Informatics Institute of Technology
Department of Computing
Software Development II Coursework Report

Module	: 4COSC010C.3: Software Development II (2022)
Module Leader	: Deshan Sumanathilake
Date of submission	: 17th of July 2023
Student ID	: 20221689/w1999416
Student First Name	: Chamidu Himantha
Student Surname	: Samaraweera Arachchige

Name	: Chamidu Himantha
Student ID	: 20221689/w1999416

Test Cases

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 100 or VFQ	Displays ‘empty’ for all queues.	Displays ‘empty’ for all Queues.	Pass
2	Add customer “Jane” to Queue 2 102 or ACQ Enter First Name: Jane Enter Second Name: Foster Burgers Required: 5	Display ‘Jane added to the queue 1 successfully”	Display ‘Jane added to the queue 1 successfully”	Pass
3	101 or VEQ	Empty Queues: Queue 2 Queue 3	Empty Queues: Queue 2 Queue 3	Pass
4	103 or RCQ Remove Customer from Queue Enter Queue Number: 1 Position: 1	Removed Customer: Jane Foster	Removed Customer: Jane Foster	Pass
5	104 or PCQ Remove a served customer	Display Removed customer: “CustomerName”	Display Removed customer: “CustomerName”	Pass
6	105 or VSF view the remaining customers sorted out in alphabetical order	Display Customers Sorted Alphabetically: “Customer Name”	Display Customers Sorted Alphabetically: “Customer Name”	Pass
7	106 or VSF Storing Program onto a File	Stores the current data onto a file	Stores the data onto a file	Pass
8	107 or LPD. Load Program Data from a File	Loads program data from a file	Loads program data from a file	Pass
9	108 or STK View the remaining burger stock when pressed 108	Displays remaining burger Stock	Displays remaining burger Stock	Pass
10	109 or AFS Add burgers to Stock Enter Quantity of Burgers to Add:	Add burgers to the stock	Add burgers to the stock	Pass

11	110 or IF View income of each queue from burgers	Get the income of each queue	Get the income of each queue	Pass
12	999 or EXT Terminating / exiting the program when 999	Displays exited the program and exits the user	Displays exited the program and exits the user	Pass
13	JavaFX GUI	Display the Graphical User Interface	-	Pass

Discussion

The Below mentioned tables include test cases for the array version and the classes version with the necessary application's waiting list. They were selected to cover topics that could relate to handling unsuccessful execution of a method or successful execution. They are selected based on the following justifications.

- The queues current status should be accurately reported if 100 is entered.
- If the number 101 is entered, then only blank queues must be shown.
- If the value 102 is entered, the customer must be added to the chosen queue otherwise, a notice should be printed to indicate that the chosen queue is full.
- If the customer's class number is 102, they should be added to the first queue with the fewest number of clients. Customers must be added to the waiting queue if all queues are already filled.
- If the value 103 is entered, the customer is taken out of the chosen queue after making sure it is empty. In 104, the first client is taken out.
- In the class version, the next customer must be added from the 103 or 104 waiting list, and the necessary amount of burgers must be served and depleted from stock.
- If the number 105 is entered, all of the clients from all queues are sorted and printed in alphabetical order.
- If 106 is entered, information about the program's present condition is saved in a save file and can be retrieved by pressing 107. Any execution error will be noted.
- Burger stock must be printed if the number 108 is input.
- If 109 is entered, the burgers that can be added to the stock is added.
- If the number 110 is entered, the income must be printed for each queue.

Since all the test cases are passed, except the GUI through JavaFX we can conclude that the program is fully functional according to the requirements.

Code :

