Queue Initialized Correctly After program starts, 101 or VEQ	Displays 'empty' for queue 3	Displays "queue 3 empty"	pass

7.	Add 100 or VFQ to view the queues before adding any customers	Display:
		X X X
		X X X
		  X X
		    X
		    X
			X X X
			X X X
			  X X
			    X
			    X
			O-Occupied
				Pass

		O-Occupied X-Unoccupied	X-Unoccupied	
8.	Add customer “Harry” to Queue 3 102 or ACQ Enter cashier number: 3 Enter Name: Harry	Display ‘Harry added to the queue 1 successfully”	Display ‘Harry added to the queue 1 successfully”	pass
9.	Add customer “Ken” to Queue 102 or ACQ Enter cashier number: 2 Enter Name: Ken	Display ‘ken added to the queue 1 successfully”	Display ‘ken added to the queue 1 successfully”	pass
10.	100 or VFQ to view the queues after adding the customers	Display: O O O O O X X X X X O-Occupied X-Unoccupied	O O O O O X X X X X O-Occupied X-Unoccupied	pass
11.	103 or RCQ to remove a customer from the queue. Cashier number no:2 Position(1-3):2	Display:” customer Ken is removed from the queue.”	Display:” Customer Ken is removed from the queue.”	pass
12.	100 or VFQ to view the queue after removing the customer ken.	Display: O O O O X X X X X X O-Occupied	O O O O X X X X X X O-Occupied X-Unoccupied	Pass

		X-Unoccupied		
13.	104 or PCQ remove served customer.	Display:” Customer Izzy has been served.”	Customer Izzy has been served.	pass
14.	105 or VCS to view customers sorted in alphabetical order	Display: Alice Harry Jane	Alice Harry Jane	pass
15.	106 or SPD to store program data in to file	Display:” Program data stored successfully”	Display:” Program data stored successfully”	pass
16.	107 or LPD to load program data to store files.	Display:” Program data stored in a text file successfully”	program data stored in a text file successfully	pass
17.	108 or STK to view the remaining burgers left.	Display:”15 burgers left.”	15 burgers left	pass
18.	Add customer “Julia” to Queue 102 or ACQ Enter cashier number : 3 Enter Name: Julia	Display ‘Julia added to the queue 3 successfully. Warning:low stock count ;10”	Julia added to queue 3 successfully warning: low stock count:10”	pass
19.	Add customer “Rosie” to Queue 102 or ACQ Enter cashier number: 2 Enter Name: Rosie	Display ‘Rosie added to queue 2 successfully warning stock count:5”	Rosie added to queue 2 successfully stock count:10”	pass
20.	108 or STK to view the no of remaining burgers left.	Display:’5 burgers left. Warning: only 5 burgers left!”	Display:’5 burgers left. Warning: only 5 burgers left!”	pass
21.	Add customer “Sam” to Queue 102 or ACQ Enter Queue: 3 Enter Name: Sam	Display ‘ customer Sam added to the queue 3 ”Warning: only 0 burgers left!”	Display ‘ customer Sam added to the queue 3 ”Warning: only 0 burgers left!”	pass

22.	Add customer "Mark" to Queue 102 or ACQ	Display:" no more burgers"	Display:"no more burgers"	pass
.				
23.	109 or AFS to add burgers to the stock. (add :30 burgers)	Display:"30 burgers remaining.total stock count :30"	"30 burgers remaining.total stock count is:30"	pass
24.	999 or EXT to exit the program	Display:" Process finished with exit code 0"	Process finished with exit code 0	pass

Class version

1.	Food Queue Initialized Correctly After program starts, 100 or VFQ before adding any customers.	Display: X X X X X X X X X X O-Occupied X-Unoccupied Waiting list:X X X X X X X X X X	X X X X X X X X X X O-Occupied X-Unoccupied Waiting list:X X X X X X X X X X	Pass
2.	Food Queue Initialized Correctly After program starts, 101 or VEQ	Displays 'empty' for all queues.	Displays Queue 1 empty Queue 2 empty Queue 3 empty	pass

