UOW-logo

Informatics Institute of Technology

Department of Computing

Software Development II Coursework Report

Module : 4COSC010C.3: Software Development II (2023)

Module Leader : Mr. Deshan Sumanathilaka

Date of submission : 17/07/2023

Student ID : <20220644 > / <w1998865 >

Student First Name : Sarah

Student Surname : Rahim

"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 : Sarah Rahim

Student ID : 20220644

**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 a Queue with 102 or ACQ  Enter Customer’s First Name: Jane  Enter Customer’s Last Name: B  Enter the number of burgers required:2 | Display ‘Customer Jane B added to queue 1.’ | Display ‘Customer Jane B added to queue 1.’ | Pass |
| 3 | Remove Customer from a Queue, 103 or RCQ  Enter the queue number: 1  Customers in Queue 1:   1. Jane B   Enter the customer number to remove: 1 | Display ‘Customer Jane B removed from queue 1.’ | Display ‘Customer Jane B removed from queue 1.’ | Pass |
| 4 | Remove a served Customer, 104 or PCQ  Enter the queue number: 3  Customers in Queue 3:  1. Zainab Rahim  2. Savinu Hasalanka  3. Paramie Jayakody  4. Tisal Thenuwara  5. Jei Skandarajah  Enter the customer number to remove:4 | Display ‘Customer Tisal Thenuwara removed from queue 3.  2 burgers added back to the stock.’ | Display ‘Customer Tisal Thenuwara removed from queue 3.  2 burgers added back to the stock.’ | Pass |
| 5 | View all customers sorted in alphabetic order, 105 or VCS | Display ‘Customer sorted by first name’ | Display ‘Customers sorted by first name:  Customer: Asiri Ekanayake, Burgers Required: 2  Customer: Diadri Weerasekara, Burgers Required: 2  Customer: Jei Skandarajah, Burgers Required: 2  Customer: Jingy Dodo, Burgers Required: 2  Customer: Paramie Jayakody, Burgers Required: 10’ | Pass |
| 6 | Store program data, 106 or SPD | Display ‘Program data stored successfully.’ | Display ‘Program data stored successfully.’ | Pass |
| 7 | Load program data, 107 or LPD | Display ‘Program data loaded successfully.’ | Display ‘Program data loaded successfully.’ | Pass |
| 8 | View Remaining burger stock, 108 or STK | Display remaining stock | Display ‘Remaining burger stock: 14’ | Pass |
| 9 | Add Burgers to stock, 109 or AFS  Enter the number of burgers to add to stock: 12 | Display ‘12 burgers added to stock.  Remaining burger stock: 26’ | Display ‘12 burgers added to stock.  Remaining burger stock: 26’ | Pass |
| 10 | View income, 110 or IFQ | Display ‘Total income: $15600’ | Display ‘Total income: $15600’ | Pass |
| 11 | Exit program, 999 or EXT | Display ‘Exiting the program...’ | Display ‘Exiting the program...’ | Pass |

**Discussion**

The test cases consisted of all the functions in the program, namely from 100 or VFQ to 999 or EXT. Each case was carefully looked through in the coursework specifications and successfully carried out in each function.

For example, in task 3 of the specifications, a waiting list had to be added to the program once the queues were full. Therefore, after filling all queues with customers, the waiting list had to be generated, thereby carrying the remaining customers. (This applied to Functions “102 or ACQ” “104 or PCQ” where, when a served customer was removed, a customer in the waiting list would be added automatically to the vacant queue.)