Main.java

```
import java.util.Scanner;



public class Main {

public static void main(String[] args) {

FoodieFaveQueueManagementSystem queueManagementSystem = new
FoodieFaveQueueManagementSystem();

Scanner scanner = new Scanner(System.in);

System.out.println("\n----Welcome To Foodies Fave Food Center----");



while (true) {

System.out.println("\nMenu:");

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 IF: Print the income of each queue");

System.out.println("999 or EXT: Exit the Program");


System.out.print("Enter your choice: ");

int choice = scanner.nextInt();


switch (choice) {

case 100:

queueManagementSystem.viewAllQueues();

break;

case 101:

queueManagementSystem.viewAllEmptyQueues();

break;

case 102:

System.out.print("Enter first name: ");

String firstName = scanner.next();

System.out.print("Enter last name: ");

String lastName = scanner.next();

System.out.print("Enter number of burgers required: ");

int burgersRequired = scanner.nextInt();

queueManagementSystem.addCustomerToQueue(firstName, lastName, burgersRequired);


break;
```

```
case 103:

System.out.print("Enter queue number: ");

int queueNumber = scanner.nextInt();

System.out.println("Queue 1 position from 1-2");

System.out.println("Queue 2 position from 1-3");

System.out.println("Queue 3 position from 1-5");

System.out.print("Enter position: ");

int position = scanner.nextInt() - 1;

queueManagementSystem.removeCustomerFromQueue(queueNumber, position);

System.out.println();

break;

case 104:

queueManagementSystem.removeServedCustomer();

break;

case 105:

queueManagementSystem.viewCustomersSortedAlphabetically();

break;

case 106:

queueManagementSystem.storeProgramData();

break;

case 107:

queueManagementSystem.loadProgramData();

break;
```

```
case 108:

queueManagementSystem.viewRemainingBurgersStock();

break;

case 109:

System.out.print("Enter quantity of burgers to add: ");

int quantity = scanner.nextInt();

queueManagementSystem.addBurgersToStock(quantity);

break;

case 110:

queueManagementSystem.printIncomeOfEachQueue();

break;

case 999:

System.out.println("Exiting the program...");

System.exit(0);

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

}
```

