

Informatics Institute of Technology

Department of Computing

Software Development II

4COSC010C.3

Coursework Report

Name : Rayaana Rilshad

IIT Student ID : 20211315

UoW ID : w1899330

Module Leader : Deshan Sumanathilaka

Date of submission : 2022/08/08

"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 : Rayaana Rilshad  
Student ID : 20211315 / w1899330

# Table of Contents

1. Test Cases.....	4
1.1. Task 1 test cases : Arrays Version .....	4
1.2. Task 2 test cases : Classes Version .....	6
1.3. Task 3 test cases : Waiting Queue .....	11
1.4. Task 4 test cases : JavaFX .....	11
2. Discussion .....	12
3. Code.....	13
3.1. Arrays Version.....	13
3.2. Classes Version.....	18
3.2.1. Main Class.....	18
3.2.2. FuelQueue Class.....	25
3.2.3. Passenger Class .....	26
3.2.4. WaitingQueue Class .....	28

## 1. Test Cases

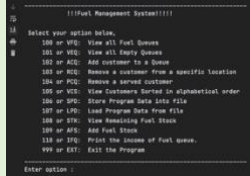
### 1.1. Task 1 test cases : Arrays Version

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Display all the menu options to the operator when program starts	Display menu option	Display menu option	Pass
2	Fuel Queue Initialized Correctly After program starts, 100 or VFQ	Displays all queues with no passengers.	Pump 1 Queue List = [null, null, null, null, null, null] Pump 2 Queue List = [null, null, null, null, null, null] Pump 3 Queue List = [null, null, null, null, null, null]	Pass
3	View all Queues 100 or VFQ	Rayaan and John in pump 1, Chamath and Ravindu in pump 2, Grande and Seevalee in pump 3	Pump 1 Queue List = [Rayaan, John, Aqeel, null, null, null] Pump 2 Queue List = [Chamath, Ravindu, null, null, null, null] Pump 3 Queue List = [Grande, Seevalee, null, null, null, null]	Pass
4	101 or VEQ before add customer	Display “Pump 1 is Empty, Pump 2 is Empty, Pump 3 is Empty”	Display “Pump 1 is Empty, Pump 2 is Empty, Pump 3 is Empty”	Pass
5	101 or VEQ after add customer to queue 1	Display “Pump 2 is Empty, Pump 3 is Empty”	Pump 2 is Empty Pump 3 is Empty	Pass
6	Add passenger “Jane” to Queue 2 102 or ACQ Enter Queue: 2 Enter Name: Jane	Add Jane to the queue 2	Pump 2 Queue List = [Jane, null, null, null, null, null]	Pass
7	Remove a customer from a specific location 103 or RCQ Enter Queue: 2 Enter specific position: 1	Display “Customer removed” &	Customer removed & Pump 1 Queue List = [Rayaan, John,	Pass

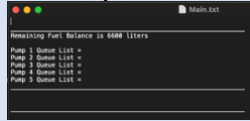
		Ravindu should be removed from the queue 2	Aqeel, null, null, null] Pump 2 Queue List = [Chamath, null, null, null, null, null] Pump 3 Queue List = [Grande, Seevalee, null, null, null, null]	
8	Remove a served customer 104 or PCQ Enter Queue: 1	Display “Served Customer Removed” & Rayaan should be removed from the queue 1	Served Customer Removed & Pump 1 Queue List = [John, Aqeel, null, null, null, null]	Pass
9	Alphabetical sort 105 or VCS Enter Queue: 1	[Aqeel, John, Rayaan, null, null, null]	Pump 1 Alphabetic list = [Aqeel, John, Rayaan, null, null, null]	Pass
10	Store program data info file after adding customers 106 or SPD	Display “Successfully wrote to the file” & wrote on the text file	Remaining Fuel Balance is 6530 liters  Pump 1 Queue List = [Rayaan, Jhon, Aqeel, null, null, null] Pump 2 Queue List = [Chamath, null, null, null, null, null] Pump 3 Queue List = [Grande, Seevalee, null, null, null, null]	Pass
11	Load program data from file after adding customers 107 or LPD	View Remaining fuel balance, Queue lists	Remaining Fuel Balance is 6530 liters  Pump 1 Queue List = [Rayaan, Jhon, Aqeel, null, null, null] Pump 2 Queue List = [Chamath, null, null, null, null, null] Pump 3 Queue List = [Grande, Seevalee, null, null, null, null]	Pass

12	View remaining fuel stock after adding 10 peoples and specific remove 2 108 or STK	Display 6600 - (10*10) + (10*2) as remaining fuel	Remaining Fuel Stock is 6520 liter	Pass
13	Add fuel stock 109 or AFS Enter adding stock: 400	Display “400 liters added to the Stock” and 6520+400 as fuel stock	Remaining Fuel Stock is 6920 liter	Pass
14	Exit the program 999 or EXT	Exit from the program	Exit from the program	Pass


## 1.2. Task 2 test cases : Classes Version

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Display all the menu options to the operator when program starts	Display menu option		Pass
2	Fuel Queue Initialized Correctly After program starts, 100 or VFQ (Before adding customers)	Displays all queues without passengers.	Pump 1 Queue List = Pump 2 Queue List = Pump 3 Queue List = Pump 4 Queue List = Pump 5 Queue List = Waiting queue is empty	Pass
3	View all fuel queues (After adding customers) 100 or VFQ	Display all queues with passengers	Pump 1 Queue List = Jane a, Kiara a, Pump 2 Queue List = Ariana a, Chamath a, Pump 3 Queue List = Rayaana a, Dinesh a, Pump 4 Queue List = Tharindu a, Pump 5 Queue List = Alvin,	Pass
4	View all fuel queues (After removing customers) 100 or VFQ	Display all queues with passengers	Displayed all queues with passengers Except waiting queue	Fail
5	View all empty queues (Before adding customers) 101 or VEQ	Display “Pump +Pump Number+ is Empty” for all queues	Pump 1 is empty Pump 2 is empty Pump 3 is empty Pump 4 is empty	Pass