**Code :**

**FoodQueue.class:**

import java.util.\*;  
import java.io.\*;  
import java.io.Serializable;  
  
public class FoodQueue {  
 private static final int *MAX\_BURGERS* = 50;  
 private static final int *WARNING\_STOCK* = 10;  
 private static final int[] *MAX\_CUSTOMERS\_PER\_QUEUE* = {2, 3, 5};  
 private static final int *BURGER\_PRICE* = 650;  
 private static final int *WAITING\_LIST\_SIZE* = 10;  
 private static Customer[] *waitingList*;  
 private static int *waitingListFront*;  
 private static int *waitingListRear*;  
 private static int *waitingListCount*;  
  
  
 private static Queue<Customer>[] *queues*;  
 private static int *burgerStock*;  
  
 @SuppressWarnings("unchecked")  
 public static void main(String[] args) {  
 *initializeQueues*();  
 *burgerStock* = *MAX\_BURGERS*;  
  
 Scanner scanner = new Scanner(System.*in*);  
 String choice;  
 do {  
 *displayMenu*();  
 choice=scanner.nextLine().toUpperCase();  
  
 try{  
 int choiceNum=Integer.*parseInt*(choice);  
 *processNumericChoice*(choiceNum);  
 }catch(NumberFormatException e){  
 *processStringChoice*(choice);  
 }  
  
 switch (choice) {  
 case "100":  
 case "VFQ":  
 *viewAllQueues*();  
 break;  
 case "101":  
 case "VEQ":  
 *viewEmptyQueues*();  
 break;  
 case "102":  
 case "ACQ":  
 *addCustomerToQueue*();  
 break;  
 case "103":  
 case "RCQ":  
 *removeCustomerFromQueue*(scanner);  
 break;  
 case "104":  
 case "PCQ":  
 *removeServedCustomer*(scanner);  
 break;  
 case "105":  
 case "VCS":  
 *viewCustomersSorted*();  
 break;  
 case "106":  
 case "SPD":  
 *storeProgramData*();  
 break;  
 case "107":  
 case "LPD":  
 *loadProgramData*();  
 break;  
 case "108":  
 case "STK":  
 *viewRemainingBurgerStock*();  
 break;  
 case "109":  
 case "AFS":  
 *addBurgersToStock*(scanner);  
 break;  
 case "110":  
 case "IFQ":  
 *viewIncome*();  
 break;  
 case "999":  
 case "EXT":  
 System.*out*.println("Exiting the program...");  
 break;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 } while (!choice.equals("999") && !choice.equals("EXT"));  
  
 scanner.close();  
 }  
  
 @SuppressWarnings("unchecked")  
 private static void initializeQueues() {  
 *waitingList* = new Customer[*WAITING\_LIST\_SIZE*];  
 *waitingListFront* = 0;  
 *waitingListRear* = -1;  
 *waitingListCount* = 0;  
 *queues* = new Queue[*MAX\_CUSTOMERS\_PER\_QUEUE*.length];  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 *queues*[i] = new LinkedList<>();  
 }  
  
 try{  
 FileReader fileReader=new FileReader("customerData.txt");  
 BufferedReader bufferedReader=new BufferedReader(fileReader);  
  
 String line;  
 while ((line = bufferedReader.readLine()) != null) {  
 String[] data = line.split(",");  
 if (data.length == 3) {  
 String firstName = data[0];  
 String lastName = data[1];  
 int burgersRequired = Integer.*parseInt*(data[2]);  
 Customer customer = new Customer(firstName, lastName, burgersRequired);  
  
 int queueIndex = *findSmallestQueue*();  
 if (queueIndex != -1) {  
 *queues*[queueIndex].add(customer);  
 System.*out*.println("Customer " + customer.getFirstName() + " " + customer.getLastName() + " added to queue " + (queueIndex + 1) + ".");  
 *burgerStock* -= burgersRequired;  
 } else {  
 if (*waitingListCount* < *WAITING\_LIST\_SIZE*) {  
 *waitingListRear* = (*waitingListRear* + 1) % *WAITING\_LIST\_SIZE*;  
 *waitingList*[*waitingListRear*] = customer;  
 *waitingListCount*++;  
 System.*out*.println("Customer " + customer.getFirstName() + " " + customer.getLastName() + " added to the waiting list.");  
 *burgerStock* -= burgersRequired;  
 } else {  
 System.*out*.println("All queues and the waiting list are full. Customer " + customer.getFirstName() + " " + customer.getLastName() + " could not be added.");  
 }  
 }  
 }  
 }  
  
 bufferedReader.close();  
 } catch (IOException e) {  
 System.*out*.println("Error reading customer data from file: " + e.getMessage());  
 }  
 }  
  
