**Vehicle Rental Software program**

**Professional Software Development 2**

**Group 7 – COM809**

**Authors**:

Patrick Donnelly

Aldrian Dungca

Alastair Dore

**Submission Date:**

**23/11/20**

Contents

[INTRODUCTION](#_gjdgxs) **3**

[DEVELOPMENT](#_30j0zll) **3**

[The Software Development Lifecycle (SDLC)](#_1fob9te) 3

[The Software Requirement Specification (SRS)](#_3znysh7) 4

[The Design Document Specification (DDS)](#_2et92p0) 5

[FUTURE DEVELOPMENTS](#_eunw84z2i0fj) **8**

[Testing](#_cvtwpfwgdsj) **9**

[Patrick Donnelly](#_7i7vufhc5hcx) 9

[Aldrian Dungca](#_r0ueq23t06pf) 12

# INTRODUCTION

The tourism industry within Ireland and Northern Ireland contributes substantially to their respective country’s economy. The island of Ireland is considered a must-see holiday destination. Estimates in 2018 reported that tourism generated 5.6billion euro for the Irish economy (Tourism – gov.ie). Unfortunately, the tourism industry, like many others have had been negatively impacted by the Covid-19 pandemic.

When domestic and international travel becomes safe once again, it is imperative that businesses within the tourism industry are fully prepared, in the interests of the economy and the livelihood of those individuals within the sector.

Despite Ireland being a favourite destination among holiday makers, its public transport infrastructure is inferior to other neighbouring countries. This dictates that visitors must arrange their own transport and this should be considered an opportunity for the Vehicle rental sector.

As a group we have produced a software system to deal with individual and multi-car (fleet) rentals and perform various other functions (details of which can be seen in the user documentation (Appendix 1)).

# DEVELOPMENT

## The Software Development Lifecycle (SDLC)

A collaborative, approach was agreed by the team.

Trello (<https://trello.com/>*)* was utilized for communication and task allocation

Github (<https://github.com/>) was used develop, review and push code for other team members to access. It was identified that although GitHub is a very useful application for collaboration, team members required further training to maximize efficient use of the application.The Intellij IDE was used by all team members.

Research into the vehicle rental industry to identify and define the system requirements was conducted and documented in the Software Requirement Specification (SRS) document. The SRS document was reviewed and the design architecture decided upon. The design architecture is outlined in the Design Document Specification (DDS). The DDS includes program preparation sheets and UML class diagrams. The DDS demonstrates the objects, classes, methods, arrays and their relationships.

## The Software Requirement Specification (SRS)

The product requirements identified by our customer (fictional) are as follows:

* Vehicles rented may be Cars, Heavy Goods Vehicles (HGV), or Multi-Person Vehicles (MPV)
* The program must count:
  + Total number of vehicles rented
  + Total cars rented
  + Total HGV rented
  + Total MPV rented
* The program must calculate the total cost of a rental based on a daily fee, the length of rental and any additional charges incurred based on any miles travelled over the contracted Daily Mileage allowance of 150. Additional miles are charged at 20p per mile
* The program should provide a function to search rentals based on:
  + Type
  + All rentals
* A ‘discount’ facility should be included which may be implemented or not i.e. during a promotion period
* Common attributes in all 3 classes must be:
  + Vehicle Make,
  + Vehicle Registration Number,
  + Type (either Car, HGV, MPV)
  + Mileage before rental
  + Duration of rental
  + Unique Rental ID;
  + Date of Order;
  + Daily fee
* Features specific to Cars:
  + Number of seats
* Features unique to Cars:
  + number of doors
* Features specific MPVs
  + Number of seats
* Features unique to HGVs
  + Height (m)
  + Load Capacity (kg)
* The system should ascertain from the user what type of vehicle they would like to place a rental order for. The system should then prompt the user to enter, only the features relevant to that vehicle type.
* The program should save all rentals to a file for future access and usage.
* The program should calculate total revenue generated from rentals

Since the rental program is executed purely on the IntelliJ IDE, the chance of the user missing information on the terminal is highly likely. To reduce the risk of this occurring, a ‘Pause’ class is required to freeze the screen for 3 seconds before showing the Main menu again. Additionally, if the user inputs the wrong data type for a given variable i.e string type instead of integer , the program should not terminate. Instead, the program should highlight to the user that the entered data type is invalid and prompt the user to try again.

## The Design Document Specification (DDS)

1. **Initial Meeting:**

**Date: 21/10/20**

The initial classes were identified as:

1. ‘**VehicleRental’** (super class -Team Member Patrick)
2. ‘**CarRental’** (subclass - extends ‘VehicleRental’ -Team Member Aldrian)
3. ‘**MPVRental’** (subclass - extends ‘VehicleRental’-Team Member Alastair)
4. ‘**HGVRental’** (subclass - extends ‘VehicleRental’ – Team Member Patrick)
5. ‘**TestRental’** – provisionally planned to create Rental order objects

The classes were divided between team members (see appendix 2). UML class diagrams (see appendix 3), initial development and preliminary testing was commenced. The process outlined above was continually reviewed and repeated as scrum cycles of the agile methodology.

1. **Second Meeting:**

**Date: 29/10/20**

Discussion regarding ‘Search’ functionality to allow user to search for historical rental orders by ‘type’ or ‘search all rentals’. (Initial code developed by Aldrian).

