UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Department of Computing

Software Development II Coursework Report

Module	: 4COSC010C.3: Software Development II
Module Leader	: Deshan Sumanathilaka
Date of submission	: 08/08/2022
Student ID	: 20211364/ w1898951
Student First Name	: Akindu
Student Surname	: Karunaratne

"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 : Akindu Dinan Manamperi Karunaratne

Student ID : 20211364 / w1898951

Test Cases

Task 1				
	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
1	Fuel Queue Initialized Correctly. After program starts, 100 or VFQ	Displays 'empty' for all queues.	Displays 'empty' for all Queues.	Pass
2	Add customer "John" to Queue 1 102 or ACQ Enter fuel queue (1,2 or 3): 1 Enter Customer name: John	Display "John was successfully added to fuel queue 1"	Display "John was successfully added to fuel queue 1"	Pass
3	Enter 'A' to add another customer or press enter to return to the menu: Pressed enter.	Displays menu options	Displays menu options	Pass
4	View all empty queues by entering 101 or VEQ	All queues display empty except for queue 1 slot 1	All queues display empty except for queue 1 slot 1	Pass
5	Add customer "Jack" to Queue 1 102 or ACQ Enter fuel queue (1,2 or 3): 1 Enter Customer name: Jack	Display "Jack was successfully added to fuel queue 1"	Display "Jack was successfully added to fuel queue 1"	Pass
6	Enter 'A' to add another customer or press enter to return to the menu: A	Displays "Enter fuel queue (1,2 or 3):" to add another customer	Displays "Enter fuel queue (1,2 or 3):" to add another customer	Pass
7	Add customer "Kimi" to Queue 2 Enter fuel queue (1,2 or 3): 2 Enter Customer name: Kimi	Display "Kimi was successfully added to fuel queue 2"	Display "Kimi was successfully added to fuel queue 2"	Pass
8	After selecting 'A' add customer "Sebastian" to queue 3 Enter fuel queue (1,2 or 3): 3 Enter Customer name: Sebastian	Display "Sebastian was successfully added to fuel queue 3"	Display "Sebastian was successfully added to fuel queue 3"	Pass
9	After selecting 'A' add customer "Akindu" to queue 3 Enter fuel queue (1,2 or 3): 3 Enter Customer name: Akindu	Display "Akindu was successfully added to fuel queue 3"	Display "Akindu was successfully added to fuel queue 3"	Pass
10	Add customer "dinan" to queue 2 Enter fuel queue (1,2 or 3): 2 Enter Customer name: dinan	Display "dinan was successfully added to fuel queue 2"	Display "dinan was successfully added to fuel queue 2"	Pass

11	Exits to menu by pressing enter key. View all queues by entering 100 or VFQ	Displays all queues with first 2 slots of each queue occupied by added customer names	Displays all queues with first 2 slots of each queue occupied by added customer names	Pass
12	Entering 105 or VCS to view customers sorted alphabetically.	Displays Fuel Queue 1 Jack John Fuel Queue 2 dinan Kimi Fuel Queue 3 Akindu Sebastian	Displays Fuel Queue 1 Jack John Fuel Queue 2 dinan Kimi Fuel Queue 3 Akindu Sebastian	Pass
13	Entering 106 or SPD to store program data	Displays “File Updated Successfully!”	Displays “File Updated Successfully!”	Pass
14	Entering 999 or EXT to exit program	Displays “Exiting the Program...” and exits program	Displays “Exiting the Program...” and exits program	Pass
15	Re-run program (Shift + F10)	Displays previous save warning on top of menu options	Displays previous save warning on top of menu options	Pass
16	Entering 107 or LPD to load program data	Displays “Data has successfully bean loaded!”	Displays “Data has successfully bean loaded!”	Pass
17	Entering 100 or VFQ to view all fuel queues	Displays previously saved queue data	Displays previously saved queue data	Pass
18	Entering 108 or STK to view remaining fuel stock	Displays “6540 Litres of Fuel remaining”	Displays “6540 Litres of Fuel remaining”	Pass
19	Entering 109 or AFS to add fuel to stock Enter amount of fuel to be added: 25	Displays “25 litres of fuel added to stock”	Displays “25 litres of fuel added to stock”	Pass

20	Entering 108 or STK to view remaining fuel stock	Displays “6565 Litres of Fuel remaining”	Displays “6565 Litres of Fuel remaining”	Pass
21	Remove customer from Queue 1 slot 2 103 or RCQ From which queue do you want to remove customer: 1 From which queue slot?: 2	Displays “Jack was successfully removed from fuel pump 1 queue slot 2”	Displays “Jack was successfully removed from fuel pump 1 queue slot 2”	Pass
22	Enter 'R' to remove another customer from a specific location or press enter to return to the menu: Pressed enter	Display menu options	Display menu options	Pass
23	Entering 108 or STK to view fuel stock	Displays updated fuel stock “6575 Litres of Fuel remaining”	Displays updated fuel stock “6575 Litres of Fuel remaining”	Pass
24	Serve customer from Queue 2 104 or PCQ From which queue was this customer served?: 2	Displays “Kimi from queue 2 was served fuel”	Displays “Kimi from queue 2 was served fuel”	Pass
25	Enter 'S' to remove another served customer or press enter to return to the menu: S Serve customer from Queue 3 From which queue was this customer served?: 3	Displays “Sebastian from queue 3 was served fuel”	Displays “Sebastian from queue 3 was served fuel”	Pass
26	Pressed enter 100 or VFQ to view all fuel queues	Displays updated fuel queue with only “John” for queue 1. For queue 2 “Dinan” has moved up to slot 1 and for queue 3 “Akindu” has moved up to slot 1 and all other remaining slots are displayed as “empty”	Displays updated fuel queue with only “John” for queue 1. For queue 2 “Dinan” has moved up to slot 1 and for queue 3 “Akindu” has moved up to slot 1 and all other remaining slots are displayed as “empty”	Pass

Validation – Task 1				
	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
27	102 or ACQ/ 103 or RCQ/ 104 or PCQ Enter fuel queue (1,2 or 3): a	Displays "Please enter an Integer Value!" and prompts user for fuel queue input again	Displays "Please enter an Integer Value!" and prompts user for fuel queue input again	Pass
28	102 or ACQ/ 103 or RCQ/ 104 or PCQ Enter fuel queue (1,2 or 3): 4	Displays "Queue number has to be between 1 - 3" and prompts user for fuel queue input again	Displays "Queue number has to be between 1 - 3" and prompts user for fuel queue input again	Pass
29	102 or ACQ Enter Customer name: 4	Displays "Customer name is in an incorrect format" and prompts user to enter name again	Displays "Customer name is in an incorrect format" and prompts user to enter name again	Pass
30	102 or ACQ Enter Customer name: (Not entering anything)	Displays "Customer name is in an incorrect format" and prompts user to enter name again	Displays "Customer name is in an incorrect format" and prompts user to enter name again	Pass
31	After filling queue 1 with 6 customers 102 or ACQ Enter fuel queue (1,2 or 3) : 1	Displays "Queue 1 is full! Please enter customer to another queue" and prompts user to select another queue	Displays "Queue 1 is full! Please enter customer to another queue" and prompts user to select another queue	Pass
32	After filling all 3 queues with 6 customers Entering 102 or ACQ	Displays "WARNING!!! All queues are full! Please wait for customers to be served to add more customers to the Queues"	Displays "WARNING!!! All queues are full! Please wait for customers to be served to add more customers to the Queues"	Pass
33	103 or RCQ From which queue slot?: 7	Displays "Slot number has to be between 1 - 6" and prompts user to enter slot number again	Displays "Slot number has to be between 1 - 6" and prompts user to enter slot number again	Pass

34	109 or AFS Enter amount of fuel to be added: x	Displays “Amount of fuel has to be an integer!” and prompts user to enter amount of fuel again.	Displays “Amount of fuel has to be an integer!” and prompts user to enter amount of fuel again.	Pass
35	109 or AFS Enter amount of fuel to be added: -20	Displays “Fuel amount has to be greater than 0 litres” and prompts user to enter fuel amount again	Displays “Fuel amount has to be greater than 0 litres” and prompts user to enter fuel amount again	Pass
36	109 or AFS Enter amount of fuel to be added: 50	Displays “WARNING!!! Maximum stock fuel centre can hold is 6600 Litres Current stock level is 6570 litres.”	Displays “WARNING!!! Maximum stock fuel centre can hold is 6600 Litres Current stock level is 6570 litres.”	Pass
37	In option menu Enter option: 23	Display “Invalid Option!!!” and displays menu to give user another chance.	Display “Invalid Option!!!” and displays menu to give user another chance.	Pass
38	Queue 1 slot 2 is empty 103 or RCQ From which queue do you want to remove customer: 1 From which queue slot?: 2	Displays “Queue slot is already empty! Please re-check the queues and try again.” And takes user to the menu	Displays “Queue slot is already empty! Please re-check the queues and try again.” And takes user to the menu	Pass
39	Queue 2 is empty 104 or PCQ From which queue was this customer served?: 2	Displays “No customer in this queue to serve! Please re-check the queues and try again.” And takes user to menu	Displays “No customer in this queue to serve! Please re-check the queues and try again.” And takes user to menu	Pass
40	All queues are empty 105 or VCS	Displays “All Queues are empty! No customers to sort” and returns user to menu	Displays “All Queues are empty! No customers to sort” and returns customer to menu	Pass

41	All queues are full 101 or VEQ	Displays “All Queues are full! No empty queues to view” and returns user to menu	Displays “All Queues are full! No empty queues to view” and returns user to menu	Pass
Task 2				
	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
42	Add customer “Eden Hazard” to Queue 102 or ACQ Enter customer's first name: Eden Enter customer's last name: Hazard Enter customer's vehicle number: CAB-2301 Enter amount of fuel requested by customer: 23	Display “Eden Hazard was successfully added to fuel queue 1”	Display “Eden Hazard was successfully added to fuel queue 1”	Pass
43	Add customer “Frank Lampard” to Queue Enter 'A' to add another customer or press enter to return to the menu: A Enter customer's first name: Frank Enter customer's last name: Lampard Enter customer's vehicle number: CBL – 1805 Enter amount of fuel requested by customer: 23	Display “Frank Lampard was successfully added to fuel queue 2”	Display “Frank Lampard was successfully added to fuel queue 2”	Pass
44	100 or VFQ to view all queues	Displays all details of Eden Hazard in queue 1 slot 1 Displays all details of Frank Lampard in queue 2 slot 1	Displays all details of Eden Hazard in queue 1 slot 1 Displays all details of Frank Lampard in queue 2 slot 1	Pass
45	After successfully adding 5 customers 1 each to 5 queues (All were requested 23 litres) 102 or ACQ add “Akindu Karunaratne” to fuel queue (Vehicle number: CBL-8858 and requested 30 litres)	Displays “Akindu Karunaratne was successfully added to fuel queue 1”	Displays “Akindu Karunaratne was successfully added to fuel queue 1”	Pass
46	108 or STK to view stock	Displays “6455 Litres of Fuel remaining”	Displays “6455 Litres of Fuel remaining”	Pass

47	Remove customer from Queue 2 slot 1 103 or RCQ From which queue do you want to remove customer: 2 From which queue slot?: 1	Displays “Frank Lampard was successfully removed from fuel pump 2 queue slot 1”	Displays “Frank Lampard was successfully removed from fuel pump 2 queue slot 1”	Pass

48	Add customer “John Doe” to Queue 102 or ACQ Enter customer's first name: John Enter customer's last name: Doe Enter customer's vehicle number: KY-5623 Enter amount of fuel requested by customer: 20	Display “John Doe was successfully added to fuel queue 2”	Display “John Doe was successfully added to fuel queue 2”	Pass
49	Serve customer from Queue 1 104 or PCQ From which queue was this customer served?: 1	Display “Eden Hazard (Vehicle Number: CAB-2301) from queue 1 was served 23 litres of fuel”	Display “Eden Hazard (Vehicle Number: CAB-2301) from queue 1 was served 23 litres of fuel”	Pass
50	Pressed enter to return to menu 100 or VFQ to view all queues	Displays all queues occupied 1 slot each. In queue 1 “Akindu Karunaratne” has moved up 1 slot. Queue 2 “John doe” is in slot 1	Displays all queues occupied 1 slot each. In queue 1 “Akindu Karunaratne” has moved up 1 slot. Queue 2 “John doe” is in slot 1	Pass
51	Serve customer from Queue 2 104 or PCQ From which queue was this customer served?: 2	Display “John Doe (Vehicle Number: KY-5623) from queue 2 was served 20 litres of fuel”	Display “John Doe (Vehicle Number: KY-5623) from queue 2 was served 20 litres of fuel”	Pass
52	Entered ‘S’ to serve another customer Serve customer from Queue 1 From which queue was this customer served?: 1	Display “Akindu Karunaratne (Vehicle Number: CBL-8858) from queue 1 was served 30 litres of fuel”	Display “Akindu Karunaratne (Vehicle Number: CBL-8858) from queue 1 was served 30 litres of fuel”	Pass

53	Pressed Enter to return to menu 110 or IFQ to view income of all queues	Displays Viewing Incomes of all Queues Fuel Queue 1: Rs. 22790.00 Fuel Queue 2: Rs. 8600.00 Fuel Queue 3: Rs. 0.00 Fuel Queue 4: Rs. 0.00 Fuel Queue 5: Rs. 0.00	Displays Viewing Incomes of all Queues Fuel Queue 1: Rs. 22790.00 Fuel Queue 2: Rs. 8600.00 Fuel Queue 3: Rs. 0.00 Fuel Queue 4: Rs. 0.00 Fuel Queue 5: Rs. 0.00	Pass

Validation – Task 2

	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
54	102 or ACQ Enter customer's first name: (Left empty)	Displays “Customer first name is in an incorrect format!” and allows user to re-enter first name	Displays “Customer first name is in an incorrect format!” and allows user to re-enter first name	Pass
55	102 or ACQ Enter customer's first name: 1	Displays “Customer first name is in an incorrect format!” and allows user to re-enter first name	Displays “Customer first name is in an incorrect format!” and allows user to re-enter first name	Pass
56	102 or ACQ Enter customer's last name: (Left empty)	Displays “Customer last name is in an incorrect format!” and allows user to re-enter first name	Displays “Customer last name is in an incorrect format!” and allows user to re-enter first name	Pass
57	102 or ACQ Enter customer's last name: 6	Displays “Customer last name is in an incorrect format!” and allows user to re-enter first name	Displays “Customer last name is in an incorrect format!” and allows user to re-enter first name	Pass
58	102 or ACQ Enter customer's vehicle number:	Displays “Vehicle number cannot be empty! Please re-enter	Displays “Vehicle number cannot be empty! Please re-enter	Pass

