Aston University
Birmingham B4 7ET
http://vle.aston.ac.uk/

Coursework Specification 2017

Antonio García-Domínguez

Module CS1410 February 23, 2017

1 Introduction

This document contains a specification of the software and other documentation that form the assessed coursework assignment for the module CS1410: Java Program Development.

Note that there are two possible levels of submission which will be assessed at different levels of credit. The full (level 1) submission will be marked out of 100 while the lesser (level 2) submission will be marked out of 75. The level 2 submission is essentially a simpler subset of the level 1 submission.

The project is to be carried out in small groups; you will be told which group you are in a separate document. Each group has been set up on Blackboard to have its own mailing list and wiki: you are strongly encouraged to use these. It is also helpful to have a way of sharing files: Dropbox[1] provides a free way of doing this that is adequate for the size of the project. The individual marks will be assigned as follows:

1. The group submission will be allocated a mark using the assessment criteria described later in this document. Thus it is in everyone's interests to make the system and its documentation as good as possible (just as in real life).
2. There will be differential marking within groups which will be *guided* by the following considerations:
   (a) each member's assessment of their contribution to the solution;
   (b) each member's performance at the oral examination described in Section 4;
   (c) individual attendance at meetings.

Note that if you do *not* attend your group's meetings regularly I may remove you from that group and you will have to complete the coursework by yourself.

If you want clarification on any aspect of the coursework (be it the problem description or what you have to submit), please ask it as a question on the Blackboard discussion board. I will read this regularly and respond. In this way, everyone has access to the same information. I am also available for office hours (bookable through WASS: see contact information on Blackboard) as is the Programming Support Officer.

[1] http://www.dropbox.com

2 Problem Description

The task is to develop a simulation of a self-service petrol station with a certain number of pumps and a shop. Different types of vehicles may come to the station to top up, (optionally) pick up some food and necessities and then pay for everything at one of the tills. The purpose of the simulation is to study what level of demand can be handled with a certain number of pumps and tills, keeping customers happy and net income high. All value ranges mentioned are uniformly randomly distributed.

You will have various types of vehicles:

- **Small cars** have 7–9 gallon tanks, and take up 1 unit of space in the queue.
- **Motorbikes** take up 0.75 units of space in the queue, and their gas tanks are smaller (5 gallons).
- **Family sedans** take up 1.5 units of space in the queue, and their gas tanks are larger (12–18 gallons).

Note that the probabilities of the purchases, the sizes of the gas tanks and their behaviour are subject to change in the future. You should make your classes flexible enough to adapt to this without major alterations to the software.

Your simulation should model the petrol station with a time resolution of 10 seconds (1 "tick"). The system has the following configuration:

1. There is a configurable number of pumps. Each pump has a queue that can fit up to 3 units of space (e.g. 2 sedans, 3 small cars or 4 motorbikes). Pumps provide 1 gallon per tick of fuel.
2. There is a configurable number of tills in the shop. Paying at the till takes 2–3 minutes.
3. Small cars and motorbikes arrive with a probability of p per tick.
4. Family sedans arrive with a probability of q per tick.
5. Customers always go to one of the least occupied queues first (both for pumps and for tills).
6. If a vehicle arrives and does not fit into any of the queues, the vehicle will simply leave.
7. Vehicles always fully top up their tanks. By default, one gallon is £1.20 for the simulation.
8. Vehicles stay in the queue for the pump while the driver goes to the shop to pay at the till.
9. The vehicle starts topping up on the next tick after it gets to the front of the queue for the pump, and the driver goes to the till on the next tick after it tops up.
10. You must track how much money was earned, and how much money was lost because of vehicles being unable to enter the station.
11. You must write test classes in JUnit 4 for at least **five** of your classes.

The level 1 system has the following additional requirements:

1. The gas station now includes a shopping area. Happy customers that refill quickly will spend some time looking around, before going to the till on the next tick and paying some additional money:

- For small cars, if the refill is done in less than 5 minutes since arriving, there is a probability of 0.3 that the driver will shop for 2–4 minutes to spend an extra £5–£10.
- For family sedans, if the refill is done in less than 10 minutes since arriving, there is a probability of 0.4 the driver will shop for 2–5 minutes to spend another £8–£16.
- Motorbike drivers in the area are thrifty and will never go to the shopping area.

You will need to track the money lost from missed sales as well. It may be interesting to track it separately from the money gained from selling fuel.

2. Trucks may also arrive at the station. They take up 2 units of space. After refilling their 30–40 gallon tank, a truck driver that refilled in 8 minutes or less will always do a purchase of £15-£20 after browsing for 4–6 minutes.