1. **Further notable Developments;**

**Date: 30/10/20**

SummaryTable’ class developed which added rental orders to an array and provided functionality to print all orders (with associated order information) from a multidimensional array (Initial code developed by Alastair).

1. **Third Meeting:**

**Date: 11/11/20**

*Main Issue discussed:*

How to add further rentals an array?

*Plan:*

Investigate how to implement an arraylist for appending new rental orders.

*Other points:*

Create a ‘MainMenu’ class to improve end user experience (Patrick to develop this functionality)

Options from MainMenu to initially include: (initial code developed by Patrick)

1. Create Rental Order

a) Car rental

b) MPV rental

c) HGV rental

1. Search Rental
2. Search by ‘Type’
3. Search all rentals

Options were extended to include:

\* Create multi-Vehicle order

\* return a vehicle

1. **Fourth Meeting:**

**Date: 13/11/20**

\**Rentals* arraylist implemented to append new rental orders (Developed by Aldrian and Patrick as an extension to Alastair’s initial code to store rentals in array)

\**RentalID* arraylist, parallel to *Rentals* arraylist, used to iterate through to identify specific RentalOrder objects based on index position.

1. **Further notable Developments:**

**Date: 16/11/20**

**\***’FileManager’ class developed for reading and writing text file and producing a list of orders created (initial code developed by Alastair)

1. **Further notable Developments:**

**Date: 17/11/20**

\*’ErrorHandler’ class to handle integer/double input errors (initial code developed by Alastair).

1. **Fifth meeting:**

**Date: 18/11/20**

\*Group discussions regarding testing and formatting.

# FUTURE DEVELOPMENTS

Implementation of a graphical user interface (GUI).

Create log in and password function that remembers user names and passwords

Restructuring the class system so that the array list of car, HGV and MPV objects can be serialised and converted to Json file and saved to a file. We tried to do this using Google’s Gson library, however it appears that our classes didn’t allow this. This would require refactoring the various vehicle rental classes to allow the program to upload all previous rental order objects created and the user could search through them, or view a summary of all orders. Currently the user is only able to search orders generated in the current session. Currently receipts generated using the toString method are saved to individual text files, and the user can retrieve these.

**Any other future development ideas guys?**

As the company expands to rent out motorbikes and bicycles in it’s push to be more environmentally friendly, and renting out mobile homes and caravans to cater for tourism after COVID, more sub-classes may be added.

When new rental orders are created, a copy of the receipt may be emailed to the customer.

Customer details may be saved inline with GDPR, and a loyalty discount may be applied to returning customers.

When the user chooses option 4. Log details of a returned vehicle, they have to remember the initial mileage. In future developments we may remind the user of the original mileage, and confirm with them if they enter a final mileage lower than the original mileage.

# TESTING

## Patrick Donnelly

1. Testing the toString method of the super class which extends to the subclasses before and after return details have been entered

CREATING CAR RENTAL

====================

Creating Car rental 1 of 1

Make: VW

Reg: ABC123

Miles before:5000

Days rented for: 5

Daily fee (£): 55

How many seats in this car? 5

How many doors in this car? 5

CAR RENTAL CREATED

Display rental details before vehicle 10001 returned:

Date of order: 20-11-2020 15:43:54

Rental ID: 10001

Vehicle Make: VW

Registration Number: ABC123

Rental Duration (Days): 5

Daily Cost: £55.00

Total Cost (before any additional fees): £275.00

Daily Mileage Allowance: 150 miles

Start Mileage: 5000 miles

End Mileage: to be recorded on vehicle return

Average Daily Miles: to be calculated on vehicle return

Additional Miles (over that contracted): to be calculated on vehicle return

Extra Fee: £0.20 per mile: to be calculated on vehicle return

Discount Percentage applied (selected at random): 0%

Total Cost (after 0 % discount applied): to be calculated on vehicle return

No. of seats in car: 5

No. of doors in car:5

Enter return details and display all details of rental order 10001:

Date of order: 20-11-2020 15:43:54

Rental ID: 10001

Vehicle Make: VW

Registration Number: ABC123

Rental Duration (Days): 5

Daily Cost: £55.00

Total Cost (before any additional fees): £275.00

Daily Mileage Allowance: 150 miles

Start Mileage: 5000 miles

End Mileage: 7000 miles

Average Daily Miles: 400.00 miles

Additional Miles (over that contracted): 1250.00

Extra Fee: £0.20 per mile: £250.00

Discount Percentage applied (selected at random): 0%

Total Cost (after 0 % discount applied): £525.00

No. of seats in car: 5

No. of doors in car:5

Patrick Donnelly

1. Testing the input from keyboard for gaining vehicle details

Single/multiple Vehicle order function:

Create a Rental order

===============

What type of vehicle rental order would you like to make?:

1. Car

2. MPV

3. HGV

2

CREATING BUS RENTAL

====================

Creating Bus rental 1 of 1

Make: Reg: **(compiler skipped entering ‘Make’)**

Corresponding Code:

System.*out*.print("Make: ");  
make = VehicleRental.*keyboard*.nextLine();

Changed to:

System.*out*.print("Make: ");  
make = VehicleRental.*keyboard*.next();

Re-test Code:

