Assessment Specification

Java Software Assignment

1. Table of Contents

2. Software Assignment

The owners of Petrol Never Unlimited Ltd are own a petrol station on the M25 and you have been tasked with the design, implement, test and document an automated demo of a Petrol Station Management app.

The implementation needs to be done in **Java 8** using **either** the Console or a Swing GUI as an output environment.

1) App Synopsis

The festive season is around the corner and many families will be travelling to their holiday destinations. This period is a fantastic opportunity for Petrol Never Unlimited Ltd to make up for the low profits during last December – where travellers were stranded on the M25 due to heavy snow. They therefore must ensure that the petrol station is working at full capacity and as efficiently as possible in order to maximise profits. The issue of efficient fuelling is paramount as drivers are normally very agitated during the summer holiday as they drive long distances with children in the car, which can be somewhat stressful to the parents. Last year, long delays in fuelling at our station resulted in many drivers turning around and fuelling at 'Petrol Nearly Unlimited Ltd', which is only 5 miles down the motorway. The management at Petrol Never Unlimited Ltd fully recognise that a fuel attendant is pivotal to the success of this operation, which is why they have decided to incentivise them by adding a commission of 1% (0.01) of the total fuel money they sell to their already very generous hourly rate of £2.49. This works out as 8 hours shift * £2.49 + 1% of the total fuel they sell during that shift.

2) App Description

The automated demo app refers to a forecourt with *9* fuel pumps arranged over *3* lanes (three pumps in each lane). Each *n* number of seconds (where *n* is random) a vehicle is created and awaits to be fuelled.

The fuelling starts when a vehicle waiting in the queue is automatically assigned to an available pump.

To add to realism of the demonstration and to account for drivers' agitation, there is a finite period of time to fuel a waiting vehicle before it leaves the forecourt. That

period is set to *Z* number of seconds (where *Z* is random). Should the vehicle not be fuelled during that period of time, it will leave the forecourt. Similarly, a vehicle that was fuelled also leaves the forecourt, only this time the amount of fuel dispensed is recorded.

A pump cannot service more than one vehicle at a time and the following **counters** have to be kept: (**1**) running total of the number of litres dispensed during the app's lifetime; (**2**) The amount of money these litres equate to; (**3**) The 1% commission; (**4**) the number of vehicles serviced, (**5**) the number of vehicles that left before they were fuelled and (**6**) keep a detailed list of each fuelling transaction – [Vehicle type, Number of litres and Pump number].

IMPORTANT NOTE: You are free to choose a sensible cost per litre as well as change the *n* and *Z* numbers as you see fit for developing this app.

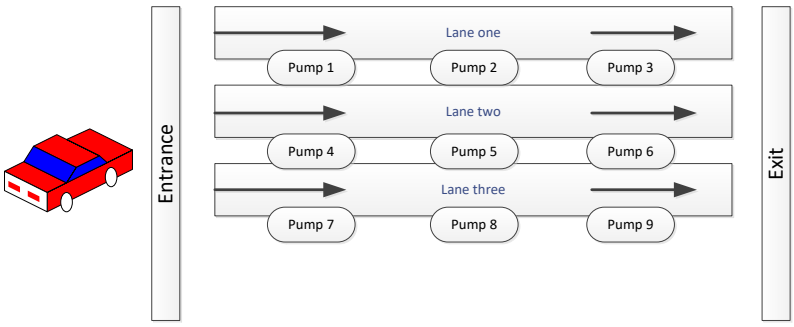

Figure 1: Forecourt Layout (Design only, not the final app UI)

3) Marking – Please read this section in detail

The assignment will be marked based on the following 5 criteria: (1) Functionality and Execution, (2) Program architecture, (3) Presentation and submission guidelines, (4) App User guide and (5) Testing.

- **All 5 criteria are important! Please do not neglect any of them.**

Time management is crucial – please make sure you leave enough time to implement and write all **five** criteria.

1. ***Functionality and execution*** (**40%**) – Bottom up approach! Start by implementing the set of functionality [**Basic**/**Intermediate**/**High Level/Very High Level**] which you are most likely to achieve within the

time available to you. When all functions have been fully implemented, attempt to implement the next set of functions. Again, only once the previous set was accomplished in FULL.

