ROBERT GORDON UNIVERSITY ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

BSc Artificial Intelligence and Data Science

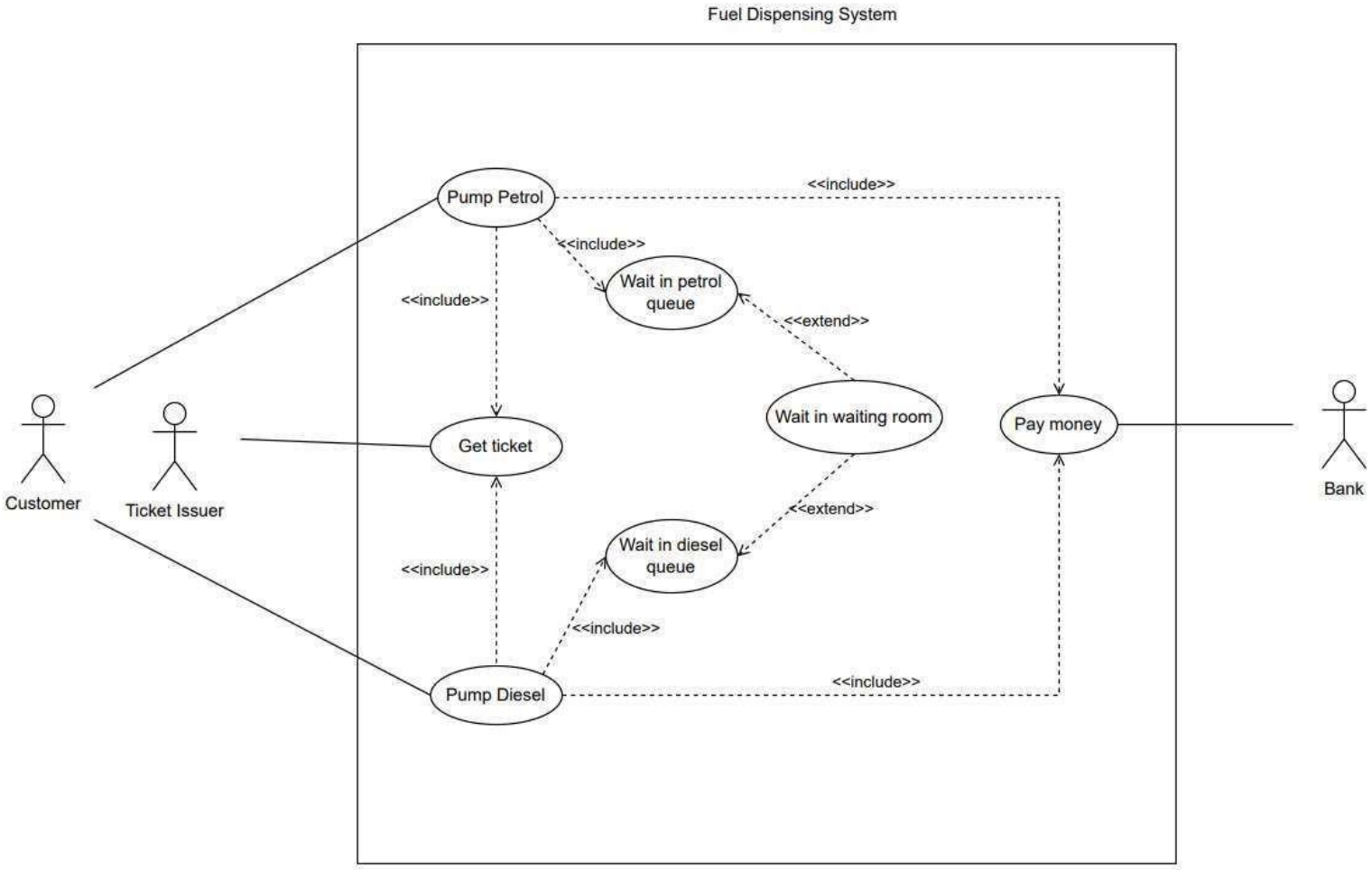
Level 05

Object Oriented Development CM2601

Coursework – Final Submission

Contents

1. Use Case Diagram – Fuel Dispensing System



2. Use Case Descriptions

2.1. Pump Petrol

Section: Main

Use case: Pump Petrol

Actor(s): Customer

Purpose: For the customer to pump petrol into their vehicle depending on its type

Overview: The customer pumps petrol to their vehicle

Preconditions: The vehicle is either a car, van, three-wheeler, motor bike or any other vehicle that requires petrol as its fuel type. The payment is already made. Customer has a ticket. Customer was waiting in the petrol queue.

Typical Course of Events

Actor Action	System Response
1. Customer pays money	5. Petrol is pumped into vehicle
2. Customer waits in petrol queue	6. Asks customer to move to another dispenser
3. Customer obtains a ticket	
4. Customer opens cap to pump petrol	

Alternative Courses

- Line 5: Tells the customer that the cap is closed and asks to open

Related Use Cases

Includes Pay money

Includes Wait in petrol queue

Includes Get ticket

2.2. Pump Diesel

Section: Main

Use case: Pump Diesel

Actor(s): Customer

Purpose: For the customer to pump diesel into their vehicle depending on its type

Overview: The customer pumps diesel to their vehicle

Preconditions: The vehicle is either a public transport vehicle or any other vehicle that requires diesel as its fuel type. The payment is already made. Customer has a ticket. Customer was waiting in the diesel queue.

Typical Course of Events

Actor Action	System Response
1. Customer pays money	5. Diesel is pumped into the vehicle
2. Customer waits in diesel queue	6. Asks customer to move to another dispenser
3. Customer gets a ticket	
4. Opens diesel cap of vehicle	

Alternative Courses

- Line 5: Customer is asked to open the cap to pump diesel if it is closed

Related Use Cases

Includes Pay money

Includes Wait in diesel queue

Includes Get ticket

2.3. Get ticket

Section: Main

Use case: Get ticket

Actor(s): Ticket issuer

Purpose: The customer needs to get a ticket with a number for the dispensing system to pump fuel on a first come first served basis (the ticket will contain a number on it)

Overview: Ony if the customer has a ticket will they be ale to be in a queue and obtain fuel for their vehicle

Preconditions: There is space (maximum is 10) in the queues to the dispensers, only then will the ticket be issued

Typical Course of Events

Actor Action	System Response
1. Issues a ticket to a customer	2. Customer takes the ticket and goes to the relevant queue

Alternative Courses

- Line 2: Customer is sent to the waiting room due to lack of slots on queue

Related Use Cases

Includes Pump Petrol

Includes Pump Diesel

2.4. Pay money

Section: Main

Use case: Pay money

Actor(s): Bank, Customer

Purpose: The customer pays money for the fuel they are about to be receiving

Overview: The customer pays money to the bank either by cash or credit

Preconditions: The customer gets fuel by the time their transaction is made

Typical Course of Events

Actor Action	System Response
1. Customer pays cash	4. Bank acknowledges payment
2. Customer pays with a debit card	5. System allows customer to pump petrol
3. Customer pays with a credit card	6. System allows customer to pump diesel

Alternative Courses

- Line 2: Debit card has limited funds, insufficient for the required amount of fuel
- Line 4: Bank declines payment and doesn't acknowledge

Related Use Cases

Includes Pump Petrol

Includes Pump Diesel

2.5. Wait in petrol queue Section:
Main

Use case: Wait in petrol queue

Actor(s): Customer

Purpose: The customer waits in a petrol queue

Overview: There are less than ten slots available for a vehicle to be in line to pump petrol to their vehicle

Preconditions: The customer must have a ticket

Typical Course of Events

Actor Action	System Response
1. The customer goes to the petrol queue	2. The customer is provided with petrol when their turn comes

Alternative Courses

- Line 2: The customer is asked to move to another queue due to lack of manpower

Related Use Cases

Includes Pump Petrol

Extends Wait in waiting room

2.6. Wait in diesel queue Section:
Main

Use case: Wait in diesel queue

Actor(s): Customer

Purpose: The customer waits in a diesel queue

Overview: There are less than ten slots available for a vehicle to be in line to pump diesel to their vehicle

Preconditions: The customer must have a ticket

Typical Course of Events

Actor Action	System Response
1. The customer goes to the diesel queue	2. The customer is provided with diesel when their turn comes

Alternative Courses

- Line 2: The customer is asked to move to another diesel queue due to lack of manpower

Related Use Cases

Includes Pump Diesel

Extends Wait in waiting room

2.7. Wait in waiting room Section:
Main

Use case: Wait in waiting room

Actor(s): Customer

Purpose: There is lack of slots available for vehicles to be in a queue at the fuel station for the required fuel type

Overview: The customer waits in the waiting room until a slot on a queue of the required fuel type becomes empty

Preconditions: The customer must have a ticket. All possible queues must be full.

Typical Course of Events

Actor Action	System Response
1. Customer obtains a ticket and goes to the waiting room	2. The system moves the customer to the queue once the queue has a slot available for another vehicle