	(Left empty)	vehicle number!" and allows user to re-enter first name	vehicle number" and allows user to re-enter first name	
59	102 or ACQ Enter amount of fuel requested by customer: a	Displays "Amount of fuel has to be an integer!" and prompts user to enter amount of fuel again.	Displays "Amount of fuel has to be an integer!" and prompts user to enter amount of fuel again.	Pass
60	102 or ACQ Enter amount of fuel requested by customer: 0	Displays "Fuel amount has to be greater than 0 litres" and prompts user to enter fuel amount again	Displays "Fuel amount has to be greater than 0 litres" and prompts user to enter fuel amount again	Pass
61	102 or ACQ Enter amount of fuel requested by customer: 0	Displays "WARNING!!! Cannot add customer as requested fuel amount is more than available stock! Press enter to return to menu and update stock to add more customers!"	Displays "WARNING!!! Cannot add customer as requested fuel amount is more than available stock! Press enter to return to menu and update stock to add more customers!"	Pass
61	After filling all 5 queues with 6 customers Entering 102 or ACQ	Displays "WARNING!!! All queues are full! Please wait for customers to be served to add more customers to the Queues"	Displays "WARNING!!! All queues are full! Please wait for customers to be served to add more customers to the Queues"	Pass
62	103 or RCQ / 104 or VCQ From which queue do you want to remove customer: 7	Displays "Queue number has to be between 1 - 5" and prompts user for fuel queue input again	Displays "Queue number has to be between 1 - 5" and prompts user for fuel queue input again	Pass
63	103 or RCQ / 104 or VCQ From which queue do you want to remove customer: b	Displays "Please enter an Integer Value!" and prompts user for fuel queue input again	Displays "Please enter an Integer Value!" and prompts user for fuel queue input again	Pass

Task 3				
	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
64	All 5 queues are full Add “John Terry” to queue 102 or ACQ Enter customer's first name: John Enter customer's last name: Terry Enter customer's vehicle number: KK-6622 Enter amount of fuel requested by customer: 30	Display “WARNING!!! All queues are full! Customer will be added to the waiting queue. John Terry was successfully added to the waiting queue!”	Display “WARNING!!! All queues are full! Customer will be added to the waiting queue. John Terry was successfully added to the waiting queue!”	Pass
65	Serve customer from queue 3 104 or VCQ From which queue do you want to remove customer: 3	Display “Akindu Karunaratne (Vehicle Number: CBL-8858) from queue 1 was served 30 litres of fuel John Terry from the waiting queue was added to fuel queue 3”	Display “Akindu Karunaratne (Vehicle Number: CBL-8858) from queue 1 was served 30 litres of fuel John Terry from the waiting queue was added to fuel queue 3”	Pass
66	Serve customer from queue 1 104 or VCQ From which queue do you want to remove customer: 1	Display “Test Customer (Vehicle Number: CAT-4370) from queue 1 was served 10 litres of fuel Waiting Queue is Empty! Therefore no customers were added to the queue”	Display “Test Customer (Vehicle Number: CAT-4370) from queue 1 was served 10 litres of fuel Waiting Queue is Empty! Therefore no customers were added to the queue”	Pass
67	All queues full Add “Niki Lauda” to queue 102 or ACQ	Display “WARNING!!! All queues are full! Customer will be added to the waiting queue. Niki Lauda was successfully added to the waiting queue!”	Display “WARNING!!! All queues are full! Customer will be added to the waiting queue. Niki Lauda was successfully added to the waiting queue!”	Pass
68	Remove customer from queue 2 slot 3	Display “Akindu Dinan was successfully	Display “Akindu Dinan was	Pass

	103 or RCQ	removed from fuel pump 2 queue slot 3 Niki Lauda from the waiting queue was added to fuel queue 2”	successfully removed from fuel pump 2 queue slot 3 Niki Lauda from the waiting queue was added to fuel queue 2”	
69	All queues and Waiting queue are full Add “Dinan Fernando” to queue 102 or ACQ Enter customer's first name: Dinan Enter customer's last name: Fernando Enter customer's vehicle number: 19-4370 Enter amount of fuel requested by customer: 15	Displays “WARNING!!! Waiting Queue is full! Cannot add more customers Press enter to return to the menu and serve customers to add more customers!	Displays “WARNING!!! Waiting Queue is full! Cannot add more customers Press enter to return to the menu and serve customers to add more customers!	Pass
Task 4				
	Test Case	**Expected Result**	**Actual Result**	**Pass/Fail**
70	111 or GUI in menu	Opens GUI in separate window	Opens GUI in separate window	Pass
71	Press “View Queue Details” button in GUI	Displays Fuel Queue on one side and Waiting queue on the other	Displays Fuel Queue on one side and Waiting queue on the other	Pass
72	Search customer by name	Displays relevant details for customer in relevant side i.e if customer is in fuel queue will display on fuel queue side	Displays relevant details for customer in relevant side i.e if customer is in fuel queue will display on fuel queue side	Pass
73	Search customer “a”	Displays customer not found	Displays customer not found	Pass

Discussion

The test cases mentioned above were chosen so I could cover all aspects of my program. I chose Test case 1 to see if the queues are initialised properly by displaying all slots of each queue as “empty”. From test case 2 and 3 I tested if the customer was getting added to the queue and from test case 4 confirmed it as the relevant slot was not displayed in the “View All Empty Queues” option. From test cases 5-10 I tested the do-while loop implemented in my

program which allows the user to enter multiple customers after selecting "102 or ACQ" as I believe that is more practical than entering 102 or ACQ every time the user wants to add a customer. Test case 11 I checked if the customers were added to the correct queues and they were. Test case 12 I tested if the "View Customers Sorted in alphabetical order" (105) menu option was working properly. This test case is valid for all 4 tasks of this coursework. When I was adding customers I ensured to add customers with the same letter for the first name and using uppercase and lowercase for the first names. The comparison was done using bubble sorting elements in the 2d array I had used and using .compareToIgnoreCase to sort customers regardless of the case of the name. Test cases 13-17 I chose to test if the storing and loading data worked. After saving the data and restarting the program a warning message is displayed saying previous save data is available. By entering 107 or LPD the program data is loaded back into the arrays and when all fuel queues are viewed the user can see the previous saved queue and stock data. Test cases 18, 19 and 20 were chose to check if menu options 108 to view fuel stock and 109 add fuel to stock were functioning correctly. As with menu option 105 these test cases were chosen so it covers the functionality in all 4 tasks. I chose test cases 21,22 and 23 show the functionality of menu option 103 to remove customer from a specific location. Just like adding customers I've given the user to remove multiple customer or press enter to return to the menu. When a customer is added 10 litres of fuel will be deducted regardless of whether the customer was served or not. So when removing the customer that 10 litres is added back to the stock as it was not served. This assumption is used in task 2 and 3 as well when the customer requests amount of fuel and is added to queue. Test cases 24,25 and 26 were chosen to show how menu option 104 works. I have given the user the option to serve multiple customers at once by entering "S" to continue or press enter to return to the menu. After removing a customer from either 103 or 104 test case 26 shows that the customers are shuffled up and the last slot is set to empty to avoid printing null. Test cases 27 to 41 were chosen to show the validation I have implemented in my program for Task 1. Every aspect of the program is validated and allows user to re-enter the input in order to continue in the program. When validating the customer name I ensured that only string inputs can be taken and when validating fuel amount and queue and slot numbers used try catch to only accept integers. Some test cases like test case 33, 36, 37,38 and 39 are common for all tasks in this program.

For Task 2 test cases 42 to 45 test the auto queue selection based on the minimum queue length. First 5 customers are added to slot 1 of each queue and the 6^{th} customer will be added back to queue 1, 7^{th} to queue 2 and so on. Test case 46 checks if the stock is updated correctly based on requested fuel amount compared to always 10 litres in task 1. Test cases 47 and 48 check if a customer is removed from option 103 or 104 from a queue and another customer is added that customer will be added to the queue where the previous customer was removed from. Test cases 49 to 53 test the functionality of the newly added menu option 110 to view incomes from all queues. The assumption I used is that the income is generated only when a customer is served and the fuel amount requested by that customer is added to the totalFuel count in the FuelQueue class. Using the totalFuel variable I was able to calculate the income for each queue separately. All other parts were tested using test cases from task 1 as mentioned before as the functionality is the same. Test cases 54-63 was chosen to check the extra validation for task 2. The program has the same validations as task 1 and in additional has validations for Customer last name, vehicle number and requested fuel amount and the range check for the fuel queue was increased from 3 to 5.

For Task 3 test cases 64 and 65 check if a customer is added to a waiting queue if all fuel queues are full and when a customer is served from menu option 104 the customer in the waiting queue is added to the end of the fuel queue which the customer was served from. Test case 66 tests if a customer is served and no customers are there in the waiting queue the relevant display message is shown to the user. I added an additional function to the code as I felt it was logical when thinking practically. When 103 option is selected it removes a customer but doesn't serve. But in a practical scenario a customer from the waiting queue will be added to that queue to replace the removed customer. To test that assumption I used test cases 67 and 68. Test case 69 is a validation test to check if the relevant warning message is displayed if the Waiting queue is also full. All other parts of the program function the same as task 2 and hence the remaining validations are the same as discussed above.

For Task 4 test case 70 checks if the GUI is loaded when option 111 or GUI is entered on the menu. Test cases 71 and 72 check the functionality within the GUI. When view queue details button is pressed both queues are loaded and if waiting queue is empty "Waiting queue is empty" is displayed. Finally test case 73 is a validation of the search feature. If no customer is found in either queue it displays the relevant error message.

Code :

Main Classes for Tasks 1,2,3 and 4 (FuelCenter.java)

Task 1

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

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights Reserved.
 * Array version for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 1.
 *
 * @author Akindu Karunaratne
 * @version 1.1 2022-08-08.
 **/

public class FuelCenter {
   private static final int numberOfPumps = 3;
   private static final int maxQueueSize = 6;

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int fuelStock = 6600; //Fuel Stock in Litres

      String[][] fuelQueue = new String[numberOfPumps][maxQueueSize]; //2D Array for
each fuel pump and its 6 queue slots

      //Initialising the empty queue slots for the 3 pumps.
      for (int i = 0; i < numberOfPumps; i++) {
         for (int j = 0; j < maxQueueSize; j++) {
            fuelQueue[i][j] = "empty";
         }
      }
      //To check if data exists from a previous run
      File temp = new File("Task1_Data.txt");
      if (temp.exists()) {
         System.out.println("\nWARNING! Data from a previous run has been found. If
you want to load that data enter '107' or 'LPD' in the menu below");
      }

      boolean menu = true; //Boolean variable menu which allows the program to loop
the displayed menu options
      while (menu) {
         //Menu options to be displayed on the console
         System.out.println("""

                ------- OPTION MENU -------

                Enter 100 or VFQ to View all Fuel Queues.
                Enter 101 or VEQ to View all Empty Queues.
                Enter 102 or ACQ to Add customer to a Queue.
                Enter 103 or RCQ to Remove a customer from a Queue(From a specific
location)
                Enter 104 or PCQ to Remove a served customer.
                Enter 105 or VCS to View Customers Sorted in alphabetical order.
                Enter 106 or SPD to Store Program Data into file.
                Enter 107 or LPD to Load Program Data from file.
                Enter 108 or STK to View Remaining Fuel Stock.
```

```
                Enter 109 or AFS to Add Fuel Stock.
                Enter 999 or EXT to Exit the Program
                """);

        System.out.print("Enter your option: "); //Taking user input for the menu
option
        String option = input.nextLine();
        System.out.println();

        //Nested if conditions to execute based on users input above
        if (option.equals("100") || option.equalsIgnoreCase("VFQ")) {
           viewAllQueues(fuelQueue); //To view all fuel queues

        } else if (option.equals("101") || option.equalsIgnoreCase("VEQ")) {
           viewEmptyQueues(fuelQueue); //To view fuel queues only with their empty
slots

        } else if (option.equals("102") || option.equalsIgnoreCase("ACQ")) {
           fuelStock = addCustomer(fuelStock, fuelQueue);

        } else if (option.equals("103") || option.equalsIgnoreCase("RCQ")) {
           fuelStock = removeCustomer(fuelStock, fuelQueue);

        } else if (option.equals("104") || option.equalsIgnoreCase("PCQ")) {
           removeServedCustomer(fuelQueue);

        } else if (option.equals("105") || option.equalsIgnoreCase("VCS")) {
           customerSorted(fuelQueue);

        } else if (option.equals("106") || option.equalsIgnoreCase("SPD")) {
           storeProgramData(fuelStock, fuelQueue);   //Stores queue and fuel stock
information to "Task1_Data.txt" file.

        } else if (option.equals("107") || option.equalsIgnoreCase("LPD")) {
           fuelStock = loadProgramData(fuelStock, fuelQueue);    //Loads data from
previous run

        } else if (option.equals("108") || option.equalsIgnoreCase("STK")) {
           System.out.println(fuelStock + " Litres of Fuel remaining"); //Displays
remaining fuel stock.

        } else if (option.equals("109") || option.equalsIgnoreCase("AFS")) {
           fuelStock = addFuelStock(fuelStock);

        } else if (option.equals("999") || option.equalsIgnoreCase("EXT")) {
           System.out.println("Exiting the Program...");
           menu = false;  //Ending loop to exit program.