3.	Add customer “Charlie Puth” to Queue 102 or ACQ Enter First Name: Charlie Enter Last Name: Puth No of burgers: 2	Display ‘Charlie Puth added successfully”	Customer Charlie Puth is added to the queue.	Pass
4.	Add customer “Maya Dominic” to Queue 102 or ACQ Enter First Name: Maya Enter Last Name:Dominic No of burgers:1	Display ‘Maya Dominic added successfully”	Customer Maya Dominic is added to the queue.	Pass
5.	Add customer “Tara Edward” to Queue 102 or ACQ Enter First Name: Tara Enter Last Name:Edward No of burgers: 3	Display ‘Tara Edward added successfully”	Customer Tara Edward is added to the queue.	Pass
6.	Add customer “Kylie Swiss” to Queue 102 or ACQ Enter First Name: Kylie Enter Last Name:Swiss No of burgers: 1	Display ‘Kylie Swiss added successfully”	Customer Kylie Swiss is added to the queue.	Pass
7.	Add customer “Amber Herd” to Queue 102 or ACQ Enter First Name: Amber Enter Last Name:Herd No of burgers: 4	Display ‘Amber Herd added successfully”	Customer Amber Herd is added to the queue.	Pass
8.	Add customer “George Pattson” to Queue 102 or ACQ Enter First Name: George Enter Last Name:Pattson No of burgers: 2	Display ‘George Pattson added successfully”	Customer George Pattson is added to the queue.	Pass
9.	Add customer “Babara Smith” to Queue 102 or ACQ Enter First Name: Babara Enter Last Name:Smith	Display ‘Babara Smith added successfully”	Customer Babara Smith is added to the queue.	Pass

	No of burgers: 1			
10.	Add customer "Angela Gomez" to Queue 102 or ACQ Enter First Name: Angela Enter Last Name:Gomez No of burgers: 2	Display 'Angela Gomez added successfully"	Customer Angela Gomez is added to the queue.	Pass
11.	Add customer "Crystal Thompsett" to Queue 102 or ACQ Enter First Name: Crystal Enter Last Name:Thompsett No of burgers: 1	Display 'Crystal Thompset added successfully"	Customer Crystal Thompsett is added to the queue.	Pass
12.	Add customer "Noel Jenner" to Queue 102 or ACQ Enter First Name: Noel Enter Last Name:Jenner No of burgers: 1	Display 'Noel Jenner added successfully"	Customer Noel Jenner is added to the queue.The queue is full.	Pass
13.	Food Queue Initialized Correctly After program starts, option 100.	Display: O O O O O O O O O O O-Occupied X-Unoccupied Waiting list:X X X X X X X X X X	Display: O O O O O O O O O O O-Occupied X-Unoccupied Waiting list:X X X X X X X X X X	pass
14.	Add customer "Chris Browns" to Queue 102 or ACQ Enter First Name: Chris Enter Last Name:Browns No of burgers: 2	Display:"Chris Browns is added to the waiting list"	Customer Chris brown is added to the waiting list.	pass

14.	Food Queue Initialized Correctly After program starts, 100 or VFQ after adding the customers.	Display: O O O O O O O O O O O-Occupied X-Unoccupied Waiting list:O X X X X X X X X X	Display: O O O O O O O O O O O-Occupied X-Unoccupied Waiting list:O X X X X X X X X X	pass
15.	103 or RCQ to remove a customer from the queue. Cashier no:1 Position(1-2):2	Display:" customer Kylie Swiss is removed from the queue."	Display:" customer Kylie Swiss is removed from the queue."	Pass
16.	104 or PCQ remove served customer.	Display:"customer Charlie puth has been served."	Display:"customer Charlie puth has been served."	pass
17.	Food Queue Initialized Correctly After program starts, 100 or after removing and serving the customer	Display: O O O X O O O O O O O-Occupied X-Unoccupied Waiting list:X X X X X X X X X	Display: O O O X O O O O O O O-Occupied X-Unoccupied Waiting list:X X X X X X X X X	pass
18.	103 or RCQ to remove a customer from the queue. Cashier no:1	Customer chris brown removed from the queue.all queues in the waiting list are	Customer chris brown removed from the queue.all queues in the waiting list are	pass

	Position(1-2):1	empty .no customer to remove	empty .no customer to remove	
18.	Food Queue Initialized Correctly After program starts, 101 or VEQ	Display: Queue 1 is empty	Display:Queue 1 is Empty.	pass
19.	105 or VCS to view customers sorted in alphabetical order	Amber Herd Angela Gomez Babara Smith Crystal Thompset George Pattson Maya Dominic Noel Jenner Tara Edward	Amber Herd Angela Gomez Babara Smith Crystal Thompset George Pattson Maya Dominic Noel Jenner Tara Edward	pass
20.	106 or SPD to store program data in to file	Display:" Program data stored successfully"	Display:" Program data stored successfully"	Pass
21.	107 or LPD to load program data to store files.	Display:" Program data stored in a text file successfully"	Display:" Program data stored in a text file successfully"	pass
22.	108 or STK to view the no of remaining burgers left.	Display:"29 burgers remaining"	Display:"29 burgers remaining"	pass
23.	109 or AFS to add burgers to the stock. (add :10 burgers)	Display:"39 burgers remaining"	Display:"39 burgers remaining"	pass
24.	110 or IFQ to print the income of each queue .	Display: Queue 1:1300 Queue 2:3900 Queue 3:5850	Display: Queue 1:1300 Queue 2:3900 Queue 3:5850	pass

