Assignment 1

Object Oriented Software Engineering (COMP3005/6005)

Due: Thursday 22 April, 11:59
Weight: 35% of the unit mark

This assignment will cover the first four topics in the unit, up to and including object dependencies.

1 Your Task

Design and implement the system described below under Problem Description. Specifically:

- Use either Java, C#, C++ or Python to implement the system described.
  - Your code should be of high quality, with consistent formatting and meaningful commenting.
  - If using Java, it is highly recommended that you use Gradle or Ant.
  - If using C#, please only use features supported by Mono.
  - If using Python, please use type hinting for all method/function parameters and return types.
- In your design, use **(a) the Strategy Pattern** and/or Template Method Pattern (one or both), and **(b) the Composite Pattern**.
  - You may also use other patterns if you wish (whether or not they are described in the unit), but that is purely optional.
- Provide a **README.txt** that briefly explains how to compile and run your code, if there are any facets of this that the marker needs to be aware of.
- Provide a UML class diagram giving a structural overview of your design.
  - Your UML must accurately represent your design.
  - Use a proper tool; e.g., Umlet, draw.io, PlantUML or others. Avoid tools that have no explicit support for UML.
  - Make the diagram layout as neat and logical as practical. Minimise crossing lines.
  - If in doubt as to whether or not to represent certain things on the diagram, err on the side of more detail. But it is generally unnecessary to show constructors and (in most cases) usage dependencies.
- Provide a response to each of the marking criteria.
  - Record this in a file called **criteria.txt**.
  - See below for more information.

2 Submission

Submit all of the above, plus a signed declaration of originality, to the appropriate area on Blackboard. Include everything in a single .zip/.tar.gz file (*please avoid* .rar, .zipx or .7z). You do not need to include any compiled code.

You must verify that your submission is correct and not corrupted. Once you have submitted, please download your own submission and thoroughly check that it is intact. You may make multiple submissions. Only your last one will be marked.

3 Marking Criteria

The assignment will be marked out of 30. There are six marking criteria, each worth 5 marks, as follows.

1. Appropriate use of containers.
2. Clear and distinct packages/(ass/interface/method responsibilities.
   - It is required that you create and use packages or namespaces, depending on the language.
   - Packages, classes, interfaces and methods must all have a clear logical purpose.
   - Packages, classes, interfaces and methods must be appropriately-named, in light of what they contain.
3. Appropriate error handling.
   - *For all possible user errors*, the program must take appropriate alternative action, and should *not* simply abort with a stack trace. (A user error means an error caused by something outside the program and the platform on which it runs.)
   - The program must *not* treat internal errors (such as a `NullPointerException`) as if they were a user errors. Ideally there should be no internal errors, but the program should be written such that, *if they did happen*, they would abort the program.
   - The program must use exception handling mechanisms appropriately to do both of the above.
4. Appropriate use of the Strategy Pattern and/or Template Method Pattern.

   We are *not* just interested in whether you can mechanically implement the parts of these patterns. You must use one (or both of them) for a practical purpose. Is it solving a problem? What is it decoupling? Are the subclasses actually doing substantively different things?

   Be warned against:
   - Creating subclass objects in the same place you're calling them.
   - Querying which subclass you're dealing with.

   The purpose of Strategy/Template Method collapses if you do these things.
5. Appropriate use of the Composite Pattern.

   Similar advice applies to the above.
6. Clear and correct UML.

3.1 Does the program run correctly?

This isn't strictly one of the six criteria, but rather across-cutting concern that affects all of them.

The marker will attempt to compile and run your code. If it does not work correctly, this will have implications for how we assess the criteria.

3.2 Your Responses to the Criteria

You must provide a response to each of the six criteria, as part of your submission. That is, for each individual criterion, write a paragraph or two justifying the choices you have made. Include all your discussion in `criteria.txt`.

This fulfils two purposes:

1. We want to know that you can articulate, in plain English, what you are doing. This is part of the task.
2. If you do something unusual/unorthodox, this is your opportunity to justify it.

One way to think about what you're doing here is this: Imagine that the marker is slightly sceptical about whether to award you the marks for a given criterion. What would you say to convince them?

Make your responses concise and to-the-point. Address each criterion separately.

4 Problem Description

Your application should **model a city's electricity usage**. It must first either *read in OR randomly generate* a set of data representing the structure of the network. Then it must **either** *write out* **OR** *display* **the network**. The user chooses which actions to perform via the command-line.

4.1 Data

The program must manage an object structure representing a hierarchical electricity network.

> **Note**
>
> For anyone who actually knows about such things, this is almost certainly *not* how a real-world electricity network actually works, so just pretend we're in a fantasy world!

Our network is a tree. At **the root is the whole city**. Each **non-root node represents a part of the city; e.g., a district, suburb, street, building; floor; or some other kind of subdivision**. **The tree's leaf nodes are of course the smallest meaningful parts of the network** (say, **individual houses**, or **apartments**, or **floors of large buildings**, though there's no difference between these for our purposes).

There is **no limit to the depth of the tree**, or to **the number of child nodes each node may have**. A **tree containing just a single root node is also valid**.