			Pump 5 is empty	
6	View all empty queues (After adding 2 customers) 101 or VEQ	Display queues 3,4,5 as empty and 1,2 as not empty	Pump 1 is not empty Pump 2 is not empty Pump 3 is empty Pump 4 is empty Pump 5 is empty	Pass
7	Add passenger “Jane” to Queue 2 102 or ACQ Enter First Name: Jane Enter First Name: Paul Enter vehicle No: AB-1234 Enter Fuel Amount: 5	Display “Jane Paul added to Pump 1 queue successfully”	Jane Paul added to Pump 1 queue successfully	Pass
8	Select the queue with the minimum length on 102 or ACQ Enter First Name: Ariana Enter First Name: Grande Enter vehicle No: AA-4567 Enter Fuel Amount: 8	Display “Ariana Grande added to Pump 2 queue successfully”	Ariana Grande added to Pump 2 queue successfully	Pass
9	Further checking selects the queue with the minimum length on 102 or ACQ Adding Jane, Ariana, Rayaana, Tharindu, Alvin, Kiara, Chamath, Dinesh orderly	Jane, Kiara in queue 1 Ariana, Chamath in queue 2 Rayaana, Dinesh in queue 3 Tharindu in queue 4 Alvin in queue 5	Pump 1 Queue List = Jane a, Kiara a, Pump 2 Queue List = Ariana a, Chamath a, Pump 3 Queue List = Rayaana a, Dinesh a, Pump 4 Queue List = Tharindu a, Pump 5 Queue List = Alvin a,	Pass
10	Wrong input for fuel amount 102 or ACQ Enter First Name: Jane Enter First Name: Paul Enter vehicle No: AB-1234 Enter Fuel Amount: aaa	Display “Enter a valid fuel amount”	Enter a valid fuel amount	Pass
11	Remove a customer from an empty location (Before adding customers) 103 or RCQ Enter Queue: 3 Enter specific position: 5	Display “Customer not available in the Pump”	Customer not available in the Pump	Pass
12	Remove a customer from a specific location 103 or RCQ Enter Queue: 3 Enter specific position: 2	Jane, Kiara in queue 1 Ariana, Chamath in queue 2 Rayaana, Dinesh in queue 3	Pump 3, queue member 2 is Successfully Removed	Pass

		<p>Tharindu in queue 4 Alvin in queue 5</p> <p>Dinesh should be removed</p>		
13	<p>Wrong input for 103 or RCQ Pump select: d and 8</p>	<p>Display “Invalid, Enter a number (1-5)” on d and “Invalid Pump” on 8</p>	<p>Invalid, Enter a number (1-5) &amp; Invalid Pump</p>	Pass
14	<p>Remove a served customer from an empty queue (Before adding customers) 104 or PCQ Enter Queue: 4</p>	<p>Display “No customers available in Pump 4”</p>	<p>No customers available in Pump 4</p>	Pass
15	<p>Remove a served customer 104 or PCQ Enter Queue: 1</p>	<p>Display “Pump 1 customer is Served Successfully” and removed Jane from queue 1</p>	<p>Pump 1 customer is Served Successfully</p>	Pass
16	<p>Wrong input for 104 or PCQ Pump select: g and 6</p>	<p>Display “Invalid, Enter a number (1-5)” on g and “Invalid Pump” on 6</p>	<p>Invalid, Enter a number (1-5) &amp; Invalid Pump</p>	Pass
17	<p>Alphabetical sort (Before adding customers) 105 or VCS</p>	<p>Display empty queues</p>	<p>Pump 1 queue in alphabetical order = [] Pump 2 queue in alphabetical order = [] Pump 3 queue in alphabetical order = [] Pump 4 queue in alphabetical order = [] Pump 5 queue in alphabetical order = []</p>	Pass
18	<p>Alphabetical sort after adding customers 105 VCS</p>	<p>Display all names in alphabetical order</p>	<p>All named displayed in alphabetical order</p>	Pass
19	<p>Store program data info file (Before adding customers) 106 or SPD</p>	<p>Display “Successfully wrote to the file” and wrote on the text file</p>	<p>Successfully wrote to the file</p> 	Pass



20	Store program data info file after adding customers 106 or SPD	Display “Successfully wrote to the file” and wrote on the text file With all customer information		Pass
21	Load program data from file (Before adding customers) 107 or LPD	Load & display the template which added on store program data	<p>Remaining Fuel Balance is 6600 liters</p> <p>Pump 1 Queue List = Pump 2 Queue List = Pump 3 Queue List = Pump 4 Queue List = Pump 5 Queue List =</p>	Pass
22	Load program data from file after adding customers 107 or LPD	View Remaining fuel balance, Queue lists and all user information with first name, second name, vehicle number and required fuel amount	<p>Remaining Fuel Balance is 6555 liters</p> <p>Pump 1 Queue List = Jane Paul, Kiara Advani, Pump 2 Queue List = Ariana Grande, Chamath Aari, Pump 3 Queue List = Rayaan Rilshad, Dinesh Karthik, Pump 4 Queue List = Tharindu Kasun, Pump 5 Queue List = Alvin Bernard,</p> <p>&lt;-- Details of Jane Paul --&gt; First Name : Jane Second Name : Paul Vehicle Number : sdfgv Required Fuel Amount : 7</p> <p>&lt;-- Details of Kiara Advani --&gt; First Name : Kiara Second Name : Advani Vehicle Number : sdfg Required Fuel Amount : 3</p> <p>&lt;-- Details of Ariana Grande --&gt; First Name : Ariana Second Name : Grande Vehicle Number : sdgf Required Fuel Amount : 9</p> <p>&lt;-- Details of Chamath Aari --&gt; First Name : Chamath Second Name : Aari Vehicle Number : sdfb Required Fuel Amount : 3</p> <p>&lt;-- Details of Rayaan Rilshad --&gt; First Name : Rayaan Second Name : Rilshad Vehicle Number : dsg Required Fuel Amount : 6</p> <p>&lt;-- Details of Dinesh Karthik --&gt;</p>	Pass