Alternative Courses

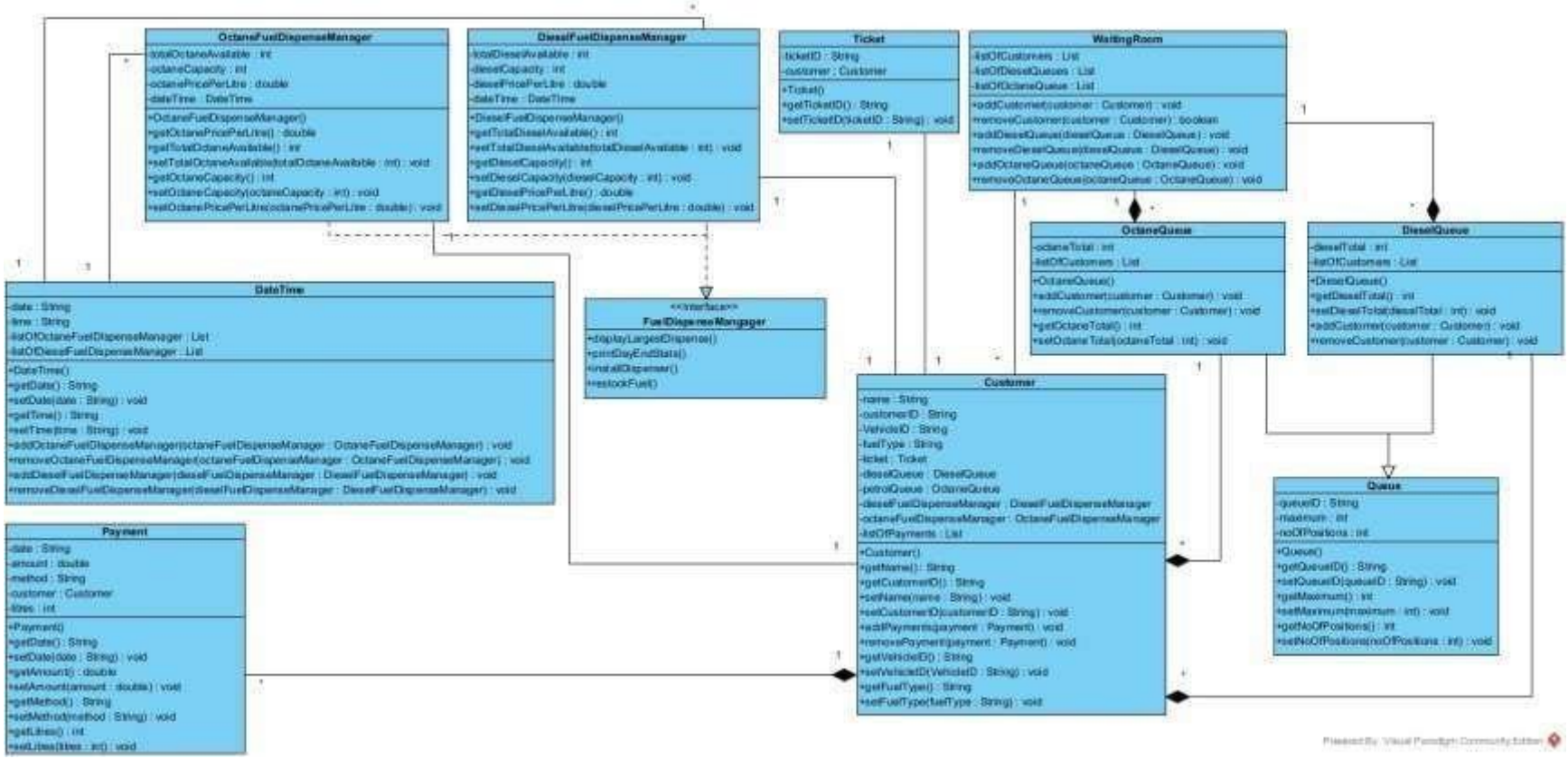
- Line 2: The available slot is for a vehicle of the other fuel type, therefore the system asks the customer to wait a little longer