FoodieFaveQueueManagementSystem.java

```
import java.io.BufferedWriter;
import java.io.File;

import java.io.FileWriter;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;


class FoodieFaveQueueManagementSystem {

private static final int MAX_QUEUE_SIZE_1 = 2;

private static final int MAX_QUEUE_SIZE_2 = 3;

private static final int MAX_QUEUE_SIZE_3 = 5;

private static final int BURGERS_STOCK = 50;

private static final int BURGER_PRICE = 650;


private List<FoodQueue> queue1;

private List<FoodQueue> queue2;

private List<FoodQueue> queue3;

private WaitingListQueue waitingListQueue;

private int burgersRemaining;


public FoodieFaveQueueManagementSystem() {

queue1 = new ArrayList<>();
}

queue2 = new ArrayList<>();

queue3 = new ArrayList<>();

waitingListQueue = new WaitingListQueue();

burgersRemaining = BURGERS_STOCK;

}


public void viewAllQueues() {

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

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

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

printQueue(queue1, 2);

printQueue(queue2, 3);

printQueue(queue3, 5);

}


private void printQueue(List<FoodQueue> queue, int maxQueueSize) {

for (int i = 0; i < maxQueueSize; i++) {

if (i < queue.size()) {

System.out.print("O ");

} else {

System.out.print("X ");

}

}
```

```
System.out.println();

}

public void viewAllEmptyQueues() {

System.out.println("Empty Queues:");

if (queue1.isEmpty()) {

System.out.println("Queue 1");

}

if (queue2.isEmpty()) {

System.out.println("Queue 2");

}

if (queue3.isEmpty()) {

System.out.println("Queue 3");

}

}


public void addCustomerToQueue(String firstName, String lastName, int burgersRequired) {

if (burgersRequired < 1 || burgersRequired > 5) {

System.out.println("Invalid number of burgers. Please enter a value between 1 and 5.");

return;

}

if (burgersRequired > burgersRemaining) {

System.out.println("Insufficient stock: " + burgersRequired);

System.out.println("Customer cannot be added to the queue.");
System.out.println("Maximum stock: 50");
return;
}

int minQueueSize = Math.min(queue1.size(), Math.min(queue2.size(), queue3.size()));

if (queue1.size() == minQueueSize) {
if (queue1.size() < MAX_QUEUE_SIZE_1) {
queue1.add(new FoodQueue(firstName, lastName, burgersRequired));
System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 1.");
} else if (queue2.size() < MAX_QUEUE_SIZE_2) {
queue2.add(new FoodQueue(firstName, lastName, burgersRequired));
System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 2.");
} else if (queue3.size() < MAX_QUEUE_SIZE_3) {
queue3.add(new FoodQueue(firstName, lastName, burgersRequired));
System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 3.");
} else {
addToWaitingList(firstName, lastName, burgersRequired);
}
} else if (queue2.size() == minQueueSize) {
if (queue2.size() < MAX_QUEUE_SIZE_2) {
queue2.add(new FoodQueue(firstName, lastName, burgersRequired));
System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 2.");
} else if (queue3.size() < MAX_QUEUE_SIZE_3) {

queue3.add(new FoodQueue(firstName, lastName, burgersRequired));

System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 3.");

} else {

addToWaitingList("\n" + firstName, lastName, burgersRequired);

}

} else if (queue3.size() == minQueueSize) {

if (queue3.size() < MAX_QUEUE_SIZE_3) {

queue3.add(new FoodQueue(firstName, lastName, burgersRequired));

System.out.println("\n" + firstName + " " + lastName + " successfully added to Queue 3.");

} else {

addToWaitingList(firstName, lastName, burgersRequired);

}

} else {

addToWaitingList(firstName, lastName, burgersRequired);

}

updateBurgersStock(burgersRequired);

}




private void addToWaitingList(String firstName, String lastName, int burgersRequired) {

FoodQueue foodQueue = new FoodQueue(firstName, lastName, burgersRequired);

if (waitingListQueue.isFull()) {
```

```
System.out.println("\nAll queues and waiting list are full. Customer cannot be added.");

} else {

waitingListQueue.enqueue(foodQueue);

System.out.println("Added customer to the waiting list.");

}

}



public void removeCustomerFromQueue(int queueNumber, int position) {

List<FoodQueue> queue = getQueueByNumber(queueNumber);

if (queue != null) {

if (position >= 0 && position < queue.size()) {

FoodQueue removedCustomer = queue.remove(position );

System.out.println("Removed customer: " + removedCustomer.getFirstName() + " " +
removedCustomer.getLastName());

if (!waitingListQueue.isEmpty()) {

FoodQueue nextCustomer = waitingListQueue.dequeue();

queue.add(nextCustomer);

System.out.println("Next customer in the waiting list added to the queue.");

}

} else {

System.out.println("Invalid position. Cannot remove customer.");

}

} else {

System.out.println("Invalid queue number. Cannot remove customer.");
```

```
}

}



public void removeServedCustomer() {

if (!queue1.isEmpty()) {

FoodQueue removedCustomer = queue1.remove(0);

System.out.println("Removed served customer from Queue 1: " +
removedCustomer.getFirstName() + " " + removedCustomer.getLastName());

if (!waitingListQueue.isEmpty()) {

FoodQueue nextCustomer = waitingListQueue.dequeue();

queue1.add(nextCustomer);

System.out.println("Next customer in the waiting list added to Queue 1." + nextCustomer);

}

} else if (!queue2.isEmpty()) {

FoodQueue removedCustomer = queue2.remove(0);

System.out.println("Removed served customer from Queue 2: " +
removedCustomer.getFirstName() + " " + removedCustomer.getLastName());

if (!waitingListQueue.isEmpty()) {

FoodQueue nextCustomer = waitingListQueue.dequeue();

queue2.add(nextCustomer);

System.out.println("Next customer in the waiting list added to Queue 2:" + nextCustomer);

}

} else if (!queue3.isEmpty()) {

FoodQueue removedCustomer = queue3.remove(0);
```

```
System.out.println("Removed served customer from Queue 3: " +
removedCustomer.getFirstName() + " " + removedCustomer.getLastName());

if (!waitingListQueue.isEmpty()) {

FoodQueue nextCustomer = waitingListQueue.dequeue();

queue3.add(nextCustomer);

System.out.println("Next customer in the waiting list added to Queue 3." + nextCustomer);

}

} else {

System.out.println("All queues are empty. Cannot remove served customer.");

}

}



public void viewCustomersSortedAlphabetically() {

List<Customer> customers = new ArrayList<>();

for (FoodQueue foodQueue : queue1) {

customers.add(new Customer(foodQueue.getFirstName(), foodQueue.getLastName()));

}

for (FoodQueue foodQueue : queue2) {

customers.add(new Customer(foodQueue.getFirstName(), foodQueue.getLastName()));

}

for (FoodQueue foodQueue : queue3) {

customers.add(new Customer(foodQueue.getFirstName(), foodQueue.getLastName()));

}

customers.sort((c1, c2) -> c1.getLastName().compareToIgnoreCase(c2.getLastName()));
```

```
System.out.println("Customers sorted alphabetically:");

for (Customer customer : customers) {

System.out.println(customer.getFirstName() + " " + customer.getLastName());

}

}




public void storeProgramData() {

try {

File file = new File("FoodieFave.txt");

FileWriter fileWriter = new FileWriter(file);

BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);



for (FoodQueue foodQueue : queue1) {

String line = "\nFrom QUEUE 1 \nFirstname: " + foodQueue.getFirstName() + " Lastname: " +
foodQueue.getLastName() + " Burgers Required " + foodQueue.getBurgersRequired();

bufferedWriter.write(line);

bufferedWriter.newLine();

}

for (FoodQueue foodQueue : queue2) {

String line = "\nFrom QUEUE 2 \nFirstname: " + foodQueue.getFirstName() + " Lastname: " +
foodQueue.getLastName() + " Burgers Required " + foodQueue.getBurgersRequired();

bufferedWriter.write(line);

bufferedWriter.newLine();
}

for (FoodQueue foodQueue : queue3) {

String line = "\nFrom QUEUE 3 \nFirstname: " + foodQueue.getFirstName() + " Lastname: " +
foodQueue.getLastName() + " Burgers Required " + foodQueue.getBurgersRequired();

bufferedWriter.write(line);

bufferedWriter.newLine();

}



bufferedWriter.close();

System.out.println("Successfully wrote the customer data to the file (FoodieFave.txt).");

} catch (IOException e) {

System.out.println("An error occurred while writing the file.");

e.printStackTrace();

}

}





/**

* Loads the program data from a file.

*/

public void loadProgramData() {

try {

File file = new File("FoodieFave.txt");
```

```
FileReader fileReader = new FileReader(file);

BufferedReader bufferedReader = new BufferedReader(fileReader);


String line;

while ((line = bufferedReader.readLine()) != null) {

System.out.println(line);

}


bufferedReader.close();

System.out.println("Data loaded from the file (FoodieFave.txt).");

} catch (IOException e) {

System.out.println("Error: File not found.");

e.printStackTrace();

}

}





public void viewRemainingBurgersStock() {

System.out.println("Remaining burgers stock: " + burgersRemaining);

}
```

```
public void addBurgersToStock(int quantity) {

if (burgersRemaining + quantity <= BURGERS_STOCK) {

burgersRemaining += quantity;

System.out.println(quantity + " burgers added to stock. Total burgers: " + burgersRemaining);

} else {

System.out.println("Cannot add burgers. Maximum stock limit is 50.");

System.out.println("Current Burger Stock : " + burgersRemaining);

}

}



private void updateBurgersStock(int burgersRequired) {

if (burgersRemaining - burgersRequired >= 0) {

burgersRemaining -= burgersRequired;

if (burgersRemaining <= 10) {

System.out.println("Warning: Low burgers stock. Remaining burgers: " + burgersRemaining);

}

} else {

System.out.println(" Insufficient burgers stock:" + burgersRequired);

System.out.println(" Customer cannot be added to the queue");

}

}
```

```
}

private List<FoodQueue> getQueueByNumber(int queueNumber) {
switch (queueNumber) {
case 1:
return queue1;
case 2:
return queue2;
case 3:
return queue3;
default:
return null;
}
}

public void printIncomeOfEachQueue() {
int incomeQueue1 = calculateQueueIncome(queue1);
int incomeQueue2 = calculateQueueIncome(queue2);
int incomeQueue3 = calculateQueueIncome(queue3);

System.out.println("Income of each queue:");
System.out.println("Queue 1: " + incomeQueue1);
System.out.println("Queue 2: " + incomeQueue2);
System.out.println("Queue 3: " + incomeQueue3);
}
```

```
private int calculateQueueIncome(List<FoodQueue> queue) {

int totalBurgersRequired = 0;

for (FoodQueue foodQueue : queue) {

totalBurgersRequired += foodQueue.getBurgersRequired();

}

return totalBurgersRequired * BURGER_PRICE;

}

}
```