			First Name : Dinesh Second Name : Karthik Vehicle Number : asdv Required Fuel Amount : 3  <-- Details of Tharindu Kasun --> First Name : Tharindu Second Name : Kasun Vehicle Number : sdgvb Required Fuel Amount : 5  <-- Details of Alvin Bernard --> First Name : Alvin Second Name : Bernard Vehicle Number : sd Required Fuel Amount : 9	
23	View remaining fuel stock (Before adding customers) 108 or STK	Display “Remaining Fuel Stock is 6600 liters”	Remaining Fuel Stock is 6600 liters	Pass
24	View remaining fuel stock after adding customers (Above program data detail) 108 or STK	6600 – (7+3+9+3+6+3+5+9)	Remaining Fuel Stock is 6555 liters	Pass
25	View fuel stock after adding stock 108 or STK (After adding fuel stock 400l)	Display “Remaining Fuel Stock is 7000 liters”	Remaining Fuel Stock is 7000 liters	Pass
26	Add fuel stock 109 or AFS Enter adding stock: 400	Display “400 liters added to the Stock”	400 liters added to the Stock	Pass
27	Wrong fuel stock for 109 or AFS Enter adding stock: abc	Display “Invalid input, add only numbers”	Invalid input, add only numbers	Pass
28	Print the income of all fuel queues (Before serving customers) 110 or IFQ Enter Option: 0	Display all pump (Rs.0 income)	Pump 1 income is Rs.0 Pump 2 income is Rs.0 Pump 3 income is Rs.0 Pump 4 income is Rs.0 Pump 5 income is Rs.0	Pass
29	Print the income of all fuel queues (After serving customers) 110 or IFQ Served two customers in pump 2 and one customer in pump 4	(0)*430 for pump1, (9+3)*430 for pump2, (0)*430 for pump3, (5)*430 for pump4, (0)*430 for pump5	Pump 1 income is Rs.0 Pump 2 income is Rs.5160 Pump 3 income is Rs.0 Pump 4 income is Rs.2150 Pump 5 income is Rs.0	Pass
30	Warning message when the stock reaches a value of 500 liters. (By adding a customer with 6100l fuel requirement)	Warning message when each time user added	!!!WARNING!!! Fuel Stock is less than 500 liters (Remaining Stock : 492)	Pass

31	Exit the program 999 or EXT	Exit from the program	Exit from the program	Pass
32	Non menu bar keys/choices 456 or ABC	Display “Invalid Key”	Invalid Key	Pass

### 1.3. Task 3 test cases : Waiting Queue

\*Before testing this, I filled all fuel pump queues with customers

	<b>Test Case</b>	<b>Expected Result</b>	<b>Actual Result</b>	<b>Pass/Fail</b>
1	Adding customers to the waiting queue automatically when all fuel queues are filled 102 or ACQ	Display “Christiano Ronaldo added to the waiting queue”	Christiano Ronaldo added to the waiting queue	Pass
2	Automatically replace a fuel queue customer with first waiting queue customer when fuel queue customer is removed. 104 or PCQ Enter remove Queue: 2 (Waiting queue first customer is Christiano Ronaldo)	Display “Pump 2 customer is Served Successfully” and show Christiano Ronaldo in queue 2 at 6th position	Pump 2 Queue List = Ariana a, Chamath a, Hasini a, Vijay a, Rilshad a, Christiano Ronaldo	Pass
3	Queue is full (Add more than 7 passengers to the waiting queue)	Display “Waiting Queue is Full”	Waiting Queue is Full	Pass
4	Circular queue	After waiting queue is filled, removing a person will give a space to add another customer instead of giving waiting queue is full message		

### 1.4. Task 4 test cases : JavaFX

## 2. Discussion

Running without errors is the top priority in a program. So, to reduce the errors, it is important to find them. That's why we use test cases. It will help to find errors and go deeply into them and correct them. So, using a better testing technique will help the tester and the programmer. In this program test case, I tested it using three criteria. That is checking the program before adding/removing customers, entering wrong inputs, and then testing after adding and removing customers.

Before everything, the program should initialize properly, load and run and be able to show the menu options to the users without any errors. So first of all, I tested that and then I go through my techniques.

Firstly, I checked all the options, before adding the customers to the queue. By performing this I should be able to check the errors of the options that can be occurred when using empty queues. As well as I can check the wrong outputs of the options. As an example, when there is an empty queue, there is no way to remove a customer from a queue. So, if I try to remove a customer from an empty queue, the program should be giving the exact error in an invalid task such as "No customer available in the pump" instead of giving traditional commands which are using in every valid tasks, like "Customer removed successfully"

Then, I went through all the options and entered wrong inputs such as added strings that need integers, added pump numbers as 10 while only 5 pumps were available in the fuel station. So, by performing this we can be able to catch `InputOutOfRangeException` and `InputMismatchException` errors. So, we can test, whether the program is running smoothly, or it is interrupted while using, by giving a wrong command. And we can check the program is giving exact commands for the user in an invalid task, instead of getting the wrong input and giving wrong command and do nothing or messing up in the program.

Finally, I tested out after adding customers to the queues. Previous two methods will check only the errors and the wrong commands on the program. But this will be checking the functionality of the program. I step by step went with all menu options.