25.	999 or EXT to exit the program	Display:” Process finished with exit code 0”	Display:” Process finished with exit code 0”	pass

Discussion

<<Discussion of how you chose your test cases to ensure that your tests cover all aspects of your program >>So first I did to do 2 test runs one for the array version which contains the task 1 and another test run for class version which contains task 2 and task 3.

In the array version we have from option 100 (which is viewing the queues) to option 999(which is exiting the programme).so I did my test run in a chronological way with more practicality, for example I started the test run with option100, which will display the queue before the customers is added, then I start to mention how the empty queues in 101 is being displayed before adding customers, which displays as all queue 1 ,2 and 3 are empty. Then I went on to add some customers in different cashiers and filled some place with 5 or 6 customers then later again displayed how the viewing queue looks after adding the customer.Also I filled queue of the cashier 1 just to show the output of the option 101, which tells the empty queue are 2 and 3 (which indirectly tells us that the queue 1 is fully filled!).After that I did option 103 part where it removes the customer from a specific location in the specific cashier. Also I used 104 to remove served customer.after that I again showed the view all queues after removing the person and served person to show how they have been removed and the customer in the back of them fills up the empty space. This was shown in the viewing the queue to make sure the reader can easily understand the process. Then its 105, which is arranging the names of the customers in an alphabetical order.Then for option106, which stores the programme data is shown and an output and option 107 to display the text file and to view it. Also for option 108 is is to show how many burgers are left out of 50 burgers from the start.Then I again added 3 more customers so that the burger count will reduce to 10 and below which will give a warning message to show theres only 10 burgers ,5 burgers and eventually 0 burgers.. After that again I added one more customer just to show that this time there will be no more burgers left and it will give an output as no burgers left.Then 109 is to add burgers to the stock and finally 999 is to exit the code.

In the class version, there’s a few alterations and the rest remains the same. Sor for viewing the queues (option 100) and for viewing empty queues (option 101) before adding any customers it gives the similar output except in the viewing queues there’s a waiting list with 10 positions. Then for option 102, which is to add customers, here it asks the first name, last name and no of burgers that person wants.So I added customers and filled all the queue and added an additional customer so a message displays all the queues are filled and that person is added to the waiting line.Then I used (option 100) to show how it has caused the change in the x and o’s and even in the waiting list (which makes the person understand easily whats happening in the code).Then as usual I removed specific person from the queue and served the customer then displayed option 100 to show how the position of x and o’s changed.After that option 105 to display the customers name in alphabetical order but this time its full name with first name and last name.and option 110 is an additional stuff which is to calculate the money made in each queues and ive calculated it based on the customers and the number of burgers have been sold where each burger costs 650rs. And the rest remains the same as array version.

This is how I did my test runs so that I don’t miss any small details and outputs .

Code :

<<paste your code>>

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

ARRAY VERSION (task 1)

```
//Uow no-W1989400
//IIT no- 20222357
//22.06.23

import java.io.*;
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

public class Foodcourt {
    //No.of available burgers
    private static int stockCount = 50;

    //customers name array
    private static String[][] cashierQueues = new String[3][];

    //length of the queue
    private static int[] queueSizes = {2, 3, 5};

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

        for (int i = 0; i < cashierQueues.length; i++) {
            cashierQueues[i] = new String[queueSizes[i]];
        }

        while (true) {
            displayMenu();
            String choice = scanner.nextLine();

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

```java
                removeCustomerFromQueue(scanner);
                break;
            case "104":
            case "PCQ":
                removeServedCustomer();
                break;
            case "105":
            case "VCS":
                viewCustomersSorted();
                break;
            case "106":
            case "SPD":
                storeProgramData();
                break;
            case "107":
            case "LPD":
                loadProgramData();
                break;
            case "108":
            case "STK":
                viewRemainingStock();
                break;
            case "109":
            case "AFS":
                addBurgersToStock(scanner);
                break;
            case "999":
            case "EXT":
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }
    }
}

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

    //printing the cashier
    for (int row = 0; row < 5; row++) {       //reference- stackoverflow
        for (int col = 0; col < 3; col++) {
            if (cashierQueues[col].length > row) {
                if (cashierQueues[col][row] != null) {
                    System.out.print(" O ");
                } else {
                    System.out.print(" X ");
                }
            } else {
                System.out.print("   ");
            }
        }
        System.out.println();
    }
```