        } else {    //Checking if an option from the menu was entered. If not program
will loop back to take user input.
           System.out.println("Invalid Option!!!\n");
        }
     }
  } //End of Main Method

  //My Methods
  /**
   * This method is to view all fuel queues with customer name if a queue slot is
occupied.
   *
   * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
   */
  public static void viewAllQueues(String[][] fuelQueue) {
```

```
      System.out.println("Viewing all Fuel Queues\n");
      System.out.println("-----------------------------------------------------------------
---------------------------------");

      for (int i = 0; i < fuelQueue.length; i++) {   //Outer loop in for loop which
loops through the rows in fuelQueue array.
         System.out.println("Fuel Queue " + (i + 1));
         for (int j = 0; j < fuelQueue[i].length; j++) { //Inner loop in for loop
which loops through the columns in fuelQueue array.
            if (fuelQueue[i][j].equals("empty")) {
               System.out.println("\tQueue slot " + (j + 1) + " is empty.");
            } else {
               System.out.println("\tQueue slot " + (j + 1) + " is occupied by " +
fuelQueue[i][j]);
            }
         }
         System.out.println();
         System.out.println("-----------------------------------------------------------------
-----------------------------------");
      }
   }

   /**
    * This method is to view all the empty slots in the fuel queues.
    *
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    */
   public static void viewEmptyQueues(String[][] fuelQueue) {
      System.out.println("Viewing all Empty Queues\n");
      System.out.println("-----------------------------------------------------------------
---------------------------------");

      //Checks if all queues are full as there will be no empty queues to display.
      if (!fuelQueue[0][5].equals("empty") && !fuelQueue[1][5].equals("empty") &&
!fuelQueue[2][5].equals("empty")) {
         System.out.println("All Queues are full! No empty queues to view");
      } else {
         for (int i = 0; i < fuelQueue.length; i++) {
            System.out.println("Fuel Queue " + (i + 1));
            for (int j = 0; j < fuelQueue[i].length; j++) {
               if (fuelQueue[i][j].equals("empty")) {    //Check if index in fuelQueue
array has any elements. If not display only the empty queues
                  System.out.println("\tQueue slot " + (j + 1) + " is empty.");
               }
            }
            System.out.println();
            System.out.println("-----------------------------------------------------------
---------------------------------------");
         }
      }
   }

   /**
    * This method is used to add a customer to a fuel queue.
    *
    * @param stock     Remaining fuel stock.
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    * @return Updated fuel stock value after customer is added (10 litres of fuel for
each customer).
    */
   public static int addCustomer(int stock, String[][] fuelQueue) {
      Scanner input = new Scanner(System.in);
      String select = "";
```

```
      int stockWarningLevel = 500;

      //do-while loop to allow user to enter multiple customers at once without
returning to the menu.
      do {
         //checks if all queues are full and returns to menu if it is.
         if (!fuelQueue[0][5].equals("empty") && !fuelQueue[1][5].equals("empty") &&
!fuelQueue[2][5].equals("empty")) {
            System.out.println("WARNING!!! All queues are full!");
            System.out.println("Please wait for customers to be served to add more
customers to the Queues\n");
            break;

         } else {
            //Method to validate queue number called.
            int pump = validateQueueSlotNumber(3, "Enter fuel queue (1,2 or 3): ",
"Queue");

            //checks if a specific fuel queue is full and gives user option to enter
another fuel queue to add customer
            if (!fuelQueue[pump - 1][maxQueueSize - 1].equals("empty")) {
               System.out.println("Queue " + pump + " is full! Please enter customer
to another queue\n");
               continue;

            } else {
               //Method to validate customer name called.
               String name = validateString("Customer name: ", "Customer name");

               for (int i = 0; i < maxQueueSize; i++) {    //Looping through the
columns of the fuelQueue array
                  if (fuelQueue[(pump - 1)][i].equals("empty")) { //Checking if slot
is empty based on the queue number user inputs.
                     fuelQueue[(pump - 1)][i] = name;             //If queue slot is
empty customer name will be set to that index position
                     System.out.println(name + " was successfully added to fuel queue
" + pump + "\n");
                     stock -= 10;         //Reducing stock level if customer is added.
                     break;
                  }
               }
            }
            System.out.print("Enter 'A' to add another customer or press enter to
return to the menu: ");
            select = input.nextLine();  //Asks user to add another customer or exit to
the option menu

         }
         System.out.println();

         if (stock <= stockWarningLevel) {
            System.out.println("WARNING! Fuel stock is below 500 Litres!\n");
//Warning message if stock reaches 500 Litres.
         }

      } while (select.equalsIgnoreCase("A"));

      return stock;
   }

   /**
    * This method is used to remove a customer from a specific queue location.
    *
```

```
    * @param stock     Fuel stock available.
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    * @return Updated fuel stock after customer was removed. (Adding 10 litres as
customer was not served).
    */
   public static int removeCustomer(int stock, String[][] fuelQueue) {
      Scanner input = new Scanner(System.in);
      String select;

      do {
         //Methods to validate queue number and slot number called.
         int pump = validateQueueSlotNumber(3, "From which queue do you want to remove
customer: ", "Queue");
         int slot = validateQueueSlotNumber(6, "From which queue slot?: ", "Slot");

         //Checks if entered queue slot is already empty. Displays error message and
returns to menu.
         if (fuelQueue[pump - 1][slot - 1].equals("empty")) {
            System.out.println("\nQueue slot is already empty! Please re-check the
queues and try again.\n");
            break;
         } else {
            System.out.println("\n" + fuelQueue[pump - 1][slot - 1] + " was
successfully removed from fuel pump "
                  + pump + " queue slot " + slot + "\n");

            for (int i = (slot - 1); i < maxQueueSize - 1; i++) {          //for loop
starts based on slot number entered by user.
               fuelQueue[(pump - 1)][i] = fuelQueue[(pump - 1)][i + 1];   //Shifts
elements up by 1 index depending on which slot was removed.
            }
            fuelQueue[(pump - 1)][maxQueueSize - 1] = "empty";     //Sets the last
index back to "empty" to avoid printing "null"
            stock += 10;                                          //Adding back the
stock as customer was not served.

            System.out.print("Enter 'R' to remove another customer from a specific
location or press enter to return to the menu: ");
            select = input.nextLine();  //Asks user to remove another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("R"));

      return stock;
   }

   /**
    * This method is to remove a served customer from a queue (Served customer is the
first in line in that particular queue).
    *
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    */
   public static void removeServedCustomer(String[][] fuelQueue) {
      Scanner input = new Scanner(System.in);
      String select;

      do {
         int pump = validateQueueSlotNumber(3, "From which queue was this customer
served?: ", "Queue");

         if (fuelQueue[pump - 1][0].equals("empty")) {
            System.out.println("\nNo customer in this queue to serve! Please re-check
```

```
the queues and try again.\n");
            break;

         } else {
            System.out.println(fuelQueue[pump - 1][0] + " from queue " + pump + " was
served fuel");

            for (int i = 0; i < maxQueueSize - 1; i++) {     //Served customer by logic
is the first in queue, hence for loop starts from first slot.
               fuelQueue[(pump - 1)][i] = fuelQueue[(pump - 1)][i + 1];   //Shifts
elements up by 1 index after first slot was removed.
            }
            fuelQueue[(pump - 1)][maxQueueSize - 1] = "empty";   //Sets the last index
back to "empty" to avoid printing "null"

            System.out.print("Enter 'S' to remove another served customer or press
enter to return to the menu: ");
            select = input.nextLine();  //Asks user to serve another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("S"));
   }

   /**
    * This method is to sort customer names in Alphabetical Order.
    *
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    */
   public static void customerSorted(String[][] fuelQueue) {
      String[][] sortedCustomers = new String[numberOfPumps][maxQueueSize]; //Creating
new array to store sorted customers

      //Checks if all queues are empty before sorting. If first slot is empty that
means the entire queue is empty.
      if (fuelQueue[0][0].equals("empty") && fuelQueue[1][0].equals("empty") &&
fuelQueue[2][0].equals("empty")) {
         System.out.println("All Queues are empty! No customers to sort");
      } else {
         for (int a = 0; a < fuelQueue.length; a++) {       //This loop copies the
customer names of fuelQueue array to sortedCustomers array
            for (int b = 0; b < fuelQueue[a].length; b++) {
               if (!(fuelQueue[a][b].equals("empty"))) {    //Checks if slot is not
empty and copies the name
                  sortedCustomers[a][b] = fuelQueue[a][b];
               } else {
                  sortedCustomers[a][b] = "e";               //If slot is empty sets
to 'e' (Used later in code)
               }
            }
         }
         for (int x = 0; x < sortedCustomers.length; x++) {             //Loop used to
bubble sort sortedCustomers array
            for (int y = 0; y < sortedCustomers[x].length; y++) {
               for (int z = 0; z < sortedCustomers[x].length - y - 1; z++) {
                  if (sortedCustomers[x][z].compareToIgnoreCase(sortedCustomers[x][z +
1]) > 0) {     //Compares to elements next to each other ignoring case.
                     String temp = sortedCustomers[x][z];
                     sortedCustomers[x][z] = sortedCustomers[x][z + 1];
                     sortedCustomers[x][z + 1] = temp;
                  }
               }
            }
```

```
         }
         System.out.println("\nViewing Customers sorted in alphabetical order \n");
         for (int i = 0; i < sortedCustomers.length; i++) {          //Loop to display
Queue and sorted customer names.
            System.out.println("Fuel Queue " + (i + 1));
            for (int j = 0; j < sortedCustomers[i].length; j++) {
               if (!(sortedCustomers[i][j].equals("e"))) {              //Skips if
slot is empty (Line 292).
                  System.out.println("\t" + sortedCustomers[i][j]);
               }
            }
            System.out.println();
         }
      }
   }

   /**
    * This method is to store queue data and fuel stock to a text file.
    *
    * @param stock     Remaining fuel stock.
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    */
   public static void storeProgramData(int stock, String[][] fuelQueue) {
      try {
         FileWriter dataFile = new FileWriter("Task1_Data.txt");  //Creating text file
"Task1_Data.txt"
         for (int i = 0; i < fuelQueue.length; i++) {
            for (int j = 0; j < fuelQueue[i].length; j++) {
               dataFile.write(fuelQueue[i][j] + "\n");   //Writing queue data to file
            }
         }
         dataFile.write(String.valueOf(stock)); //Writing remaining stock level to
file
         dataFile.close();
         System.out.println("File Updated Successfully!");

      } catch (IOException e) {
         System.out.println("An error occurred!: " + e);
      }
   }

   /**
    * This method loads the saved file back into the program.
    *
    * @param stock     Remaining fuel stock.
    * @param fuelQueue 2D Array collection Fuel Queues (Each Pump and its 6 Slots).
    * @return Fuel stock level.
    */
   public static int loadProgramData(int stock, String[][] fuelQueue) {
      try {
         File loadedFile = new File("Task1_Data.txt");
         Scanner readFile = new Scanner(loadedFile);

         while (readFile.hasNext()) {
            for (int i = 0; i < fuelQueue.length; i++) {
               for (int j = 0; j < fuelQueue[i].length; j++) {
                  fuelQueue[i][j] = readFile.nextLine();    //Reading the queue data
from "Task1_Data.txt" file and loads back to fuelQueue array.
               }
            }
            stock = readFile.nextInt();   //Reading the stock level from
"Task1_Data.txt" file.
         }
```

```
         readFile.close();
         System.out.println("Data has successfully bean loaded!");

      } catch (FileNotFoundException e) {
         System.out.println("File does not exist!\n");   //Try-catch to display error
message if file is not created first
      }
      return stock;
   }

   /**
    * This method is used to update the existing fuel stock level.
    *
    * @param stock Available fuel stock.
    * @return Updated fuel stock level.
    */
   public static int addFuelStock(int stock) {
      int maxStock = 6600;
      int newStock = validateFuelAmount();

      //Checking if total stock will be less than 6600 litres to add if not display
error message
      if (stock + newStock <= maxStock){
         stock += newStock;   //Adds the stock amount added by the user.
         System.out.println(newStock + " litres of fuel added to stock");

      }else {
         System.out.println("WARNING!!! Maximum stock fuel center can hold is 6600
Litres");
         System.out.println("Current stock level is " + stock + " litres.");
      }

      return stock;
   }

   /**
    * This method is used to validate strings in this code.
    *
    * @param displayMessage The text to display to user when asking for string input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated string name.
    */
   public static String validateString(String displayMessage, String errorMessage) {
      Scanner input = new Scanner(System.in);
      String stringName = "";
      boolean validateString = true;

      while (validateString) {   // If conditions not satisfied program will loop back
to ask for input.
         System.out.print("Enter " + displayMessage); //User input for string.
         stringName = input.nextLine();

         //stringName.matches("[a-zA-Z]+") section of code taken from
https://www.tutorialkart.com/java/how-to-check-if-string-contains-only-alphabets-in-
java/
         if (stringName.matches("[a-zA-Z]+") && !(stringName.isEmpty())) {
            validateString = false;
         } else {
            System.out.println(errorMessage + " is in an incorrect format!\n");
         }
      }
      return stringName;
   }
```

```
   /**
    * This method is used to validate queue number or slot number.
    *
    * @param value          Used to check the range depending on queue (3) or queue
slot (6).
    * @param displayMessage The text to display to user when asking for input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated integer value for queue number or slot number
    */
   public static int validateQueueSlotNumber(int value, String displayMessage, String
errorMessage) {
      Scanner input = new Scanner(System.in);
      int number = 0;
      boolean validateInteger = true;

      while (validateInteger) {  // If conditions not satisfied program will loop back
to ask for input.
         try {
            System.out.print(displayMessage);
            number = Integer.parseInt(input.nextLine());

            if (number >= 1 && number <= value) {
               validateInteger = false;
            } else {
               System.out.println(errorMessage + " number has to be between 1 - " +
value + "\n");
            }
         } catch (NumberFormatException e1) {
            System.out.println("Please enter an Integer Value!\n");
         }
      }
      return number;
   }
   /**
    * This method is used to validate the fuel amount.
    *
    * @return Validated fuel amount.
    */
   public static int validateFuelAmount() {
      Scanner input = new Scanner(System.in);
      int fuelAmount = 0;
      boolean validateFuel = true;

      while (validateFuel) {  // If conditions not satisfied program will loop back to
ask for input.
         try {
            System.out.print("Enter amount of fuel to be added: ");
            fuelAmount = Integer.parseInt(input.nextLine());

