CITS5501 2020 project

Version: 0.2
Date: 5 May 2020
Please check the CITS5501 website to ensure that you have the latest version.

The goal of this assignment is to assess your understanding of class and unit testing, syntax-based testing, and Alloy modelling.

- The assignment contributes **35%** towards your final mark this semester, and is to be completed as individual work. It is marked out of 35.
- The deadline for this assignment is **23:59 pm, Sunday 31st May**.
- The assignment is to be done individually.
- Submit your assignment as either one zipped file, or multiple individual files, using **cssubmit**. Please do not use other archive file formats (e.g. `7z`, `.dmg`, `.rar`, `.sqx`).
- Your submission should contain a PDF report containing answers to questions 1–2, plus an Alloy "`.als`" file with your answer to question 3.
- You are expected to have read and understood the University Guidelines on Academic Conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort.
- You must submit your project before the submission deadline above. There are significant Penalties for Late Submission (click the link for details).

You are part of the software development team for Exron, a power generation company. Specifically, you work in the trade division, which trades in the energy resources market (electricity, gas, and carbon emissions).

The company trades in these commodities just as investors might trade in shares or futures – for example, a power producer (company A) could agree on a contract with another company (company B) to provide 500 Megawatts-hours for a given price in 10 months' time; and company B can sell its rights under the contract on the commodities market. Your company can buy and sell contracts in order to ensure it has enough supply for customers, or to make profit from excess supply.

Your team has been tasked with developing a new interface to the company's existing trading systems, which will allow traders to specify transactions to make using a specially-designed programming language (that is, a *domain-specific language*).

1. Classes and unit tests

Consider the following partially-specified classes:

- Interface `TradingOrganization`.

  Has getter and setter methods for:

  - `organizationID`, a `String`
  - `registeredAddress`, a `String`
  - `name`, a `String`
  - `emailAddress`, a `String`

- Interface `Formula`.

  Has a method:
  `double calculateResult()`

- class `Contract`

  Has instance variables:

  - `Date deliveryStartDate`
  - `Date deliveryEndDate`

- class `DieselContract`

  - Inherits from `Contract`.

  Has instance variables:

  - `ContractStorageDatabase db`
  - `double gallonsPerDay`
  - `TradingOrganization otherContractParty`
  - `Formula priceCalculationFormula`

  Has a method:
  `void finalizeContract()`.

When `finalizeContract` is executed on a properly initialized `DieselContract` object, the details of the contract are stored in a persistent database (the `db`) instance variable, and electronic authorization of the contract is performed over the network using the `otherContractParty`'s email address.

Answer the following questions. If you need to make any assumptions, state what they are.

a. Identify two plausible *class invariants*, each of which applies to one of these classes. (If both your invariants apply to the same class, that is fine.)

   Explain what the invariants are. What implications would it have for the behaviour of the system if the invariants were broken?

   ($\frac{1}{4}$ to $\frac{1}{3}$ page) [**5 marks**]

b. If writing a JUnit unit test for the `finalizeContract` method, what *test fixtures* might be required? Give at least two examples.

Sketch out code for a test class plus fixtures using a language and testing framework of your choice (Java or Python; if using another language, check with the unit coordinator first).

You do *not* need to write the actual tests; just class instance variables and any fixture-related code.

(Up to a page) **[5 marks]**

c. For the fixtures you described in question 1(b): if you were writing an integration test, would your approach be any different? Explain how and why.

($\frac{1}{4}$ to $\frac{1}{3}$ page) **[5 marks]**

d. If you were designing tests for the `finalizeContract()` method using input space partitioning, what would the *parameters* be? Identify at least two *characteristics* you could use for performing input space partitioning. Explain why they are appropriate, and how you would use them in writing unit tests.

($\frac{1}{2}$ to 1 page) **[5 marks]**

2. Contracts specification language

Pricing formulas for contracts can be specified by traders using a dedicated language. A fragment of the BNF syntax for the language is:

```
<quantityVar> ::= "diesel" <grade> | "windpower" <grade>
                  | "coalpower <grade>"
<operator>    ::= "+" | "*" | "-" | "/"
<builtinfunc> ::= "min(" <quantityVar> "," <dateRange> ")"
                    | "max(" <quantityVar> "," <dateRange> ")"
                    | "avg(" <quantityVar> "," <dateRange> ")"
<expression>  ::= <builtinfunc> <operator> <builtinfunc> |
                  <builtinfunc> <operator> "(" <expression> ")"
```

We assume that `<grade>` and `<dateRange>` are specified elsewhere, and here we will assume they are terminal symbols. (Or, if you like, you can assume they have only one possible production.)

a. Write down two sample strings from the language defined by this grammar. Given our assumptions, explain whether it is plausible to write exhaustive tests for the language.

($\frac{1}{3}$ to $\frac{1}{2}$ page) **[5 marks]**

b. If you wanted to achieve Terminal Symbol Coverage (TSC), how many tests would be needed? What about for Production Coverage (PDC)? Show any working and explain your reasoning.

($\frac{1}{2}$ to 1 page) **[5 marks]**

3. Formal methods

Write code for signatures and facts in Alloy which will do the following:

- Declare the existence of a "diesel contract" type.
- Declare the existence of a "formula" and "trading organization" types.
- Declare appropriate relationships and cardinalities for the above types.

Submit your work as an Alloy ("`.als`") file. [**5 marks**] Include explanatory comments in your model code.

Report requirements

Your report should be in PDF format, and use A4 size pages. The font for body text should be between 9 and 12 points. It should contain numbered headings, with useful heading titles. Any diagrams, charts or tables used must be legible and large enough to read. All pages (except the cover, if you have one) should be numbered. If you give scholarly references, you may use any standard citation style you wish, as long as it is consistent. Cover sheets, diagrams, charts, tables, bibliographies and reference lists do not count towards any page-count maximums.