```
        System.out.println("X - Not Occupied   O - Occupied");
    }
    //to check if the queue is empty or not
    private static void viewAllEmptyQueues() {
        if (cashierQueues[0][0] == null){
            System.out.println("queue 1 is empty");
        }
        if (cashierQueues[1][0] == null){
            System.out.println("queue 2 is empty");
        }
        if (cashierQueues[2][0] == null){
            System.out.println("queue 3 is empty");
        }
    }

    //adding the customer to the queue
    private static void addCustomerToQueue(Scanner scanner) {
        if (stockCount>0){
            try {
                System.out.print("Enter the cashier number (1, 2, or 3): ");
                int cashierNumber = Integer.parseInt(scanner.nextLine());

                if (cashierNumber >= 1 && cashierNumber <= 3) {
                    int queueIndex = cashierNumber - 1;
                    if (cashierQueues[queueIndex].length > 0 &&
cashierQueues[queueIndex][cashierQueues[queueIndex].length - 1] == null) {
                        System.out.print("Enter the customer name: ");
                        String customerName = scanner.nextLine();

                        for (int i = 0; i < cashierQueues[queueIndex].length;
i++) {
                            if (cashierQueues[queueIndex][i] == null) {
                                cashierQueues[queueIndex][i] = customerName;
                                break;
                            }
                        }

                        updateStockCount();
                        System.out.println("Customer " + customerName + "
added to the queue.");
                    } else {
                        System.out.println("The queue is full. Customer
cannot be added.");
                    }
                } else {
                    System.out.println("Invalid cashier number.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input");
            }
        } else {
            System.out.println("Not enough burgers");
        }
    }
    //removing the customer from the queue
    private static void removeCustomerFromQueue(Scanner scanner) {
        try {
```

```java
            System.out.print("Enter the cashier number (1, 2, or 3): ");
            int cashierNumber = Integer.parseInt(scanner.nextLine());
//reference - w3school.com

            if (cashierNumber >= 1 && cashierNumber <= 3) {
                int queueIndex = cashierNumber - 1;
                if (cashierQueues[queueIndex].length > 0 &&
cashierQueues[queueIndex][0] != null) {
                    System.out.print("Enter the customer position (1 to " +
cashierQueues[queueIndex].length + "): ");
                    int position = Integer.parseInt(scanner.nextLine());

                    if (position >= 1 && position <=
cashierQueues[queueIndex].length && cashierQueues[queueIndex][position - 1]
!= null) {
                        String customerName =
cashierQueues[queueIndex][position - 1];
                        cashierQueues[queueIndex][position - 1] = null;
                        shiftCustomersLeft(cashierQueues[queueIndex],
position - 1);

                        updateStockCount();
                        System.out.println("Customer '" + customerName + "'
removed from the queue.");
                    } else {
                        System.out.println("Invalid customer position.");
                    }
                } else {
                    System.out.println("The queue is empty. No customer to
remove.");
                }
            } else {
                System.out.println("Invalid cashier number.");
            }
        } catch (NumberFormatException e){
            System.out.println("Invalid input");
        }
    }
    //removing the served customers
    private static void removeServedCustomer() {
        for (int i = 0; i < cashierQueues.length; i++) {
            String[] queue = cashierQueues[i];
            if (queue.length > 0 && queue[0] != null) {
                String customerName = queue[0];
                queue[0] = null;
                shiftCustomersLeft(queue, 0);

                updateStockCount();
                System.out.println("Customer '" + customerName + "' removed
from the queue.");
                return;
            }
        }

        System.out.println("All queues are empty. No customer to remove.");
    }
    //to view the customers in the alphabetical order
    private static void viewCustomersSorted() {
        String[] allCustomers = getAllCustomers();
        if (allCustomers.length > 0) {
            bubbleSort(allCustomers);
            System.out.println("Customers Sorted in alphabetical order:");
            for (String customer : allCustomers) {
                System.out.println(customer);
            }
        } else {
            System.out.println("No customers in the queues.");
        }
    }

    public static void bubbleSort(String[] arr) {
        int n = arr.length;
        boolean swapped;