Before testing all the functions using this method, I begin to test the program by Adding customers, removing customers from a specific location, and removing served customers (The first customer in the queue should be removed). In the arrays version, I added customers by selecting queue numbers. But in the classes version, I only added customers because the program should automatically add the customer to the queue with the minimum length. After doing those things, I get a note of all the options I did in the program. Then I looked at the view of all empty queue options and view all queues options. So, I can confirm the functionality of adding, removing customers and the empty queue, and viewing all queue options. Furthermore, I separately checked the alphabetical sorting function. Next, I viewed the data file before performing the store method and then I stored data to the file and checked the loaded file. If the file is updated with the data which I used in the program, we can confirm that the store and load function works properly. Fuel stock should be updated in adding customers and removing customers before serving. And also, it should be updated when adding stocks. By checking with previously entered details and the added fuel stock, we can check the functionality of adding fuel stock and view the remaining stock. Additionally, we can get further confirmations of adding and removing customers functionality by checking fuel stocks. Furthermore, we can test the print income function by checking the fuel stock. Before testing the waiting queue, I filled all the fuel queues with the customers. Then I filled the waiting queue to check the waiting queue functionality and waiting queue full message. Then I removed a customer from a fuel queue and tested out the automatic replacement function and circular queue.

I hope I covered all the aspect of my program by choosing the test cases using above mentioned methods.

## 3. Code

### 3.1. Arrays Version

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

public class ArraysVersion {
    static int TotFuel = 6600;
    static int PerCustomer = 10;

    static String[] Pump1 = new String[6];
    static String[] Pump2 = new String[6];
    static String[] Pump3 = new String[6];

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("""

                                !!!Fuel Management System!!!!

                                Select your option below,
                                100 or VFQ:  View all Fuel Queues
                                101 or VEQ:  View all Empty Queues
                                102 or ACQ:  Add customer to a Queue
                                103 or RCQ:  Remove a customer from a specific location
                                104 or PCQ:  Remove a served customer
                                105 or VCS:  View Customers Sorted in alphabetical order
                                106 or SPD:  Store Program Data into file
                                107 or LPD:  Load Program Data from file
                                108 or STK:  View Remaining Fuel Stock
                                109 or AFS:  Add Fuel Stock
                                999 or EXT:  Exit the Program

                                """);

        while (true) {
            System.out.print("Enter option : ");
            String choice = scanner.next();

            if (choice.equals("100") || choice.equalsIgnoreCase("VFQ")) {
                //Ignoring lower & upper case differences (w3schools)
                AllFuelQueue();
            } else if (choice.equals("101") ||
choice.equalsIgnoreCase("VEQ")) {
                EmptyQueue();
            } else if (choice.equals("102") ||
choice.equalsIgnoreCase("ACQ")) {
```

```

        AddCustomer();
    } else if (choice.equals("103") ||
choice.equalsIgnoreCase("RCQ")) {
        RemoveSpecific();
    } else if (choice.equals("104") || choice.equals("PCQ")) {
        RemoveServed();
    } else if (choice.equals("105") ||
choice.equalsIgnoreCase("VCS")) {
        AlphabeticSort();
    } else if (choice.equals("106") ||
choice.equalsIgnoreCase("SPD")) {
        StoreData();
    } else if (choice.equals("107") ||
choice.equalsIgnoreCase("LPD")) {
        LoadData();
    } else if (choice.equals("108") ||
choice.equalsIgnoreCase("STK")) {
        System.out.println("Remaining Fuel Stock is " + TotFuel + "
liters");
    } else if (choice.equals("109") || choice.equals("AFS")) {
        AddStock();
    } else if (choice.equals("999") ||
choice.equalsIgnoreCase("EXT")) {
        System.exit(0);
    } else {
        System.out.println("Invalid Key");
    }
}

static void AllFuelQueue() {
    System.out.println("Pump 1 Queue List = " + Arrays.toString(Pump1));
    System.out.println("Pump 2 Queue List = " + Arrays.toString(Pump2));
    System.out.println("Pump 3 Queue List = " + Arrays.toString(Pump3));
} //View all Fuel Queues

static void EmptyQueue() {
    if (Pump1[0] == null) {
        System.out.println("Pump 1 is Empty");
    }if (Pump2[0] == null) {
        System.out.println("Pump 2 is Empty");
    }if (Pump3[0] == null) {
        System.out.println("Pump 3 is Empty");
    }else
        System.out.println("No Empty Queue Available");
} //View all Empty Queues

static void AddCustomer() {
    Scanner scan = new Scanner(System.in);
    System.out.print("Select the Pump : ");
    String PumpNum = scan.next();
    String YesNo;
    boolean opt = true;
    try {
        do {
            if (opt) { //(opt == true)
                if (PumpNum.equals("1")) {

```

```

        System.out.print("Add Customer : ");
        String CustName = scan.next();
        for (int j = 0; j < 6; j++) {
            if (Pump1[j] == null) {
                Pump1[j] = CustName;
                TotFuel = TotFuel-PerCustomer;
                break;
            }
        }
    }if (PumpNum.equals("2")) {
        System.out.print("Add Customer : ");
        String CustName = scan.next();
        for (int j = 0; j < 6; j++) {
            if (Pump2[j] == null) {
                Pump2[j] = CustName;
                TotFuel = TotFuel-PerCustomer;
                break;
            }
        }
    }if (PumpNum.equals("3")) {
        System.out.print("Add Customer : ");
        String CustName = scan.next();
        for (int j = 0; j < 6; j++) {
            if (Pump3[j] == null) {
                Pump3[j] = CustName;
                TotFuel = TotFuel-PerCustomer;
                break;
            }
        }
    }if (TotFuel<=500){
        System.out.println(" !!!WARNING!!! Fuel Stock is less
than 500 liters ");
    }
    System.out.print("Add another customer? (y for Yes |
AnyKey for No) : ");
    YesNo = scan.next();
    if (!YesNo.equalsIgnoreCase("y")) {
        opt = false;
    }
    } else {
        break;
    }
    } while (true);
} catch (InputMismatchException e) {
    System.out.println("Ender valid num");
}
scan.nextLine();
} //Add customer to a Queue

static void RemoveSpecific(){ //Remove Array Element in
Java (https://www.youtube.com/watch?v=r1L\_71N-5rs)
    Scanner scan = new Scanner(System.in);
    System.out.print("Select the Pump : ");
    String PumpNum = scan.next();
    System.out.print("Select the target : ");
    int target = scan.nextInt();
    int logicalsize = 6;

```