CREATING BUS RENTAL

====================

Creating Bus rental 1 of 1

Make: VW **(compiler now stopped to allow user input)**

Reg: ABC123

Tested using Multi-Vehicle option

CREATING BUS RENTAL

====================

Creating Bus rental 1 of 2

Make: VW **(compiler allowed user input)**

Reg: HDt342

Miles before:500

Days rented for:

Creating Bus rental 2 of 2

Make: VW **(compiler allowed user input)**

Reg: JDT23

Miles before:500

## Aldrian Dungca

### Testing the ‘Search’ class

The function of the ‘Search’ class is to allow the user to search rental orders created during the current session. It allows for 2 search scenarios:

1. Search by rental type (Car,MPV,HGV)
2. Search all rentals

Initially, the development group stored all rental orders in an Array. However, during the first iteration of testing, it was discovered that arrays are of fixed length and therefore did not allow for adding extra elements after the array is declared. This meant the user had to hardcode in the rental objects to the array after they were created. It was decided by the development team on the third meeting to investigate ArrayLists and determine whether they could be utilized instead of Arrays. This proved to be a valuable decision as it was determined that Arraylists can grow or shrink in size dynamically. The transition from Array to ArrayList is shown below:

public static void searchByType(VehicleRental[] rentalsArray)

public static void searchByType(ArrayList<VehicleRental> Rentals)

To fully utilize the ArrayList, each subclass (CarRental, HGVRental and MPVRental) needed to be updated to include a method which creates an order and automatically adds that order to the Rentals ArrayList. An example of a car order being added to the ArrayList is shown below:

(the ArrayLists below are declared in the super-class)

*//arraylist to store all vehicle rentals*

public static ArrayList<VehicleRental> *Rentals*=new ArrayList<>();

public static ArrayList<Integer> *RentalsID*=new ArrayList<>();

(this method is within the CarRental sub-class)

public static void makeCarOrder(int num) {

*//creating object*

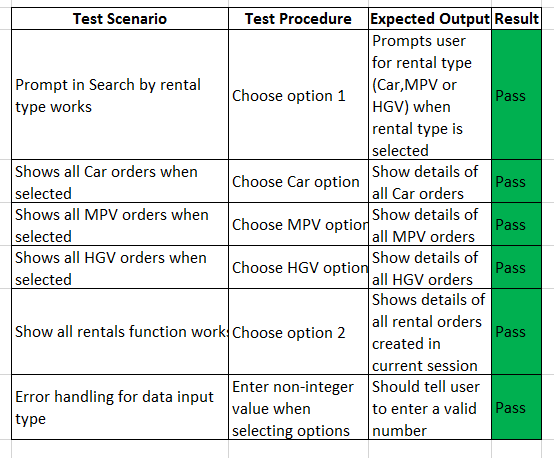
CarRental car1 = new CarRental(type, make, reg, milesBefore, days, fee, seats, doors);

*//adding created object to arraylist*

*Rentals*.add(car1);

Additional minor updates to the Search class were also required to implement the ArrayList, this included using .get function and using .size as opposed to .length.

Once the updates were completed, the development group tested all scenarios for the Search class. The results of this testing is shown in the table below:



**Testing ‘makeCarOrder’ method**

The aim of the makeCarOrder method is to initialize a new CarRental object and add this new object into the Rentals ArrayList so it can be accessed by other classes such as the Search class and SummaryTable class.

This class was tested for functionality and the results are illustrated on the table below:

Alastair Dore

### Testing ‘MPVRental’ class

The aim of the MPVRental class is to contain the blue-prints for an MPVRental order object, it’s attributes and methods to instantiate and display the attributes. The testBusOrder method was written to create an rental object quickly without user input.

A new class, TestMPVRental was created to test this class and methods:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Procedure** | **Expected Output** | **Result** |
| Instantiate MPVRental Object | Instantiate object | No error thrown | Pass |
| Values displayed from toString method equal those entered | Print object contents | Check values equal those entered | Pass |
| Date equals date of rental order | Print object contents | Check date and time match date and time | Pass |
| Check set and get methods | Call set and get methods | Output from get equals set | All passed except getMake. All passed except getMake. Found the error in setMake in VehicleRental. Fixed and passed |
| Test all values were updated | Print object contents | Values equal those inputted | Pass |
| File was saved to correct folder | Open file in folder | Values equal those inputted | Pass |
| Test MPV order is correctly generated through main menu | Run program and choose menu item one. Then check values | Values equal those inputted | Pass |

Alastair Dore

**Other Tests**

### Testing ‘Pause’ class

The aim of the MPVRental class is to pause the display to allow the user to view the order, or summary table before returning to the main menu.

A few calls to this method were made from the TestMPVRental class with different time values and timed against a timer. All times were correct.

**Testing ‘ErrorHandler’ class**

The aim of this class is to prevent the program ending when a user enters text when a number or decimal is expected.

To test this class we tried entering text when a number was expected and it passed.