Each **node in the tree has a name consisting of letter and/or digit characters (no spaces or other symbols)**, and **all nodes must have different names**. Each **leaf node also has a series of "power consumption" numbers (non-negative real numbers); one for each of several "categories"**. A category is one of the following:

Category	Abbreviation
Weekday morning	dm
Weekday afternoon	da
Weekday evening	de
Weekend morning	em
Weekend afternoon	ea
Weekend evening	ee
Heatwave	h
Special event	s

The abbreviations will be used in the file format shown below. The exact meaning of these categories isn't that important for our purposes. Suffice it to say that each leaf node has a particular power consumption *for each category*.

4.2 Command-Line

All user input occurs via the command-line. The exact method of executing the program will depend on the language and build system, so here's what we mean first:

Language/Environment	Command
Java, using `gradle run`:	`./gradlew run --args='[arguments]'`
Java, directly invoking a *jar* file:	`java -jar program.jar [arguments]`
C#, using Mono:	`mono program.exe [arguments]`
Python:	`python program.py [arguments]`
C++:	`./program [arguments]`

All user input must occur via this method. The user must supply the following:

- Either **"-g"** for generate, or **"-r" for read**, where the latter must be followed by an input filename.
- Either **"-d"** for display, or **"-w" for write**, where the latter must be followed by an output filename (which cannot be the same as the input filename, if provided).

So, if the application is Java based, and we want to use Gradle, the following are all valid means of invocation:

```
./gradlew run --args='-g -d'
./gradlew run --args='-g -w outputdata.csv -d'
./gradlew run --args='-r inputdata.csv -d'
./gradlew run --args='-r inputdata.csv -w outputdata.csv'
```

It is an **error if neither `-g` nor `-r` are provided**, or **if both are provided**, or **if `-r` is not followed by a filename**. Similarly for `-d` and `-w`. Optionally, you may allow `-w` to be written before `-g`/`-r`. Or you can require the user to provide them in the order indicated above.)

In the case of such an error, the program must say what went wrong.

4.3 Reading and Writing

If `-r` is given, the program must read the data file specified. If `-w` is given, the program must write its data to the (other) file specified.

> **Note**
>
> In the event that both `-r` and `-w` are given, both apply as normal, i.e., the data should be read from one file and, if valid, written to another. The validation requirement means the data cannot just be blindly copied.

The same file format must be used for both reading and writing the data. The data file should contain one line per tree node:

- The first line of the file specifies the name of the root node:

  [name]
- A line representing a non-root, non-leaf node contains a name and a parent node name, separated by a comma:

  [name],[parent-name]

  The parent node (which may or may not be the root) must be specified on an earlier line of the same file.
- For leaf nodes, the line contains the node name, parent node name, and the power consumption for one or more categories:

  [name],[parent-name],[cat1]:[power1],[cat2]:[power2],...

  Each *[catN]* is an abbreviation of a power consumption category, and each *[powerN]* is a non-negative real number specifying the corresponding power consumption. The categories can be given in any order, and can also be omitted, in which case the power consumption is assumed to be zero for that category.

Example

```
city
northside,city
southside,city
building7,northside,h=175.015,dm=550.8
building2,southside,s=644.2,em=540.1,da=97.3
building5,southside,dm=8950
```

It is an error if an input file does not exist or is not in the expected format, or if some storage-related error occurs during either reading or writing. **In such cases, the program must gracefully exit with an appropriate error message**.

4.4 Random Generation

If `-g` is given, the entire tree must be auto-generated. The method for doing this is simplistic, and does not reflect the limits on what a tree can look like in general.

1. Generate a random **tree depth between 2 and 5** (inclusive, where 1 means the tree consists only of a root node).
2. Generate the root node, and generate a name for it.
3. Until the tree has reached the required depth:
   - For each existing node:
     - Generate a number of child nodes, **between 2 and 5 (inclusive)**.
     - Generate a name for each new node.
4. For each leaf node, and for each power consumption category, generate a random number between **10 and 10000**.

The names of the nodes do not necessarily need to be randomly chosen, but must be unique.

4.5 Displaying

If `-d` is given, two things must happen:

1. The tree must be displayed (after it has been read in or generated).

   To display the tree, output the names of the nodes, one per line, with indentation used to indicate which node is a child of which other node. The root node has no indentation.

   Example (based on the file above)

   ```
   city
       northside
           building1
       southside
           building2
           building3
   ```

   (If you wish, you may use ASCII/Unicode line-drawing characters to draw tree-like lines from parent to child nodes.)
2. The *total* power consumption for each category (across the whole city) must be calculated and displayed.

   For a given category, you can calculate the total power consumption by simply adding up the power consumption of each leaf node for that category.

   Example (not based on any previous example)

   ```
   Weekday morning    : 2005.88
   Weekday afternoon  : 4748.7
   Weekday evening    : 5539.9
   Weekend morning    : 3221.0
   Weekend afternoon  : 5193.2
   Weekend evening    : 7883.2
   Heatwave           : 11298.8
   Special event      : 6887.5
   ```

5 Academic Integrity

Please see the *Coding and Academic Integrity Guidelines* on Blackboard.

In summary, this is an assessable task. If you use someone else's work or assistance to help complete part of the assignment, where it's intended that you complete it yourself, you will have compromised the assessment. You will not receive marks for any parts of your submission that are not your own original work. Further, if you do not *reference* any external sources that you use, you are committing plagiarism and/or collusion, and penalties for academic misconduct may apply.

Curtin also provides general advice on academic integrity at academicintegrity.curtin.edu.au.

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.