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

            // If no two elements were swapped in the inner loop,
            // the array is already sorted
            if (!swapped)
                break;
        }
    }
    //to store the programme data
    private static void storeProgramData() {
        try (PrintWriter writer = new PrintWriter("program_data.txt")) {
            writer.println(stockCount);
            for (String[] queue : cashierQueues) {
                for (String customer : queue) {
                    if (customer != null) {
                        writer.println(customer);
                    }
                }
            }
            System.out.println("Program data stored successfully.");
        } catch (IOException e) {
            System.out.println("Error storing program data: " +
e.getMessage());
        }
    }
    //loading the programme data
    private static void loadProgramData() {
        try (Scanner scanner = new Scanner(new File("program_data.txt"))) {
            stockCount = Integer.parseInt(scanner.nextLine());
```

```java
            for (String[] queue : cashierQueues) {
                Arrays.fill(queue, null);
            }

            int queueIndex = 0;
            while (scanner.hasNextLine()) {
                String customer = scanner.nextLine();
                if (customer.trim().isEmpty()) {
                    queueIndex++;
                } else {
                    int availableIndex = -1;
                    for (int i = 0; i < cashierQueues[queueIndex].length;
i++) {
                        if (cashierQueues[queueIndex][i] == null) {
                            availableIndex = i;
                            break;
                        }
                    }

                    if (availableIndex != -1) {
                        cashierQueues[queueIndex][availableIndex] = customer;
                    } else {
                        System.out.println("Warning: Some customers could not
be loaded due to insufficient queue space.");
                    }
                }
            }

            System.out.println("Program data loaded successfully.");
        } catch (FileNotFoundException e) {
            System.out.println("Program data file not found. Starting with
default values.");
        }
    }
    //to view the remaining stock of burgers left
    private static void viewRemainingStock() {
        System.out.println("Remaining burgers in stock: " + stockCount);
    }

    //to add the burgers to the stock
    private static void addBurgersToStock(Scanner scanner) {
        try {
            System.out.print("Enter the number of burgers to add: ");
            int quantity = Integer.parseInt(scanner.nextLine());
            stockCount += quantity;
            System.out.println("Burgers added to stock. Total stock count: "
+ stockCount);
        }  catch (Exception e){
            System.out.println("Invalid input");
        }
    }

    private static void updateStockCount() {
        stockCount -= 5;
        if (stockCount <= 10) {
            System.out.println("Warning: Low stock count. Remaining burgers
in stock: " + stockCount);
```

```java
        }
    }

    private static void shiftCustomersLeft(String[] queue, int startIndex) {
        for (int i = startIndex; i < queue.length - 1; i++) {
            queue[i] = queue[i + 1];
        }
        queue[queue.length - 1] = null;
    }

    private static String[] getAllCustomers() {
        int totalCustomers = 0;
        for (String[] queue : cashierQueues) {
            totalCustomers += countNonNullElements(queue);
        }

        String[] allCustomers = new String[totalCustomers];
        int index = 0;
        for (String[] queue : cashierQueues) {
            for (String customer : queue) {
                if (customer != null) {
                    allCustomers[index] = customer;
                    index++;
                }
            }
        }

        return allCustomers;
    }


    private static int countNonNullElements(String[] array) {
        int count = 0;
        for (String element : array) {
            if (element != null) {
                count++;
            }
        }
        return count;
    }
    //display the menu
    private static void displayMenu() {
        System.out.println("**********************************");
        System.out.println("*   Foodies Fave Food Center    *");
        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.print("Enter your choice: ");
    }
}
```

CLASS VERSION (TASK 2 AND 3)

```java
import java.io.*;
import java.util.Arrays;
import java.util.Scanner;

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;
    }
}

class FoodQueue {
    private Customer[] customers;
    private int maxLength;
    private int currentLength;

    public FoodQueue(int maxLength) {
        this.maxLength = maxLength;
        this.currentLength = 0;
        this.customers = new Customer[maxLength];
    }

    public boolean isFull() {
        return currentLength == maxLength;
    }

    public boolean isEmpty() {
        return currentLength == 0;
    }
    public int getCurrentLength() {
        return currentLength;
    }

    public void addCustomer(Customer customer) {
        if (!isFull()) {
            customers[currentLength] = customer;
            currentLength++;
        }
    }

    //removing the customer form the specific location
    public Customer removeCustomer(int position) {
        if (position >= 0 && position < currentLength) {
            Customer customer = customers[position];
            shiftCustomersLeft(position);
            currentLength--;
            return customer;
        }
        return null;
    }

    private void shiftCustomersLeft(int startIndex) {
        for (int i = startIndex; i < currentLength - 1; i++) {
            customers[i] = customers[i + 1];
        }
        customers[currentLength - 1] = null;
    }

    //geting the income of the sold burgers in each queue
    public double getIncome() {
        double income = 0;
        for (Customer customer : customers) {
            if (customer != null) {
                income += customer.getBurgersRequired() * 650;
            }
        }
        return income;
    }

    public Customer[] getCustomers() {
        return customers;
    }
}

public class Foodcourt {
    // No. of available burgers
    private static int stockCount = 50;