Related Use Cases

Extends Wait in petrol queue

Extends Wait in diesel queue

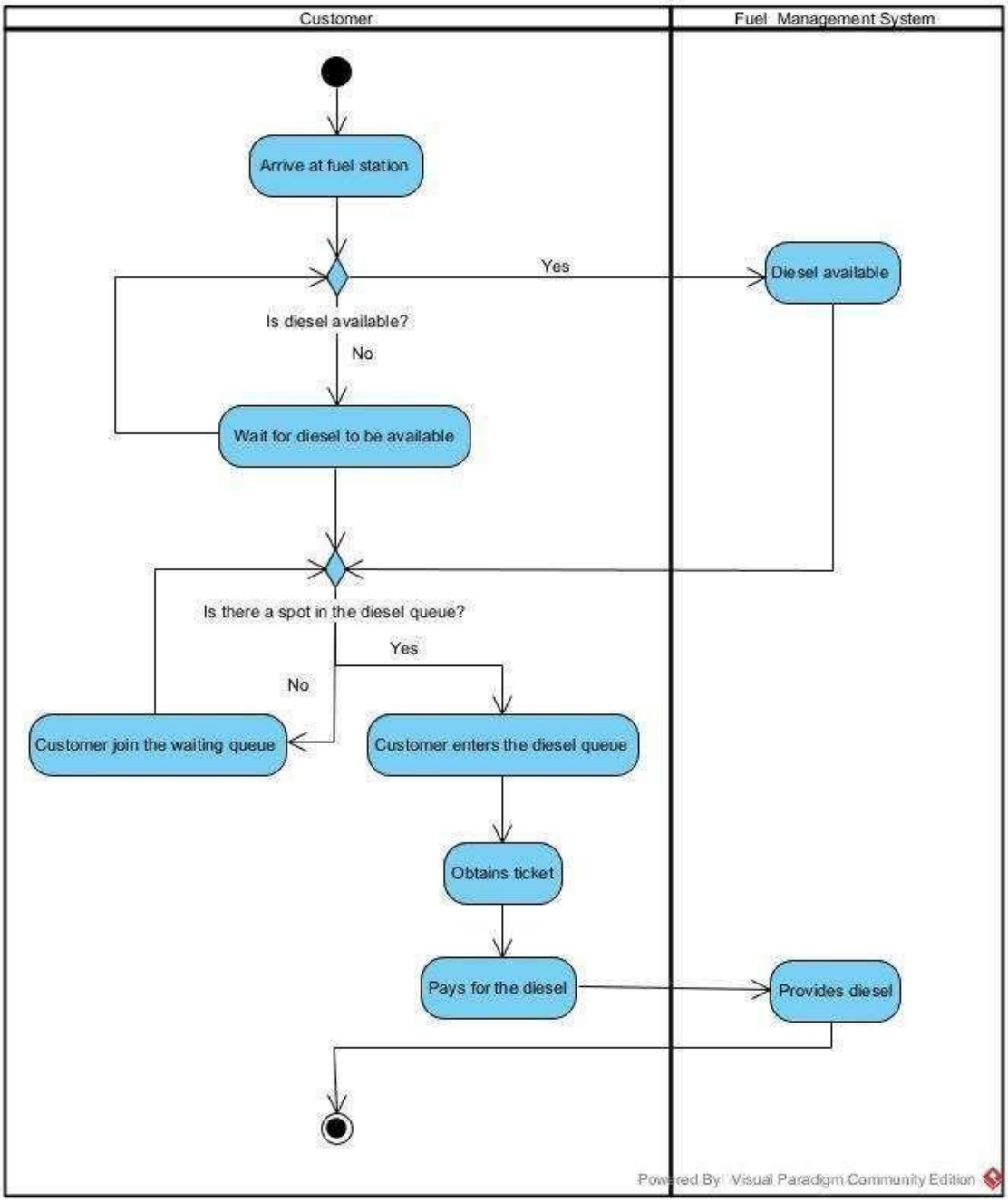
3. Class Diagram



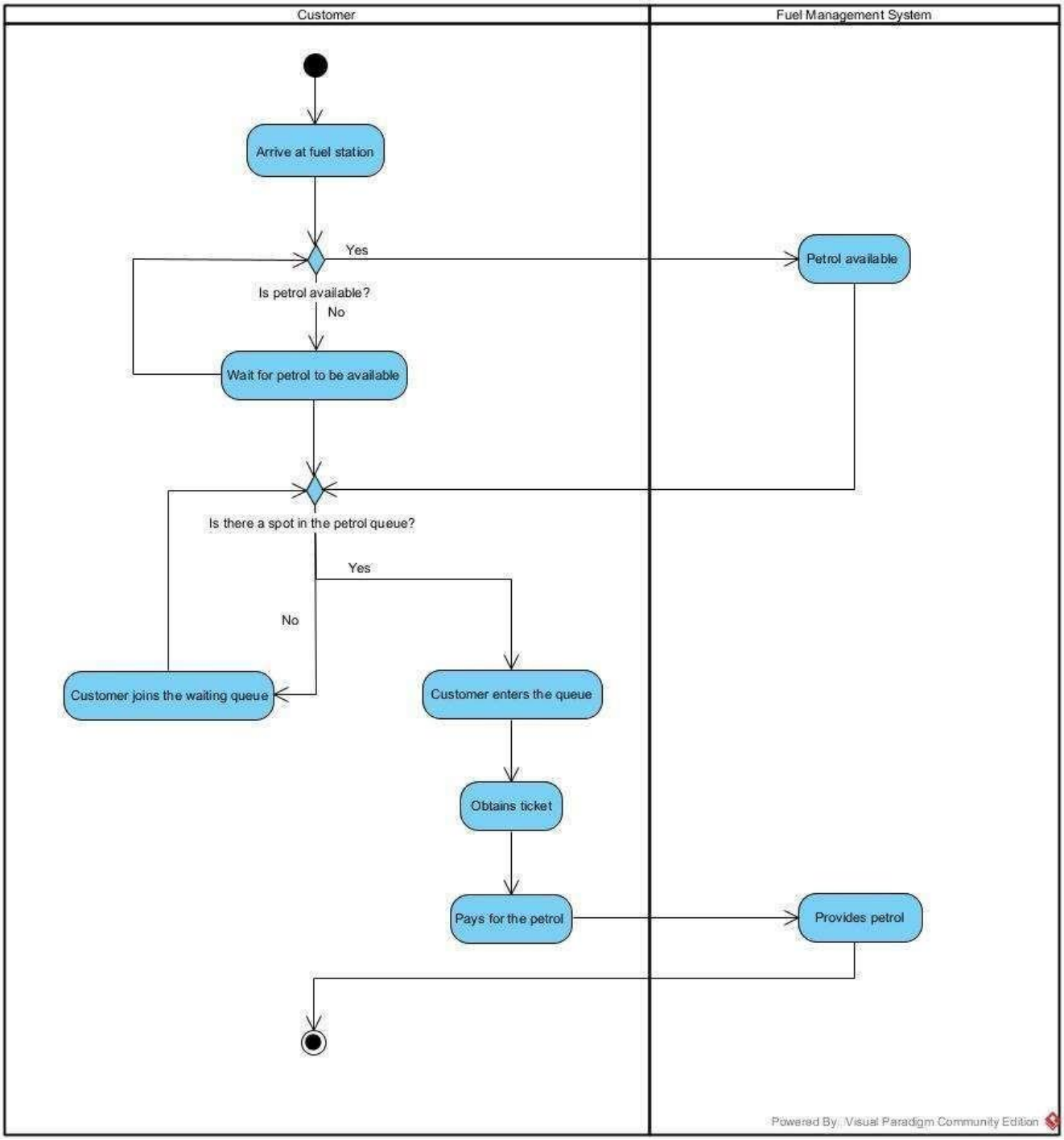
ROBERT GORDON UNIVERSITY ABERDEEN

4. Activity Diagram

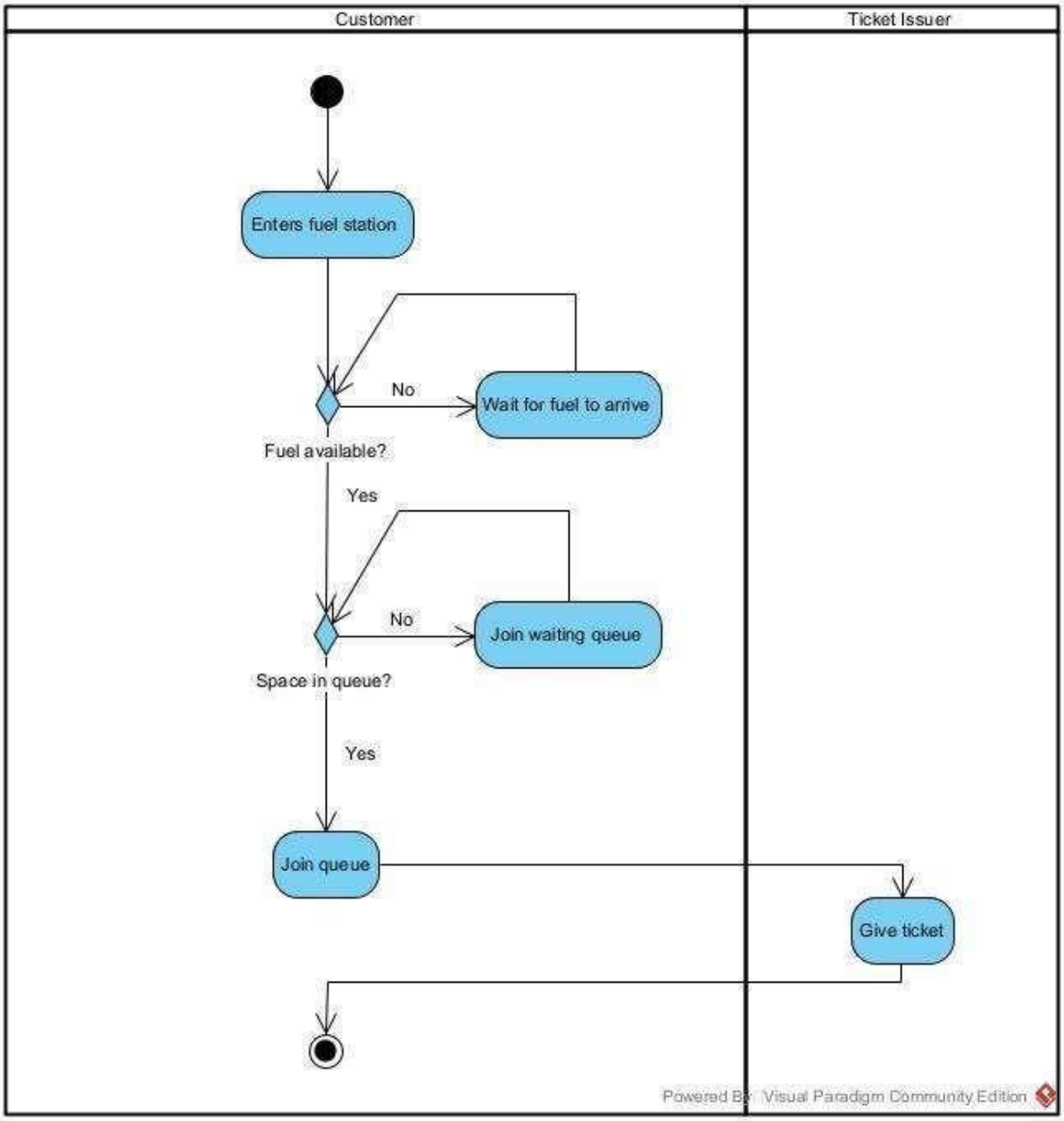
4.1. Pump diesel



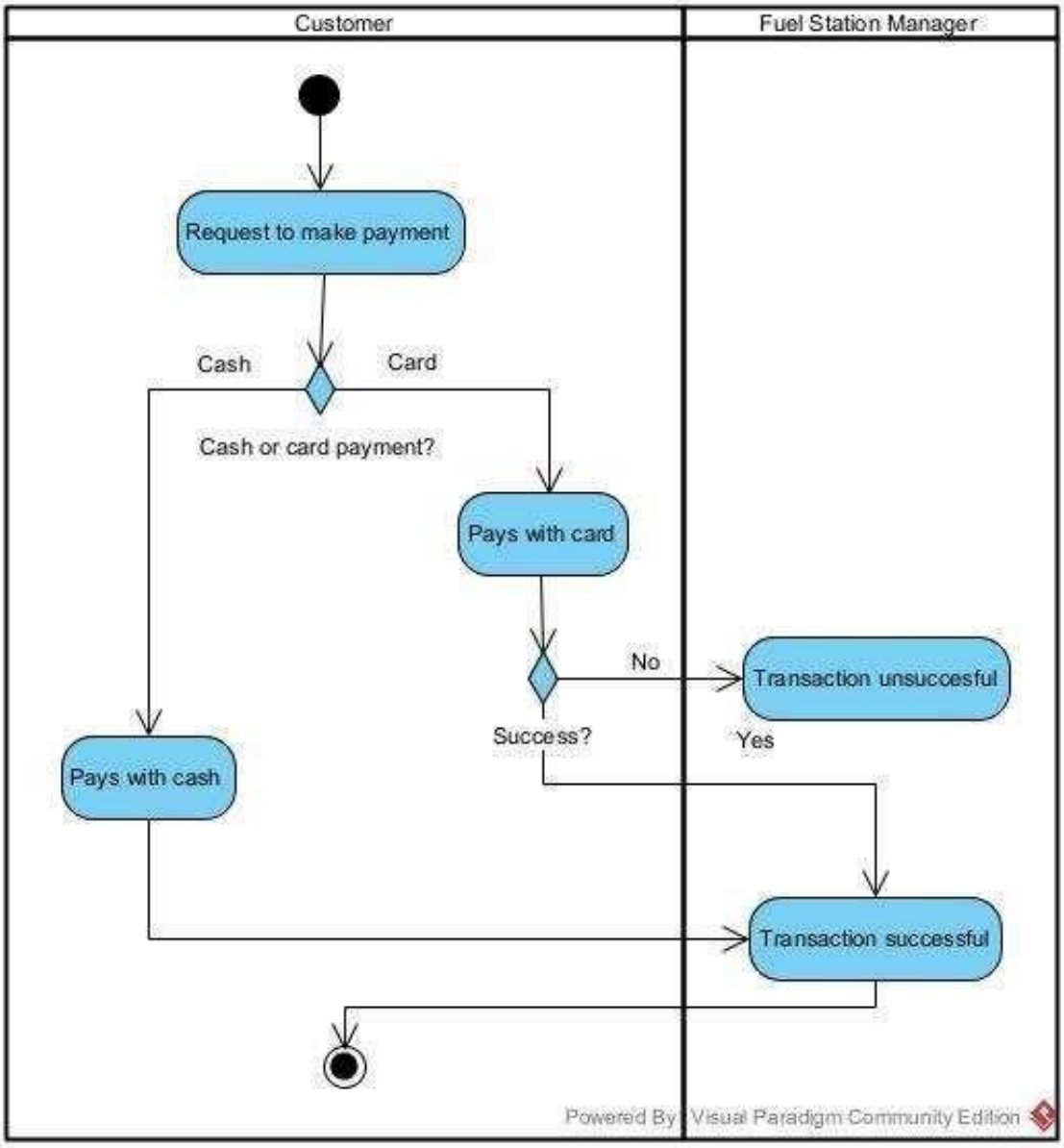
4.2. Pump Petrol



4.3. Get Ticket

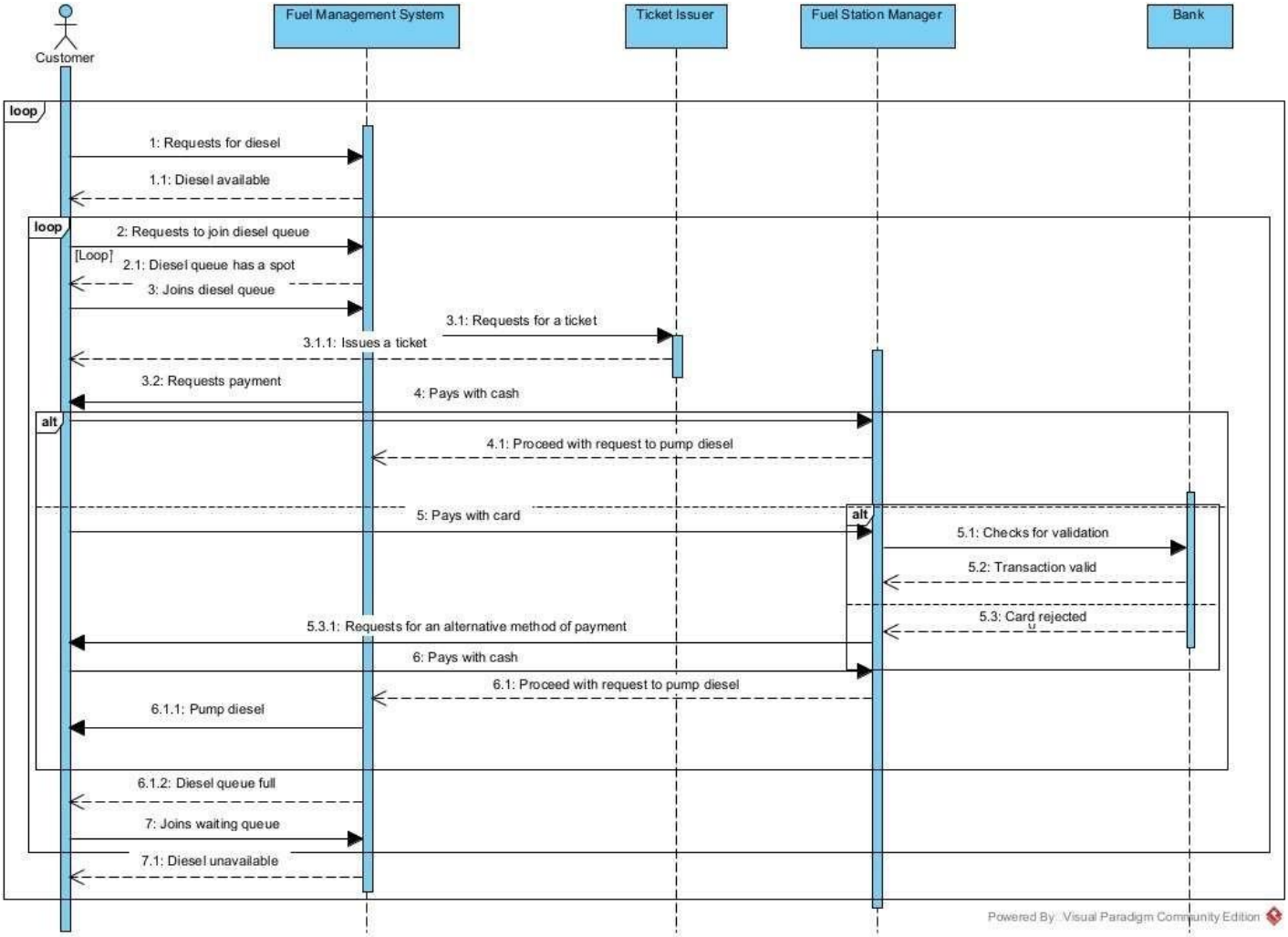


4.4. Make Payment



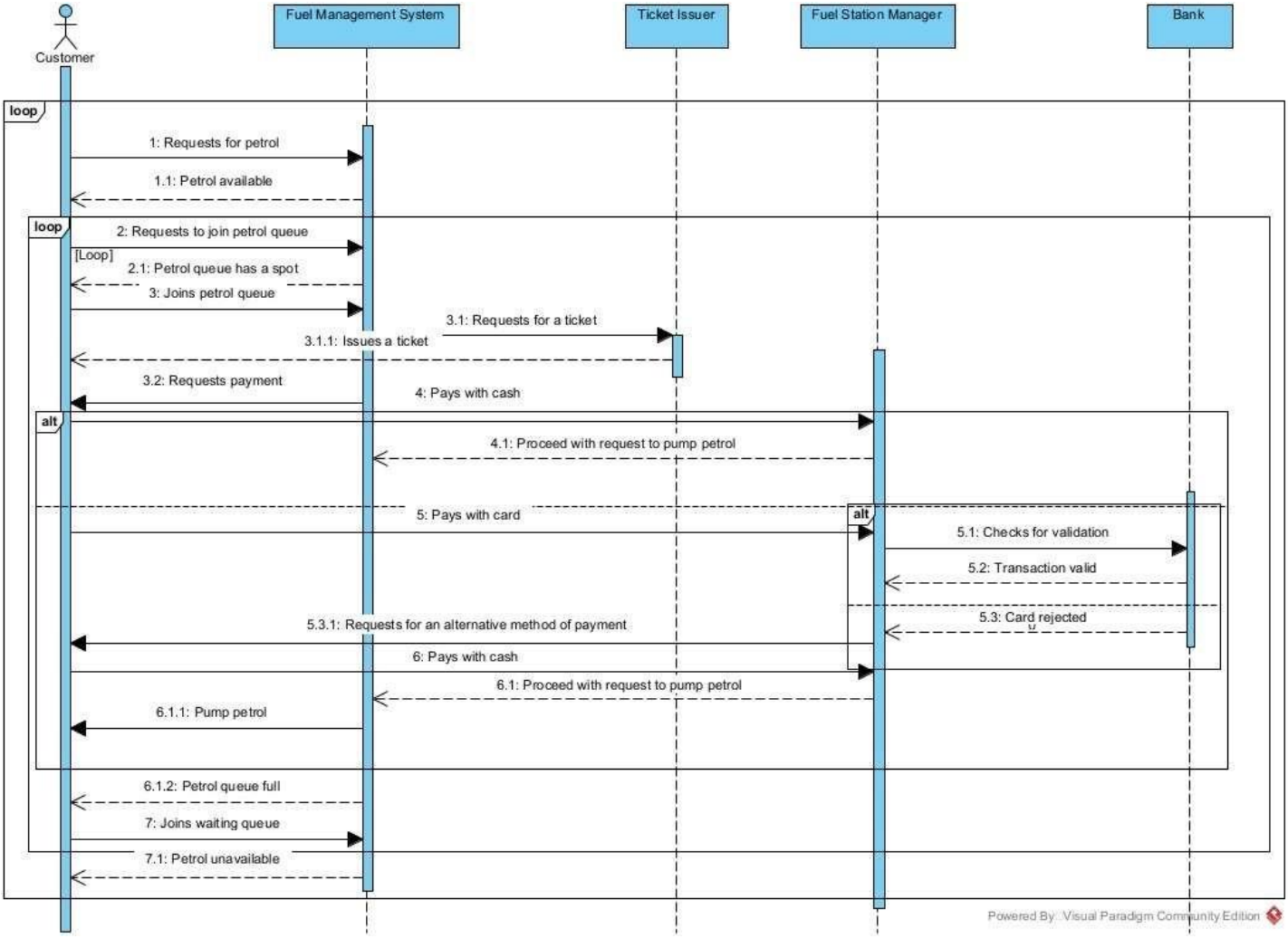
5. Sequence Diagram

5.1. Pump Diesel


Customer
Fuel Management System
Ticket Issuer
Fuel Station Manager
Bank
loop
1: Requests for diesel
1.1: Diesel available
loop
2: Requests to join diesel queue
[Loop]
2.1: Diesel queue has a spot
3: Joins diesel queue
3.1: Requests for a ticket
3.1.1: Issues a ticket
3.2: Requests payment
4: Pays with cash
alt
4.1: Proceed with request to pump diesel
5: Pays with card
alt
5.1: Checks for validation
5.2: Transaction valid
5.3: Card rejected
5.3.1: Requests for an alternative method of payment
6: Pays with cash
6.1: Proceed with request to pump diesel
6.1.1: Pump diesel
6.1.2: Diesel queue full
7: Joins waiting queue
7.1: Diesel unavailable
Powered By Visual Paradigm Community Edition