**Other Tests**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Procedure** | **Expected Output** | **Result** |
| Main menu | Enter number out of range | Display menu again | Pass |
| Enter text | Ask for a valid number | Pass |
| Create MPV Rental order from menu option 1 | Enter text | Displays correct output | Pass |
| Object was saved to the array | Display summary list option 7 | Displays correct details | Pass |
| Order can be through option 3 | Choose Search by rental type and search all rentals | Displays the correct order | Pass |
| MPV can be returned using option 4, and the corect details are entered | Choose option 4 and enter a number greater than the original mileage. Then view the order. | Mileage is updated | Pass |
| Order has been saved to the folder | Choose option 9 | File exists. Open it and check the values are correct | Pass |

Alastair Dore

### Testing ‘FileManager’ class

The aim of the MPVRental class is to check if the folder order\_receipts exists and if not create it. Write an order to a file and save it in order\_receipts, display a list of all orders in the directory, and open a specified file.

A new class, TestFileManager was created to test this class and methods:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Procedure** | **Expected Output** | **Result** |
| create folder if none exists | Delete old folder. Call createFolder method. | New folder created | Pass |
| Doesn't over-write folder if folder already exists | Call createFolder method. | Folder contains same contents as before | Pass |
| write() method writes a file to the correct folder | create testBusOrder, call write method. | File name matches the time of order, and contents equal those in the MPVRental class | Pass |
| getAllReceipts() displays all files in folder | call method and check the list is the same as the contents of the folder | The list is the same as the contents of the folder | Pass |
| getAllReceipts() doesn't crash if the folder is empty | Delete the folder and run getAllReceipts | "No Orders" is displayed | Pass |
| getAllReceipts() doesn't crash if you enter a decimal or text | Enter text, number, decimal when prompted | Asks for a valid input | Pass |
| getAllReceipts() doesn't crash if you enter a number out of range | Enter a number out of range | Asks for a number within the range | Pass |
| getAllReceipts() and read() open the correct file | Enter a number within range and check the contents against the file contents | The file contents are the same as what is displayed | Pass |

Alastair Dore

### Testing ‘FileManager’ class

The aim of the Summary class is to display a summary table of orders generated in the current session. Also to show the total mileage and revenue for that session.

|  |  |  |
| --- | --- | --- |
| **Test Procedure** | **Expected Output** | **Result** |
| Choose option 1 and create one of each type of order |  | Pass |
| Choose option 7 | All orders are present. Columns are correct and totals are correct | Pass |
| Choose option 4, update the final mileage for each order |  | Pass |
| Choose option 7 | Columns and total values are calculated correctly | Pass |

Appendix 1- (User documentation)

Vehicle Rental System

(User documentation)

i.

Vehicle Rental System

Index Page

Introduction iii.

Rent a single Vehicle iii.

Rent Multiple Vehicles iv.

Search Existing orders v.

Log details of a returning Vehicle vii.

Update Discount list viii.

Display number of rental orders viii.

Print Summary list of all rentals ix.

Display Total Revenue x.

Access rental receipts on our Database x.

Quit xi.

ii

**Introduction:**

This Vehicle rental software package was designed to manage the rental orders of Cars, MPVs and HGVs.

This document will guide you through the various functions which can be performed by the program.

Main Menu - consists of a Switch statement which prompts user to make a selection

**Rent a single Vehicle**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

To rent a single vehicle, select option 1 by entering the number on the keyboard and pressing enter

Create a single vehicle order will display the ‘Create a Rental order option’.

Create a Rental order

===============

What type of vehicle rental order would you like to make?

1. Car

2. MPV

3. HGV

Make your selection by entering the number for the Vehicle type on the keyboard and pressing enter.

Enter the Vehicle details prior to rental

iii.

CREATING CAR RENTAL

====================

Creating Car rental 1 of 1

Make: VW

Reg: ABC123

Miles before:2000

Days rented for: 5

Daily fee (£): 55

How many seats in this car? 5

How many doors in this car? 5

**Rent Multiple Vehicles**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

To rent multiple vehicles, select option 2 from the main menus by entering the number on the keyboard and pressing enter

Create a multi-vehicle (fleet) rental order

===========================================

What type of vehicle rental order would you like to make?:

1. Car

2. MPV

3. HGV

1

Please enter number of cars you wish to rent

2

Make your selection by entering the number for the Vehicle type on the keyboard and pressing enter.

Next enter the number of Vehicles you wish to rent when prompted.

Please enter number of cars you wish to rent

2

CREATING CAR RENTAL

====================

Creating Car rental 1 of 2

Make: VW

Reg: ABC123

Miles before:500

Days rented for: 5

Daily fee (£): 55

How many seats in this car? 5

How many doors in this car? 5

CAR RENTAL CREATED

====================

Creating Car rental 2 of 2

Make: Audi

Reg: DEF456

Miles before:400

Days rented for: 4

Daily fee (£): 44

How many seats in this car? 4

How many doors in this car? 4

**Search Existing orders**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

Select option 3

v.