            if (fuelAmount > 0) {
               validateFuel = false;
            }else {
               System.out.println("Fuel amount has to be greater than 0 litres\n");
            }
         } catch (NumberFormatException e) {
            System.out.println("Amount of fuel has to be an integer!\n");
         }
      }
      return fuelAmount;
   }
}
```

Task 2

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

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights
Reserved.
 * Classes version for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 2.
 *
 * @author Akindu Karunaratne
 * @version 1.0 2022-08-08.
 **/

public class FuelCenter implements Serializable {
   private static final double fuelPrice = 430.00;
   private static final int numberOfPumps = 5;
   private static final int maxQueueSize = 6;

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int fuelStock = 6600; //Fuel Stock in Litres

      FuelQueue[] fuelQueues = new FuelQueue[numberOfPumps]; //Creating array
of FuelQueue objects with size for the number of pumps (5).

      initialise(fuelQueues); //Calling initialise method

      //To check if data exists from a previous run
      File temp = new File("Task2_Data.txt");
      if (temp.exists()) {
         System.out.println("\nWARNING! Data from a previous run has been
found. If you want to load that data enter '107' or 'LPD' in the menu
below");
      }
      boolean menu = true; //Boolean variable menu which allows the program
to loop the displayed menu options
      while (menu) {
         //Menu options to be displayed on the console
         System.out.println("""

                ------- OPTION MENU -------

                Enter 100 or VFQ to View all Fuel Queues.
                Enter 101 or VEQ to View all Empty Queues.
                Enter 102 or ACQ to Add customer to a Queue.
                Enter 103 or RCQ to Remove a customer from a Queue (From a
specific location).
                Enter 104 or PCQ to Remove a served customer.
                Enter 105 or VCS to View Customers Sorted in alphabetical
order.
                Enter 106 or SPD to Store Program Data into file.
                Enter 107 or LPD to Load Program Data from file.
                Enter 108 or STK to View Remaining Fuel Stock.
                Enter 109 or AFS to Add Fuel Stock.
                Enter 110 or IFQ to View income of each fuel queue.
```

```
                Enter 999 or EXT to Exit the Program
                """);

        System.out.print("Enter your option: "); //Taking user input for the
menu option
        String option = input.nextLine();
        System.out.println();

        if (option.equals("100") || option.equalsIgnoreCase("VFQ")) {
           viewAllQueues(fuelQueues); //To view all fuel queues

        } else if (option.equals("101") || option.equalsIgnoreCase("VEQ")) {
           viewEmptyQueues(fuelQueues); //To view fuel queues only with
their empty slots

        } else if (option.equals("102") || option.equalsIgnoreCase("ACQ")) {
           fuelStock = addCustomer(fuelQueues, fuelStock); //Add customer
method called.

        } else if (option.equals("103") || option.equalsIgnoreCase("RCQ")) {
           fuelStock = removeCustomer(fuelQueues, fuelStock); //Remove
customer(specific location) method called.

        } else if (option.equals("104") || option.equalsIgnoreCase("PCQ")) {
           removeServedCustomer(fuelQueues); //Remove served customer method
called.

        } else if (option.equals("105") || option.equalsIgnoreCase("VCS")) {
           customerSorted(fuelQueues);   //customerSorted method called to
view customers sorted in alphabetical order.

        } else if (option.equals("106") || option.equalsIgnoreCase("SPD")) {
           storeProgramData(fuelQueues, fuelStock);   //Stores queue and
fuel stock information to "Task2_Data.txt" file.

        } else if (option.equals("107") || option.equalsIgnoreCase("LPD")) {
           fuelStock = loadProgramData(fuelQueues, fuelStock);    //Loads
data from previous run

        } else if (option.equals("108") || option.equalsIgnoreCase("STK")) {
           System.out.println(fuelStock + " Litres of Fuel remaining");
//Displays remaining fuel stock.

        } else if (option.equals("109") || option.equalsIgnoreCase("AFS")) {
           fuelStock = addFuelStock(fuelStock); //addFuelStock method called
to add fuel to stock.

        } else if (option.equals("110") || option.equalsIgnoreCase("IFQ")) {
           viewIncome(fuelQueues); //viewIncome method called to view income
of all fuel queues.

        } else if (option.equals("999") || option.equalsIgnoreCase("EXT")) {
           System.out.println("Exiting the Program...");
           menu = false;  //Ending loop to exit program.

        } else {    //Checking if an option from the menu was entered. If
not program will loop back to take user input.
```

```
            System.out.println("Invalid Option!!!\n");
         }
      }
   } //End of Main Method

   //My methods

   /**
    * This method initialises the FuelQueue object arrays and assigns a
Passenger object for each queue
    *
    * @param queueRef FuelQueue object array.
    */
   private static void initialise(FuelQueue[] queueRef) {
      for (int i = 0; i < queueRef.length; i++) {
         queueRef[i] = new FuelQueue(i + 1, maxQueueSize, 0); //Assigns
FuelQueue object to each array index for each pump
         for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
            //Assigns Passenger object to each queue slot in each pump.
            queueRef[i].assignCustomer(new Passenger("empty", "empty",
"empty", 0));
         }
      }
   }

   /**
    * This method is to view all fuel queues with customer name if a queue
slot is occupied.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewAllQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Fuel Queues\n");
      System.out.println("-----------------------------------------------------
-----------------------------------------");

      for (FuelQueue fuelQueue : queueRef) {   //Outer loop in for loop which
loops through the rows in fuelQueue array.
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() +
"\n");
         for (int i = 0; i < fuelQueue.getQueueSize(); i++) { //Inner loop in
for loop which loops through the columns in fuelQueue array.
            if
(fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");
            } else {
               System.out.println("\tQueue slot " + (i + 1) + " is occupied
by; ");
               System.out.println(fuelQueue.getCustomerDetails(i));
//Printing details of each customer in queues
            }
         }
         System.out.println();
         System.out.println("-----------------------------------------------
-----------------------------------------");
      }
   }
```

```
   /**
    * This method is to view all the empty slots in the fuel queues.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewEmptyQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Empty Queues\n");
      System.out.println("-------------------------------------------------------
-----------------------------------------");

      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() +
"\n");

         for (int i = 0; i < fuelQueue.getQueueSize(); i++) {

            //Check if index in queueRef array has any elements. If not
display only the empty queues
            if
(fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");
            }
         }
         System.out.println();
         System.out.println("-------------------------------------------------
-----------------------------------------");
      }
   }

   /**
    * This method is used to add a customer to a fuel queue.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int addCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select = "";
      int stockWarningLevel = 500;
      int fullSlots = 0;   //Number of queue slots full
      int fullQueues;
      int[] occupiedQueueSlots = new int[numberOfPumps]; //Array to store
available slots in each pump.

      do {
         //For loop to check each fuel queue to check how many slots are full
         for (int i = 0; i < queueRef.length; i++) {
            for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
               if
(!queueRef[i].getCustomerDetails(j).getFirstName().equals("empty")) {
                  fullSlots++;
               }
            }
            occupiedQueueSlots[i] = fullSlots; //Assigns number of full slots
to relevant pump
```

```
            fullSlots = 0;
         }
         int pump = 0;
         int minLength = occupiedQueueSlots[pump]; //Sets fuel pump with
minimum queue length to the first pump by default.

         //For loop to check which pump has the minimum number of full slots
(Queue with minimum length).
         for (int i = 1; i < occupiedQueueSlots.length; i++) {
            if (occupiedQueueSlots[i] < minLength) {
               minLength = occupiedQueueSlots[i];
               pump = i; //Sets pump to queue with minimum length to be used
later.
            }
         }
         //Calling method to check how many queues are full
         fullQueues = fullQueueChecker(occupiedQueueSlots);

         //Check if all queues are full and if so returns user to menu. If
not adds customer
         if (fullQueues == numberOfPumps) {
            System.out.println("WARNING!!! All queues are full!");
            System.out.println("Please wait for customers to be served to add
more customers to the Queues\n");

         } else {
            //Validation methods for customers first name, last name, vehicle
number and requested fuel amount called.
            String firstName = validateString("customer's first name: ",
"Customer first name");
            String lastName = validateString("customer's last name: ",
"Customer last name");
            String vehicleNumber = validateVehicle();
            int fuelAmount = validateFuelAmount("requested by customer");

            if (fuelAmount < stock) {
               for (int i = 0; i < maxQueueSize; i++) {    //Looping through
the columns of the fuelQueue array
                  if
(queueRef[pump].getCustomerDetails(i).getFirstName().equals("empty")) {
//Checks if slot is empty to add and replace

                     //Adds customer using setters from Passenger class to
pump assigned from line 198.

queueRef[pump].getCustomerDetails(i).setFirstName(firstName);

queueRef[pump].getCustomerDetails(i).setLastName(lastName);

queueRef[pump].getCustomerDetails(i).setVehicleNumber(vehicleNumber);

queueRef[pump].getCustomerDetails(i).setFuelAmount(fuelAmount);

                     stock -=
queueRef[pump].getCustomerDetails(i).getFuelAmount(); //Stock updated with
requested fuel amount reduced
                     System.out.println("\n" +
```

```
queueRef[pump].getCustomerDetails(i).getFullName() + " was successfully added
to fuel queue " + (pump + 1) + "\n");
                  }
               }
            } else {
               System.out.println("WARNING!!! Cannot add customer as
requested fuel amount is more than available stock!");
               System.out.println("Press enter to return to menu and update
stock to add more customers!\n");
            }
            System.out.print("Enter 'A' to add another customer or press
enter to return to the menu: ");
            select = input.nextLine();  //Asks user to add another customer
or exit to the option menu
            System.out.println();
         }
         if (stock <= stockWarningLevel) {
            System.out.println("WARNING! Fuel stock is below 500 Litres!\n");
//Warning message if stock reaches 500 Litres.
         }

      } while (select.equalsIgnoreCase("A"));

      return stock;
   }

   /**
    * This method is used to remove a customer from a specific queue
location.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int removeCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select = "";
      do {
         //Calling validation method for Queue number and Slot number.
         int pump = validateQueueSlotNumber(5, "fuel queue do you want to
remove customer: ", "Queue");
         int slot = validateQueueSlotNumber(6, "queue slot do you want to
remove customer: ", "Slot");

         //Checks if slot is already empty. If so returns user to menu.
         if (queueRef[pump - 1].getCustomerDetails(slot -
1).getFirstName().equals("empty")) {
            System.out.println("\nQueue slot is already empty! Please re-
check the queues and try again.\n");

         } else {
            System.out.println("\n" + queueRef[pump -
1].getCustomerDetails(slot - 1).getFullName() + " was successfully removed
from fuel pump "
                  + pump + " queue slot " + slot + "\n");
            //Adding back customers requested fuel to stock as customer was
not served
```

```
            stock += queueRef[pump - 1].getCustomerDetails(slot -
1).getFuelAmount();

            //Removing customer by calling removeCustomer method from
FuelQueue class
            queueRef[pump - 1].removeCustomer(slot - 1);

            //Initialising last index in ArrayList with new Passenger object
to avoid printing null.
            queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
"empty", 0));

            System.out.print("Enter 'R' to remove another customer from a
specific location or press enter to return to the menu: ");
            select = input.nextLine();  //Asks user to remove another
customer or exit to the option menu
            System.out.println();
         }

      } while (select.equalsIgnoreCase("R"));

      return stock;
   }

   /**
    * This method is to remove a served customer from a queue.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void removeServedCustomer(FuelQueue[] queueRef) {
      Scanner input = new Scanner(System.in);
      String select = "";
      do {
         //Queue number validation method called.
         int pump = validateQueueSlotNumber(5, "fuel queue was customer
served?: ", "Queue");

         //Stores amount of fuel the customer requested using getter from
Passenger class.
         int requestedFuel = queueRef[pump -
1].getCustomerDetails(0).getFuelAmount();

         //Sets the requested fuel to the total fuel amount served in that
queue. Used when calculating income for each queue.
         queueRef[pump - 1].setTotalFuel(requestedFuel);

         //Checks if queue is already empty
         if (queueRef[pump -
1].getCustomerDetails(0).getFirstName().equals("empty")) {
            System.out.println("\nNo customer in this queue to serve! Please
re-check the queues and try again.\n");

         } else {
            System.out.println("\n" +
                  queueRef[pump - 1].getCustomerDetails(0).getFullName() +
" (Vehicle Number: " +
                  queueRef[pump -
```

```
1].getCustomerDetails(0).getVehicleNumber() + ") from queue " + pump + " was
served " +
                    queueRef[pump - 1].getCustomerDetails(0).getFuelAmount()
+ " litres of fuel\n"
            );

            //Removes customer using removeCustomer method from FuelQueue
class. Slot is 0 as the customer served is the first customer in the queue.
            queueRef[pump - 1].removeCustomer(0);

            //Initialising last index in ArrayList with new Passenger object
to avoid printing null.
            queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
"empty", 0));

            System.out.print("Enter 'S' to remove another served customer or
press enter to return to the menu: ");
            select = input.nextLine();  //Asks user to serve another customer
or exit to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("S"));
   }

   /**
    * This method is to sort customer names of each queue in Alphabetical
Order.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void customerSorted(FuelQueue[] queueRef) {
      System.out.println("Viewing Customers sorted in alphabetical order\n");

      //For each loop to print each queue separately.
      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber());
         fuelQueue.customersSorted(); //Calling customersSorted method from
Fuel Queue class.
         System.out.println();
      }
   }

   /**
    * This method stores queue data and fuel stock to a text file using
serialization.
    * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
serialization adapted from this link.</a>
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    */
   public static void storeProgramData(FuelQueue[] queueRef, int stock) {
      try {
         FileOutputStream fileOutputStream = new
FileOutputStream("Task2_Data.txt");
         ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
```

```
            for (FuelQueue fuelQueue : queueRef) {
               objectOutputStream.writeObject(fuelQueue);
            }
            objectOutputStream.writeObject(stock);
            objectOutputStream.close();
            System.out.println("Data successfully stored to file!");

         } catch (IOException e) {
            System.out.println("An error occurred!: " + e);
         }
      }

      /**
       * This method loads the saved file back into the program using
deserialization.
       * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
deserialization adapted from this link.</a>
       *
       * @param queueRef FuelQueue object array
       * @param stock    Remaining fuel stock.
       * @return Updated fuel stock.
       */
      public static int loadProgramData(FuelQueue[] queueRef, int stock) {
         try {
            FileInputStream fileInputStream = new
FileInputStream("Task2_Data.txt");
            ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);

            for (int i = 0; i < queueRef.length; i++) {
               queueRef[i] = (FuelQueue) objectInputStream.readObject();
            }
            stock = (int) objectInputStream.readObject();
            objectInputStream.close();
            System.out.println("File successfully loaded!");

         } catch (IOException e) {
            System.out.println("File does not exist!" + e);
         } catch (ClassNotFoundException e1) {
            System.out.println("Class not found!");
         }
         return stock;
      }

      /**
       * This method is used to add fuel to the existing stock.
       *
       * @param stock Remaining fuel stock.
       * @return Updated fuel stock.
       */
      public static int addFuelStock(int stock) {
         int maxStock = 6600;
         int newStock = validateFuelAmount("to be added to stock");

         //Checking if total stock will be less than 6600 litres to add if not
display error message
```

```
      if (stock + newStock <= maxStock) {
         stock += newStock;   //Adds the stock amount added by the user.
         System.out.println(newStock + " litres of fuel added to stock");

      } else {
         System.out.println("WARNING!!! Maximum stock fuel center can hold is
6600 Litres");
         System.out.println("Current stock level is " + stock + " litres.");
      }

      return stock;
   }

   /**
    * This method is used to view the total income of all queues.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewIncome(FuelQueue[] queueRef) {
      System.out.println("Viewing Incomes of all Queues\n");

      //For each loop to display total income of each queue.
      for (FuelQueue fuelQueue : queueRef) {
         //printf used to format income to 2 decimal places. Total fuel from
each queue is taken from getTotalFuel method from FuelQueue class.
         System.out.printf("\tFuel Queue " + fuelQueue.getQueueNumber() + ":
Rs. %.2f\n", (fuelQueue.getTotalFuel() * fuelPrice));
      }
   }

   /**
    * This method is used to validate strings in this code.
    *
    * @param displayMessage The text to display to user when asking for
string input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated string name.
    */
   public static String validateString(String displayMessage, String
errorMessage) {
      Scanner input = new Scanner(System.in);
      String stringName = "";
      boolean validateString = true;

      while (validateString) {   // If conditions not satisfied program will
loop back to ask for input.
         System.out.print("Enter " + displayMessage);
         stringName = input.nextLine();

         //stringName.matches("[a-zA-Z]+") section of code taken from
https://www.tutorialkart.com/java/how-to-check-if-string-contains-only-
alphabets-in-java/
         if (stringName.matches("[a-zA-Z]+") && !(stringName.isEmpty())) {
            validateString = false;
         } else {
            System.out.println(errorMessage + " is in an incorrect
format!\n");
```

```
         }
      }
      return stringName;
   }

   /**
    * This method is used to validate customer's vehicle number.
    *
    * @return Validated vehicle number.
    */
   public static String validateVehicle() {
      Scanner input = new Scanner(System.in);
      String vehicleName = "";
      boolean validateVehicle = true;

      while (validateVehicle) {  // If conditions not satisfied program will
loop back to ask for input.
         System.out.print("Enter customer's vehicle number: ");
         vehicleName = input.nextLine();

         if (!(vehicleName.isEmpty())) {
            validateVehicle = false;
         } else {
            System.out.println("Vehicle number cannot be empty! Please re-
enter vehicle number\n");
         }
      }
      return vehicleName;
   }

   /**
    * This method is used to validate queue number or slot number.
    *
    * @param value          Used to check the range depending on queue (3) or
queue slot (6).
    * @param displayMessage The text to display to user when asking for
input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated integer value for queue number or slot number
    */
   public static int validateQueueSlotNumber(int value, String
displayMessage, String errorMessage) {
      Scanner input = new Scanner(System.in);
      int number = 0;
      boolean validateInteger = true;

      while (validateInteger) {  // If conditions not satisfied program will
loop back to ask for input.
         try {
            System.out.print("From which " + displayMessage);
            number = Integer.parseInt(input.nextLine());

            if (number >= 1 && number <= value) {
               validateInteger = false;
            } else {
               System.out.println(errorMessage + " number has to be between 1
- " + value + "\n");
```

```
            }
         } catch (NumberFormatException e1) {
            System.out.println("Please enter an Integer Value!\n");
         }
      }
      return number;
   }

   /**
    * This method is used to validate the fuel amount entered by user.
    *
    * @param displayMessage The text to display to user when asking for
amount of fuel for customer or adding to stock.
    * @return Validated fuel amount.
    */
   public static int validateFuelAmount(String displayMessage) {
      Scanner input = new Scanner(System.in);
      int fuelAmount = 0;
      boolean validateFuel = true;

      while (validateFuel) {  // If conditions not satisfied program will
loop back to ask for input.
         try {
            System.out.print("Enter amount of fuel " + displayMessage + ":
");
            fuelAmount = Integer.parseInt(input.nextLine());

            if (fuelAmount > 0) {
               validateFuel = false;
            } else {
               System.out.println("Fuel amount has to be greater than 0
litres\n");
            }
         } catch (NumberFormatException e) {
            System.out.println("Amount of fuel has to be an integer!\n");
         }
      }
      return fuelAmount;
   }

   /**
    * This method checks how many queues are full.
    *
    * @param occupiedSlots Array with available slots of each queue.
    * @return Number of full queues.
    */
   public static int fullQueueChecker(int[] occupiedSlots) {
      int fullQueues = 0;

      for (int fullSlots : occupiedSlots) {
         if (fullSlots == maxQueueSize) {
            fullQueues++;
         }
      }
      return fullQueues;
   }
}
```