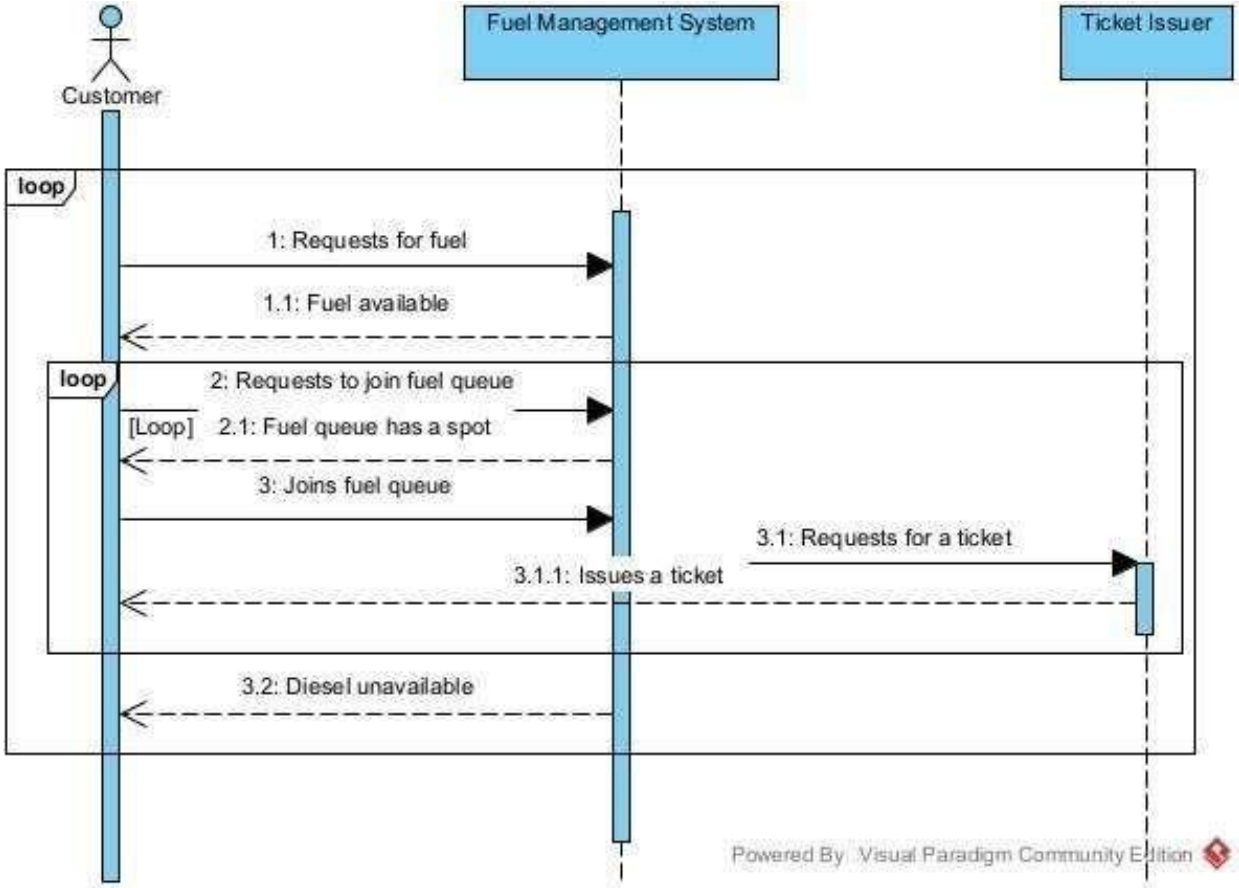
5

.2. Pump Petrol



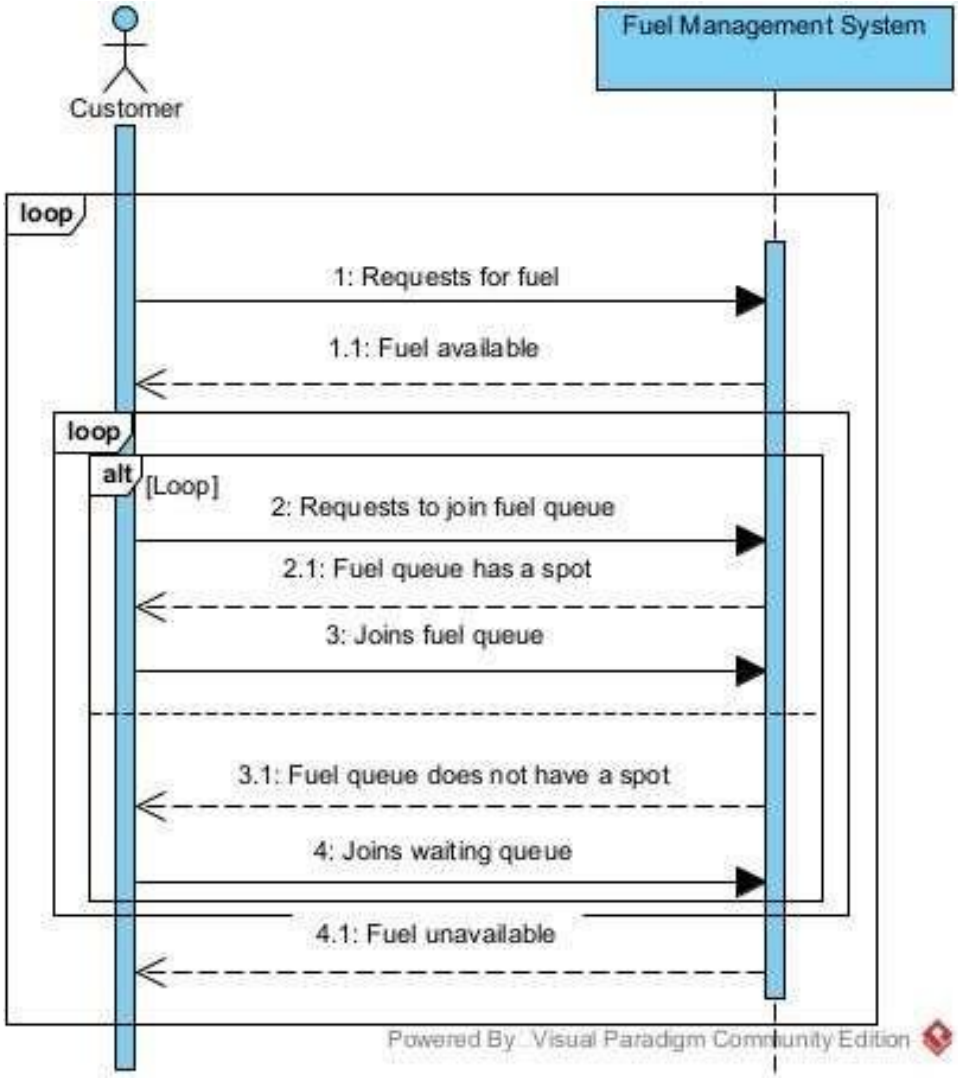
.3. Get Ticket

5



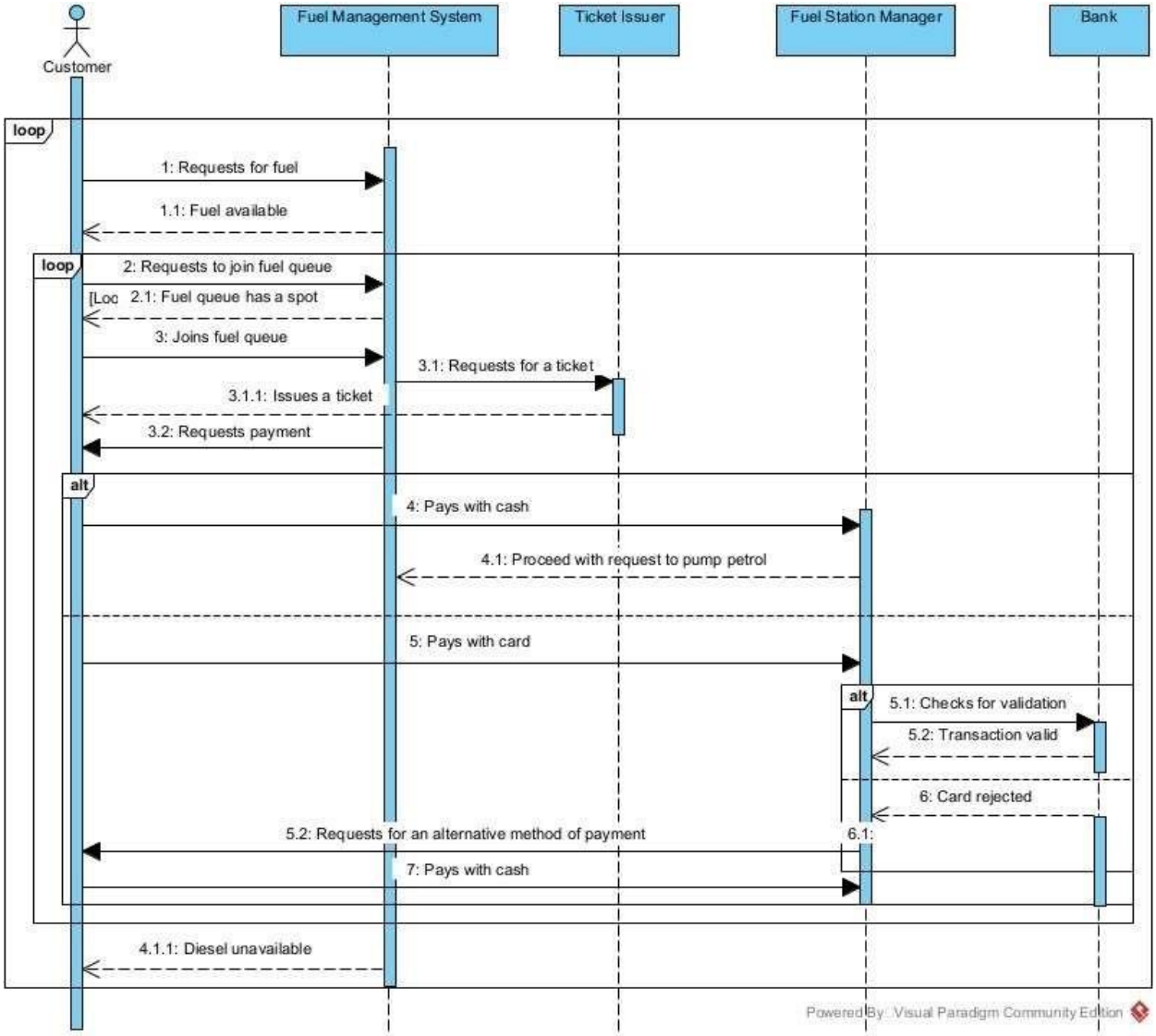
.4. Join Queue

5



.5. Make Payment

5


Customer
Fuel Management System
Ticket Issuer
Fuel Station Manager
Bank
loop
1: Requests for fuel
1.1: Fuel available
loop
2: Requests to join fuel queue
[Loc 2.1: Fuel queue has a spot
3: Joins fuel queue
3.1: Requests for a ticket
3.1.1: Issues a ticket
3.2: Requests payment
alt
4: Pays with cash
4.1: Proceed with request to pump petrol
5: Pays with card
alt
5.1: Checks for validation
5.2: Transaction valid
6: Card rejected
5.2: Requests for an alternative method of payment
6.1:
7: Pays with cash
4.1.1: Diesel unavailable
Powered By: Visual Paradigm Community Edition