Search Vehicles

===============

Your Options (please enter a number):

1. Search By Rental Type

2. Search all rentals

Select option 1 or 2 accordingly

Date of order: 20-11-2020 01:17:05

Rental ID: 10002

Vehicle Make: VW

Registration Number: ABC123

Rental Duration (Days): 5

Daily Cost: £55.00

Total Cost (before any additional fees): £275.00

Daily Mileage Allowance: 150 miles

Start Mileage: 500 miles

End Mileage: to be recorded on vehilce return

Average Daily Miles: to be calculated on vehicle return

Additional Miles (over that contracted): to be calculated on vehicle return

Extra Fee: £0.20 per mile:to be calculated on vehicle return

Discount Percentage applied (selected at random): 0%

Total Cost (after 0 % discount applied): to be calculated on vehicle return

No. of seats in car: 5

No. of doors in car:5

Date of order: 20-11-2020 01:18:34

Rental ID: 10003

Vehicle Make: Audi

Registration Number: DEF456

Rental Duration (Days): 4

Daily Cost: £44.00

Total Cost (before any additional fees): £176.00

Daily Mileage Allowance: 150 miles

Start Mileage: 400 miles

End Mileage: to be recorded on vehilce return

Average Daily Miles: to be calculated on vehicle return

Additional Miles (over that contracted): to be calculated on vehicle return

Extra Fee: £0.20 per mile:to be calculated on vehicle return

Discount Percentage applied (selected at random): 0%

Total Cost (after 0 % discount applied): to be calculated on vehicle return

No. of seats in car: 4

No. of doors in car:4

Full details of Ordered vehicles will be displayed

vi.

**Log details of a returning Vehicle**

**Vehicle Rentals**

**===============**

**What would you like to do?**

**Your Options (please enter a number):**

**1. Make a single Vehicle rental order**

**2. Make a multi-Vehicle (fleet) Vehicle Rental order**

**3. Search existing orders**

**4. Log details of a returned vehicle**

**5. Update Discount List**

**6. Display number of rental orders**

**7. Print Summary list of all rentals**

**8. Display Total Revenue**

**9. Access rental receipts on our Database**

**0. Quit?**

**4**

**Return a Vehicle**

**================**

**Please enter the Vehicle rental ID you wish to return:**

**10001**

**Please enter end mileage of Rental id: 10001**

**2000**

To log details of a Vehicle being returned, select option 4 and then enter the Rental ID of the Vehicle being returned – in the example RentalID 10001 is being returned.

Enter the mileage of the vehicle on its return.

vii**.**

**Update Discount list**

**Vehicle Rentals**

**===============**

**What would you like to do?**

**Your Options (please enter a number):**

**1. Make a single Vehicle rental order**

**2. Make a multi-Vehicle (fleet) Vehicle Rental order**

**3. Search existing orders**

**4. Log details of a returned vehicle**

**5. Update Discount List**

**6. Display number of rental orders**

**7. Print Summary list of all rentals**

**8. Display Total Revenue**

**9. Access rental receipts on our Database**

**0. Quit?**

**5**

**This is the array of percentage discounts: 0, 21, 8, 35, 4, 28, 27, 4, 15, 13**

To update the discount list, select option 5. This automatically updates the list

The discount list is a list of 10 numbers (representing a percentage value). The 10 values are generated at random. When the discount list has been updated, each new rental order will randomly be allocated a percentage discount which will be applied to the customers’ final bill.

If the update Discount list is now performed at the beginning of a session, there will be no discounts applied.

This feature can be utilized intermittently, for example – when running a time limited promotion.

**Display number of rental orders**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

6

Number of Vehicle Rentals

=========================

viii.

Your Options (please enter a number):

1. Number of Car rentals

2. Number of MPV rentals

3. Number of HGV rentals

4. Total number of rentals

4

Total number of Vehicle rentals: 3

To display rental totals, select option 6 and then select the rental type you wish to get the total of. In the example the ‘Total Number of rentals’ was requested and the number 3 was displayed

**Print Summary list of all rentals**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

To print a summary of all rentals, select option 7

The summary table will be displayed as shown below

Summary of Rentals

==================

Summary of all Vehicle rental Orders:

=================

Display Daily Rental Summary

Date OrderID Type Miles Travelled Cost(£)

20-11-2020 10001 Car 0 725.0

20-11-2020 10002 Car 0 425.0

20-11-2020 10003 Car 0 576.0

ix.

Total number of buses rented: 0

Total number of cars rented: 3

Total number of HGVs rented: 0

Total number of Vehicles rented 3

Total mileage for all vehicles rented: 0

Total revenue from all vehicles rented: £0.00

**Display Total Revenue**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

To display total Revenue – select option 8

Total Revenue

=============

Total revenue generated from all rentals: £1,726.00

**Access rental receipts on our Database**

Vehicle Rentals

===============

What would you like to do?

Your Options (please enter a number):

1. Make a single Vehicle rental order

2. Make a multi-Vehicle (fleet) Vehicle Rental order

3. Search existing orders

4. Log details of a returned vehicle

5. Update Discount List

6. Display number of rental orders

7. Print Summary list of all rentals

8. Display Total Revenue

9. Access rental receipts on our Database

0. Quit?

x.

To access receipts on the system select option 9. The first ID is the date and time of order created, which will be unique. The second ID is the rental ID generated by the system (which is not shown in the image below as it was added since).