  
 private static void processNumericChoice(int choice) {  
 // Placeholder implementation for processing numeric choice  
 System.*out*.println("Processing numeric choice: " + choice);  
 }  
  
 private static void processStringChoice(String choice) {  
 // Placeholder implementation for processing string choice  
 System.*out*.println("Processing string choice: " + choice);  
 }  
  
 private static void displayMenu() {  
 System.*out*.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println(" \* Cashiers \*");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("X X X");  
 System.*out*.println("X X X");  
 System.*out*.println(" X X");  
 System.*out*.println(" X");  
 System.*out*.println(" X");  
 System.*out*.println("\nMenu Options:");  
 System.*out*.println("100 or VFQ: View all customers in queues");  
 System.*out*.println("101 or VEQ: View empty queues");  
 System.*out*.println("102 or ACQ: Add customer to a queue");  
 System.*out*.println("103 or RCQ: Remove customer from a queue");  
 System.*out*.println("104 or PCQ: Remove a served customer");  
 System.*out*.println("105 or VCS: View all customers sorted in alphabetic order");  
 System.*out*.println("106 or SPD: Store program data");  
 System.*out*.println("107 or LPD: Load program data");  
 System.*out*.println("108 or STK: View remaining burger stock");  
 System.*out*.println("109 or AFS: Add burgers to stock");  
 System.*out*.println("110 or IFQ: View income");  
 System.*out*.println("999 or EXT: Exit program");  
 System.*out*.println("\nEnter your choice:");  
 }  
  
 private static void viewAllQueues() {  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 System.*out*.println("\nQueue " + (i + 1) + ":");  
 if (*queues*[i].isEmpty()) {  
 System.*out*.println("No customers in the queue.");  
 } else {  
 for (Customer customer : *queues*[i]) {  
 System.*out*.println(customer);  
 }  
 }  
 }  
 }  
  
 private static void viewEmptyQueues() {  
 boolean foundEmptyQueue = false;  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 if (*queues*[i].isEmpty()) {  
 System.*out*.println("Queue " + (i + 1) + " is empty.");  
 foundEmptyQueue = true;  
 }  
 }  
 if (!foundEmptyQueue) {  
 System.*out*.println("No empty queues found.");  
 }  
 }  
  
 private static void addCustomerToQueue() {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter customer's first name: ");  
 String firstName = scanner.nextLine();  
  
 System.*out*.print("Enter customer's last name: ");  
 String lastName = scanner.nextLine();  
  
 System.*out*.print("Enter the number of burgers required: ");  
 int burgersRequired = scanner.nextInt();  
  
 Customer customer = new Customer(firstName, lastName, burgersRequired);  
  
 int queueIndex = *findSmallestQueue*();  
 if (queueIndex != -1) {  
 *queues*[queueIndex].add(customer);  
 System.*out*.println("Customer " + customer.getFirstName() + " " + customer.getLastName() + " added to queue " + (queueIndex + 1) + ".");  
 *burgerStock* -= burgersRequired; // Subtract from burger stock  
 } else {  
 if (*waitingListCount* < *WAITING\_LIST\_SIZE*) {  
 *waitingListRear* = (*waitingListRear* + 1) % *WAITING\_LIST\_SIZE*;  
 *waitingList*[*waitingListRear*] = customer;  
 *waitingListCount*++;  
 System.*out*.println("Customer " + customer.getFirstName() + " " + customer.getLastName() + " added to the waiting list.");  
 *burgerStock* -= burgersRequired; // Subtract from burger stock  
 } else {  
 System.*out*.println("All queues and the waiting list are full. Customer could not be added.");  
 }  
 }  
 }  
  
  
 private static int findSmallestQueue() {  
 int minSize = Integer.*MAX\_VALUE*;  
 int minIndex = -1;  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 if (*queues*[i].size() < minSize && *queues*[i].size() < *MAX\_CUSTOMERS\_PER\_QUEUE*[i]) {  
 minSize = *queues*[i].size();  
 minIndex = i;  
 }  
 }  
 return minIndex;  
 }  
  