```

        if (PumpNum.equals("1")) {
            for (int i = target; i < logicalsize - 1; i++) {
                Pump1[i] = Pump1[i + 1];
            }
            Pump1[logicalsize - 1] = null;
            logicalsize--;
            TotFuel=TotFuel+PerCustomer;
            System.out.println("Customer Removed");
        }
        if (PumpNum.equals("2")) {
            for (int i = target; i < logicalsize - 1; i++) {
                Pump2[i] = Pump2[i + 1];
            }
            Pump2[logicalsize - 1] = null;
            logicalsize--;
            TotFuel=TotFuel+PerCustomer;
            System.out.println("Customer Removed");
        }
        if (PumpNum.equals("3")) {
            for (int i = target; i < logicalsize - 1; i++) {
                Pump3[i] = Pump3[i + 1];
            }
            Pump3[logicalsize - 1] = null;
            logicalsize--;
            TotFuel=TotFuel+PerCustomer;
            System.out.println("Customer Removed");
        }
        if (!(PumpNum.equals("1") || PumpNum.equals("2") || PumpNum.equals("3"))) {
            System.out.println("Invalid Pump Number");
        }
    }
    //Remove a customer from a specific location

    static void RemoveServed() { //Remove Array Element in
        Java(https://www.youtube.com/watch?v=r1L\_71N-5rs)
        int logicalsize = 6;
        Scanner scan = new Scanner(System.in);
        System.out.print("Select the Pump Number to remove the Served
Customer : "); //First customer in the list will be removed
        String PumpNum = scan.next();
        if (PumpNum.equals("1")) {
            for (int i = 0; i < logicalsize - 1; i++) {
                Pump1[i] = Pump1[i + 1];
            }
            Pump1[logicalsize - 1] = null;
            logicalsize--;
            System.out.println("Served Customer Removed");
        }
        if (PumpNum.equals("2")) {
            for (int i = 0; i < logicalsize - 1; i++) {
                Pump2[i] = Pump2[i + 1];
            }
            Pump2[logicalsize - 1] = null;
            logicalsize--;
            System.out.println("Served Customer Removed");
        }
        if (PumpNum.equals("3")) {
            for (int i = 0; i < logicalsize - 1; i++) {
                Pump3[i] = Pump3[i + 1];
            }
            Pump3[logicalsize - 1] = null;
            logicalsize--;
            System.out.println("Served Customer Removed");
        }
    }
}

```



```

    }
    //Remove a served customer

    static void SortName(String[] arr, int n) {
//https://www.geeksforgeeks.org/sorting-strings-using-bubble-sort-2/
        String temp;
        for (int j = 0; j < n - 1; j++) {    // Sorting strings using bubble
sort
            for (int i = j + 1; i < n; i++)
                if (!(arr[j] == null || arr[i] == null)){
                    if (arr[j].compareToIgnoreCase(arr[i]) > 0) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                    }
                }
        }
    } //Belongs to AlphabeticSort
    static void AlphabeticSort() {    //https://www.geeksforgeeks.org/sorting-
strings-using-bubble-sort-2/
        Scanner scan = new Scanner(System.in);
        System.out.print("Select the Pump Number to sort in Alphabetically :
");
        String PumpNum = scan.next();
        if (PumpNum.equals("1")) {
            String[] Pump1copy = Arrays.copyOf(Pump1, Pump1.length);
//https://www.youtube.com/watch?v=ZJFSr-qjts(How to copy arrays using
Arrays.copyOf() method in java?)
            int n = Pump1copy.length;
            SortName(Pump1copy, n);
            System.out.println("Pump 1 Alphabetic list = " +
Arrays.toString(Pump1copy));
        } if (PumpNum.equals("2")) {
            String[] Pump2copy = Arrays.copyOf(Pump2, Pump2.length);
            int n = Pump2copy.length;
            SortName(Pump2copy, n);
            System.out.println("Pump 2 Alphabetic list = " +
Arrays.toString(Pump2copy));
        } if (PumpNum.equals("3")) {
            String[] Pump3copy = Arrays.copyOf(Pump3, Pump3.length);
            int n = Pump3copy.length;
            SortName(Pump3copy, n);
            System.out.println("Pump 3 Alphabetic list = " +
Arrays.toString(Pump3copy));
        }
    } //View Customers Sorted in alphabetical order

    static void StoreData() {
//https://www.w3schools.com/java/java_files_create.asp
        try {
            FileWriter myWriter = new FileWriter("FuelQueue.txt");
            myWriter.write("Remaining Fuel Balance is "+TotFuel+"
liters"+"\\n"+"\\n"+
                "Pump 1 Queue List = "+Arrays.toString(Pump1)+"\\n"+
                "Pump 2 Queue List = "+Arrays.toString(Pump2)+"\\n"+
                "Pump 3 Queue List = "+Arrays.toString(Pump3)+"\\n");
            myWriter.close();

```