Task 3

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

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights Reserved.
 * Classes version for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 3.
 *
 * @author Akindu Karunaratne
 * @version 1.0 2022-08-08.
 **/

public class FuelCenter implements Serializable {
   private static final double fuelPrice = 430.00;
   private static final int numberOfPumps = 5;
   private static final int maxQueueSize = 6;
   private static final int waitingQueueSize = 6;
   private static WaitingQueue waitingQueue = new WaitingQueue(waitingQueueSize);

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int fuelStock = 6600;

      FuelQueue[] fuelQueues = new FuelQueue[numberOfPumps]; //Creating array of
FuelQueue objects with size for the number of pumps (5).

      initialise(fuelQueues);

      //To check if data exists from a previous run
      File temp = new File("Task3_Data.txt");
      if (temp.exists()) {
         System.out.println("\nWARNING! Data from a previous run has been found. If
you want to load that data enter '107' or 'LPD' in the menu below");
      }

      boolean menu = true; //Boolean variable menu which allows the program to loop
the displayed menu options
      while (menu) {
         //Menu options to be displayed on the console
         System.out.println("""

                  ------- OPTION MENU -------

                  Enter 100 or VFQ to View all Fuel Queues.
                  Enter 101 or VEQ to View all Empty Queues.
                  Enter 102 or ACQ to Add customer to a Queue.
                  Enter 103 or RCQ to Remove a customer from a Queue(From a specific
location)
                  Enter 104 or PCQ to Remove a served customer.
                  Enter 105 or VCS to View Customers Sorted in alphabetical order.
                  Enter 106 or SPD to Store Program Data into file.
                  Enter 107 or LPD to Load Program Data from file.
                  Enter 108 or STK to View Remaining Fuel Stock.
                  Enter 109 or AFS to Add Fuel Stock.
                  Enter 110 or IFQ to Print income of a selected queue.
                  Enter 999 or EXT to Exit the Program
                  """);

         System.out.print("Enter your option: "); //Taking user input for the menu
option
         String option = input.nextLine();
```

```
            System.out.println();

            if (option.equals("100") || option.equalsIgnoreCase("VFQ")) {
                viewAllQueues(fuelQueues); //To view all fuel queues

            } else if (option.equals("101") || option.equalsIgnoreCase("VEQ")) {
                viewEmptyQueues(fuelQueues); //To view fuel queues only with their empty
slots

            } else if (option.equals("102") || option.equalsIgnoreCase("ACQ")) {
                fuelStock = addCustomer(fuelQueues, fuelStock); //Add customer method
called.

            } else if (option.equals("103") || option.equalsIgnoreCase("RCQ")) {
                fuelStock = removeCustomer(fuelQueues, fuelStock); //Remove
customer(specific location) method called.

            } else if (option.equals("104") || option.equalsIgnoreCase("PCQ")) {
                fuelStock = removeServedCustomer(fuelQueues, fuelStock); //Remove served
customer method called.

            } else if (option.equals("105") || option.equalsIgnoreCase("VCS")) {
                customerSorted(fuelQueues);   //customerSorted method called to view
customers sorted in alphabetical order.

            } else if (option.equals("106") || option.equalsIgnoreCase("SPD")) {
                storeProgramData(fuelQueues, fuelStock);   //Stores queue and fuel stock
information to "Task2_Data.txt" file.

            } else if (option.equals("107") || option.equalsIgnoreCase("LPD")) {
                fuelStock = loadProgramData(fuelQueues, fuelStock);    //Loads data from
previous run

            } else if (option.equals("108") || option.equalsIgnoreCase("STK")) {
                System.out.println(fuelStock + " Litres of Fuel remaining"); //Displays
remaining fuel stock.

            } else if (option.equals("109") || option.equalsIgnoreCase("AFS")) {
                fuelStock = addFuelStock(fuelStock); //addFuelStock method called to add
fuel to stock.

            } else if (option.equals("110") || option.equalsIgnoreCase("IFQ")) {
                viewIncome(fuelQueues); //viewIncome method called to view income of all
fuel queues.

            } else if (option.equals("999") || option.equalsIgnoreCase("EXT")) {
                System.out.println("Exiting the Program...");
                menu = false;  //Ending loop to exit program.

            } else {    //Checking if an option from the menu was entered. If not program
will loop back to take user input.
                System.out.println("Invalid Option!!!\n");
            }
        }
    }

    /**
     * This method initialises the FuelQueue object arrays and assigns a passenger
object for each queue
     *
     * @param queueRef FuelQueue object array.
     */
```

```
   private static void initialise(FuelQueue[] queueRef) {
      for (int i = 0; i < queueRef.length; i++) {
         queueRef[i] = new FuelQueue(i + 1, maxQueueSize, 0); //Assigns FuelQueue
object to each array index for each pump
         for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
            //Assigns Passenger object to each queue slot in each pump.
            queueRef[i].assignCustomer(new Passenger("empty", "empty", "empty", 0));
         }
      }
   }

   /**
    * This method is to view all fuel queues with customer name if a queue slot is
occupied.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewAllQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Fuel Queues\n");
      System.out.println("-----------------------------------------------------------------
---------------------------------");

      for (FuelQueue fuelQueue : queueRef) {   //Outer loop in for loop which loops
through the rows in fuelQueue array.
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() + "\n");

         for (int i = 0; i < fuelQueue.getQueueSize(); i++) { //Inner loop in for loop
which loops through the columns in fuelQueue array.
            if (fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");

            } else {
               System.out.println("\tQueue slot " + (i + 1) + " is occupied by; ");
               System.out.println(fuelQueue.getCustomerDetails(i)); //Printing details
of each customer in queues
            }
         }
         System.out.println();
         System.out.println("-----------------------------------------------------------
-----------------------------------");
      }
   }

   /**
    * This method is to view all the empty slots in the fuel queues.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewEmptyQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Empty Queues\n");
      System.out.println("-----------------------------------------------------------------
---------------------------------");

      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() + "\n");