 private static void removeCustomerFromQueue(Scanner scanner) {  
 System.*out*.print("Enter the queue number: ");  
 int queueNumber = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 if (queueNumber >= 1 && queueNumber <= *MAX\_CUSTOMERS\_PER\_QUEUE*.length) {  
 Queue<Customer> queue = *queues*[queueNumber - 1];  
  
 if (queue.isEmpty()) {  
 System.*out*.println("No customers in the queue.");  
 } else {  
 System.*out*.println("Customers in Queue " + queueNumber + ":");  
 int count = 1;  
  
 for (Customer customer : queue) {  
 System.*out*.println(count + ". " + customer.getFirstName() + " " + customer.getLastName());  
 count++;  
 }  
  
 System.*out*.print("Enter the customer number to remove: ");  
 int customerNumber = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 if (customerNumber >= 1 && customerNumber <= queue.size()) {  
 Customer removedCustomer = null;  
 count = 1;  
 Iterator<Customer> iterator = queue.iterator();  
  
 while (iterator.hasNext()) {  
 Customer customer = iterator.next();  
 if (count == customerNumber) {  
 removedCustomer = customer;  
 iterator.remove();  
 break;  
 }  
 count++;  
 }  
  
 if (removedCustomer != null) {  
 System.*out*.println("Customer " + removedCustomer.getFirstName() + " " + removedCustomer.getLastName() +  
 " removed from queue " + queueNumber + ".");  
 } else {  
 System.*out*.println("Invalid customer number.");  
 }  
 } else {  
 System.*out*.println("Invalid customer number.");  
 }  
 }  
 } else {  
 System.*out*.println("Invalid queue number.");  
 }  
 }  
  
  
  
  
 private static void removeServedCustomer(Scanner scanner) {  
 System.*out*.print("Enter the queue number: ");  
 int queueNumber = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 if (queueNumber >= 1 && queueNumber <= *MAX\_CUSTOMERS\_PER\_QUEUE*.length) {  
 Queue<Customer> queue = *queues*[queueNumber - 1];  
  
 if (queue.isEmpty()) {  
 System.*out*.println("No customers in the queue.");  
 } else {  
 System.*out*.println("Customers in Queue " + queueNumber + ":");  
 int count = 1;  
  
 for (Customer customer : queue) {  
 System.*out*.println(count + ". " + customer.getFirstName() + " " + customer.getLastName());  
 count++;  
 }  
  
 System.*out*.print("Enter the customer number to remove: ");  
 int customerNumber = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 if (customerNumber >= 1 && customerNumber <= queue.size()) {  
 Customer removedCustomer = null;  
 count = 1;  
 Iterator<Customer> iterator = queue.iterator();  
 while (iterator.hasNext()) {  
 removedCustomer = iterator.next();  
 if (count == customerNumber) {  
 iterator.remove();  
 System.*out*.println("Customer " + removedCustomer.getFirstName() + " " + removedCustomer.getLastName() + " removed from queue " + queueNumber + ".");  
 break;  
 }  
 count++;  
 }  
  
 if (removedCustomer != null) {  
 *burgerStock* += removedCustomer.getBurgersRequired(); // Add to burger stock  
 System.*out*.println(removedCustomer.getBurgersRequired() + " burgers added back to the stock.");  
  
 // Check if there are customers in the waiting list  
 if (*waitingListCount* > 0) {  
 Customer nextCustomer = *waitingList*[*waitingListFront*];  
 *waitingListFront* = (*waitingListFront* + 1) % *WAITING\_LIST\_SIZE*;  
 *waitingListCount*--;  
  
 *queues*[queueNumber - 1].add(nextCustomer);  
 System.*out*.println("Customer " + nextCustomer.getFirstName() + " " + nextCustomer.getLastName() + " added to queue " + queueNumber + " from the waiting list.");  
 }  
 }  
 } else {  
 System.*out*.println("Invalid customer number.");  
 }  
 }  
 } else {  
 System.*out*.println("Invalid queue number.");  
 }  
 }  
  
  
 private static void viewCustomersSorted() {  
 List<Customer> allCustomers = new ArrayList<>();  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 allCustomers.addAll(*queues*[i]);  
 }  
 if (allCustomers.isEmpty()) {  
 System.*out*.println("No customers in the queues.");  
 } else {  
 Collections.*sort*(allCustomers, Comparator.*comparing*(Customer::getFirstName, String.*CASE\_INSENSITIVE\_ORDER*));  
 System.*out*.println("Customers sorted by first name:");  
 for (Customer customer : allCustomers) {  
 System.*out*.println(customer);  
 }  
 }  
 }  
  