5

6. Implementation

Main

```
import java.sql.*;
import java.util.InputMismatchException; import
java.util.Scanner;

public class Main extends Thread{
    public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        boolean valid = true;
int option = 0;
        String systemOperatorUsername = "";
        String systemOperatorPassword = "";
Queue queue = null;          int recordCount =
0;
        String[] vehicles = new String[10];
int systemOperatorOption = 0;
        String vehicleNum = null;
int dispenserNumber = 0;          int
numberOfPositions =0;          String
vehicleType = null;
        String fuelType = null;
boolean isARecord = false;
        PreparedStatement psmt = null;
        ResultSet rs = null;
int ticketNumber = 0;
        CommonWaitingQueue waitingQueue = null;
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/fuel_management_s
ystem?autoReconnect=true&useSSL=false","root","rashadha@73");
        // to get all records in particular queue who is not still pumped
        String query2 = "       SELECT                          " +
                                ticket_num,vehicle_num
                "       FROM                " +
                "       dispenser_detail
                        WHERE dispenser_num = ? and fuel_type = ?
                        and
                             ";
fuel_pumped_or_not = ?;          get the vehicle that's not added to q
        a                                 received
ticket
        String query3 =          "select ticket_num, vehicle_num,
                 " +             vehicle_type,                          ";
                     "from       ticket_management_system" +
fuel_type                 "     where add_to_queue_or_not = ? and rown" +
                / query     ck if there is available vehicles in w
                String query
                   = "fuel_t select queue_entry_num, vehicle_type,

                           "
```

```
                                pe from dispenser_detail" +


                            "where dispenser_number = ? and rownum =";
                             1;
// query to add vehicle to a queue which is in common waiting
```

```
queue
        String query5 = "UPDATE dispenser_detail " +
                "SET dispenser_number = REPLACE(dispenser_number,?,?) " +
                "WHERE queue_entry_num = ?;";
        // query to take the first vehicle from the queue
        String query6 =  "select queue_entry_num, fuel_type " +
                "from dispenser_detail " +
               "where fuel_pumped_or_not = ? and  dispenser_number = ? and
rownum = 1;";          //
        int queueEntryNumber = 0;
        boolean isARecordInWaiting = false;
        // query to enter to the queue
        String queryToEnqueue = "insert into dispenser_detail" +
"(ticket_num, vehicle_num, vehicle_type, fuel_type, dispenser_num)" +
                "values(?, ?, ?, ?);";
        // query to enter a vehicle to ticket_counter table which received
a ticket
        String query = "insert into ticket_counter " +
                "(vehicle_num, vehicle_type, fuel_type)" +
                "values (?, ?, ?)";
        // query to check remain fuel amount
        String query7 = "SELECT available_amount" +
                " FROM fuel_availability " +
                " WHERE fuel_type = ?;";
        String  queryToReadObject  =  "select  vehicle_num,  vehicle_type,
fuel_type" +
                "from ticket_management_system" +
                "where add_to_queue_or_not = ? and rownum = 1;";
        DieselFuelDispenseManager dieselFuelDispenseManager = null;
        OctaneFuelDispenseManager octaneFuelDispenseManager = null;
        DateClass date = null;
        DispenserOperator dispenserOperator = null;
MultiThreading thread = null;
        // query to update dispensed vehicle detail
        String query8 = "UPDATE dispenser_detail " +
                "SET fuel_pumped_or_not = REPLACE(fuel_pumped_or_not,?,?),
" +
                "dispensed_fuel_amount = REPLACE(dispensed_fuel_amount, ?,
?), " +
                "paid_amount = REPLACE(paid_amount,?,?), " +
                "dispensed_date = REPLACE(dispensed_date,?,?) " +
                "WHERE queue_entry_num = ?;";
        // query to add to common waiting queue who received a ticket if
there is no availabe position in particular queue
        String queryToAddToCommonWaitingQueue = "insert into " +
                "dispenser_detail " +
                "(ticket_num, vehicle_num, vehicle_type, fuel_type, " +
                "dispenser_num) " +
                "values (?, ?, ?, ?, ?);";
        String query9 = "UPDATE fuel_availability " +
```

```
                "SET available_amount = ? " +
                "WHERE fuel_type = ?;";
Scanner sc = new Scanner(System.in);          do
        {
            System.out.println("input according to the option: "); // 1 --
> customer, 2--> system operator

try
{
                option = sc.nextInt();
                if (!(option <= 6 && option >= 1))
                {
                    throw new IllegalArgumentException();
                }
}
            catch(IllegalArgumentException e)
            {
                System.out.println("Your input is invalid");
valid = false;
            }
            catch(InputMismatchException e)
            {
                System.out.println("You input is invalid");
            }
        }while (!valid);

        if (option == 1)
        {
            // to display number of available positions in
queue              psmt = con.prepareStatement(query2);
for (int i = 0; i<4; i++)
            {
                psmt.setInt(1,i+1);
psmt.setString(2, "petrol");
psmt.setString(3,null);                 rs =
psmt.executeQuery();

                while (rs.next())
                {
                    vehicleNum = rs.getString(3);
vehicles[recordCount] = vehicleNum;
recordCount+=1;
                }
                con.close();
                queue = new Queue(waitingQueue,vehicles,0);
queue.displayNumberOfPositions(fuelType,i+1);
```

```
            }
recordCount = 0;
            for (int i = 0; i<3; i++)
            {
                psmt.setInt(1,i+1);
psmt.setString(2, "diesel");
psmt.setString(3,null);                 rs =
psmt.executeQuery();

                while (rs.next())
                {
                    vehicleNum = rs.getString(3);
vehicles[recordCount] = vehicleNum;
recordCount+=1;
                }
                con.close();
                queue = new Queue(waitingQueue,vehicles,0);
queue.displayNumberOfPositions(fuelType,i+1);
            }
            System.out.print("Enter vehicle Number: ");
            vehicleNum = sc.next();
            System.out.println("Enter vehicle type (eg: \"car\" or " +
                    "\"van\" or \"motor bike\" or \"three wheeler\"" +
" or \"public transport\" or \"other\"): ");             do{
try                 {
                    vehicleType = sc.next().toLowerCase();
                    if (!(vehicleType == "car" || vehicleType == "van"
                     || vehicleType == "motor bike" || vehicleType ==
"three wheeler"
                    || vehicleType == "public transport" || vehicleType ==
"other"))
                    {
                        throw new IllegalArgumentException();
                    }
}
                catch (IllegalArgumentException e)
{
                    System.out.println("Your entry is invalid");
valid = false;
                }
            }while (!valid);



            System.out.println("Enter fuel type(eg: \"92Octane\" or
\"Diesel\"): ");
do {
try
                {
                    fuelType = sc.next();
```

```
                    if (fuelType == "92Octane" || fuelType == "Diesel")
                        throw new IllegalArgumentException();
                }
                catch(IllegalArgumentException e)
                {
                    System.out.println("Your entry is invalid");
valid = false;
                }
            }while (!valid);

            Customer customer = new
Customer(vehicleNum,vehicleType,fuelType);
            TicketCounter ticketCounter = new TicketCounter();
ticketCounter.issueTicket(customer);
            ticketCounter.saveCustomerDetails(con,query);
        }
        if (option == 2) // system operator
        {
do{
            System.out.print("Enter the username: ");
try{
                systemOperatorUsername = sc.next();
System.out.print("Enter the password: ");
systemOperatorPassword = sc.next();
                if (!(systemOperatorUsername =="groupActivity" &&
systemOperatorPassword == "dudupi"))
                {
                    throw new IllegalArgumentException();
                }
}
            catch (IllegalArgumentException e)
            {
                System.out.println("Your entry is invalid");
            }
        }while (!(systemOperatorUsername =="rashadha" &&
systemOperatorPassword == "dudupi"));

        do{
            System.out.print("Enter an option: "); // 1--> enQueue

            try
{
                systemOperatorOption = sc.nextInt();
                if (systemOperatorOption <= 6 && systemOperatorOption
>= 1)
                    throw new IllegalArgumentException();
            }
            catch (IllegalArgumentException e)
            {
```

```
                    System.out.println("Your entry is invalid");
                    valid = false;
                }
                catch (InputMismatchException e)
                {
                    System.out.println("Your entry is invalid");
                }
            }while (!valid);
            vehicles = new String[10];

            if (systemOperatorOption == 1) // add to queue
            {
                // check in the common waiting queue, if not check in the
ticket counter
                psmt = con.prepareStatement(query4);
psmt.setInt(1, 0);                rs =
psmt.executeQuery();

                if (rs.next())
                {
                    isARecordInWaiting = true;
queueEntryNumber = rs.getInt(1);
vehicleType = rs.getString(4);
fuelType = rs.getString(5);
                }
                con.close();

                if (isARecordInWaiting)
                {
                    if (fuelType == "diesel") {
                        if (vehicleType == "public transport") {
                            // 1
                            dispenserNumber = 1;

                            queue = new Queue(waitingQueue, vehicles, 0);
queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() != 0) {
waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }
} else // 2,3
                        {
                            dispenserNumber = 2;

                            queue = new Queue(waitingQueue, vehicles, 0);
```

```
queue.isFull(psmt, con, query2,
```

```
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() == 0) {
dispenserNumber = 3;                                recordCount = 0;
                                vehicles = new String[10];


                                queue = new Queue(waitingQueue, vehicles,
0);
                                queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() != 0) {
                                    waitingQueue = new
CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                                }
                            } else {
                                waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }
                        }
                    }
                    else if (fuelType == "petrol")
                    {
                        if (vehicleType == "car" || vehicleType == "van")
                        {
                            // 1 or 2
                            dispenserNumber = 1;


                            queue = new Queue(waitingQueue, vehicles, 0);
queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() == 0) {
dispenserNumber = 2;                                recordCount = 0;
                                vehicles = new String[10];
                                 queue = new Queue(waitingQueue,
vehicles,
0);
```

```
                                queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
```

```
                if (queue.getNumOfPositions() != 0) {
```

```
                                    waitingQueue = new
CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                                }
                            } else {
                                waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }

                        }
                        else if (vehicleType == "three wheeler")
                        {
                            // 3
                            dispenserNumber = 3;
                            queue = new Queue(waitingQueue, vehicles, 0);
queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() != 0) {
waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }
                        }
                        else if (vehicleType == "motor bike")
                        {
                            // 4
                            dispenserNumber = 4;
                            queue = new Queue(waitingQueue, vehicles, 0);
queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() != 0) {
waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }
```

```
                        }
else
                        {
                            // 2
                            dispenserNumber = 2;
                            queue = new Queue(waitingQueue, vehicles, 0);
```

```
queue.isFull(psmt, con, query2,
```

```
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 1);
if (queue.getNumOfPositions() != 0) {
waitingQueue = new CommonWaitingQueue();

waitingQueue.removeFromCommonWaitingQueue(psmt, con, query5,
dispenserNumber, queueEntryNumber);
                            }
}

                    }

                }

else
                {
                    // check in ticket counter
psmt = con.prepareStatement(query3);
psmt.setString(1,null);                     rs =
psmt.executeQuery();

                    if (rs.next())
                    {
                        isARecord = true;
                        ticketNumber = rs.getInt(1);
vehicleNum = rs.getString(2);
vehicleType = rs.getNString(3);
fuelType = rs.getString(4);
                    }
                    con.close();
recordCount = 0;                        if
(!isARecord)
                    {
                        if (fuelType == "diesel")
                        {
                            if (vehicleType == "public transport")
                            {
                                // 1
                                dispenserNumber = 1;

                                queue = new Queue(waitingQueue,vehicles,0);
queue.isFull(psmt, con, query2, dispenserNumber, fuelType, vehicleNum,
recordCount, ticketNumber, rs, queryToEnqueue, vehicleType,
queryToAddToCommonWaitingQueue, 0);
```

```
                    }
                    else // 2,3
                    {
                        dispenserNumber = 2;
```

```
                                queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 0);
numberOfPositions = queue.getNumOfPositions();
                                if (numberOfPositions == 0)
                                {
                                    dispenserNumber = 3;
recordCount = 0;
                                    vehicles  = new String[10];


                                    queue = new
Queue(waitingQueue,vehicles,0);
                                    queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 0);
                                }
                            }
                        }
                        else if (fuelType == "petrol")
                        {
                            if (vehicleType == "car" || vehicleType ==
"van")
                            {
                                // 1 or 2
                                dispenserNumber = 1;
                                queue = new Queue(waitingQueue,vehicles,0);
queue.isFull(psmt, con, query2, dispenserNumber, fuelType, vehicleNum,
recordCount, ticketNumber, rs, queryToEnqueue, vehicleType,
queryToAddToCommonWaitingQueue, 0);
numberOfPositions = queue.getNumOfPositions();

                                if (numberOfPositions == 0)
                                {
                                    // add to common waiting queue
dispenserNumber = 3;
recordCount = 0;
                                    vehicles  = new String[10];


                                    queue = new
Queue(waitingQueue,vehicles,0);
                                    queue.isFull(psmt, con, query2,
dispenserNumber, fuelType, vehicleNum, recordCount, ticketNumber, rs,
queryToEnqueue, vehicleType, queryToAddToCommonWaitingQueue, 0);
                                }
```

```
}
```

```
queue = new
Queue(waitingQueue,vehicles,0);
```

```
else if (vehicleType == "three wheeler")
```

```
                            {
                                // 3
                                dispenserNumber = 3;
                                queue = new Queue(waitingQueue,vehicles,0);
queue.isFull(psmt, con, query2, dispenserNumber, fuelType, vehicleNum,
recordCount, ticketNumber, rs, queryToEnqueue, vehicleType,
queryToAddToCommonWaitingQueue, 0);
                            }
                            else if (vehicleType == "motor bike")
                            {
                                // 4
                                dispenserNumber = 4;
                                queue = new Queue(waitingQueue,vehicles,0);
queue.isFull(psmt, con, query2, dispenserNumber, fuelType, vehicleNum,
recordCount, ticketNumber, rs, queryToEnqueue, vehicleType,
queryToAddToCommonWaitingQueue, 0);
                            }
else
                            {
                                // 2
                                dispenserNumber = 2;
                                queue = new Queue(waitingQueue,vehicles,0);
queue.isFull(psmt, con, query2, dispenserNumber, fuelType, vehicleNum,
recordCount, ticketNumber, rs, queryToEnqueue, vehicleType,
queryToAddToCommonWaitingQueue, 0);                             }

                        }
                    }
                }
            }
            //
}
        if (option == 3) // dispense operator
        {
//              MultiThreading thread1 = new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,1,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator);
//              MultiThreading thread2 = new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,2,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator);
//              MultiThreading thread3 = new
```

```
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,3,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator);
//            MultiThreading thread4 = new
```

```
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,4,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator);
//            MultiThreading thread5 = new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,1,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator);
//            MultiThreading thread6 = new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,2,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator);
//            MultiThreading thread7 = new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,3,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator);

            new Thread(new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,1,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,2,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,3,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"petrol",query7,query8,query9,4,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,450,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,1,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,2,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator));             new Thread(new
MultiThreading(psmt,con,query6,"diesel",query7,query8,query9,3,queueEntryN
umber,rs,isARecord,queue,dieselFuelDispenseManager,octaneFuelDispenseManag
er,date,430,dispenserOperator));
        }
    }      @Override      public  void run()
    {
```