Historical Orders on our system

===========================

Enter number of order you'd like to see

0 Car\_receipt\_17112020\_125233.txt

1 Car\_receipt\_17112020\_210232.txt

2 Car\_receipt\_17112020\_210746.txt

33 MPV\_receipt\_17112020\_141056\_10001.txt

34 MPV\_receipt\_17112020\_141057\_10002.txt

35 MPV\_receipt\_17112020\_141058\_10003.txt

36 MPV\_receipt\_17112020\_143744\_10001.txt

37 MPV\_receipt\_17112020\_143746\_10002.txt

Then select the order number you wish to see

Date of order: 18-11-2020 00:23:47

Rental ID: 10002

Vehicle Make:

Registration Number: HSUD345

Rental Duration (Days): 7

Daily Cost: £70.00

Total Cost (before any additional fees): £490.00

Daily Mileage Allowance: 150 miles

Start Mileage: 3000 miles

End Mileage: to be recorded on vehilce return

Average Daily Miles: to be calculated on vehicle return

Additional Miles (over that contracted): to be calculated on vehicle return

Extra Fee: £0.20 per mile:to be calculated on vehicle return

Discount Percentage applied (selected at random): 18%

Total Cost (after 18 % discount applied): to be calculated on vehicle return

No. of seats in car: 5

No. of doors in car:4

**Quit**

To quit the program, select ‘0’ from the main menu.

**Any other input - Default**

To prevent the program crashing, or ending abruptly, a default case was added to the end of the switch statement. This means that if the user enters another number, it will display the menu again.

Appendix 2

**Vehicle Rental Program**

Design:

(\*please note that as a team we used a collaborative and agile approach and in doing so there were classes which required input from multiple team members\*)

**Patrick**

VehicleRental Class (Super Class)

HGVRental Class (sub Class)

MainMenu Class (class created for program functionality) (added to by Alastair and Aldrain)

VehicleRentalManager class (it only calls the main menu ()) (class created for program functionality)

**Aldrian**

CarRental class (sub Class)

Search class (class created for program functionality)

SummaryTable Class (contributed by Alastair and Patrick (class created for program functionality)

**Alastair**

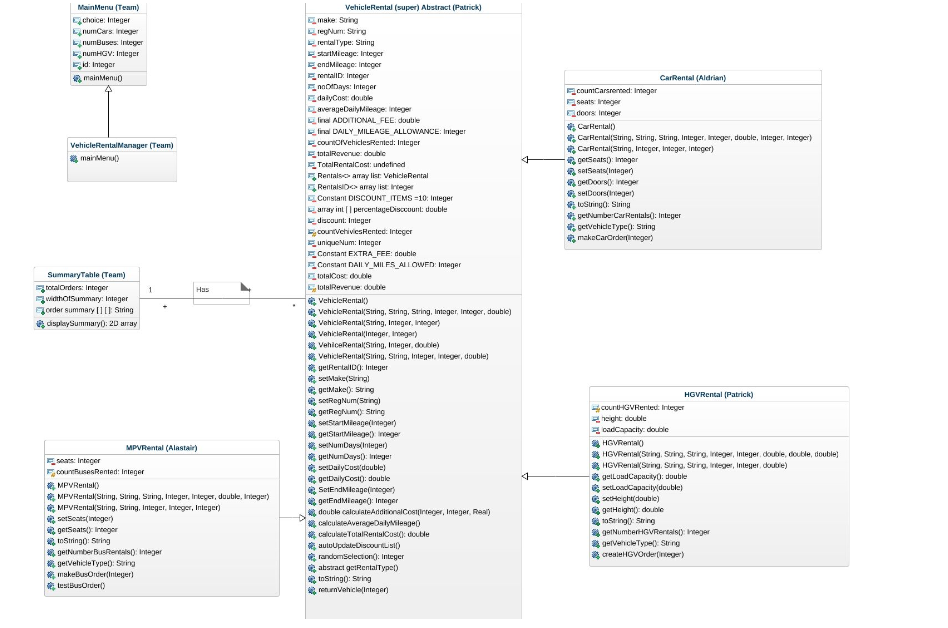
MPVRental class (sub Class)

ErrorHandler class (class created for program functionality)

FileManager Class (Contributed to by Adrian) (class created for program functionality)

Pause class (class created for program functionality)

Appendix 3 (UML Diagram)



Appendix 4 - Program Prep Sheet (VehicleRental.java)

Program Preparation Sheet

**Name**: Patrick Donnelly **Registration No**: B00823962

**Date**: 21/10/20

**Class Name and Description**: VehicleRental.java

An abstract class which is a used by subclasses to create vehicle rentals of various types.

**Program Design Data Requirements**

Declare ArrayList ArrayList<VehicleRental> Rentals=new Arraylist<>()

Declare another ArrayList ArrayList<VehicleRental> RentalsID=new Arraylist<>()

Declare -unique number

int uniqueNum = 10001

Declare rentalID

Int rentalID;

Declare and initialise countVehicles

Static int countVehiclesRented=0

Declare constant number of discounts

final int DISCOUNT\_ITEMS = 10

Declare and initialise array for discount items:

int [] percentageDiscount = new int[DISCOUNT\_ITEMS]

update Discount array

Declare discount variable which will be initialised from int discount = pick

Random array index selector

randomSelection(){

int pick = percentageDiscount[math.random\*discount\_Items]

return pick

Default Constructor method() VehicleRental()

rentalID=uniqueNum;

uniqueNum++;

countVehiclesRented++;

Altezzprnative constructor()

rentalID=uniqueNum;

uniqueNum++;

countVehiclesRented++;

VehicleRental(String Type, String Make,

String RegNum, int MilesBefore, int lengthDays,  
 double dailyFee)

Alternative constructor()

rentalID=uniqueNum,uniqueNum++;

countVehiclesRented++;

VehicleRental(String itasRegNum, int MilesBefore, int lengthDays)

Get/Set methods for:

