Curtin University — Department of Computing

Object Oriented Software Engineering (COMP2003)

2016, semester 2

Assignment

Due: Wednesday 26 October, 5:00 pm.

Weight: 20% of the unit mark.

This assignment assesses your ability to take complex requirements and come up with a *good* design. You must produce working code, but this is not *sufficient* to get any marks. You will get marks based on the quality of your design, and by how well you demonstrate your depth of understanding of the design options.

1 Problem Overview

You have been asked to design and implement a training system for company directors. In the best tradition of economics, we shall start by making things up.

The training system is intended to show people what would happen when they make certain buy/sell decisions. For our simplified purposes, a company owns a set of property. Each item of property is worth money, and so can be bought and sold. Over time, that property also generates an income for its owner (although in some situations that income might be negative; i.e. a cost to the company).

A company director (in training) can propose a *plan*. A plan consists of several proposed decisions, each one occurring at a particular point in time (in the future), and involving either buying or selling property.

The training system will simulate the consequences of a given plan over a number of years. For simplicity, all events in the simulation will take place at the start of a particular year, and nothing changes *during* each year. The system will work from a set of input files that contain information about all property.

2 Property

All forms of property have:

- A monetary value (in $) – what the property is worth; and
- A profit (also in $) – the overall amount of money the owner gets in a particular year (which could be negative).

A company can own the following types of property:

Bank account. Each company always owns exactly one of these, and it cannot be bought or sold. Rather, the balance increases or decreases when other property is bought or sold, and when a profit or loss is made (see "companies" below for details). The company can go into debt, when the balance is negative. There are no upper or lower limits on the bank account balance.

Profit calculation: if the balance is positive, the company receives an amount of "interest" equal to 5% of the balance. For instance, if the account balance is $1,000,000 during 2017, then the company receives $50,000 in interest at the start of 2018. If the account balance is negative, the same interest calculation is applied, but the the result will be negative (that is, a loss rather than a profit).

Business units. These are factories, mines, call centres, etc. that employ people to produce products and services. Business units can be bought or sold, and so they have an overall monetary value.

Profit calculation: a business unit generates revenue from whatever it sells, but must also pay wages to its employees. At the start of the simulation, each business unit has a particular yearly revenue, and a particular yearly wage payment (representing the combined yearly wages of all its employees). The yearly revenue can change, from year-to-year and from unit to unit. Wages can also change, but when they do, they change for the whole simulation.

Companies. Companies can buy and sell other companies. However, there cannot be a cycle in ownership relationships. That is, a company cannot own itself, nor can two companies own each other, nor can n companies form an ownership circle.

Profit calculation: Say s is the sum of the profit of whatever the company owns. If $s \leq 0$, then the company's profit is 0, and its bank balance is reduced by s. Otherwise, the company's profit is $0.5s$, and its bank balance also increases by $0.5s$.

Every item of property has exactly one owner, which can either be a company within the simulation, or an "unnamed owner". The simulation must of course keep track of who owns what, although unnamed owners have no information to keep track of.

3 System Behaviour

The training system should take five command-line parameters: three filenames (for input files described in Section 4), a start year, and an end year. The simulation must run from the start year to the end year, inclusive, simulating a series of events and their consequences. There should be no user interaction!

While running, the simulation contains one or more companies, and zero or more business units. One of the companies is the *primary company* – the one that will be doing the buying and/or selling.

At the very start of the simulated time period, the bank account balance for all companies begins at $0. All property-specific values are specified in one of the input files.

At the start of each year, the following things must happen, in this order:

1. Companies' bank accounts are updated to reflect the profit or loss they have made over the previous year. However, this only happens from the second year onwards (since nothing happens prior to the first year).

2. Companies' names and bank account balances should be outputted in an easy-to-read format, along with the current year.

3. A set of zero or more *unplanned events* take place that affect income and monetary value. Unplanned events can happen in any combination. Several events of the same type can also happen at once, in which case their effects are cumulative. The events are *not* random, though – they are specified in an input file.

   The types of possible unplanned events are as follows:

   - Wages may increase or decrease by 5% for all business units.
   - The revenue for *a specific* business unit can increase or decrease by 5%.
   - The monetary value of *a specific* business unit or company can increase or decrease by 5%.

   These are all multiplicative changes. For instance, an increase in wages means that the simulation must multiply each business units' wages by 1.05. For a decrease, the multiplier is 0.95. (FYI, if an increase and decrease were to happen in the same year, they would not exactly cancel each other out, because $1.05 \times 0.95 \neq 1$.)

4. A set of zero or more buy/sell decisions are made, according to the plan proposed by the director-in-training. These are likewise specified in an input file.