   An unhappy truck driver will let the other truck drivers know about the bad service. This will make it less likely that they come. Trucks arrive with a probability of t, which is initially $t_0 = 0.02$. An unhappy truck driver will reduce t by 20% of its current value: $t' = 0.8t$. A happy truck driver will increase it by 5% of its current value, up to the original value of t: $t' = \min\{1.05t, t_0\}$.

3. A GUI written using the Swing toolkit should be provided and allow the user to set values such as p and q, the price of the gallon, and the period of time that the simulation should run for.

   Note: a level 2 submission does not need to provide such a GUI, but still needs a clean interface to the main model.

The aim of running the simulation is to decide for each station configuration (number of pumps and tills) which level of activity it is best suited for: that is, which are the values of p and q that report the highest net income (raw income minus missed sales).

To do this, the simulation should be run for four hours (1440 ticks) for all independent combinations of the values below, and the results should be averaged over 10 different seeds for the random number generator:

- p: 0.01, 0.02, 0.03, 0.04, and 0.05.
- q: same options as p.
- Pumps: 1, 2 and 4.
- Tills: 1, 2 and 4.
- (**Level 1 only**) With and without trucks (see paragraph below).

For a level 1 submission, you should run separate studies with and without trucks. The owners of the gas station are currently wondering if they should allow trucks to refill at the station or not, so you should compare for each station configuration if it is better to allow trucks or not.

3 Design Notes

The following information and ideas may be useful in developing the simulation system. You will also find some of the ideas and structures from the lab classes helpful.

1. Think generically! You should aim to write a small library of classes that can be used to build the scenario described above, but that would also support similar scenarios (more types of vehicles, smarter queueing) without change to the library classes. This is one of the assessment criteria for the design.

2. When deciding which type of vehicle will arrive, first generate a random number between 0 and 1. If it is less than or equal to p, it will be a small car. If it is between p and $2p$, it will be a motorbike. If it is between $2p$ and $2p + q$, it will be a sedan. Otherwise, no vehicle will arrive. **Note**: trucks would add another range between $2p + q$ and $2p + q + t$.

3. The program should be run with a command-line interface. However, you should develop your simulation model with a clean interface so that a GUI could be added in the future. The GUI would be used to set values such as p and q, the number of pumps, and the period of time that the simulation should be run for.

4. You must develop all the classes yourself, with the exception of the standard library classes (e.g. the random number generator `Random`, Java Collections classes, the Swing library, exception classes, and the usual classes for I/O) and the `LabelledSlider` class. Note that the `LabelledSlider` class has been modified to provide floating point (rather than integer) values as this is more appropriate for defining probabilities.

5. The statistics should be printed to standard output (`System.out`) or written to a file. If they are printed to standard output, in Unix-based systems (GNU/Linux or Mac) you can then use output redirection to put the values into a file. The command

   ```
   java bar > foo
   ```

   overwrites the contents of `foo` each time the program `bar` is run. To append the results of each run you should use the syntax

   ```
   java bar >> foo
   ```

   If you are running the program from Eclipse, you can use the options in the "Common" tab to redirect the output to a file.

4 Deliverables

The submission date for the assignment is **5:00pm on Thursday 4th May**. Late submissions will be treated under the standard rules for Computer Science, with an **absolute** deadline of one week after which submissions will not be marked. This is necessary so that feedback can be given before the start of exams and to spread out coursework deadlines. The lateness penalty will be 10% of the available marks for each working day. There will also be a short oral exam for each group on their submission: more details will be given nearer the time; each group member will be asked questions about their role in the coursework, but there is no need to prepare a presentation.

Submission will be *on-line* through the Blackboard site; your submission should be contained in a single zip file with a name containing the name of the group. It can be submitted by any one of the group members. Your submission should include:

1. A complete set of source files for the project (including the JUnit test classes);

2. HTML documentation for the system generated from the commented code using `javadoc` (I do *not* need printed copy of this: electronic versions are adequate);

3. A written report (see below).

Remind yourself of the Computer Science regulations on collusion and plagiarism. Collaboration is acceptable within certain well-defined limits. If collusion is detected, then the marks of all parties (even the person who supplied the solution) are reduced. If you think that cooperative working has gone beyond what is acceptable, then discuss the matter with me so that we can agree on the action to be taken.