- *I recommend creating a new project for each level you have attempted so that you don't end up losing code you might want to reuse later on.*

2. ***Program architecture and authorship*** (**40%**) – Use the good programming guidelines and design features (pseudocode, activity diagrams, etc.) throughout the development cycle of the app. Adhering to guidelines retrospectively is time consuming and often leads to bugs.
3. ***Submission and presentation*** (**3%**) – See details below and **please** adhere to the presentation and submission instructions.
4. ***App User guide*** (**7%**) – See details below.
5. ***Testing*** (**10%**) – The mark for this section takes into account bugs that are **not** documented in your test plan. Please **test** your program, and **test it well -** reporting all the results!
6. ***Total Marks - 100%***

4) Program Functionality and Execution

Before deciding which level of functionality to implement, please read carefully what is required in each level. You need to assess whether or not you are able to implement the level of functionality you chose within the time available to you. It is better to scale up the implementation should you have time. Please make sure that you leave enough time for carrying out the testing phase as well as writing the user guide. **Also, please feel free to change the numbers of seconds and litres per second to any value that you wish for developing this app, as the functionality should remain the same regardless of what these values are.**

Basic functionality: (for a maximum potential mark of 40% to 50%)

A. A new vehicle is created every *1.5* seconds. (No need to randomise this timer at this level)

B. When a vehicle is sent to a pump, the fuelling process will take *8* seconds. At the end of the *8* seconds the number of litres dispensed is recorded and the vehicle leaves the forecourt, freeing the pump for use.
C. The Pump is capable of dispensing *1.5* litres / second.
D. At this level, a newly created vehicle can wait until it is sent to a pump without a time limit.
E. Counters 1 to 4 and 6 have to be implemented (See App Description for detail, look for **counters** keyword).
F. At this level, there is only one type of fuel (Unleaded) and one type of vehicle that ever gets serviced by the petrol station.

Intermediate functionality: (for a maximum potential mark of 50% to 60%)

A. Same as above. **The below several points will override any information given at the previous level, where relevant.**
B. A new vehicle is created randomly every *1500* to *2200* milliseconds. And there can only be one car waiting to be serviced.
C. When a vehicle is sent to a pump, the fuelling time will be random and take between *7000* and *9000* milliseconds. At the end, the number of litres dispensed is recorded and the vehicle leaves the pump.
D. Newly created vehicles can only be fuelled within *1500* milliseconds of their creation. Failure to service them within this time frame will remove the vehicle from the forecourt.
E. There are 3 types of fuels (Unleaded, Diesel and LPG – Classes, ideally) and 3 types of vehicles (Car, Van and HGV – Classes, ideally). Both the type of vehicle and fuel of newly created vehicles will be selected randomly. At this level HGVs and Vans can run on any fuel type.
F. All pumps contain all types of fuel.
G. All counters have to be shown, but with the addition of two extra counters totalling the number of litres dispensed for the other two fuel types (Diesel and LPG).

High level functionality: (for a maximum potential mark of 60% to 70%)

A. Same as above. **The below several points will override any information given at the previous level, where relevant.**
B. A new vehicle is created randomly every *1500* to *2200* milliseconds. And the queue of vehicles waiting to be serviced can reach *5*. At this level, the time frame to service a vehicle is random between *1000* and *2000* milliseconds
C. Newly created vehicles will be created with a random amount of fuel already in their tank (which cannot be greater than a quarter of their total tank capacity). Fuelling time will be based on the above amount.
D. The random amount of fuel in a newly created vehicle will be proportionate to the vehicle size. Each vehicle type will have the following maximum amount in their tank: Cars - maximum *40* litres, Vans - maximum *80* litres and HGV - maximum *150* litres. HGV can only run on Diesel. Vans on both Diesel and LPG and finally cars on all three types of fuel.
E. A queuing system has to be implemented in the forecourt as each lane has three pumps. As per figure 1, a vehicle which uses pump 1 blocks the way for new vehicles to get to pumps 2 or 3. The situation is similar for lanes 2 and 3.

Very High level functionality: (for a potential mark of 70% to 100%)

For a mark of 70% **PLUS** (up to the maximum of 100%) you will need to exhibit exceptional code elegance, noticeable additional functionality (e.g. Windows Forms, highly dynamic and featureful Console UI, etc.) that enhances the app user experience as well as the realistic 'feel' of the demo, together with exemplary User guide and testing.