```

        System.out.println("Successfully wrote to the file");
    } catch (IOException e) {
        System.out.println("An error occurred");
        e.printStackTrace();
    }
} //Store Program Data into file

static void LoadData() {
//https://www.w3schools.com/java/java_files_read.asp
    try {
        File myObj = new File("FuelQueue.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            System.out.println(data);
        }
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred");
        e.printStackTrace();
    }
} //Load Program Data from file

static void AddStock() {
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter adding Stock : ");
    int AddFuel = scan.nextInt();
    TotFuel = TotFuel + AddFuel;
    System.out.println(AddFuel + " liters added to the Stock");
} //Add Fuel Stock
}

```

## 3.2. Classes Version

### 3.2.1. Main Class

```

import java.io.FileNotFoundException;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class Main {
    static int totalFuel = 6600;           // Total remaining fuel stock
    static int pricePerLiter = 430;        // Price of a fuel liter
    static int customersPerQ = 6;          // Number of customers that can
    be hold in a queue
    static int noOfPumps = 5;              // Number of pumps in the Fuel
    center

    static FuelQueue[] allQueues = new FuelQueue[5];
    static WaitingQueue waitingQ = new WaitingQueue(); // Waiting Queue
}

```

Array

```
public static void main(String[] args) {
    createQueue();
    Scanner scanner = new Scanner(System.in);
    System.out.println("""

                                !!!Fuel Management System!!!!

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

                                """);

    while (true) {
        System.out.print("Enter option : ");
        String choice = scanner.next();

        switch (choice) {
            case "100", "VFQ", "vfq" -> allFuelQueue();
            case "101", "VEQ", "veq" -> emptyQueue();
            case "102", "ACQ", "acq" -> addCustomer();
            case "103", "RCQ", "rcq" -> removeSpecific();
            case "104", "PCQ", "pcq" -> removeServed();
            case "105", "VCS", "vcs" -> alphabeticSort();
            case "106", "SPD", "spd" -> storeData();
            case "107", "LPD", "lpd" -> loadData();
            case "108", "STK", "stk" -> System.out.println("Remaining
Fuel Stock is " + totalFuel + " liters");
            case "109", "AFS", "afs" -> addStock();
            case "110", "IFQ", "ifq" -> printFuelQIncome();
            case "999", "EXT", "ext" -> System.exit(0);
            default -> System.out.println("Invalid Key");
        }
    }
}

static void createQueue() {
    FuelQueue Queue1 = new FuelQueue(1);
    FuelQueue Queue2 = new FuelQueue(2);
    FuelQueue Queue3 = new FuelQueue(3);
    FuelQueue Queue4 = new FuelQueue(4);
    FuelQueue Queue5 = new FuelQueue(5);
}
```

```

        allQueues[0] = Queue1;
        allQueues[1] = Queue2;
        allQueues[2] = Queue3;
        allQueues[3] = Queue4;
        allQueues[4] = Queue5;
    }

    static void allFuelQueue() {
        for (int i = 0; i < noOfPumps; i++) {
            System.out.print("Pump " + (i + 1) + " Queue List = ");
            ArrayList<Passenger> Passen = allQueues[i].getQueue();
            for (Passenger p : Passen) {
                System.out.print(p.getFullname() + ", ");
            }
            System.out.println();
        }
        if (!waitingQ.isEmpty()) {
            System.out.print("Waiting Queue List = ");
            Passenger[] waiting = waitingQ.getWaitingQ();
            for (Passenger qp : waiting) {
                System.out.print(qp.getFullname() + ", ");
            }
        } else {
            System.out.println("Waiting list is empty");
        }
        System.out.println();
    } //View all Fuel Queues

    static void emptyQueue() {
        for (int i = 0; i < noOfPumps; i++){
            if (allQueues[i].getCustomerLength() == 0) {
                System.out.println("Pump " + (i + 1) + " is empty"); //
            } else {
                System.out.println("Pump " + (i + 1) + " is not empty");
            }
        }
    } //View all Empty Queues

    static void addCustomer() {
        Scanner scan = new Scanner(System.in);
        String YesNo;
        boolean opt = true;
        try {
            do {
                if (opt) {
                    System.out.print("Add first name : ");
                    String firstName = scan.next();
                    System.out.print("Add last name : ");
                    String lastName = scan.next();
                    System.out.print("Add vehicle number : ");
                    String vehicleNo = scan.next();
                    System.out.print("Required fuel amount : ");
                    int fuelAmount = scan.nextInt();
                }
            } while (opt);
        } catch (Exception e) {
            System.out.println("Invalid input");
        }
    }

```

```

        Passenger passenger = new Passenger(firstName, lastName,
vehicleNo, fuelAmount);

        if (!(allQueues[1].getCustomerLength() == customersPerQ))
{
    // This will check last pump's last customer. This will run till it
fills.

        for (int i = 0; i < customersPerQ; i++) {
            if (allQueues[0].getCustomerLength() == i) {
                allQueues[0].setAddCustomer(passenger);
                System.out.println(firstName + " " + lastName
+ " added to Pump 1 queue successfully");
                totalFuel = totalFuel -
passenger.getRequiredFuel();
                break;
            }
            if (allQueues[1].getCustomerLength() == i) {
                allQueues[1].setAddCustomer(passenger);
                System.out.println(firstName + " " + lastName
+ " added to Pump 2 queue successfully");
                totalFuel = totalFuel -
passenger.getRequiredFuel();
                break;
            }
            if (allQueues[2].getCustomerLength() == i) {
                allQueues[2].setAddCustomer(passenger);
                System.out.println(firstName + " " + lastName
+ " added to Pump 3 queue successfully");
                totalFuel = totalFuel -
passenger.getRequiredFuel();
                break;
            }
            if (allQueues[3].getCustomerLength() == i) {
                allQueues[3].setAddCustomer(passenger);
                System.out.println(firstName + " " + lastName
+ " added to Pump 4 queue successfully");
                totalFuel = totalFuel -
passenger.getRequiredFuel();
                break;
            }
            if (allQueues[4].getCustomerLength() == i) {
                allQueues[4].setAddCustomer(passenger);
                System.out.println(firstName + " " + lastName
+ " added to Pump 5 queue successfully");
                totalFuel = totalFuel -
passenger.getRequiredFuel();
                break;
            }
        }
    } else {
        // Runs If the last pump's last customer
filled
        if (!waitingQ.isFull()) { // To check space
availability in waiting queue
            waitingQ.enqueue(passenger);
            System.out.println(firstName+" "+lastName+" added
to the waiting queue");
            waitingQ.show();
        } else {

```