Customer.java

```
class Customer {

private String firstName;

private String lastName;


public Customer(String firstName, String lastName) {

this.firstName = firstName;

this.lastName = lastName;
}

public String getFirstName() {
return firstName;
}

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

FoodQueue.java

```
class FoodQueue {
private static final int MAX_WAITING_LIST_SIZE = 5;

private String firstName;
private String lastName;
private int burgersRequired;

public FoodQueue(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;

}

}
```

WaitingListQueue.java

```
class WaitingListQueue {

private static final int MAX_SIZE = 5;


private FoodQueue[] queue;

private int front;

private int rear;

private int size;


public WaitingListQueue() {

queue = new FoodQueue[MAX_SIZE];

front = 0;

rear = -1;

size = 0;

}


public boolean isFull() {

return size == MAX_SIZE;

}


public boolean isEmpty() {
```

```
return size == 0;

}


public void enqueue(FoodQueue foodQueue) {

if (isFull()) {

System.out.println("Waiting list is full. Customer cannot be added.");

return;

}

rear = (rear + 1) % MAX_SIZE;

queue[rear] = foodQueue;

size++;

}


public FoodQueue dequeue() {

if (isEmpty()) {

System.out.println("Waiting list is empty. No customer to remove.");

return null;

}

FoodQueue removedCustomer = queue[front];

front = (front + 1) % MAX_SIZE;

size--;

return removedCustomer;

}
```

```
}
```

<<END>>