Students are encouraged to research and utilise built in methods in Java 8 to aid with the Random and Timer aspects of this assignment (e.g. Random and Timer classes). This, however, is not necessary for all levels of implementation. Please note that you must acknowledge and reference all sources of information and code examples you use. You may also **discuss** your ideas with your colleagues, however

please bear in mind that **any code or report you write should be 100% your own work, written individually** and **not in collaboration** with said colleagues.

ALL levels of functionalities have to implement either a **Console** or **Swing** based Graphical User Interface (GUI) which displays the forecourt, denoting (in any way you want) newly created cars, free and busy pumps, counters, etc. The idea is to make the screen simple, clear and easy for the user to use (using simple characters - **See example below).** If implementing a Console based GUI, then please **look at clearing the Console window and redrawing the UI every time a change happens to give the impression of a dynamic app**.

```
                ********1 BUSY*************2 AVAIL***********3 Avail********
CAR→            ************************************************************
Counter A       ********4 BUSY*************5 AVAIL***********6 BUSY********
Counter B       ************************************************************
Counter C       ********7 BUSY*************8 AVAIL***********9 BUSY********
```

Console GUI Pump layout example

5) Program Architecture and Authorship

The program architecture must be based on Object Oriented Design Methodology.

- There must be a minimum of 5 classes, complete with their member fields, Constructors and Methods. You may use static Classes, which should only contain static attributes and methods, to hold functionality and data relevant for the entire app, not just a particular Class like Vehicle, Pump, etc.
- Where possible, fields are required to be private, only accessible via accessors (**getters** and **setters**). Please look into Encapsulation.
- The structure needs to be easily maintainable and extendable to accommodate more pumps/lanes/petrol types/ etc.
- Clearly separate User Interface functionality objects (e.g. draw) from the data manipulation objects (e.g. add/subtract fuel amounts, calculate costs etc.).
- Name classes, variables, methods etc. with meaningful, clear names using consistent, appropriate capitalisation.
- Be well documented with // comments within and around methods, classes and other definitions.
- Have a consistent, appropriate layout including good use of indentation, white-space and individual files for each class.

6) Submission & Presentation

The work must be submitted according to the following instructions. This ensures reduced risk for you and maximum convenience for the marking staff.

1. ***Create a ZIP archive file*** containing:
    a. The complete project folder of the source code for the program
    b. The documentation in electronic form as a Word document (.doc, .docx) or Adobe (.pdf) file. Excel workbooks (.xls, .xlsx) are acceptable for the testing section.
    c. Clearly Mark which functionality level you have implemented [**Basic** | **Intermediate** | **High** | **V.High**].
2. ***Clearly label the ZIP archive file*** with your Student and module information
3. ***Keep a copy of your work*** in a safe place just in case something happens to it after hand-in.

7) App User Guide

This should be a clearly written, structured and presented document describing how the app works and what can be visually seen in the demo. It should contain an exciting app synopsis that would entice the reader to buy the full app along with a simple set of instructions of how to run the demo app. The file format should be either PDF or DOCX.

8) Testing

Full documentation of the tests carried out throughout the development of the software.

A proposed test plan and log layouts can be seen below. You need not copy these exactly, but whatever plan/log you choose to use, they must be easily read, accurate, and thorough i.e. all operations (if/else/method/class/object etc.) in your code are expected to have been exhaustively tested and recorded with enough detail to repeat the tests.

Table 1 - Test Plan (plan tests)

Test #	Test Purpose	Expected outcome
1.	Ensure the cars are correctly being created	Cars should appear in the queue every X number of seconds
…	…	…

Table 2 - Test Log (record test results)

Test #	Input Data	Actual outcome	Comments
1.	N/A	Cars are being added to queue with correct data	All OK
…	…	…	…

You may choose to compile this documentation using an Excel spreadsheet if you prefer.

9) Other Notes

- The intended audience for your documentation is to be considered that of your fellow students. However, please avoid unintended plagiarism by sharing the actual assignment output!
- You should include references to any sources you draw upon.
- A program that does not perform all the requirements will not necessarily be considered a failure – it can still earn marks for what it can do and how it does it. Clearly document any shortcomings