Your report (which must be a PDF file[2]) should contain:

- a brief description of the design (3–4 pages) distinguishing between library components (that could be re-used) and client code (that is specific to the scenario described above). This description should explain the rationale behind design decisions.
- (**Level 1 only**) a brief description of the changes that would be required to your library classes to support simulations with the following properties: more types of vehicles; multiple types of fuel with different prices; parking away from the pump during shopping; vehicles breaking down during the simulation (0.5 page).
- a diagram of the class hierarchy (which may be hand drawn and scanned) using UML notation (1–2 pages);
- (**Level 1 only**) a sequence diagram for *one* of the main scenarios in the simulation (1 page);
- the results of the simulations in tabular form;
- a brief (< 0.5 page) discussion of the results and their implications;
- a complete listing of your source code as PDF files (which can be generated by opening the files in WordPad or NotePad and using the 'Print' command as described above);
- instructions on how to build and run the program.

The report should clearly state whether your submission is Level 1 or Level 2, and which group the submission is from. You may use any compiler or development environment you like to develop the program but the submitted software *must* compile under the standard Oracle JDK version 8.0 using the Eclipse environment.

As well as the group submission, each person in the group should submit their own single page report analysing their contribution to the work:

- a brief description of their contribution;
- a discussion of what they have learned from carrying out the implementation.

[2]You can create a PDF file easily from any other format on the lab machines. Simply open the file in its normal programme (e.g. Microsoft Word) and select the 'Print' command; instead of sending it to your normal printer, select PDF-Creator (PDF-Exchange will probably also work) and it will be output as a PDF file. Alternatively, Word documents can be exported directly as PDF files.

Note that marks will be awarded for the *quality* of the implemented software, including a flexible and extendible design that takes account of the design criteria discussed in Section 2 of the lecture notes, good commenting, robust and defensive programming, and a consistent programming style.

Level 1 The approximate breakdown of marks is 25 for design and its documentation, 70 for the implementation, and 5 for the results and discussion, for a total of 100 marks. This coursework counts for 20% of the total assessment of the module.

Level 2 The approximate breakdown of marks is 20 for design and its documentation, 50 for the implementation, and 5 for the results and discussion, for a total of 75 marks. This level of coursework counts for 20% of the total assessment of the module, but you will only be awarded up to a maximum of 15% because of the reduced level of effort required.

5 Milestones

Experience has shown that the most successful groups are those that work together in a structured way and follow a sensible lifecycle. To encourage this, I propose the following milestones.

Requirements Analysis: you need to analyse the requirements defined in this document to generate an overall system architecture (e.g. using the noun-verb method). You should start this at the tutorial in week 6 (Tuesday 28th February). Write the results up on the group wiki by the end of Friday 3rd March so that it can be reviewed.

Design: you need to create a UML class diagram for your detailed design. You should start this at the tutorial in week 7 (Tuesday 7th March) at the latest and be prepared to discuss your design in the lab on Monday 13th March. The design should be evaluated by running some scenarios. This can be done in the lab.

Unit Tests: you can write some of your unit tests before implementing the body of the code. This can start once the design is agreed, preferably in the lab on Monday 13th March. You can also write all the class skeletons during weeks 8 and 9.

GUI Design: you can sketch your GUI design (assuming you are developing a level 1 submission) as a "wireframe" at the tutorial in week 8 (Tuesday 14th March).

You *must* have a solid and detailed design specification to share amongst the team by the end of week 8 (Friday 17th March) so that the system can be developed during weeks 10 and 11 and the Easter break with some hope of integrating the software successfully on your return. Note that the submission deadline is on Thursday of your week back.

The marking criteria are shown on the following page.

My comments on your code and design will use the following key:

Strengths

1. Good use of encapsulation in the design.
2. Good subsystem decomposition (including model/interface).
3. Well structured GUI code.
4. Clear separation between service and client functions.
5. Good use of inheritance and dynamic binding for run-time polymorphism.
6. Good use of abstract base classes and interfaces.
7. Object creation correctly performed.
8. Avoiding global variables.
9. Appropriate use of `final` and `static` in member functions.
10. Appropriate use of `final` variables in place of numeric constants.
11. Headers with filename, description, author, and version in all files.
12. Clear design documentation.
13. Unit tests that adequately cover potential errors.

Weaknesses

A Design does not show a clear division of responsibilities between classes.

B Design too problem specific.

C Code repetition due to poor design.

D Use of type variables in place of inheritance and function overriding.

E Poorly designed class interface, making class easy to misuse.

F Inadequate commenting.

G Poor separation of model and interface.

H Poor use of Swing components in GUI.

I Poor use of generics.