    // Customers queues
    private static FoodQueue[] cashierQueues = new FoodQueue[3];
    private static FoodQueue waitingList = new FoodQueue(10);

    // Length of the queue
    private static int[] queueSizes = {2, 3, 5};
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < cashierQueues.length; i++) {
            cashierQueues[i] = new FoodQueue(queueSizes[i]);
        }

        while (true) {
            displayMenu();
            String choice = scanner.nextLine();

            switch (choice) {
                case "100":
                case "VFQ":
                    viewAllQueues();
                    break;
                case "101":
                case "VEQ":
                    viewAllEmptyQueues();
                    break;
                case "102":
                case "ACQ":
                    addCustomerToQueue(scanner);
                    break;
                case "103":
                case "RCQ":
                    removeCustomerFromQueue(scanner);
                    break;
                case "104":
                case "PCQ":
                    removeServedCustomer();
                    break;
                case "105":
                case "VCS":
                    viewCustomersSorted();
                    break;
                case "106":
                case "SPD":
                    storeProgramData();
                    break;
                case "107":
                case "LPD":
                    loadProgramData();
                    break;
                case "108":
                case "STK":
                    viewRemainingStock();
                    break;
                case "109":
                case "AFS":
                    addBurgersToStock(scanner);
                    break;
                case "110":
                case "IFQ":
                    printIncomeOfEachQueue();
                    break;
                case "999":
```

```
                case "EXT":
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
            }
        }
    }

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

        String[] queueOne = new String[2];
        String[] queueTwo = new String[3];
        String[] queueThree = new String[5];
        String temp1 = Arrays.toString(cashierQueues[0].getCustomers());
        String[] customers1 = temp1.replace("[", "").replace("]",
"").split(",");
        int m = 0;
        for (String i : customers1) {
            if (i.trim().equals("null")) {
                queueOne[m] = "  X  ";
            } else {
                queueOne[m] = "  O  ";
            }
            m++;
        }
        String temp2 = Arrays.toString(cashierQueues[1].getCustomers());
        String[] customers2 = temp2.replace("[", "").replace("]",
"").split(",");
        int k = 0;
        for (String i : customers2) {
            if (i.trim().equals("null")) {
                queueTwo[k] = "  X  ";
            } else {
                queueTwo[k] = "  O  ";
            }
            k++;
        }
        String temp3 = Arrays.toString(cashierQueues[2].getCustomers());
        String[] customers3 = temp3.replace("[", "").replace("]",
"").split(",");
        int l = 0;
        for (String i : customers3) {
            if (i.trim().equals("null")) {
                queueThree[l] = "  X  ";
            } else {
                queueThree[l] = "  O  ";
            }
            l++;
        }

        String[][] queueLen = { queueOne, queueTwo, queueThree };
```

```java
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 3; y++) {
                if ((y == 0 && x < 2) || (y == 1 && x < 3) || (y == 2)) {
                    System.out.print(queueLen[y][x]);
                } else {
                    System.out.print("     ");
                }
            }
            System.out.println();
        }

        System.out.print("\nWaiting List - ");
        for (int i=0; i<10;) {
            if (i<waitingList.getCurrentLength()) {
                System.out.print(" O ");
            } else {
                System.out.print(" X ");
            }
            i++;
        }
        System.out.println("\n\nX - Not Occupied   O - Occupied");
    }

    //viewing the empty queues
    private static void viewAllEmptyQueues() {
        for (int i = 0; i < cashierQueues.length; i++) {
            if (cashierQueues[i].isEmpty()) {
                System.out.println("Queue " + (i + 1) + " is empty");
            }
        }
    }

    //Adding customers to the queue
    private static void addCustomerToQueue(Scanner scanner) {
        if(stockCount>0){
            try {
                int queueIndex = getShortestQueueIndex();
                if (queueIndex != -1) {
                    System.out.print("Enter the customer's first name: ");
                    String firstName = scanner.nextLine();

                    System.out.print("Enter the customer's last name: ");
                    String lastName = scanner.nextLine();
                    int burgersRequired;
                    while (true) {
                        System.out.print("Enter the number of burgers
required: ");
                        burgersRequired =
Integer.parseInt(scanner.nextLine());
                        if (burgersRequired <= stockCount) {
                            break;
                        } else {
                            System.out.println("we only have " + stockCount +
" burgers left!");
                        }
                    }
                    Customer customer = new Customer(firstName, lastName, 
burgersRequired);
                    cashierQueues[queueIndex].addCustomer(customer);

                    stockCount -= burgersRequired;
                    if (stockCount <= 10) {
                        System.out.println("Warning: Low stock count. 
Remaining burgers in stock: " + stockCount);
                    }
                    System.out.println("Customer added to the queue.");
                } else {
                    System.out.print("Enter the customer's first name: ");
                    String firstName = scanner.nextLine();