```

        System.out.println("Waiting Queue is Full");
    }
    }if (totalFuel<=500){
        System.out.println(" !!!WARNING!!! Fuel Stock is less
than 500 liters (Remaining Stock : "+totalFuel+"");
    }
    System.out.print("Add another customer? (y for Yes |
AnyKey for No) : ");
    YesNo = scan.next();
    if (!YesNo.equalsIgnoreCase("y")) {
        opt = false;
    }
    } else {
        break;
    }
    } while (true);
} catch (InputMismatchException e) {
    System.out.println("Ender a valid fuel amount ");
}
scan.nextLine();
} //Add customer to a Queue

static void removeSpecific() {
    try {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.print("Select the Pump (1-5) : ");
            int pumpNum = scan.nextInt();

            if (pumpNum > noOfPumps) {
                System.out.println("Invalid Pump");
            } else {
                Scanner pos = new Scanner(System.in);
                System.out.print("Enter the Queue Position Number (1-6) :
");

                int position = pos.nextInt();
                if (position > customersPerQ) {
                    System.out.println("Invalid position");
                } else {
                    allQueues[pumpNum - 1].setRemoveCustomer(position -
1);

                    if (!waitingQ.isEmpty()) {
                        for (int i = 0; i < noOfPumps; i++) {
                            if (allQueues[i].getCustomerLength() <
customersPerQ) {
                                allQueues[i].setAddCustomer(waitingQ.getFrontObj());
                                waitingQ.dequeue();
                                break;
                            }
                        }
                    }
                    System.out.println("Pump " + pumpNum + ", queue
member " + position + " is Successfully Removed");
                }
            }
        }
    }
}

```

```

        } catch (InputMismatchException e) {
            System.out.println("Invalid, Enter a number (1-5)");
        }
    } catch (IndexOutOfBoundsException e) {
        System.out.println("Customer not available in the Pump");
    }
} //Remove a customer from a specific location

static void removeServed() {
    try {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.print("Select the Pump (1-5) to remove the Served
Customer : ");
            int pumpNum = scan.nextInt();
            if (pumpNum > noOfPumps) {
                System.out.println("Invalid Pump");
            } else {
                int customerIncome = pricePerLiter * allQueues[pumpNum -
1].getCustomer(0).getRequiredFuel();
                allQueues[pumpNum - 1].setRemoveCustomer(0); //First
customer in the list will be removed
                allQueues[pumpNum - 1].setInncome(customerIncome);
                if (!waitingQ.isEmpty()) {
                    for (int i = 0; i < noOfPumps; i++) {
                        if (allQueues[i].getCustomerLength() <
customersPerQ) {
                            allQueues[i].setAddCustomer(waitingQ.getFrontObj());
                            waitingQ.dequeue();
                            break;
                        }
                    }
                    System.out.println("Pump " + pumpNum + " customer is
Served Successfully");
                }
            }
        } catch (IndexOutOfBoundsException e) {
            System.out.println("No customers available in the Pump ");
        }
    } catch (InputMismatchException e) {
        System.out.println("Invalid, Enter a number (1-5)");
    }
} //Remove a served customer

static void alphabeticSort() {
    for (int i=0; i<noOfPumps; i++){
        ArrayList<Passenger> Passen = allQueues[i].getQueue(); //copy
of Queue
        ArrayList<String> customerNames = new ArrayList<>();
        for (Passenger p : Passen) {
            customerNames.add(p.getFullname());
        }
        customerNames.sort(String.CASE_INSENSITIVE_ORDER);
        System.out.println("Pump " + (i+1) + " queue in alphabetical

```

```

order = " + customerNames);
    }
    } //View Customers Sorted in alphabetic order

    static void storeData() { // Java Create and Write To Files
        (https://www.w3schools.com/java/java_files_create.asp)
        try {
            FileWriter myWriter = new FileWriter("Main.txt");

myWriter.write("\n_____
\n");
            myWriter.write("Remaining Fuel Balance is "+totalFuel+"
liters"+"\\n"+"\\n");
            for (int i = 0; i < noOfPumps; i++) {
                myWriter.write("Pump " + (i + 1) + " Queue List = ");
                ArrayList<Passenger> Passen = allQueues[i].getQueue();
//copy of Queue
                for (Passenger p : Passen) {
                    myWriter.write(p.getFirstName() + " " + p.getSecondName()
+ ", ");
                }
                myWriter.write("\\n");
            }

myWriter.write("_____
\\n \\n");
            for (int i = 0; i < noOfPumps; i++) {
                ArrayList<Passenger> Passen = allQueues[i].getQueue();
//copy of Queue
                for (Passenger p : Passen) {
                    myWriter.write("<-- Details of " + p.getFirstName() + "
"+ p.getSecondName() + " --> \\n");
                    myWriter.write("First Name : " + p.getFirstName() + "\\n"
+
                                "Second Name : " + p.getSecondName() + "\\n" +
                                "Vehicle Number : " + p.getVehicleNo() + "\\n" +
                                "Required Fuel Amount : " + p.getRequiredFuel()
+ "\\n");
                    myWriter.write("\\n");
                }
            }

myWriter.write("\\n_____
\\n");
            myWriter.close();
            System.out.println("Successfully wrote to the file");
        } catch (IOException e) {
            System.out.println("An error occurred");
            e.printStackTrace();
        }
    } //Store Program Data into file

    static void loadData() { //Java Read Files
        (https://www.w3schools.com/java/java_files_read.asp)

```