Multi-Threading

```
mport com.sun.jdi.event.ThreadStartEvent;

import java.sql.Connection;
import
java.sql.PreparedStatement;
import java.sql.ResultSet; import
java.sql.SQLException; import
java.text.ParseException; import
java.text.SimpleDateFormat;
import java.util.Date;
import
java.util.InputMismatchException;
import java.util.Random; import
java.util.Scanner;

public class MultiThreading  extends Random implements Runnable
{
    private PreparedStatement psmt;
private Connection con;
     public MultiThreading(PreparedStatement psmt, Connection con, String
query6, String fuelType, String query7, String query8, String query9, int
dispenserNumber, int queueEntryNumber, ResultSet rs, boolean isARecord,
Queue queue, DieselFuelDispenseManager dieselFuelDispenseManager,
OctaneFuelDispenseManager octaneFuelDispenseManager, DateClass date,
float fuelPrice, DispenserOperator dispenserOperator) {         this.psmt
= psmt;         this.con = con;         this.query6 = query6;
this.fuelType = fuelType;         this.query7 = query7;
this.query8 = query8;         this.query9 = query9;
        this.dispenserNumber = dispenserNumber;
this.queueEntryNumber = queueEntryNumber;
this.rs = rs;
        this.isARecord = isARecord;
this.queue = queue;
        this.dieselFuelDispenseManager =
dieselFuelDispenseManager;         this.octaneFuelDispenseManager
= octaneFuelDispenseManager;         this.date = date;
this.fuelPrice = fuelPrice;
        this.dispenserOperator = dispenserOperator;
    }
    private String query6, fuelType, query7, query8,
query9;     private int dispenserNumber, queueEntryNumber;
private ResultSet rs;     private boolean isARecord;
private Queue queue;
    private DieselFuelDispenseManager dieselFuelDispenseManager;




    private OctaneFuelDispenseManager octaneFuelDispenseManager;
```

```
    private float fuelAmount, paidAmount; // don't get this as a parameter
in constructor
    private final Scanner sc = new
Scanner(System.in);     private DateClass date;
private float fuelPrice;
    private String date2; // no neeed to consider aas a parameter in
constructor
    private final SimpleDateFormat sdf1 = new SimpleDateFormat("dd-mmyyyy");
    private java.util.Date date3; // no need in constructor parameter
private java.sql.Date sqlDate; // no need in constructor parameter
private DispenserOperator dispenserOperator;     private float
remainingFuelAmount; // no need to in cons.
    @Override
public void run() {
try {
            queue.peakForFirstVehicle(psmt,con, query6,dispenserNumber,
rs, queueEntryNumber, fuelType, isARecord);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        if (isARecord)
        {
            // check for the availibility
if (fuelType == "petrol")
            {
                this.octaneFuelDispenseManager = new
OctaneFuelDispenseManager(0);
                this.remainingFuelAmount =
this.octaneFuelDispenseManager.checkForTheAvailability(psmt,con,query7,rs)
;
                if (this.remainingFuelAmount > 50)
                {
                    // enter fuel amout
                    this.fuelAmount = enterFuelAmount();
this.paidAmount = calculateThePaidAmount();
this.date = new DateClass();
                    this.date2 = this.date.displayTodayDate();
                    // convert to sql date
try {
                        this.date3 =
sdf1.parse(date2);                      } catch
(ParseException e) {                        throw new
RuntimeException(e);
                    }
                    this.sqlDate = new
java.sql.Date(this.date3.getTime());
                    this.dispenserOperator = new DispenserOperator();
this.dispenserOperator.pumpFuel(psmt, con, query8,
```

```
fuelAmount, paidAmount, sqlDate, queueEntryNumber);
```

```
                    // update repository

this.octaneFuelDispenseManager.updateTheRepository(psmt, con, query9,
fuelAmount);                 }                 else
                {
                    System.out.println("Not enough petrol fuel in
the repository");                 }             }             else
{
                this.dieselFuelDispenseManager = new
DieselFuelDispenseManager(0);
                this.remainingFuelAmount =
this.octaneFuelDispenseManager.checkForTheAvailability(psmt,con,query7,rs)
;
                if (this.remainingFuelAmount > 50)
                {
                    this.fuelAmount = enterFuelAmount();
this.paidAmount = calculateThePaidAmount();
this.date = new DateClass();
                    this.date2 = this.date.displayTodayDate();
                    // convert to sql date
try {
                        this.date3 =
sdf1.parse(date2);                     } catch
(ParseException e) {                         throw new
RuntimeException(e);
                    }
                    this.sqlDate = new
java.sql.Date(this.date3.getTime());
                    // pump fuel and update database
                    this.dispenserOperator = new DispenserOperator();
this.dispenserOperator.pumpFuel(psmt, con, query8, fuelAmount, paidAmount,
sqlDate, queueEntryNumber);
                    // update repository

this.dieselFuelDispenseManager.updateTheRepository(psmt, con, query9,
fuelAmount);                 }                 else
                {
                    System.out.println("Not enough diesel fuel in the
repository");
                }
            }
}
else
```

```
        {
            System.out.println("No vehicles available in "+fuelType+"
"+dispenserNumber+" queue");
        }
}
    public float enterFuelAmount()
    {
        float fuelAmount =
0;          boolean valid =
true;          do {
try{
                System.out.print("Enter fuel amount: ");
fuelAmount = sc.nextFloat();
            }
            catch (InputMismatchException e)
            {
                System.out.println("You entry is invalid");
valid = false;
            }
        }while (!valid);
return fuelAmount;
    }
    public float calculateThePaidAmount()
    {
        return this.paidAmount = this.fuelAmount * this.fuelPrice;
}

//    public Date enter
}
```

Fuel Dispense Manager Interface

```
import java.sql.Connection;
import
java.sql.PreparedStatement;
import java.sql.ResultSet;

public interface FuelDispenseManager
{
    public float checkForTheAvailability(PreparedStatement psmt,
Connection con, String query7, ResultSet rs);
    public void updateTheRepository(PreparedStatement psmt,
Connection con, String query9, float fuelAmount);     public void
installDispenser();     public void printEachDayStationDetails();
public void printLargestAmountDispensedVehicle();     public void
printDispenserWisedDetails();     public void stockFuelInDispenser();
}
```

Diesel Fuel Dispense Manager

```
import java.sql.Connection; import
java.sql.PreparedStatement; import
java.sql.ResultSet; import
java.sql.SQLException;
 public class DieselFuelDispenseManager implements
FuelDispenseManager{     public DieselFuelDispenseManager(float
remaining92OctaneAmount) {          this.remaining92OctaneAmount =
remaining92OctaneAmount;     }
    public float getRemaining92OctaneAmount() {
return remaining92OctaneAmount;
    }
    public void setRemaining92OctaneAmount(float remaining92OctaneAmount)
{
        this.remaining92OctaneAmount =
remaining92OctaneAmount;     }
    private float remaining92OctaneAmount;
    @Override
    public float checkForTheAvailability(PreparedStatement psmt,
Connection con, String query7, ResultSet rs) {
try {
            psmt = con.prepareStatement(query7);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(1, "diesel");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            rs = psmt.executeQuery();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        try {
            if (rs.next())
            {
                this.remaining92OctaneAmount = rs.getFloat(2);
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            con.close();
```

```
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return this.remaining92OctaneAmount;
    }

    @Override
    public void updateTheRepository(PreparedStatement psmt,
Connection con, String query9, float fuelAmount) {         try {
            psmt = con.prepareStatement(query9);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setFloat(1, this.remaining92OctaneAmount - fuelAmount);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(2, "92Octane");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            con.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        System.out.println("Diesel  repository  is  updated");
}

    @Override
    public void installDispenser() {

    }

    @Override
    public void printEachDayStationDetails() {
    }

    @Override
    public void printLargestAmountDispensedVehicle() {
    }

    @Override
```

```
    public void printDispenserWisedDetails() {
```

```
    }
     @Override
    public void stockFuelInDispenser() {

    }


}
```

Octane Fuel Dispense Manager

```
import java.sql.Connection; import
java.sql.PreparedStatement; import
java.sql.ResultSet; import
java.sql.SQLException;

public class OctaneFuelDispenseManager implements FuelDispenseManager{
     public OctaneFuelDispenseManager(float remaining92OctaneAmount)
{          this.remaining92OctaneAmount = remaining92OctaneAmount;
}
    public float getRemaining92OctaneAmount() {
return remaining92OctaneAmount;
    }
    public void setRemaining92OctaneAmount(float remaining92OctaneAmount)
{
        this.remaining92OctaneAmount =
remaining92OctaneAmount;     }
    private float remaining92OctaneAmount;
    @Override
    public float checkForTheAvailability(PreparedStatement psmt,
Connection con, String query7, ResultSet rs) {
try {
            psmt = con.prepareStatement(query7);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(1, "92Octane");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            rs = psmt.executeQuery();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        try {
            if (rs.next())
            {
                this.remaining92OctaneAmount = rs.getFloat(2);
}
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
```

```
            con.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return this.remaining92OctaneAmount;
    }

    @Override
    public void updateTheRepository(PreparedStatement psmt, Connection
con, String query9, float fuelAmount)
    {
try {
            psmt = con.prepareStatement(query9);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setFloat(1, this.remaining92OctaneAmount - fuelAmount);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(2, "92Octane");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            con.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        System.out.println("92Octane Repository is updated");
}

    @Override
    public void installDispenser() {

    }

    @Override
    public void printEachDayStationDetails() {
    }

    @Override
    public void printLargestAmountDispensedVehicle() {
    }
```

```
@Override
```

```
    public void printDispenserWisedDetails() {

    }
     @Override
    public void stockFuelInDispenser() {

    }

}
```

Dispenser Operator

```
import java.sql.Connection; import
java.sql.Date;
import java.sql.PreparedStatement; import
java.sql.SQLException;

public class DispenserOperator
{     public void pumpFuel(PreparedStatement psmt, Connection con, String
query8, float dispensedFuelAmount, float paidAmount, Date dispensedDate,
int queueEntryNumber)
    {
try {
            psmt = con.prepareStatement(query8);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(1, null);
} catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(2, "YES");
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(3, null);
} catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setFloat(4, dispensedFuelAmount);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(5, null);
} catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setFloat(6, paidAmount);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setString(7, null);
} catch (SQLException e) {
```

```
            throw new RuntimeException(e);
        }
try {
            psmt.setDate(8, dispensedDate);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            psmt.setInt(9, queueEntryNumber);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
try {
            con.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        System.out.println("Fuel pumped and remove from the queue");
    }
}
```

Customer public class

Customer

```
{
    public Customer(String vehicleNum, String vehicleType, String
fuelType) {
        this.vehicleNum = vehicleNum;
this.vehicleType = vehicleType;
this.fuelType = fuelType;
    }
    public String getVehicleNum() {
return vehicleNum;
    }
    public void setVehicleNum(String vehicleNum) {
this.vehicleNum = vehicleNum;
    }
    public String getVehicleType() {
return vehicleType;
    }
    public void setVehicleType(String vehicleType) {
this.vehicleType = vehicleType;
    }
    public String getFuelType() {
return fuelType;
    }
    public void setFuelType(String fuelType) {
this.fuelType = fuelType;
    }      private String
vehicleNum;     private String
vehicleType;     private String
fuelType; }
```