String make, String regNum,

String rentalType;

int startMileage, int numDays

double dailyCost;

int endMileage;

totalRentalCost;

protected static double totalRevenue;

\*Ren

Get methods for:

String dateID, String dateOfOrder

Int totalMileage, , int rentalID

V

ScalcculateAverageDailyMiles yMiles**()**

(double) (endMileage-startMileage)/numDays;

' averageDailyMiles,

final double EXTRA\_FEE\_PER\_MILE=0.2;

final int doublecDAILY\_MILES\_ALLOWED=150;

(calcu'clateAverageDailyMiles() - DAILY\_MILES\_ALLOWED)\*

numDays\*EXTRA\_FEE\_PER\_MILE

toString

returnVehicle(int rentalNum)

int endMiles;

int requiredNumber=rentalNum, position =0;

boolean found =false;

while(!found)&&(position< arrayList rentals){

if (num !=rentalNum){

position++

}

else{

found=true

}

}

If(found){

endMiles = keyboard

Rentals.get(position).setEndMiles(endmiles)

totalRev += Rentals.get(position).calculateTotalRentalCost();

}

else{

rental not found

}

Appendix5 - Program Prep Sheet (HGVRental.java)

Program Preparation Sheet

**Name**: Patrick Donnelly **Registration No**: B00823962

**Date**: 21/10/20

**Class Name and Description**: HGVRental.java

A subclass to save attributes for an MPV rental order in order to create, display and MPV rental orders. Inherits from the VehicleRental superclass.

**Program Design Data Requirements**

Declare countHGV

int countHGVRented=0;

Declare height and load capacity variables

double height, loadCapacity

Default Constructor method

HGVRental(){

countHGVRented++

Alternative Constructor method

countHGVRented++

HGVRental(String type)

Super(type)

Alternative Constructor Method

height=HGVheight;

loadCapacity=HGVloadCapacity;

countHGVRented++;++

HGVRental(String HGVType,String HGVMake,

String HGVReg,int HGVMilesBefore,

int lengthDays,double dailyFee,double

HGVheight,double HGVloadCapacity ){

super(HGVType,HGVMake,

HGVReg,HGVMilesBefore,lengthDays,dailyFee)

Alternative constructor Method

height=0;

loadCapacity=0;

countHGVRented++;

HGVRental(String HGVType, String HGVMake,

String HGVReg,int HGVMilesBefore,

int lengthDays, double dailyFee){

super(HGVType, HGVMake, HGVReg,

HGVMilesBefore,lengthDays, dailyFee);

Get/Set methods for:

loadCapacity/height

toString method

if(height =0)

“height not declared”

else “height is “ height

if(loadCapacity =0)

“load Capacity not declared”

else “load capacity is “ load capacity

getNumberHGVRentals()

return countHGVRented

getVehicleType()

return ("HGV")

createHGVOrder(int num)

for loop (num = number of times)

prompt user for make,reg, miles before, days, daily fee

height, load capacity.

HGVRental thisHGV=new HGVRental(type,make,reg,milesBefore,days,fee,height,loadCapacity);

Append thisHGV to Rentals arraylist (VehicleRental class)

Append thisHGV(ID) to RentalsID arraylist (VehicleRental class)

Save to file

FileManager.write(thisHGV);

Appendix6- Program Prep Sheet (MainMenu.java)

Program Preparation Sheet

**Name**: Patrick Donnelly **Registration No**: B00823962

**Date**: 11/11/20

**Class Name and Description**: MainMenu.java

Displays a menu to the user and prompts user selection to perform various functions

**Program Design Data Requirements**

Declare local variables

int choice

Output – 10 options to user

choice =kb.nextInt

mainMenu(){

Switch statement;

Case1

Create order

Nested switch

3 options

case 1 – makeCarOrder(1)

case2 – makeBusOrder(1)

case3- createHGVOrder(1)

case2

multi-vehicle (fleet) rental order

Nested switch

3 options

case 1 –

“how many”

numCars =kb.nextInt

makeCarOrder(numCars)

case2 –

“how many”

numBus =kb.nextInt

makeBusOrder(numBus)

case3-

“how many”

numHGV =kb.nextInt

makeHGVOrder(numHGV)

int numCars, numBus, numHGV

case 3

Search Vehicles

Nested switch

2 options

case 1 –

searchByType

case2 –

SearchAll

case4

Return a Vehicle

“Enter vehicle ID”

Id=kb.nextInt

Call method returnVehilce(id)

int id

case5

update discount list

autoUpdateDiscountList()

case6

Number of Vehicle Rentals

Nested switch

4 options

case 1 –

car

getNumberCarRentals()

case2 –

MPV

getNumberCarRentals

case3 –

HGV

countHGVRented

case4 –

Total rentals

countVehiclesRented

case7

Summary

Call SummaryTable.displaySummary()

case8

Revenue

VehicleRental.totalRevenue

case9

Receipts

FileManager.getAllReceipts()

case0

Quit

break;

mainMenu

end switch

end mainMenu(){

end class

Appendix7- Program Prep Sheet (VehicleRentalManager.java)

Program Preparation Sheet

**Name**: Patrick Donnelly **Registration No**: B00823962

**Date**: 18/11/20

**Class Name and Description**: VehicleRentalManager.java

Contains the main() method

A program which is used to initiate the program via the main Menu

**Program Design Data Requirements**

public static void main() {

MainMenu.mainMenu();

}//main

Program Preparation Sheet

**Name**: Alastair Dore **Registration No**: Basdfaasdfasdf

**Date**: 21/10/20

**Class Name and Description**: MPVRental.java

A subclass to save attributes for an MPV rental order in order to create, display and MPV rental orders. Inherits from the VehicleRental superclass.