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

                    System.out.print("Enter the number of burgers required: 
");
                    int burgersRequired = 
Integer.parseInt(scanner.nextLine());

                    Customer customer = new Customer(firstName, lastName, 
burgersRequired);
                    waitingList.addCustomer(customer);

                    stockCount -= burgersRequired;
                    if (stockCount <= 10) {
                        System.out.println("Warning: Low stock count. 
Remaining burgers in stock: " + stockCount);
                    }
                    System.out.println("Customer added to the waiting 
list.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input");
            }
        } else {
            System.out.println("No burgers left");
        }
    }

    //getting the shortest queue index
    private static int getShortestQueueIndex() {
        int minLength = Integer.MAX_VALUE;
        int shortestIndex = -1;

        for (int i = 0; i < cashierQueues.length; i++) {
            int currentLength = cashierQueues[i].getCurrentLength();
            if (currentLength < minLength) {
                minLength = currentLength;
                shortestIndex = i;
            }
        }
        if (minLength==2 && shortestIndex==0){
            if (cashierQueues[2].getCurrentLength()==5) {
                shortestIndex=-1;
            } else if (cashierQueues[1].getCurrentLength()==3) {
                shortestIndex=shortestIndex + 2;
            }else if(cashierQueues[0].getCurrentLength()==2){
                shortestIndex=shortestIndex + 1;
            }
        }

        return shortestIndex;
    }

    //Removing the customer from the queue
    private static void removeCustomerFromQueue(Scanner scanner) {
        try {
            System.out.print("Enter the cashier number (1, 2, or 3): ");
            int cashierNumber = Integer.parseInt(scanner.nextLine());

            int queueIndex = cashierNumber - 1;
            if (queueIndex >= 0 && queueIndex < cashierQueues.length) {
                FoodQueue queue = cashierQueues[queueIndex];
                if (!queue.isEmpty()) {
                    System.out.print("Enter the customer position (1 to " +
queue.getCurrentLength() + "): ");
                    int position = Integer.parseInt(scanner.nextLine());

                    Customer customer = queue.removeCustomer(position - 1);
                    if (customer != null) {
                        stockCount += customer.getBurgersRequired();
                        if (stockCount <= 10) {
                            System.out.println("Warning: Low stock count.
Remaining burgers in stock: " + stockCount);
                        }
                        System.out.println("Customer '" +
customer.getFirstName() + " " + customer.getLastName() +
                                "' removed from the queue.");

                        if (!waitingList.isEmpty()) {
                            Customer nextCustomer =
waitingList.removeCustomer(0);

cashierQueues[getShortestQueueIndex()].addCustomer(nextCustomer);
                            stockCount -= nextCustomer.getBurgersRequired();
                            System.out.println("Next customer in the waiting
list placed in the food queue.");
                        } else {
                            System.out.println("All queues and the waiting
list are empty. No customer to remove.");
                        }
                    } else {
                        System.out.println("Invalid customer position.");
                    }
                } else {
                    System.out.println("The queue is empty. No customer to
remove.");
                }
            } else {
                System.out.println("Invalid cashier number.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid input");
        }
    }

    //removing served customer
    private static void removeServedCustomer() {
        for (FoodQueue queue : cashierQueues) {
            if (!queue.isEmpty()) {
                Customer customer = queue.removeCustomer(0);
                if (customer != null) {
                    updateStockCount();
                    System.out.println("Customer '" + customer.getFirstName()
+ " " + customer.getLastName() +
                            "' removed from the queue.");
                    if (!waitingList.isEmpty()) {
                        Customer nextCustomer =
waitingList.removeCustomer(0);

cashierQueues[getShortestQueueIndex()].addCustomer(nextCustomer);
                        stockCount -= nextCustomer.getBurgersRequired();
                        System.out.println("Next customer in the waiting list
placed in the food queue.");
                    } else {
                        System.out.println("All queues and the waiting list
are empty. No customer to remove.");
                    }
                    return;
                }
            }
        }