Queue `import org.apache.poi.xssf.usermodel.XSSFWorkbook;`

```
 import java.io.FileInputStream;
import
java.io.FileNotFoundException;
import java.io.IOException; import
java.sql.Connection; import
java.sql.PreparedStatement; import
java.sql.ResultSet; import
java.sql.SQLException;

public class Queue
{
    public Queue(CommonWaitingQueue waitingQueue, String[] vehicleQueue,
int numOfPositions) {
        this.waitingQueue = waitingQueue;
this.vehicleQueue = vehicleQueue;
this.numOfPositions = numOfPositions;
    }
    public CommonWaitingQueue getWaitingQueue() {
return waitingQueue;
    }
    public void setWaitingQueue(CommonWaitingQueue waitingQueue) {
this.waitingQueue = waitingQueue;
    }

    //    public Queue(String[] vehicleQueue) {
//        this.vehicleQueue = vehicleQueue;
//    }
    private CommonWaitingQueue waitingQueue;

    public String[] getVehicleQueue() {
return vehicleQueue;
    }
    public void setVehicleQueue(String[] vehicleQueue) {
this.vehicleQueue = vehicleQueue;
    }      private String[] vehicleQueue = new
String[10];

//    public Queue(String[] vehicleQueue, int numOfPositions) {
//        this.vehicleQueue = vehicleQueue;
//        this.numOfPositions =
numOfPositions; //    }      public int
getNumOfPositions() {         return
numOfPositions;
```

```
    }
```

```
    public void setNumOfPositions(int numOfPositions) {
this.numOfPositions = numOfPositions;
    }
    private int numOfPositions;

    public void displayNumberOfPositions(String fuelType,int
dispenserNumber)
    {
        for (int i = 0; i<this.vehicleQueue.length; i++)
        {
            if (this.vehicleQueue[i]==null)
            {
                this.numOfPositions += 1;
            }
}
        if (fuelType == "petrol")
        {
            System.out.println("No of remaining positions in petrol
dispenser "+dispenserNumber+" = "
                    +this.numOfPositions);
        }
else
        {
            System.out.println("No of remaining positions in diesel
dispenser "+dispenserNumber+" = "
                    +this.numOfPositions);
        }
}
    public void isFull(PreparedStatement psmt, Connection con, String
query2, int dispenserNumber, String fuelType, String vehicleNum, int
recordCount
    , int ticketNumber, ResultSet rs, String queryToEnqueue, String
vehicleType, String queryToAddToCommonWaitingQueue, int fromWaiting)
throws SQLException {
        psmt = con.prepareStatement(query2);
psmt.setInt(1,dispenserNumber);
psmt.setString(2, fuelType);
psmt.setString(3,null);          rs =
psmt.executeQuery();
         while
(rs.next())
        {
            vehicleNum = rs.getString(3);
ticketNumber = rs.getInt(1);
            this.vehicleQueue[recordCount] = vehicleNum;
recordCount+=1;
```

```
        }
```

```
        con.close();
        displayNumberOfPositions(fuelType,dispenserNumber);
```

```
        if (fromWaiting == 0)
        {
            if (numOfPositions == 0)
            {
                if (fuelType == "diesel" && dispenserNumber == 2)
                {
                    return;
                }
                else if (fuelType == "petrol" && dispenserNumber == 1)
                {
                    return;
                }
                this.waitingQueue = new CommonWaitingQueue();
this.waitingQueue.addToCommonQueue(psmt, con,
queryToAddToCommonWaitingQueue, ticketNumber, vehicleNum,
vehicleType, fuelType);             }             else
                addToQueue(psmt, con, queryToEnqueue, ticketNumber,
vehicleNum, vehicleType, fuelType, dispenserNumber);
        }
}
    public void addToQueue(PreparedStatement psmt, Connection con, String
queryToEnqueue, int ticketNumber,
                           String vehicleNum, String vehicleType,
String fuelType, int dispenserNumber) throws SQLException {
psmt = con.prepareStatement(queryToEnqueue);         psmt.setInt(1,
ticketNumber);         psmt.setString(2, vehicleNum);
psmt.setString(3,vehicleType);         psmt.setString(4, fuelType);
psmt.setInt(5, dispenserNumber);           psmt.executeQuery();
con.close();
        System.out.println("added to "+fuelType+" dispenser queue
"+dispenserNumber);
    }         public boolean peakForFirstVehicle(PreparedStatement psmt,
Connection con, String query6, int dispenserNumber, ResultSet rs, int
queueEntryNumber, String fuelType, boolean isARecord) throws SQLException
{
        psmt =
con.prepareStatement(query6);
psmt.setString(1, null);
psmt.setInt(2, dispenserNumber);         rs
= psmt.executeQuery();

        if (rs.next())
```

```
        {
            queueEntryNumber = rs.getInt(1);
            fuelType = rs.getString(5);
isARecord = true;
        }
        con.close();
return isARecord;
    }
}
```

Common Waiting Queue

```
import java.sql.Connection; import
java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;


public class CommonWaitingQueue
{
    private ArrayList<String> commonQueue = new ArrayList<>();

    public void addToCommonQueue(PreparedStatement psmt, Connection con,
String queryToAddToCommonWaitingQueue,
                                 int ticketNumber, String
vehicleNum, String vehicleType, String fuelType) throws SQLException
{         psmt =
con.prepareStatement(queryToAddToCommonWaitingQueue);
psmt.setInt(1, ticketNumber);         psmt.setString(2, vehicleNum);
psmt.setString(3,vehicleType);          psmt.setString(4, fuelType);
psmt.setInt(5, 0);         psmt.executeQuery();         con.close();
        System.out.println("Added to common waiting queue");
    }
    public void removeFromCommonWaitingQueue(PreparedStatement psmt,
Connection con, String query5,
                                             int dispenserNum,
int queueEntryNumber) throws SQLException {         psmt =
con.prepareStatement(query5);         psmt.setInt(1,0);
        psmt.setInt(2,dispenserNum);
psmt.setInt(3, queueEntryNumber);
psmt.executeQuery();
con.close();
        System.out.println("Vehicle is removed from a waiting queue");
    }
}
```

Date Class

```
import java.time.LocalDateTime; import
java.time.format.DateTimeFormatter;
 public class
DateClass
{
    private  DateTimeFormatter  dtf  =  DateTimeFormatter.ofPattern("dd-
MMyyyy");
    private LocalDateTime now = LocalDateTime.now();
public String displayTodayDate()
    {
        return dtf.format(now);
    }
}
```

Ticket Counter

```
import java.sql.Connection;
import
java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Scanner;

public class TicketCounter
{     private String fuelType;     private
double fuelAmount;     private String
vehicleNumber;     private String
vehicleType;     private String
dispenserNumber;     private String
ticketNumber;     private DateClass date;
private String[] vehicleQueue;     private
CommonWaitingQueue waitingQueue;
    private Queue vehicleList = new Queue(waitingQueue, vehicleQueue,10);


    public String getFuelType() {
return fuelType;
    }
    public void setFuelType(String fuelType) {
this.fuelType = fuelType;
    }
    public double getFuelAmount() {
return fuelAmount;
    }
    public void setFuelAmount(double fuelAmount) {



        this.fuelAmount = fuelAmount;
```

```
    }                   public    String
getVehicleNumber() {            return
vehicleNumber;
```

```
    }
    public void setVehicleNumber(String vehicleNumber) {
this.vehicleNumber = vehicleNumber;
    }
    public String getVehicleType() {
return vehicleType;
    }
    public void setVehicleType(String vehicleType) {
this.vehicleType = vehicleType;
    }
    public String getDispenserNumber() {
return dispenserNumber;
    }
    public void setDispenserNumber(String dispenserNumber) {
this.dispenserNumber = dispenserNumber;
    }
    public String getTicketNumber() {
return ticketNumber;
    }
    public void setTicketNumber(String ticketNumber) {
this.ticketNumber = ticketNumber;
    }
    public DateClass getDate() {
return date;
    }
    public void setDate(DateClass date) {
this.date = date;
    }      public Queue
getVehicleList() {          return
vehicleList;
    }      public void setVehicleList(Queue
vehicleList) {          this.vehicleList =
vehicleList;     }
```

```
    public TicketCounter() {
```

```
    }
    public ArrayList<Customer> getCustomers() {
        return customers;
    }
    public void setCustomers(ArrayList<Customer> customers) {
this.customers = customers;
    }

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

    public void issueTicket() {
        Scanner input = new Scanner(System.in);

        System.out.println("Enter the vehicle number: ");
setVehicleNumber(input.nextLine());

        System.out.println("Fuel type:");
        System.out.println("1 --> 92 Octane");
        System.out.println("2 --> Diesel");

        int fuelTypeNumber = input.nextInt();
         switch (fuelTypeNumber)
{             case 1:
                setFuelType("92 Octane");
allocateDispenser("92 Octane");
break;              case 2:
                setFuelType("Diesel");
allocateDispenser("Diesel");
break;              case 0:
break;              default:
                System.out.println("Invalid input.");
        }

        System.out.println("Enter the fuel amount: ");
setFuelAmount(input.nextDouble());
    }      private void allocateDispenser(String
fuelType) {
        Scanner input = new Scanner(System.in);
if (fuelType.equals("92 Octane")) {
System.out.println("Vehicle Type: ");
```

```
        System.out.println("1 --> Car");
```

```
            System.out.println("2 --> Van");
            System.out.println("3 --> Three wheeler");
System.out.println("4 --> Motor bike");
            System.out.println("5 --> Other vehicle");

            int vehicleTypeNumber = input.nextInt();
             switch (vehicleTypeNumber)
{                 case 1:
                    setVehicleType("Car");
setDispenserNumber("P1 and P2");
break;                  case 2:
                    setVehicleType("Van");
setDispenserNumber("P1 and P2");
break;                  case 3:
                    setVehicleType("Three
wheeler");
setDispenserNumber("P3");
break;                  case 4:
                    setVehicleType("Motor bike");
setDispenserNumber("P4");
break;                  case 5:
                    setVehicleType("Other
vehicle");
setDispenserNumber("P2");
break;                  case 0:
break;                  default:
                    System.out.println("Invalid input.");
            }
        } else if (fuelType.equals("Diesel")) {
            System.out.println("Vehicle Type: ");
            System.out.println("1 -->  Public transport vehicles");
System.out.println("2 --> Other vehicles");

            int vehicleTypeNumber = input.nextInt();
             switch (vehicleTypeNumber)
{                 case 1:
                    setVehicleType("Public transport
vehicles");                         setDispenserNumber("D1");
break;                  case 2:
                    setVehicleType("Other vehicles");
setDispenserNumber("D2 and D3");
break;
```

```
        case 0:
```

```
                    break;
default:
                    System.out.println("Invalid input.");
            }
}
        ticket();



    }
    public void ticket() {
        for (int i = 0; i < 1; i++) {
            System.out.println("* * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *");
            System.out.println("*    --------------------                ----
---------     *");
            System.out.println("*          " + getVehicleNumber() + "
" + getDate() + "           *");
            System.out.println("*    --------------------                ----
---------     *");
            System.out.println("*               -------------------------
*");
            System.out.println("*             " + getFuelType() + "
Dispenser " + getDispenserNumber() + "      *");
            System.out.println("*                    Ticket Number: " +
getTicketNumber() + "                    *");
            System.out.println("*               -------------------------
*");
            System.out.println("* * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *\n");
        }
    }
    public void saveCustomerDetails(Connection con, String query) throws
ClassNotFoundException, SQLException {
        PreparedStatement stmt = con.prepareStatement(query);
stmt.setString(1, customers.get(customers.size()-
1).getVehicleNum());
        stmt.setString(2, customers.get(customers.size()-
1).getVehicleType());
        stmt.setString(3, customers.get(customers.size()1).getFuelType());

stmt.execute();
con.close();
    }      public void issueTicket(Customer
customer)     {
```

```
            customers.add(customer);
        }
}
```