   The primary company can buy any business units that it does not already own. It can also buy any companies it does not already own, *provided* this would not create an ownership cycle (i.e. it cannot buy itself, or its owner, or its owner's owner, etc.). When buying something, the primary company's bank account will decrease by the value of the property bought. If the previous owner is a company within the simulation, that company's bank account increases by the same amount.

   Note: a buy decision is *still valid* even if the company does not have enough money. In that case, its bank account balance simply becomes negative, implying that it has gone into debt.

   The primary company can sell anything it does own, *except* for its bank account. When selling something, for simplicity, we always sell it to an "unnamed buyer" (who becomes the unnamed owner). The bank account of the primary company will increase by the current value of the property it sells.

4 Input Files

The training system will take its input from three different CSV (Comma Separated Value) files:

- The *property file* describes every business unit and company in the simulation;
- The *event file* contains a list of unplanned events;
- The *plan file* contains a list of buy and sell decisions.

The system should detect errors in the training files and report them appropriately.

4.1 The Property File

The property file contains one row for each item of property (plus one heading row), with six columns:

- The unique name of the item of property;
- The property type: "C" for company, or "B" for business unit;
- The name of the initial owner of the property, or empty if the owner is unnamed;
- The monetary value (in $) of the property;
- The revenue (in $) of the business unit, or empty if the property is a company;
- The wages (in $) of the business unit, or empty if the property is a company.

The primary company is the first company listed in the file (but not necessarily the first item of property).

For example:

```
Name,Type,Owner,Worth,Revenue,Wages
AmazingCorp,C,,1000000,,
Cannington factory,B,AmazingCorp,5000000,100000,50000
Tumbleweed Co,C,AmazingCorp,500000,,
Giblet Inc,C,,7500000,,
Kalgoorlie mine,B,Giblet Inc,12000000,800000,300000
32 Bakers Street,B,AmazingCorp,500000,10000,8000
99 Dread Pirate Lane,B,,400000,45000,55000
```

4.2 The Events File

The events file contains one row for each unplanned event (plus one heading row), with three columns:

- The year when the event happens;
- The event type: "W+" for wages increase (across all business units), "R+" for business unit revenue increase (for a specific business unit), or "V+" for monetary value increase (for a specific business unit or company), or "W-", "R-" or "V-" for corresponding decreases;

- The name of the affected property (if the event type starts with "R" or "V"), or empty if the event does not concern a specific item of property ("W").

In a valid file, the events will be listed in chronological order. For instance:

```
Year,Event,Property
2017,R-,32 Bakers Street
2017,V+,Tumbleweed Co
2018,W-,
2018,V-,Tumbleweed Co
2019,W+,
2019,W+,
2019,R+,Cannington factory
```

4.3 The Plan File

The plan file contains one transaction per row (plus one heading row), with three columns

- The year when the transaction happens;
- The transaction type: "B" for buy, or "S" for sell.
- The name of the property to be bought or sold.

Like the events file, the transactions in the plan file should be listed in chronological order. For instance:

```
Year,Buy/Sell,Property
2017,S,Cannington factory
2017,B,Kalgoorlie mine
2018,S,32 Bakers Street
2018,B,Tumbleweed Co
2019,S,Giblet Inc
```

5 Your Task

Design and implement this system, providing the following:

(a) Your design, expressed in UML, containing all significant classes, class relationships, and significant methods and fields.

(b) Your complete, well-documented code, in Java, C++ or Python (your choice). Do not use any third-party code without approval, in writing, from the lecturer.

Provide clear instructions for compiling and running your code.

Then, in 1–2 pages total, discuss the following issues:

(c) Where have you used polymorphism, and why? Discuss any design patterns you've used that incorporate polymorphism.

(d) How does your design achieve testability? That is, what have you done to make unit testing easier?

(e) Discuss two plausible alternative design choices, and explain their pros and cons.

(To do well here, show that you understand the range of possible solutions and their tradeoffs. If you simply say "I could have used pattern Y instead of X", you will get zero marks.)

6 Submission

Submit your entire assignment electronically, via Blackboard, before the deadline.

Submit one .zip or .tar.gz file containing:

A declaration of originality – whether scanned or photographed or filled-in electronically, but in any case *complete*.
Your source code – a set of ordinary .java, .py or .cpp/.h files.
Your UML – one or more .pdf, .png or .jpg files.
Everything else – exactly one .pdf file.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your newest submission will be marked. The late submission policy (see the Unit Outline) will be strictly enforced.

7 Academic Misconduct – Plagiarism and Collusion

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from *anyone* else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it.

These things are considered **plagiarism** or **collusion**.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for *you* to solve them *on your own*.

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by Turnitin and/or other systems to detect plagiarism and/or collusion.

End of Assignment