    }

    //viewing the full name of customers being placed in alphabetical order.
    private static void viewCustomersSorted() {
        Customer[] allCustomers = getAllCustomers();
        if (allCustomers.length > 0) {
            Arrays.sort(allCustomers, (c1, c2) ->
c1.getLastName().compareToIgnoreCase(c2.getLastName()));
            System.out.println("Customers Sorted in alphabetical order:");
            for (Customer customer : allCustomers) {
                System.out.println(customer.getFirstName() +
customer.getLastName());
            }
        } else {
            System.out.println("No customers in the queues.");
        }
    }

    //storing the programme data
    private static void storeProgramData() {
        try (PrintWriter writer = new PrintWriter("program_data.txt")) {
            writer.println(stockCount);
            for (FoodQueue queue : cashierQueues) {
                boolean x = false;
                Customer[] customers = queue.getCustomers();
                for (Customer customer : customers) {
                    if (customer != null) {
                        writer.println(customer.getFirstName());
                        writer.println(customer.getLastName());
                        writer.println(customer.getBurgersRequired());
                        x=true;
                    }
                }
                if (x){
                    writer.println(); /*Empty line to separate queues*/
                }
            }
            System.out.println("Program data stored successfully.");
        } catch (IOException e) {
            System.out.println("Error storing program data: " +
e.getMessage());
        }
    }

    //loading the programme data
    private static void loadProgramData() {
        try (Scanner scanner = new Scanner(new File("program_data.txt"))) {
            stockCount = Integer.parseInt(scanner.nextLine());

            for (int i = 0; i < cashierQueues.length; i++) {
                cashierQueues[i] = new FoodQueue(queueSizes[i]);
            }

            int queueIndex = 0;
            while (scanner.hasNextLine()) {
                String firstName = scanner.nextLine();
                String lastName = scanner.nextLine();
                int burgersRequired = Integer.parseInt(scanner.nextLine());

                if (queueIndex >= 0 && queueIndex < cashierQueues.length) {
                    FoodQueue queue = cashierQueues[queueIndex];
                    Customer customer = new Customer(firstName, lastName,
burgersRequired);
                    if (!queue.isFull()) {
                        queue.addCustomer(customer);
                    } else {
                        System.out.println("Warning: Some customers could not
be loaded due to insufficient queue space.");
                    }
                }

                if (scanner.hasNextLine() &&
scanner.nextLine().trim().isEmpty()) {
                    queueIndex++;
                }
            }

            System.out.println("Program data loaded successfully.");
        } catch (FileNotFoundException e) {
            System.out.println("Program data file not found. Starting with
default values.");
        }
    }

    //viewing the remaining stock
    private static void viewRemainingStock() {
        System.out.println("Remaining burgers in stock: " + stockCount);
    }

    //adding the burgers to the stock
    private static void addBurgersToStock(Scanner scanner) {
        try {
            System.out.print("Enter the number of burgers to add: ");
            int quantity = Integer.parseInt(scanner.nextLine());
            stockCount += quantity;
            System.out.println("Burgers added to stock. Total stock count: "
+ stockCount);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input");
        }
    }

    //printing the income of the sold burgers of each queue
    private static void printIncomeOfEachQueue() {
        for (int i = 0; i < cashierQueues.length; i++) {
            FoodQueue queue = cashierQueues[i];
            double income = queue.getIncome();
            System.out.println("Income of Queue " + (i + 1) + ": " + income);
        }
    }

    //displaying a warning message for the low stock count
    private static void updateStockCount() {
        stockCount -= 5;
        if (stockCount <= 10) {
            System.out.println("Warning: Low stock count. Remaining burgers
in stock: " + stockCount);
        }
    }

    private static Customer[] getAllCustomers() {
        int totalCustomers = 0;
        for (FoodQueue queue : cashierQueues) {
            totalCustomers += queue.getCurrentLength();
        }
        totalCustomers += waitingList.getCurrentLength();

        Customer[] allCustomers = new Customer[totalCustomers];
        int index = 0;
        for (FoodQueue queue : cashierQueues) {
            Customer[] customers = queue.getCustomers();
            for (Customer customer : customers) {
                if (customer != null) {
                    allCustomers[index] = customer;
                    index++;
                }
            }
        }
        Customer[] waitingCustomers = waitingList.getCustomers();
        for (Customer customer : waitingCustomers) {
            if (customer != null) {
                allCustomers[index] = customer;
                index++;
            }
        }

        return allCustomers;
    }

    //Displaying the menu options
    private static void displayMenu() {
        System.out.println("**********************************");
        System.out.println("*   Foodies Fave Food Center    *");
        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: Print the income of each queue");
        System.out.println("999 or EXT: Exit the Program");
        System.out.println("**********************************");
        System.out.print("Enter your choice: ");
    }
}
```

<<END>>