**Program Design Data Requirements**

Declare countBusesRented

protected static int countBusesRented=0;

Declare number of seats

private int seats;

Default Constructor method

public MPVRental(){

super();

seats = 2;

countBusesRented++;

}

alternative constructor1

public MPVRental(String MPVtype, String busMake,String busReg,int busMilesBefore,int lengthDays,double dailyFee, int busSeats){

super(MPVtype, busMake,busReg,busMilesBefore,lengthDays,dailyFee);

seats=busSeats;

countBusesRented++;

}

alternative constructor2 is not used but it is here incase it is needed in future developments

public MPVRental(String MPVtype, String busReg,int busMilesBefore,int lengthDays,int busSeats){

super(busReg,busMilesBefore,lengthDays);

seats=busSeats;

countBusesRented++;

}

Get/Set methods for:

number of MPV seats, number of bus rentals, vehicle type

toString method to display details of the order, and save and write to a receipt file

Inherited from VehicleRental superclass

public String toString(){

return super.toString()+"\n"+

"No. of seats in bus: "+seats+"\n";

}

makeBusOrder method to create an MPV rental order by getting input from the user. The details of the new order are then added to the array lists of vehicle rentals and rental IDs, and the receipts are saved to a file, and displayed to the customer. There is a pause of 3 seconds to allow the user to see the receipt before returning to the main menu.

A testBusOrder method was created to speed up the testing process. An MPV rental order object can be generated automatically with default values without asking for user input.

Program Preparation Sheet

**Name**: Alastair Dore **Registration No**: Basdfaasdfasdf

**Date**: 21/10/20

**Class Name and Description**: Pause.java

Contains a simple method to allow the program to pause for a specified number of seconds. This utilizes the programming technique of threading which allows a program to create multiple threads and do more than one thing at a time. The Thread.sleep method needs to be within a try catch clause.

**Program Design Data Requirements**

int secs;

public static void pause(int secs)

{

try

{

Thread.sleep(secs\*1000);

} catch (InterruptedException e)

{

e.printStackTrace();

}

**Name**: Alastair Dore **Registration No**: Basdfaasdfasdf

**Date**: 21/10/20

**Class Name and Description**: ErrorHandler.java

Contains two methods, one to receive an integer input from the user, and one to receive a Double (decimal) input.

These were added to stop the program crashing if a user entered text when asked to enter a number.

The request for user input is placed within a try catch clause, and this is placed within a do loop in order to repeat until the user enters a number value.

The try catch error handling methods were abstracted to a separate class so that the members of our team were able to handle user input errors without having to add try catch clauses and repeating code.

**Program Design Data Requirements**

getIntegerInput()

int thisInt;

do loop

try clause

receive input from user

catch any invalid input and display "Please enter a valid number"

continue do loop until valid an integer is entered

if try catch clause is successful, return thisInt

getDoubleInput()

int thisDouble;

do loop

try clause

receive input from user

catch any invalid input and display "Please enter a valid decimal number"

continue do loop until valid an double is entered

if try catch clause is successful, return thisDouble

**Name**: Alastair Dore **Registration No**: Basdfaasdfasdf

**Date**: 21/10/20

**Class Name and Description**: FileManager.java

The methods are static as they are called a number of times from different classes.

**Program Design Data Requirements**

String filename which will store the filename to be written or read

String FILE\_PATH which stores the name of the directory to hold the order receipts

private static void create()

creates a file named filename. Placed within a try catch clause incase it is unsuccessful.

public static void createFolder()

checks if the FILE\_PATH exists, and if not, it creates a directory

public static void getAllReceipts()

Checks if any orders are in the FILE\_PATH directory.

Gets a list of all order receipts within the directory storing them in an array named orders[], and uses a for an enhanced for loop, also known as a for-each loop to display each one. I could also have used a normal for loop, but I wanted to learn to use a for-each loop . The filenames are displayed with an index number.

Use a try clause to get the index number from the user of the order receipt file that they want to see. The try clause was used to catch errors if a number was entered out of bounds of the length of the array of orders[]. Ie., if the user tried to enter an index that was too high. This was placed within a do loop to repeat until a valid index number was entered.

The method read() is then called to open and and display the file.

public static void write(VehicleRental vehicleRental)

vehicle rental object

String rentalType

String dateID

int rentalID

String FILE\_PATH

String fileName

The createFolder() method is called to create a folder if one doesn’t exist.

fileName is generated from FILE\_PATH and vehicleRental object attributes.

public static String read()

Open the specified file and read each line to a string variable named text.

The contents of the file are returned to the method which called it, eg., getAllReceipts.

**Name**: Alastair Dore, Aldrian Dungca, Patrick Donelly

**Date**: 21/10/20

**Class Name and Description**: SummaryTable.java

A 2D array is used.

Column headers are added to the top of each column.

Loop through each row of the table, entering attributes from the VehicleOrder objects saved in the Rentals array.

Display the table using a nested for loop