```

        try {
            File myObj = new File("Main.txt");
            Scanner myReader = new Scanner(myObj);
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                System.out.println(data);
            }
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred");
            e.printStackTrace();
        }
    } //Load Program Data from file

    static void addStock() {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter adding Stock : ");
            int addFuel = scan.nextInt();
            totalFuel = totalFuel + addFuel;
            System.out.println(addFuel + " liters added to the Stock");
        } catch (InputMismatchException e) {
            System.out.println("Invalid input, add only numbers");
        }
    } //Add Fuel Stock

    static void printFuelQIncome() {
        for (int i = 0; i < noOfPumps; i++) {
            System.out.println("Pump " + (i + 1) + " income : Rs." +
allQueues[i].getInncome());
        }
    } // Print the income of Fuel queues
}

```

### 3.2.2. FuelQueue Class

```

import java.util.ArrayList;

public class FuelQueue {
    private ArrayList<Passenger> Queue = new ArrayList<>(); // ArrayList
    declaration

    private int pump;
    private int income = 0;

    //Constructor method
    public FuelQueue(int pump) {
        this.pump = pump;
    }
}

```

```

    }

    public void setAddCustomer (Passenger passenger){           // Setter for
adding passenger to the Queue
        Queue.add(passenger);
    }

    public void setRemoveCustomer (int position){               // Setter for remove
specific passenger from Queue
        Queue.remove(position);
    }

    public ArrayList<Passenger> getQueue() {                   // Getter for the ArrayList
        return Queue;
    }

    public int getCustomerLength() {                           // Getter for the customer array
length
        return Queue.size();
    }

    public void setInncome(int pumpIncome){
        income = income + pumpIncome;
    }

    public Passenger getCustomer(int position){
        return Queue.get(position);
    }

    public int getInncome(){
        return income;
    }
}

```

### 3.2.3. Passenger Class

```

public class Passenger {

    private String firstName;
    private String secondName;
    private String vehicleNo;
    private int requiredFuel;

    //Passenger constructor with firstName, lastName, vehicleNo and
requiredFuel
    Passenger(String firstName, String secondName, String vehicleNo, int
requiredFuel) {
        this.firstName = firstName;
        this.secondName = secondName;
    }
}

```

```

        this.vehicleNo = vehicleNo;
        this.requiredFuel = requiredFuel;
    }

    //setters are to update the current value of the attribute
    //getters are to get the value of the attribute

    public void setFirstName(String firstName) {    // Setter for the
firstName
        this.firstName = firstName;
    }

    public String getFirstName() {    // Getter for the firstName
        return firstName;
    }

    public void setSecondName(String secondName) {    // Setter for the
secondName
        this.secondName = secondName;
    }

    public String getSecondName() {    // Getter for the secondName
        return secondName;
    }

    public void setVehicleNo(String vehicleNo) {    // Setter for the
vehicleNo
        this.vehicleNo = vehicleNo;
    }

    public String getVehicleNo() {    // Getter for the vehicleNo
        return vehicleNo;
    }

    public void setRequiredFuel(int requiredFuel) {    // Setter for the
requiredFuel
        this.requiredFuel = requiredFuel;
    }

    public int getRequiredFuel() {    // Getter for the requiredFuel
        return requiredFuel;
    }

    public String getFullname() {
        return getFirstName() + " " + getSecondName();
    }
}

```

### 3.2.4. WaitingQueue Class

```
//References : | Telusko YouTube | Queue Implementation using Java / Circular  
Array Complete playlist of Data Structure Using Java  
// P1:https://youtu.be/PvDoT79oHTs P2:https://youtu.be/8K1rt6v5mJ4  
P3:https://youtu.be/23DjZA7AuOY  
  
public class WaitingQueue {  
    private Passenger[] waitingList = new Passenger[7];    // Array  
    declaration  
    private int size;  
    private int front;  
    private int rear;  
  
    public void enqueue(Passenger passenger) {    // To insert passengers  
to the queue  
        if (!isFull()) {  
            waitingList[rear] = passenger;  
            rear = (rear + 1) % waitingList.length;  
            size = size + 1;  
        }else  
            System.out.println("Queue is full");  
    }  
  
    public Passenger dequeue() {    // To remove passengers from the queue  
(front will increase by +1)  
        Passenger passenger = waitingList[front];  
        if (!isEmpty()) {  
            front = (front + 1) % waitingList.length;  
            size = size - 1;  
        }else  
            System.out.println("Waiting Queue is Empty");  
        return passenger;  
    }  
  
    public int getSize() {  
        return size;  
    }  
  
    public boolean isEmpty() {    // Check the list is empty or not  
        return getSize() == 0;  
    }  
  
    public boolean isFull() {    // Check the list is full or not  
        return getSize() == waitingList.length;  
    }  
  
    public Passenger getFrontObj() {    // Getter to get front object in the  
list  
        return waitingList[front];  
    }  
  
    public void show() {  
        System.out.print("Waiting Queue : ");  
    }  
}
```

```
        for (int i = 0; i < size; i++){
            System.out.print(waitingList[(front+i) % waitingList.length] + "
");
        }
        System.out.println();
    }

    public Passenger[] getWaitingQ() {        // Getter for the ArrayList
        return waitingList;
    }
}
```

← ← END → →