         for (int i = 0; i < fuelQueue.getQueueSize(); i++) {

            //Check if index in queueRef array has any elements. If not display only
the empty queues
            if (fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");
            }
```

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

   }

   /**
    * This method is used to add a customer to a fuel queue. If all queues are full
the customer will be added to the waiting queue from this method.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int addCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select;
      int stockWarningLevel = 500;
      int fullSlots = 0;   //Number of queue slots full
      int fullQueues;
      int[] occupiedQueueSlots = new int[numberOfPumps]; //Array to store available
slots in each pump.

      do {
         if (stock <= stockWarningLevel) {
            System.out.println("WARNING! Fuel stock is below 500 Litres!\n");
//Warning message if stock reaches 500 Litres.
         }

         //Validation methods for customers first name, last name, vehicle number and
requested fuel amount called.
         String firstName = validateString("customer's first name: ", "Customer first
name");
         String lastName = validateString("customer's last name: ", "Customer last
name");
         String vehicleNumber = validateVehicle();
         int fuelAmount = validateFuelAmount("requested by customer");

         if (fuelAmount < stock) {
            //For loop to check each fuel queue to check how many slots are full
            for (int i = 0; i < queueRef.length; i++) {
               for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
                  if
(!queueRef[i].getCustomerDetails(j).getFirstName().equals("empty")) {
                     fullSlots++;
                  }
               }
               occupiedQueueSlots[i] = fullSlots; //Assigns number of full slots to
relevant pump
               fullSlots = 0;
            }
            int pump = 0;
            int minLength = occupiedQueueSlots[pump]; //Sets fuel pump with minimum
queue length to the first pump by default.

            //For loop to check which pump has the minimum number of full slots (Queue
with minimum length).
            for (int i = 1; i < occupiedQueueSlots.length; i++) {
               if (occupiedQueueSlots[i] < minLength) {
                  minLength = occupiedQueueSlots[i];
                  pump = i; //Sets pump to queue with minimum length to be used later.
```

```
                }
            }
            //Calling method to check how many queues are full
            fullQueues = fullQueueChecker(occupiedQueueSlots);

            //Checks if all queues are full. If so tries to add customer to the
waiting queue.
            if (fullQueues == numberOfPumps) {

                //Check if waiting queue is full. If waiting queue is full user is
given instructions to return to menu.
                if (waitingQueue.isFull()) {
                    System.out.println("\nWARNING!!! Waiting Queue is full! Can not add
more customers\n");
                    System.out.println("Press enter to return to the menu and serve
customers to add more customers!\n");

                } else {
                    //Display message to show user customer was added to waiting queue.
                    System.out.println("\nWARNING!!! All queues are full! Customer will
be added to the waiting queue");

                    //enQueue method called from WaitingQueue class. Passenger object
passed as a parameter to add new customer in waiting queue.
                    waitingQueue.enQueue(new Passenger(firstName, lastName,
vehicleNumber, fuelAmount));
                    System.out.println("\n" + firstName + " " + lastName + " was
successfully added to the waiting queue!\n");
                }

            } else {
                for (int i = 0; i < maxQueueSize; i++) {    //Looping through the
columns of the fuelQueue array
                    if
(queueRef[pump].getCustomerDetails(i).getFirstName().equals("empty")) { //Checks if
slot is empty to add and replace

                        //Adds customer using setters from Passenger class to pump
assigned from line 210.
                        queueRef[pump].getCustomerDetails(i).setFirstName(firstName);
                        queueRef[pump].getCustomerDetails(i).setLastName(lastName);

queueRef[pump].getCustomerDetails(i).setVehicleNumber(vehicleNumber);
                        queueRef[pump].getCustomerDetails(i).setFuelAmount(fuelAmount);
                        queueRef[pump].setQueueNumber(pump + 1);
                        stock -= queueRef[pump].getCustomerDetails(i).getFuelAmount();
//Stock updated with requested fuel amount reduced

                        System.out.println("\n" +
queueRef[pump].getCustomerDetails(i).getFullName() + " was successfully added to fuel
queue " + (pump + 1) + "\n");
                        break;
                    }
                }
            }
         }else {
            System.out.println("WARNING!!! Cannot add customer as requested fuel
amount is more than available stock!");
            System.out.println("Press enter to return to menu and update stock to add
more customers!\n");
         }
         System.out.print("Enter 'A' to add another customer or press enter to return
to the menu: ");
```

```
         select = input.nextLine();  //Asks user to add another customer or exit to
the option menu
         System.out.println();

      } while (select.equalsIgnoreCase("A"));

      return stock;
   }

   /**
    * This method is used to remove a customer from a specific queue location.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int removeCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select = "";
      do {
         //Calling validation method for Queue number and Slot number.
         int pump = validateQueueSlotNumber(5, "fuel queue do you want to remove
customer: ", "Queue");
         int slot = validateQueueSlotNumber(6, "queue slot do you want to remove
customer: ", "Slot");

         //Checks if slot is already empty. If so returns user to menu.
         if (queueRef[pump - 1].getCustomerDetails(slot -
1).getFirstName().equals("empty")) {
            System.out.println("\nQueue slot is already empty! Please re-check the
queues and try again.\n");

         } else {
            System.out.println("\n" + queueRef[pump - 1].getCustomerDetails(slot -
1).getFullName() + " was successfully removed from fuel pump "
                  + pump + " queue slot " + slot);

            //Adding back customers requested fuel to stock as customer was not served
            stock += queueRef[pump - 1].getCustomerDetails(slot - 1).getFuelAmount();

            //Removing customer by calling removeCustomer method from FuelQueue class
            queueRef[pump - 1].removeCustomer(slot - 1);

            //Checking if waiting queue is empty to add customer to fill the queue
            if (waitingQueue.isEmpty()) {
               System.out.println("\nWaiting Queue is Empty! Therefore no customers
were added to the queue");

               //If waiting queue is empty assigns last index of array list with a new
Passenger object to avoid printing null.
               queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
"empty", 0));

            } else {
               //If there is a customer in waiting queue added to the fuel queue based
on which queue customer was removed from
               Passenger waitingCustomer = waitingQueue.deQueue();
               queueRef[pump - 1].assignCustomer(waitingCustomer);
               System.out.println(waitingCustomer.getFullName() + " from the waiting
queue was added to fuel queue " + pump);
               stock -= waitingCustomer.getFuelAmount(); //Reducing stock from waiting
queue customer
            }
```

```
            System.out.print("\nEnter 'R' to remove another customer from a specific
location or press enter to return to the menu: ");
            select = input.nextLine();  //Asks user to remove another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("R"));

      return stock;
   }

   /**
    * This method is to remove a served customer. If there is a customer in the
waiting queue that customer will be automatically
    * added to the fuel queue from this method.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int removeServedCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select = "";
      do {
         //Queue number validation method called.
         int pump = validateQueueSlotNumber(5, "fuel queue was customer served?: ",
"Queue");

         //Stores amount of fuel the customer requested using getter from Passenger
class.
         int requestedFuel = queueRef[pump - 1].getCustomerDetails(0).getFuelAmount();

         //Sets the requested fuel to the total fuel amount served in that queue. Used
when calculating income for each queue.
         queueRef[pump - 1].setTotalFuel(requestedFuel);

         //Checks if queue is already empty
         if (queueRef[pump - 1].getCustomerDetails(0).getFirstName().equals("empty"))
{
            System.out.println("\nNo customer in this queue to serve! Please re-check
the queues and try again.\n");

         } else {
            System.out.println("\n" +
                  queueRef[pump - 1].getCustomerDetails(0).getFullName() + "
(Vehicle Number: " +
                  queueRef[pump - 1].getCustomerDetails(0).getVehicleNumber() + ")
from queue " + pump + " was served " +
                  queueRef[pump - 1].getCustomerDetails(0).getFuelAmount() + "
litres of fuel\n"
            );
            //Removes customer using removeCustomer method from FuelQueue class. Slot
is 0 as the customer served is the first customer in the queue.
            queueRef[pump - 1].removeCustomer(0);

            //Checking if waiting queue is empty to add customer to fill the queue
            if (waitingQueue.isEmpty()) {
               System.out.println("\nWaiting Queue is Empty! Therefore no customers
were added to the queue");

               //If waiting queue is empty assigns last index of array list with a new
Passenger object to avoid printing null.
               queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
```

```
"empty", 0));

            } else {
               //If there is a customer in waiting queue added to the fuel queue based
on which queue customer was removed from
               Passenger waitingCustomer = waitingQueue.deQueue();
               queueRef[pump - 1].assignCustomer(waitingCustomer);
               System.out.println(waitingCustomer.getFullName() + " from the waiting
queue was added to fuel queue " + pump);
               stock -= waitingCustomer.getFuelAmount(); //Reducing stock from waiting
queue customer
            }
            System.out.print("\nEnter 'S' to remove another served customer or press
enter to return to the menu: ");
            select = input.nextLine();  //Asks user to serve another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("S"));

      return stock;
   }

   /**
    * This method is to sort customer names of each queue in Alphabetical Order.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void customerSorted(FuelQueue[] queueRef) {
      System.out.println("Viewing Customers sorted in alphabetical order\n");

      //For each loop to print each queue separately.
      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber());
         fuelQueue.customersSorted(); //Calling customersSorted method from Fuel Queue
class.
         System.out.println();
      }
   }

   /**
    * This method stores queue data and fuel stock to a text file using serialization.
    * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
serialization adapted from this link.</a>
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    */
   public static void storeProgramData(FuelQueue[] queueRef, int stock) {
      try {
         FileOutputStream fileOutputStream = new FileOutputStream("Task3_Data.txt");
         ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);

         for (FuelQueue fuelQueue : queueRef) {
            objectOutputStream.writeObject(fuelQueue);
         }
         objectOutputStream.writeObject(stock);
         objectOutputStream.writeObject(waitingQueue);
         objectOutputStream.close();
         System.out.println("Data successfully stored to file!");

      } catch (IOException e) {
```

```
            System.out.println("An error occurred!: " + e);
        }
    }

    /**
     * This method loads the saved file back into the program using deserialization.
     * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
deserialization adapted from this link.</a>
     *
     * @param queueRef FuelQueue object array
     * @param stock    Remaining fuel stock.
     * @return Updated fuel stock.
     */
    public static int loadProgramData(FuelQueue[] queueRef, int stock) {
        try {
            FileInputStream fileInputStream = new FileInputStream("Task3_Data.txt");
            ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

            for (int i = 0; i < queueRef.length; i++) {
                queueRef[i] = (FuelQueue) objectInputStream.readObject();
            }
            stock = (int) objectInputStream.readObject();
            waitingQueue = (WaitingQueue) objectInputStream.readObject();
            objectInputStream.close();
            System.out.println("File successfully loaded!");

        } catch (IOException e) {
            System.out.println("File does not exist!\n");
        } catch (ClassNotFoundException e1) {
            System.out.println("Class not found!\n");
        }
        return stock;
    }

    /**
     * This method is used to add fuel to the existing stock.
     *
     * @param stock Remaining fuel stock.
     * @return Updated fuel stock.
     */
    public static int addFuelStock(int stock) {
        int maxStock = 6600;
        int newStock = validateFuelAmount("to be added to stock");

        //Checking if total stock will be less than 6600 litres to add if not display
error message
        if (stock + newStock <= maxStock) {
            stock += newStock;   //Adds the stock amount added by the user.
            System.out.println(newStock + " litres of fuel added to stock");

        } else {
            System.out.println("WARNING!!! Maximum stock fuel center can hold is 6600
Litres");
            System.out.println("Current stock level is " + stock + " litres.");
        }

        return stock;
    }

    /**
     * This method is used to view the total income of all queues.
     *
     * @param queueRef FuelQueue object array.
```

```
    */
   public static void viewIncome(FuelQueue[] queueRef) {
      System.out.println("Viewing Incomes of all Queues\n");

      //For each loop to display total income of each queue.
      for (FuelQueue fuelQueue : queueRef) {
         //printf used to format income to 2 decimal places. Total fuel from each
queue is taken from getTotalFuel method from FuelQueue class.
         System.out.printf("\tFuel Queue " + fuelQueue.getQueueNumber() + ": Rs.
%.2f\n", (fuelQueue.getTotalFuel() * fuelPrice));
      }
   }

   /**
    * This method is used to validate strings in this code.
    *
    * @param displayMessage The text to display to user when asking for string input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated string name.
    */
   public static String validateString(String displayMessage, String errorMessage) {
      Scanner input = new Scanner(System.in);
      String stringName = "";
      boolean validateString = true;

      while (validateString) {
         System.out.print("Enter " + displayMessage);
         stringName = input.nextLine();

         if (stringName.matches("[a-zA-Z]+") && !(stringName.isEmpty())) {
            validateString = false;
         } else {
            System.out.println(errorMessage + " is in an incorrect format!\n");
         }
      }
      return stringName;
   }

   /**
    * This method is used to validate customer's vehicle number.
    *
    * @return Validated vehicle number.
    */
   public static String validateVehicle() {
      Scanner input = new Scanner(System.in);
      String vehicleName = "";
      boolean validateVehicle = true;

      while (validateVehicle) {
         System.out.print("Enter customer's vehicle number: ");
         vehicleName = input.nextLine();

         if (!(vehicleName.isEmpty())) {
            validateVehicle = false;
         } else {
            System.out.println("Vehicle number cannot be empty! Please re-enter
vehicle number\n");
         }
      }
      return vehicleName;
   }

   /**
```

```
    * This method is used to validate queue number or slot number.
    *
    * @param value          Used to check the range depending on queue (5) or queue
slot (6).
    * @param displayMessage The text to display to user when asking for input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated integer value for queue number or slot number
    */
   public static int validateQueueSlotNumber(int value, String displayMessage, String
errorMessage) {
      Scanner input = new Scanner(System.in);
      int number = 0;
      boolean validateInteger = true;

      while (validateInteger) {
         try {
            System.out.print("From which " + displayMessage);
            number = Integer.parseInt(input.nextLine());

            if (number >= 1 && number <= value) {
               validateInteger = false;
            } else {
               System.out.println(errorMessage + " number has to be between 1 - " +
value + "\n");
            }
         } catch (NumberFormatException e1) {
            System.out.println("Please enter an Integer Value!\n");
         }
      }
      return number;
   }

   /**
    * This method is used to validate the fuel amount entered by user.
    *
    * @param displayMessage The text to display to user when asking for amount of fuel
for customer or adding to stock.
    * @return Validated fuel amount.
    */
   public static int validateFuelAmount(String displayMessage) {
      Scanner input = new Scanner(System.in);
      int fuelAmount = 0;
      boolean validateFuel = true;

      while (validateFuel) {  // If conditions not satisfied program will loop back to
ask for input.
         try {
            System.out.print("Enter amount of fuel " + displayMessage + ": ");
            fuelAmount = Integer.parseInt(input.nextLine());

            if (fuelAmount > 0) {
               validateFuel = false;
            } else {
               System.out.println("Fuel amount has to be greater than 0 litres\n");
            }
         } catch (NumberFormatException e) {
            System.out.println("Amount of fuel has to be an integer!\n");
         }
      }
      return fuelAmount;
   }

   /**
```

```
    * This method checks how many queues are full.
    *
    * @param occupiedSlots Array with available slots of each queue.
    * @return Number of full queues.
    */
   public static int fullQueueChecker(int[] occupiedSlots) {
      int fullQueues = 0;

      for (int fullSlots : occupiedSlots) {
         if (fullSlots == maxQueueSize) {
            fullQueues++;
         }
      }
      return fullQueues;
   }
}
```