  
  
 private static void storeProgramData() {  
 try {  
 FileWriter fileWriter = new FileWriter("programData.txt");  
 BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);  
  
 // Store queue details  
 for (int i = 0; i < *MAX\_CUSTOMERS\_PER\_QUEUE*.length; i++) {  
 Queue<Customer> queue = *queues*[i];  
 bufferedWriter.write("Queue " + (i + 1) + ":\n");  
  
 if (queue.isEmpty()) {  
 bufferedWriter.write("No customers in the queue.\n");  
 } else {  
 for (Customer customer : queue) {  
 bufferedWriter.write("Customer: " + customer.getFirstName() + " " + customer.getLastName() +  
 ", Burgers Required: " + customer.getBurgersRequired() + "\n");  
 }  
 }  
  
 bufferedWriter.write("\n");  
 }  
  
 // Store waiting list details  
 bufferedWriter.write("Waiting List:\n");  
 if (*waitingListCount* == 0) {  
 bufferedWriter.write("No customers in the waiting list.\n");  
 } else {  
 for (int i = *waitingListFront*; i <= *waitingListRear*; i++) {  
 int index = i % *WAITING\_LIST\_SIZE*;  
 Customer customer = *waitingList*[index];  
 bufferedWriter.write("Customer: " + customer.getFirstName() + " " + customer.getLastName() +  
 ", Burgers Required: " + customer.getBurgersRequired() + "\n");  
 }  
 }  
  
 bufferedWriter.close();  
 fileWriter.close();  
 System.*out*.println("Program data stored successfully.");  
 } catch (IOException e) {  
 System.*out*.println("Error storing program data: " + e.getMessage());  
 }  
 }  
  
 private static void loadProgramData() {  
 try {  
 FileReader fileReader = new FileReader("programData.txt");  
 BufferedReader bufferedReader = new BufferedReader(fileReader);  
  
 String line;  
 while ((line = bufferedReader.readLine()) != null) {  
 System.*out*.println(line);  
 }  
  
 bufferedReader.close();  
 fileReader.close();  
 System.*out*.println("Program data loaded successfully.");  
 } catch (IOException e) {  
 System.*out*.println("Error loading program data: " + e.getMessage());  
 }  
 }  
  
  
 private static void viewRemainingBurgerStock() {  
 if(*burgerStock*<=*WARNING\_STOCK*){  
 System.*out*.println("Remaining burger stock: " + *burgerStock*);  
 System.*out*.println("WARNING: Burger stock is running low!");  
 } else {  
 System.*out*.println("Remaining burger stock: " + *burgerStock*);  
 }  
 }  
  
 private static void addBurgersToStock(Scanner scanner) {  
 System.*out*.print("Enter the number of burgers to add to stock: ");  
 int burgersToAdd = scanner.nextInt();  
 *burgerStock* += burgersToAdd;  
 System.*out*.println(burgersToAdd + " burgers added to stock.");  
 *viewRemainingBurgerStock*(); //Display the total burger stock  
 return; //Exit the method  
 }  
  
 private static void viewIncome() {  
 int totalIncome = (*MAX\_BURGERS* - *burgerStock*) \* *BURGER\_PRICE*;  
 System.*out*.println("Total income: $" + totalIncome);  
 }  
}

**Customer class:**

import java.io.Serializable;  
public class Customer implements Serializable {  
 private final String firstName;  
 private final String lastName;  
 private final 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;  
 }  
  
 @Override  
 public String toString() {  
 return "Customer: " + firstName + " " + lastName + ", Burgers Required: " + burgersRequired;  
 }  
}