Aston University

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

Coursework Specification 2017

Antonio García-Domínguez

Module CS1410 February 23, 2017

Things to consider // Requirements // specific numbers/algorithms // optional/Level1 // Expansion // Track

**simulation of a self-service petrol station**

certain number of pumps and a shop.

Different types of vehicles may come to the station to top up,

(optionally) pick up some [things] 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.

The level 1 system has the following additional requirements:

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:

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.

All value ranges mentioned are uniformly randomly distributed.

**You will have various types of vehicles:**

Small cars : 7–9 gallon tanks, : up 1 units of space in the queue. if (refill, done in less than 5 minutes since arriving), … probability of 0.3, will shop for 2–4 minutes, spend extra £5–£10. arrive probability of *p* per tick.

Motorbikes : 5 gallon tank, : 0.75 units of space in the queue. will never shop. arrive probability of *p* per tick.

Family sedans : 12–18 gallon tanks, : 1.5 units of space in the queue. if (refill, done in less than 10 minutes since arriving), … probability of 0.4, will shop for

2–5 minutes, spend extra £8–£16. Arrive probability of *q* per tick.

Trucks have 30–40 gallon tanks, take up 2 units of space, refilling in 8 minutes or less, driver will always shop, for 4-6 minutes, purchase of £15-£20.

Unhappy truck driver will let other truck drivers know about bad service & make it less likely that they come. Trucks arrive with a probability is initially *t*0 =0*.*02.

Unhappy truck driver will reduce *t* by 20% of its current value: *t*0 =0*.*8*t*. A happy truck driver will increase it by 5% of its current value, up to the original value of *t*: *t*0 = min{1*.*05*t,t*0}.

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.

10 seconds (1 “tick”).

**There is a configurable number of pumps**

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.

There is a configurable number of tills in the shop.

Paying at the till takes 2–3 minutes.

Customers always go to one of the least occupied queues first (both for pumps and for tills).

If a vehicle arrives and does not fit into any of the (pump) queues, the vehicle will simply leave.

You must track how much money was earned, and how much money was lost because of vehicles being unable to enter the station.

Vehicles always fully top up their tanks.

By default, one gallon is £1.20 for the simulation.

Vehicles stay in the queue for the pump while the driver goes to the shop to pay at the till.

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.

You must write test classes in JUnit 4 for at least five of your classes.

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.

**Simulations**

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.

**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.

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.

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 2*p*, it will be a motorbike. If it is between 2*p* and 2*p* + *q*, it will be a sedan. Otherwise, no vehicle will arrive. Note: trucks would add another range between 2*p* + *q* and 2*p* + *q* + *t*.

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.

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.

The statistics should be printed to standard output (System.out) or written to a file.

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

**Strengths**

Good use of encapsulation in the design.

Good subsystem decomposition (including model/interface).

Well structured GUI code.

Clear separation between service and client functions.

Good use of inheritance and dynamic binding for run-time polymorphism.

Good use of abstract base classes and interfaces.

Object creation correctly performed.

Avoiding global variables.

Appropriate use of final and static in member functions.

Appropriate use of final variables in place of numeric constants.

Headers with filename, description, author, and version in all files.

Clear design documentation.

Unit tests that adequately cover potential errors.

**Weaknesses**

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

Design too problem specific.

Code repetition due to poor design.

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

Poorly designed class interface, making class easy to misuse.

Inadequate commenting.

Poor separation of model and interface.

Poor use of Swing components in GUI.

Poor use of generics.