Task 4

```
package com.example.task4;

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

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights Reserved.
 * JavaFX for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 4.
 *
 * @author Akindu Karunaratne
 * @version 1.0 2022-08-08.
 **/

public class FuelCenter implements Serializable {
   static final double fuelPrice = 430.00;
   static final int numberOfPumps = 5;
   static final int maxQueueSize = 6;
   static final int waitingQueueSize = 6;
   private static FuelQueue[] fuelQueues = new FuelQueue[numberOfPumps];
   private static WaitingQueue waitingQueue = new WaitingQueue(waitingQueueSize);

   public static FuelQueue[] getFuelQueues(){
      return fuelQueues;
   }
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int fuelStock = 6600;

      initialise(fuelQueues);

      //To check if data exists from a previous run
      File temp = new File("Task3_Data.txt");
      if (temp.exists()) {
         System.out.println("\nWARNING! Data from a previous run has been found. If
you want to load that data enter '107' or 'LPD' in the menu below");
      }

      while (true) {
         //Menu options to be displayed on the console
         System.out.println("""

                 ------- OPTION MENU -------
```

```
                Enter 100 or VFQ to View all Fuel Queues.
                Enter 101 or VEQ to View all Empty Queues.
                Enter 102 or ACQ to Add customer to a Queue.
                Enter 103 or RCQ to Remove a customer from a Queue(From a specific
location)
                Enter 104 or PCQ to Remove a served customer.
                Enter 105 or VCS to View Customers Sorted in alphabetical order.
                Enter 106 or SPD to Store Program Data into file.
                Enter 107 or LPD to Load Program Data from file.
                Enter 108 or STK to View Remaining Fuel Stock.
                Enter 109 or AFS to Add Fuel Stock.
                Enter 110 or IFQ to Print income of a selected queue.
                Enter 111 or GUI to View Fuel Queues in GUI
                Enter 999 or EXT to Exit the Program
                """);

        System.out.print("Enter your option: "); //Taking user input for the menu
option
        String option = input.nextLine();

        System.out.println();

        if (option.equals("100") || option.equalsIgnoreCase("VFQ")) {
           viewAllQueues(fuelQueues); //To view all fuel queues

        } else if (option.equals("101") || option.equalsIgnoreCase("VEQ")) {
           viewEmptyQueues(fuelQueues); //To view fuel queues only with their empty
slots

        } else if (option.equals("102") || option.equalsIgnoreCase("ACQ")) {
           fuelStock = addCustomer(fuelQueues, fuelStock); //Add customer method
called.

        } else if (option.equals("103") || option.equalsIgnoreCase("RCQ")) {
           fuelStock = removeCustomer(fuelQueues, fuelStock); //Remove
customer(specific location) method called.

        } else if (option.equals("104") || option.equalsIgnoreCase("PCQ")) {
           fuelStock = removeServedCustomer(fuelQueues, fuelStock); //Remove served
customer method called.

        } else if (option.equals("105") || option.equalsIgnoreCase("VCS")) {
           customerSorted(fuelQueues);   //customerSorted method called to view
customers sorted in alphabetical order.

        } else if (option.equals("106") || option.equalsIgnoreCase("SPD")) {
           storeProgramData(fuelQueues, fuelStock);   //Stores queue and fuel stock
information to "Task2_Data.txt" file.

        } else if (option.equals("107") || option.equalsIgnoreCase("LPD")) {
           fuelStock = loadProgramData(fuelQueues, fuelStock);    //Loads data from
previous run

        } else if (option.equals("108") || option.equalsIgnoreCase("STK")) {
           viewFuelStock(fuelStock);     //viewFuelStock method called to display
remaining fuel stock.

        } else if (option.equals("109") || option.equalsIgnoreCase("AFS")) {
           fuelStock = addFuelStock(fuelStock); //addFuelStock method called to add
fuel to stock.

        } else if (option.equals("110") || option.equalsIgnoreCase("IFQ")) {
           viewIncome(fuelQueues); //viewIncome method called to view income of all
```

```
fuel queues.

         } else if (option.equals("111") || option.equalsIgnoreCase("GUI")) {
            Task4Application.main(args);

         } else if (option.equals("999") || option.equalsIgnoreCase("EXT")) {
            System.out.println("Exiting the Program...");
            System.exit(0);  //Ending loop to exit program.

         } else {    //Checking if an option from the menu was entered. If not program
will loop back to take user input.
            System.out.println("Invalid Option!!!\n");
         }
      }
   }

   /**
    * This method initialises the FuelQueue object arrays and assigns a passenger
object for each queue
    *
    * @param queueRef FuelQueue object array.
    */
   private static void initialise(FuelQueue[] queueRef) {
      for (int i = 0; i < queueRef.length; i++) {
         queueRef[i] = new FuelQueue(i + 1, maxQueueSize, 0); //Assigns FuelQueue
object to each array index for each pump
         for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
            //Assigns Passenger object to each queue slot in each pump.
            queueRef[i].assignCustomer(new Passenger("empty", "empty", "empty", 0));
         }
      }
   }

   /**
    * This method is to view all fuel queues with customer name if a queue slot is
occupied.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewAllQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Fuel Queues\n");
      System.out.println("-----------------------------------------------------------------
--------------------------------");

      for (FuelQueue fuelQueue : queueRef) {   //Outer loop in for loop which loops
through the rows in fuelQueue array.
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() + "\n");

         for (int i = 0; i < fuelQueue.getQueueSize(); i++) { //Inner loop in for loop
which loops through the columns in fuelQueue array.
            if (fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");

            } else {
               System.out.println("\tQueue slot " + (i + 1) + " is occupied by; ");
               System.out.println(fuelQueue.getCustomerDetails(i));
            }
         }
         System.out.println();
         System.out.println("-----------------------------------------------------------------
----------------------------------");
      }
   }
```

```
   /**
    * This method is to view all the empty slots in the fuel queues.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewEmptyQueues(FuelQueue[] queueRef) {
      System.out.println("Viewing all Empty Queues\n");
      System.out.println("-----------------------------------------------------------------
---------------------------------");

      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber() + "\n");

         for (int i = 0; i < fuelQueue.getQueueSize(); i++) {

            //Check if index in queueRef array has any elements. If not display only
the empty queues
            if (fuelQueue.getCustomerDetails(i).getFirstName().equals("empty")) {
               System.out.println("\tQueue slot " + (i + 1) + " is empty.");
            }
         }
         System.out.println();
         System.out.println("-----------------------------------------------------------
-----------------------------------");
      }

   }

   /**
    * This method is used to add a customer to a fuel queue. If all queues are full
the customer will be added to the waiting queue from this method.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int addCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select;
      int stockWarningLevel = 500;
      int fullSlots = 0;   //Number of queue slots full
      int fullQueues;
      int[] occupiedQueueSlots = new int[numberOfPumps]; //Array to store available
slots in each pump.

      do {
         if (stock <= stockWarningLevel) {
            System.out.println("WARNING! Fuel stock is below 500 Litres!\n");
//Warning message if stock reaches 500 Litres.
         }

         //Validation methods for customers first name, last name, vehicle number and
requested fuel amount called.
         String firstName = validateString("customer's first name: ", "Customer first
name");
         String lastName = validateString("customer's last name: ", "Customer last
name");
         String vehicleNumber = validateVehicle();
         int fuelAmount = validateFuelAmount("requested by customer");

         //For loop to check each fuel queue to check how many slots are full
         for (int i = 0; i < queueRef.length; i++) {
```

```
            for (int j = 0; j < queueRef[i].getQueueSize(); j++) {
               if (!queueRef[i].getCustomerDetails(j).getFirstName().equals("empty"))
{
                  fullSlots++;
               }
            }
            occupiedQueueSlots[i] = fullSlots; //Assigns number of full slots to
relevant pump
            fullSlots = 0;
         }
         int pump = 0;
         int minLength = occupiedQueueSlots[pump]; //Sets fuel pump with minimum queue
length to the first pump by default.

         //For loop to check which pump has the minimum number of full slots (Queue
with minimum length).
         for (int i = 1; i < occupiedQueueSlots.length; i++) {
            if (occupiedQueueSlots[i] < minLength) {
               minLength = occupiedQueueSlots[i];
               pump = i; //Sets pump to queue with minimum length to be used later.
            }
         }
         //Calling method to check how many queues are full
         fullQueues = fullQueueChecker(occupiedQueueSlots);

         //Checks if all queues are full. If so tries to add customer to the waiting
queue.
         if (fullQueues == numberOfPumps) {

            //Check if waiting queue is full. If waiting queue is full user is given
instructions to return to menu.
            if (waitingQueue.isFull()) {
               System.out.println("\nWARNING!!! Waiting Queue is full! Can not add
more customers\n");
               System.out.println("Press enter to return to the menu and serve
customers to add more customers!\n");

            } else {
               //Display message to show user customer was added to waiting queue.
               System.out.println("\nWARNING!!! All queues are full! Customer will be
added to the waiting queue");

               //enQueue method called from WaitingQueue class. Passenger object
passed as a parameter to add new customer in waiting queue.
               waitingQueue.enQueue(new Passenger(firstName, lastName, vehicleNumber,
fuelAmount));
               System.out.println("\n" + firstName + " " + lastName + " was
successfully added to the waiting queue!\n");
            }

         } else {
            for (int i = 0; i < maxQueueSize; i++) {    //Looping through the columns
of the fuelQueue array
               if
(queueRef[pump].getCustomerDetails(i).getFirstName().equals("empty")) { //Checks if
slot is empty to add and replace

                  //Adds customer using setters from Passenger class to pump assigned
from line 210.
                  queueRef[pump].getCustomerDetails(i).setFirstName(firstName);
                  queueRef[pump].getCustomerDetails(i).setLastName(lastName);

queueRef[pump].getCustomerDetails(i).setVehicleNumber(vehicleNumber);
```

```
                    queueRef[pump].getCustomerDetails(i).setFuelAmount(fuelAmount);
                    queueRef[pump].setQueueNumber(pump + 1);
                    stock -= queueRef[pump].getCustomerDetails(i).getFuelAmount();
//Stock updated with requested fuel amount reduced

                    System.out.println("\n" +
queueRef[pump].getCustomerDetails(i).getFullName() + " was successfully added to fuel
queue " + (pump + 1) + "\n");
                    break;
                }
            }
        }
        System.out.print("Enter 'A' to add another customer or press enter to return
to the menu: ");
        select = input.nextLine();  //Asks user to add another customer or exit to
the option menu
        System.out.println();

     } while (select.equalsIgnoreCase("A"));

     return stock;
  }

  /**
   * This method is used to remove a customer from a specific queue location.
   *
   * @param queueRef FuelQueue object array.
   * @param stock    Remaining fuel stock.
   * @return Updated fuel stock.
   */
  public static int removeCustomer(FuelQueue[] queueRef, int stock) {
     Scanner input = new Scanner(System.in);
     String select = "";
     do {
        //Calling validation method for Queue number and Slot number.
        int pump = validateQueueSlotNumber(5, "fuel queue do you want to remove
customer: ", "Queue");
        int slot = validateQueueSlotNumber(6, "queue slot do you want to remove
customer: ", "Slot");

        //Checks if slot is already empty. If so returns user to menu.
        if (queueRef[pump - 1].getCustomerDetails(slot -
1).getFirstName().equals("empty")) {
           System.out.println("\nQueue slot is already empty! Please re-check the
queues and try again.\n");

        } else {
           System.out.println("\n" + queueRef[pump - 1].getCustomerDetails(slot -
1).getFullName() + " was successfully removed from fuel pump "
                 + pump + " queue slot " + slot);

           //Adding back customers requested fuel to stock as customer was not served
           stock += queueRef[pump - 1].getCustomerDetails(slot - 1).getFuelAmount();

           //Removing customer by calling removeCustomer method from FuelQueue class
           queueRef[pump - 1].removeCustomer(slot - 1);

           //Checking if waiting queue is empty to add customer to fill the queue
           if (WaitingQueue.isEmpty()) {
              System.out.println("Waiting Queue is Empty! Therefore no customers were
added to the queue");

              //If waiting queue is empty assigns last index of array list with a new
```

```
Passenger object to avoid printing null.
                queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
"empty", 0));

            } else {
                //If there is a customer in waiting queue added to the fuel queue based
on which queue customer was removed from
                Passenger waitingCustomer = waitingQueue.deQueue();
                queueRef[pump - 1].assignCustomer(waitingCustomer);
                System.out.println(waitingCustomer.getFullName() + " from the waiting
queue was added to fuel queue " + pump);
                stock -= waitingCustomer.getFuelAmount(); //Reducing stock from waiting
queue customer
            }
            System.out.print("\nEnter 'R' to remove another customer from a specific
location or press enter to return to the menu: ");
            select = input.nextLine();  //Asks user to remove another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("R"));

      return stock;
   }

   /**
    * This method is to remove a served customer. If there is a customer in the
waiting queue that customer will be automatically
    * added to the fuel queue from this method.
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int removeServedCustomer(FuelQueue[] queueRef, int stock) {
      Scanner input = new Scanner(System.in);
      String select = "";
      do {
         //Queue number validation method called.
         int pump = validateQueueSlotNumber(5, "fuel queue was customer served?: ",
"Queue");

         //Stores amount of fuel the customer requested using getter from Passenger
class.
         int requestedFuel = queueRef[pump - 1].getCustomerDetails(0).getFuelAmount();

         //Sets the requested fuel to the total fuel amount served in that queue. Used
when calculating income for each queue.
         queueRef[pump - 1].setTotalFuel(requestedFuel);

         //Checks if queue is already empty
         if (queueRef[pump - 1].getCustomerDetails(0).getFirstName().equals("empty"))
{
            System.out.println("\nNo customer in this queue to serve! Please re-check
the queues and try again.\n");
         } else {
            System.out.println("\n" +
                  queueRef[pump - 1].getCustomerDetails(0).getFullName() + "
(Vehicle Number: " +
                  queueRef[pump - 1].getCustomerDetails(0).getVehicleNumber() + ")
from queue " + pump + " was served " +
                  queueRef[pump - 1].getCustomerDetails(0).getFuelAmount() + "
litres of fuel\n"
```

```
            );

            //Removes customer using removeCustomer method from FuelQueue class. Slot
is 0 as the customer served is the first customer in the queue.
            queueRef[pump - 1].removeCustomer(0);

            //Checking if waiting queue is empty to add customer to fill the queue
            if (WaitingQueue.isEmpty()) {
               System.out.println("Waiting Queue is Empty! Therefore no customers were
added to the queue");

               //If waiting queue is empty assigns last index of array list with a new
Passenger object to avoid printing null.
               queueRef[pump - 1].assignCustomer(new Passenger("empty", "empty",
"empty", 0));

            } else {
               //If there is a customer in waiting queue added to the fuel queue based
on which queue customer was removed from
               Passenger waitingCustomer = waitingQueue.deQueue();
               queueRef[pump - 1].assignCustomer(waitingCustomer);
               System.out.println(waitingCustomer.getFullName() + " from the waiting
queue was added to fuel queue " + pump);
               stock -= waitingCustomer.getFuelAmount(); //Reducing stock from waiting
queue customer
            }
            System.out.print("\nEnter 'S' to remove another served customer or press
enter to return to the menu: ");
            select = input.nextLine();  //Asks user to serve another customer or exit
to the option menu
            System.out.println();
         }
      } while (select.equalsIgnoreCase("S"));

      return stock;
   }

   /**
    * This method is to sort customer names of each queue in Alphabetical Order.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void customerSorted(FuelQueue[] queueRef) {
      System.out.println("Viewing Customers sorted in alphabetical order\n");

      //For each loop to print each queue separately.
      for (FuelQueue fuelQueue : queueRef) {
         System.out.println("Fuel Queue " + fuelQueue.getQueueNumber());
         fuelQueue.customersSorted(); //Calling customersSorted method from Fuel Queue
class.
         System.out.println();
      }
   }

   /**
    * This method stores queue data and fuel stock to a text file using serialization.
    * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
serialization adapted from this link.</a>
    *
    * @param queueRef FuelQueue object array.
    * @param stock    Remaining fuel stock.
    */
   public static void storeProgramData(FuelQueue[] queueRef, int stock) {
```

```
      try {
         FileOutputStream fileOutputStream = new FileOutputStream("Task3_Data.txt");
         ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);

         for (FuelQueue fuelQueue : queueRef) {
            objectOutputStream.writeObject(fuelQueue);
         }
         objectOutputStream.writeObject(stock);
         objectOutputStream.writeObject(waitingQueue);
         objectOutputStream.close();
         System.out.println("Data successfully stored to file!");

      } catch (IOException e) {
         System.out.println("An error occurred!: " + e);
      }
   }

   /**
    * This method loads the saved file back into the program using deserialization.
    * <a href="https://www.geeksforgeeks.org/serialization-in-java/">Code for
deserialization adapted from this link.</a>
    *
    * @param queueRef FuelQueue object array
    * @param stock    Remaining fuel stock.
    * @return Updated fuel stock.
    */
   public static int loadProgramData(FuelQueue[] queueRef, int stock) {
      try {
         FileInputStream fileInputStream = new FileInputStream("Task3_Data.txt");
         ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

         for (int i = 0; i < queueRef.length; i++) {
            queueRef[i] = (FuelQueue) objectInputStream.readObject();
         }
         stock = (int) objectInputStream.readObject();
         waitingQueue = (WaitingQueue) objectInputStream.readObject();
         objectInputStream.close();
         System.out.println("File successfully loaded!");

      } catch (IOException e) {
         System.out.println("File does not exist!\n");
      } catch (ClassNotFoundException e1) {
         System.out.println("Class not found!\n");
      }
      return stock;
   }

   /**
    * This method is used to view the remaining fuel stock.
    *
    * @param stock Remaining fuel stock.
    */
   public static void viewFuelStock(int stock) {
      System.out.println(stock + " Litres of Fuel remaining"); //Displays remaining
fuel stock.
   }

   /**
    * This method is used to add fuel to the existing stock.
    *
    * @param stock Remaining fuel stock.
    * @return Updated fuel stock.
```

```
    */
   public static int addFuelStock(int stock) {
      int maxStock = 6600;
      int newStock = validateFuelAmount("to be added to stock");

      //Checking if total stock will be less than 6600 litres to add if not display
error message
      if (stock + newStock <= maxStock) {
         stock += newStock;   //Adds the stock amount added by the user.
         System.out.println(newStock + " litres of fuel added to stock");

      } else {
         System.out.println("WARNING!!! Maximum stock fuel center can hold is 6600
Litres");
         System.out.println("Current stock level is " + stock + " litres.");
      }

      return stock;
   }

   /**
    * This method is used to view the total income of all queues.
    *
    * @param queueRef FuelQueue object array.
    */
   public static void viewIncome(FuelQueue[] queueRef) {
      System.out.println("Viewing Incomes of all Queues\n");

      //For each loop to display total income of each queue.
      for (FuelQueue fuelQueue : queueRef) {
         //printf used to format income to 2 decimal places. Total fuel from each
queue is taken from getTotalFuel method from FuelQueue class.
         System.out.printf("\tFuel Queue " + fuelQueue.getQueueNumber() + ": Rs.
%.2f\n", (fuelQueue.getTotalFuel() * fuelPrice));
      }
   }

   /**
    * This method is used to validate strings in this code.
    *
    * @param displayMessage The text to display to user when asking for string input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated string name.
    */
   public static String validateString(String displayMessage, String errorMessage) {
      Scanner input = new Scanner(System.in);
      String stringName = "";
      boolean validateString = true;

      while (validateString) {
         System.out.print("Enter " + displayMessage);
         stringName = input.nextLine();

         if (stringName.matches("[a-zA-Z]+") && !(stringName.isEmpty())) {
            validateString = false;
         } else {
            System.out.println(errorMessage + " is in an incorrect format!\n");
         }
      }
      return stringName;
   }

   /**
```

```
    * This method is used to validate customer's vehicle number.
    *
    * @return Validated vehicle number.
    */
   public static String validateVehicle() {
      Scanner input = new Scanner(System.in);
      String vehicleName = "";
      boolean validateVehicle = true;

      while (validateVehicle) {
         System.out.print("Enter customer's vehicle number: ");
         vehicleName = input.nextLine();

         if (!(vehicleName.isEmpty())) {
            validateVehicle = false;
         } else {
            System.out.println("Vehicle number cannot be empty! Please re-enter
vehicle number\n");
         }
      }
      return vehicleName;
   }

   /**
    * This method is used to validate queue number or slot number.
    *
    * @param value          Used to check the range depending on queue (5) or queue
slot (6).
    * @param displayMessage The text to display to user when asking for input.
    * @param errorMessage   Text to display if error occurs.
    * @return Validated integer value for queue number or slot number
    */
   public static int validateQueueSlotNumber(int value, String displayMessage, String
errorMessage) {
      Scanner input = new Scanner(System.in);
      int number = 0;
      boolean validateInteger = true;

      while (validateInteger) {
         try {
            System.out.print("From which " + displayMessage);
            number = Integer.parseInt(input.nextLine());

            if (number >= 1 && number <= value) {
               validateInteger = false;
            } else {
               System.out.println(errorMessage + " number has to be between 1 - " +
value + "\n");
            }
         } catch (NumberFormatException e1) {
            System.out.println("Please enter an Integer Value!\n");
         }
      }
      return number;
   }

   /**
    * This method is used to validate the fuel amount entered by user.
    *
    * @param displayMessage The text to display to user when asking for amount of fuel
for customer or adding to stock.
    * @return Validated fuel amount.
    */
```

```
   public static int validateFuelAmount(String displayMessage) {
      Scanner input = new Scanner(System.in);
      int fuelAmount = 0;
      boolean validateFuel = true;

      while (validateFuel) {  // If conditions not satisfied program will loop back to
ask for input.
         try {
            System.out.print("Enter amount of fuel " + displayMessage + ": ");
            fuelAmount = Integer.parseInt(input.nextLine());

            if (fuelAmount > 0) {
               validateFuel = false;
            } else {
               System.out.println("Fuel amount has to be greater than 0 litres\n");
            }
         } catch (NumberFormatException e) {
            System.out.println("Amount of fuel has to be an integer!\n");
         }
      }
      return fuelAmount;
   }

   /**
    * This method checks how many queues are full.
    *
    * @param occupiedSlots Array with available slots of each queue.
    * @return Number of full queues.
    */
   public static int fullQueueChecker(int[] occupiedSlots) {
      int fullQueues = 0;

      for (int fullSlots : occupiedSlots) {
         if (fullSlots == maxQueueSize) {
            fullQueues++;
         }
      }
      return fullQueues;
   }
}
```

FuelQueue and Passenger classes for Task 2,3 and 4

FuelQueue.java

```
import java.io.Serializable;
import java.util.ArrayList;
public class FuelQueue implements Serializable {

   //Attributes
   private final ArrayList<Passenger> customerQueue;
   private int queueNumber;
   private final int queueSize;
   private int totalFuel;

   //Constructor
   public FuelQueue(int queueNumber, int queueSize, int totalFuel) {
      this.queueNumber = queueNumber;
      this.totalFuel = totalFuel;
      this.queueSize = queueSize;
      this.customerQueue = new ArrayList<>(queueSize);
   }
   public Passenger getCustomerDetails(int queueSlot) {
      return customerQueue.get(queueSlot);
   }
   public int getQueueNumber() {
      return queueNumber;
   }
   public void setQueueNumber(int queueNumber) {
      this.queueNumber = queueNumber;
   }
   public int getQueueSize() {
      return queueSize;
   }
   public int getTotalFuel() {
      return totalFuel;
   }
   public void setTotalFuel(int totalFuel) {
      this.totalFuel += totalFuel;
   }
   public void assignCustomer(Passenger customer) {
      customerQueue.add(customer);
   }
   public void removeCustomer(int slot) {
      customerQueue.remove(slot);
   }
   //Method to sort customers in alphabetical order
   public void customersSorted() {
      ArrayList<String> sortedCustomers = new ArrayList<>();

      for (Passenger customer : customerQueue) {
         if (!customer.getFirstName().equals("empty")) {
            sortedCustomers.add(customer.getFullName());
         }
      }
      sortedCustomers.sort(String.CASE_INSENSITIVE_ORDER);

      for (String sortedCustomer : sortedCustomers) {
         if (!(sortedCustomer.equals("empty"))) {
            System.out.println("\t" + sortedCustomer);
         }
      }
   }
```

Passenger.java

```
import java.io.Serializable;

public class Passenger implements Serializable {

   //Attributes
   private String firstName;
   private String lastName;
   private String vehicleNumber;
   private int fuelAmount;

   //Constructor
   public Passenger(String firstName, String lastName, String vehicleNumber,
int fuelAmount) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.vehicleNumber = vehicleNumber;
      this.fuelAmount = fuelAmount;
   }
   public String getFirstName() {
      return this.firstName;
   }
   public String getFullName() {
      return firstName + " " + lastName;
   }
   public String getVehicleNumber() {
      return this.vehicleNumber;
   }
   public int getFuelAmount() {
      return this.fuelAmount;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
   public void setVehicleNumber(String vehicleNumber) {
      this.vehicleNumber = vehicleNumber;
   }
   public void setFuelAmount(int fuelAmount) {
      this.fuelAmount = fuelAmount;
   }

   //Overriding the toString() method
   public String toString(){
      return "\t\tFirst Name     :  " + firstName +
             "\n\t\tLast Name      :  " + lastName +
             "\n\t\tVehicle Number :  " + vehicleNumber +
             "\n\t\tAmount of Fuel :  " + fuelAmount + " litres\n";
   }
}
```

WaitingQueue class used for Task 3 and Task 4

WaitingQueue.java

```
import java.io.Serializable;

public class WaitingQueue implements Serializable {

   //Attributes
   private int front;
   private int rear;
   private final int maxSize;
   private int numCustomers;
   private final Passenger[] waitingQueue;

   //Constructor
   public WaitingQueue(int maxSize) {
      this.maxSize = maxSize;
      this.waitingQueue = new Passenger[maxSize];
      this.front = 0;
      this.rear = -1;
      this.numCustomers = 0;
   }

   //Method to add customer to waiting queue
   public void enQueue(Passenger data) {
      rear = (rear + 1) % maxSize;
      waitingQueue[rear] = data; //Adding Passenger object to array index
with rear pointer.
      numCustomers++; //Increases number of customers by 1
   }

   //Method to remove customer from waiting queue
   public Passenger deQueue() {
      Passenger temp = waitingQueue[front]; //Getting Passenger object from
front pointer
      front = (front + 1) % maxSize;
      numCustomers--; //Decreases number of customers by 1

      return temp;
   }

   //Methods to check if waiting queue is empty or full
   public boolean isEmpty(){
      return (numCustomers == 0);
   }
   public boolean isFull(){
      return (numCustomers == maxSize);
   }

}
```

Task4Application, Task4Controller and Task4.fxml used for GUI

Task4.fxml

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

<?import javafx.scene.Cursor?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<!-->AnchorPane.setMinSize(20,20);-->

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="550.0" prefWidth="750.0" style="-fx-background-color:
#000;" xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.task4.Task4Controller">
   <children>
      <Label alignment="CENTER" prefHeight="100.0" prefWidth="750.0" style="-fx-
background-color: #a6051a;" text="FUEL QUEUE MANAGEMENT SYSTEM" textFill="WHITE">
         <font>
            <Font name="System Bold" size="24.0" />
         </font>
      </Label>
      <TextField fx:id="searchBar" layoutX="14.0" layoutY="112.0" prefHeight="47.0"
prefWidth="226.0" promptText="Search Customer by First Name" style="-fx-background-
color: grey;">
         <font>
            <Font size="14.0" />
         </font>
         <cursor>
            <Cursor fx:constant="DEFAULT" />
         </cursor></TextField>
      <TextArea fx:id="fuelQueues" editable="false" layoutX="8.0" layoutY="242.0"
prefHeight="293.0" prefWidth="369.0" style="-fx-background-color: #87ceeb;">
         <font>
            <Font name="Calibri" size="14.0" />
         </font></TextArea>
      <Button layoutX="395.0" layoutY="113.0" mnemonicParsing="false"
onAction="#viewQueues" prefHeight="41.0" prefWidth="330.0" style="-fx-background-
color: #28282b; -fx-border-color: red; -fx-border-width: 2px;" text="View Queue
Details" textFill="WHITE">
         <font>
            <Font name="Calibri" size="23.0" />
         </font>
         <cursor>
            <Cursor fx:constant="HAND" />
         </cursor></Button>
      <Button layoutX="250.0" layoutY="112.0" mnemonicParsing="false"
onAction="#searchCustomer" prefHeight="47.0" prefWidth="83.0" style="-fx-background-
color: #28282b; -fx-border-color: red; -fx-border-width: 1.5px;" text="Search"
textFill="WHITE">
         <font>
            <Font name="Calibri" size="18.0" />
         </font>
         <cursor>
            <Cursor fx:constant="HAND" />
```

```
         </cursor>
      </Button>
      <TextArea fx:id="waitingQueue" editable="false" layoutX="375.0" layoutY="242.0"
prefHeight="293.0" prefWidth="369.0" style="-fx-background-color: #87ceeb;">
         <font>
            <Font name="Calibri" size="14.0" />
         </font>
      </TextArea>
      <Label alignment="CENTER" layoutX="132.0" layoutY="201.0" prefHeight="25.0"
prefWidth="150.0" text="Fuel Queues" textFill="WHITE">
         <font>
            <Font name="Calibri" size="24.0" />
         </font>
      </Label>
      <Label alignment="CENTER" layoutX="460.0" layoutY="196.0" prefHeight="40.0"
prefWidth="200.0" text="Waiting Queue" textFill="WHITE">
         <font>
            <Font name="Calibri" size="24.0" />
         </font>
      </Label>
   </children>
</AnchorPane>
```

Task4Application.java

```
package com.example.task4;

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

import java.io.*;

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights Reserved.
 * JavaFX for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 4.
 *
 * @author Akindu Karunaratne
 * @version 1.0 2022-08-08.
 **/

public class Task4Application extends Application{
   @Override
   public void start(Stage stage) throws IOException {
      FXMLLoader fxmlLoader = new
FXMLLoader(Task4Application.class.getResource("Task4.fxml"));
      Scene scene = new Scene(fxmlLoader.load(), 750, 550);
      stage.setTitle("Fuel Center");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) {
      launch();
   }
}
```

Task4Controller.java

```
package com.example.task4;

import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

/**
 * COPYRIGHT (C) 2022 Akindu Karunaratne (w1898951/20211364). All Rights Reserved.
 * JavaFX for a Fuel Queue Management System in a fuel center.
 * Solves Software Development 2 (4COSC010.3) Coursework 1 Task 4.
 *
 * @author Akindu Karunaratne
 * @version 1.0 2022-08-08.
 **/
public class Task4Controller {
   @FXML
   private TextArea fuelQueues;
   @FXML
   private TextArea waitingQueue;
   @FXML
   private TextField searchBar;

   @FXML
   protected void viewQueues() {
      String queueDetails = "";  //String to store queue details
      String waitingDetails = "";   //String to store waiting queue details.
      FuelQueue[] queueNames = FuelCenter.getFuelQueues();  //Getting Fuel Queue data
from Main class
      Passenger[] waitingNames = WaitingQueue.getWaitingQueue(); //Getting Waiting
queue data from Main class

      for (int i = 0; i < FuelCenter.numberOfPumps; i++) {   //Outer loop in for loop
which loops through the rows in fuelQueue array.
         queueDetails += "Fuel Queue " + (i + 1) + "\n";
         for (int j = 0; j < FuelCenter.maxQueueSize; j++) { //Inner loop in for loop
which loops through the columns in fuelQueue array.
            if (queueNames[i].getCustomerDetails(j).getFirstName().equals("empty")) {
               queueDetails += "\tQueue slot " + (j + 1) + " is empty.\n";

            } else {
               queueDetails += "\tQueue slot " + (j + 1) + " is occupied by; \n";
               queueDetails += queueNames[i].getCustomerDetails(j);
            }
            queueDetails += "\n";
         }
         fuelQueues.setText(queueDetails);
      }
      if (!WaitingQueue.isEmpty()) {
         for (int i = 0; i < waitingNames.length; i++) {
            if (waitingNames[i] != null) {
               waitingDetails += "Customer " + (i + 1) + "\n";
               waitingDetails += "\tFirst name           : " +
waitingNames[i].getFirstName() + "\n";
               waitingDetails += "\tLast name            : " +
waitingNames[i].getLastName() + "\n";
               waitingDetails += "\tVehicle number  : " +
waitingNames[i].getVehicleNumber() + "\n";
               waitingDetails += "\tAmount of fuel   : " +
waitingNames[i].getFuelAmount() + " litres\n";
               waitingDetails += "\n";
```

```
            }
         }
      } else {
         waitingDetails += "Waiting queue is empty! \n";
      }
      waitingQueue.setText(waitingDetails);
   }

   @FXML
   protected void searchCustomer() {
      String queueDetails = "";
      String waitingDetails = "";
      boolean isQueueNameFound = false;
      boolean isWaitingNameFound = false;

      //Setting text areas to empty to display the results
      fuelQueues.setText("");
      waitingQueue.setText("");
      FuelQueue[] queueNames = FuelCenter.getFuelQueues();
      Passenger[] waitingNames = WaitingQueue.getWaitingQueue();

      String customerName = searchBar.getText(); //Getting input from user

      for (int i = 0; i < FuelCenter.numberOfPumps; i++) {
         for (int j = 0; j < FuelCenter.maxQueueSize; j++) {
            //Checking if customer found in fuel queue
            if
(queueNames[i].getCustomerDetails(j).getFirstName().equalsIgnoreCase(customerName)) {
               queueDetails += "Fuel Queue " + queueNames[i].getQueueNumber() + "\n";
               queueDetails += queueNames[i].getCustomerDetails(j) + "\n";
               isQueueNameFound = true;
            }
         }
      }
      for (int x = 0; x < waitingNames.length; x++) {
         //Checking if customer found in waiting queue
         if (waitingNames[x] != null &&
waitingNames[x].getFirstName().equalsIgnoreCase(customerName)) {
            waitingDetails += "Customer " + (x + 1) + "\n";
            waitingDetails += "\tFirst name           : " +
waitingNames[x].getFirstName() + "\n";
            waitingDetails += "\tLast name            : " +
waitingNames[x].getLastName() + "\n";
            waitingDetails += "\tVehicle number   : " +
waitingNames[x].getVehicleNumber() + "\n";
            waitingDetails += "\tAmount of fuel    : " +
waitingNames[x].getFuelAmount() + " litres\n";
            waitingDetails += "\n";
            isWaitingNameFound = true;
         }
      }
      //Error if customer not found in fuel queue
      if (!isQueueNameFound){
         queueDetails += "Customer not found in fuel queue\n";
      }
      //Error if customer not found in waiting queue.
      if (!isWaitingNameFound){
         waitingDetails += "Customer not found in waiting queue\n";
      }
      fuelQueues.setText(queueDetails);
      waitingQueue.setText(waitingDetails);
   }
}
```