Assignment 1

Object Oriented Software Engineering (COMP2003/6005)
Semester 1, 2022

Due: Thursday 14 April, 23:59
Weight: 35% of the unit mark

Contents

This assignment will cover the first four topics in the unit, up to and including error handling and logging.

20152428

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.
  - If using Java, you must also use PMD as in the practical worksheet code, and it's thus highly recommended that you use Gradle.
  - 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 Decorator Pattern.
- 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, but provide the diagram in PDF or PNG form.
  - 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 the marking criteria.
  - Record this in a file called `criteria.txt`.
  - See below for more information.

2 Submission

Submit all of the above 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 do not need to sign a declaration of originality, but the act of submitting will be taken as agreement with the principles of academic integrity.

20152428

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

3.1 Demo

You are principally being assessed on *design*. However, we also need to ensure you produce a working app, and running demos is an efficient way to do this.

The logistics will vary between locations (this unit runs simultaneously in Perth, Miri and Sri Lanka), but the following approaches are being considered:

- Uploading a short video.
- Scheduling a real-time online demo (e.g., over Blackboard Collaborate or other video conferencing tools).
- Scheduling a face-to-face demo.

Further announcements will be made on which approach(es) will actually be used.

In any case:

- You will be asked to *download* your own Blackboard submission, possibly perform a cryptographic hash (e.g., `sha256sum`) on it, unzip it, build it, and run it.
- You will be provided with *different test data* from that shown in this specification, to prove that your design handles any test data.

If your app does not work as expected, you should expect a mark penalty.

3.2 Criteria

The assignment will be marked out of 35. There are seven marking criteria, as follows:

1. **3 marks** – General code quality, including addressing PMD rules.
    - The code must be reasonably-well commented.
    - If using Java, you must also use PMD, and use the same PMD "ruleset" — "`oose-pmd-rules.xml`" — used in the practical worksheet code.

- There should be no PMD rule violations.
- You may *suppress* PMD rules, in specific places, if appropriate, as long as you include a comment justifying it. If you're not certain what would be appropriate, ask in advance (in general terms).

- If using another language, you must find and use another such "linting" tool in an equivalent manner.

2. **3 marks –** Appropriate use of containers.
    - This is at your discretion, but you must find reasonable uses for containers within your code.
3. **5 marks –** Clear and distinct class/interface/method responsibilities.
    - Classes, interfaces and methods must all have a clear logical purpose.
    - Classes, interfaces and methods must be appropriately-named, in light of what they contain.
4. **6 marks –** Appropriate error handling and logging.
    - The program must use exception handling appropriately. It must gracefully handle external errors, without compromising the debugging of internal errors.
    - The program must include a reasonable set of logging statements.
5. **6 marks –** 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.

6. **6 marks –** Appropriate use of the Decorator Pattern.

Similar advice applies to the above.

7. **6 marks –** Clear and correct UML.

3.3 Your Responses to the Criteria

You must provide a response to each of criteria 2–6, as part of your submission. For each of those critera, write a paragraph or two justifying the choices you have made. Include this 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

The application will be a terminal-based maze game.

4.1 Maze Structure and Concepts

The maze consists of a rectangular grid, with at least one row and at least one column. We'll number rows from 0 at the top, and columns starting from 0 at the left. There are horizontal and vertical "walls" and "doors" *between* some grid squares, and nothing between other grid squares. Just to emphasise: the walls and doors do not occupy grid squares, but exist between them.

There is always a wall around the entire outside perimeter of the maze.

Each door has one of six colours (1–6), and can be "opened" by a key of a corresponding colour. Keys can be found in certain grid squares around the maze. Grid squares may also contain messages.

The player's icon also occupies one grid square at any given point in time.

4.2 Input File

The input file is a text file containing a maze structure. The first line contains the maze size: first the number of rows, then a space, then the number of columns. (There is no maximum size, but there must be at least one row and one column.)

Each other line starts with a code, indicating what *kind* of information it contains, then a sequence of several integers. Here are the different kinds of codes and what they do, with examples of the integers that follow:

- "`S 0 3`" indicates that the player starts at location (0,3) in the maze (meaning row 0, column 3).
- "`E 6 1`" indicates that location (6,1) (row 6, column 1) is an end point in the maze. (There can be one or more of these.)
- "`K 3 5 4`" indicates that a colour-4 (blue) key is at location (3,5).
- "`WH 2 3`" indicates a horizontal wall north (above) of square (2,3) (and thus south of square (1,3)).
- "`WV 2 3`" indicates a vertical wall west (left) of square (2,3) (and thus east of square (2,2)).
- "`DH 2 3 4`" indicates a colour-4 (blue) horizontal *door* north of (2,3).
- "`DV 2 3 4`" indicates a colour-4 (blue) vertical door west of (2,3).
- "`M 3 4 You step into the light...`" indicates that a message is associated with location (3,4). The message is free-form text that starts after the space after the second number, and occupies the remainder of the line.

Importantly:

- These lines may occur in any order.
- Not all locations need to be mentioned in the file.
  - The shortest valid file contains just the maze size, plus one "`S ...`" line and one "`E ...`" line (though this will create a "maze" that is just a bare, rectangular room).
- Locations can be mentioned multiple times. For instance, a single grid square might contain keys of all six different colours, as well as three messages, as well as having walls/doors on all sides.

- It is valid to have multiple keys and doors of the same colour, and even (for keys) in the same square.
- It is valid to use any subset of the six colours; e.g., 2 and 5.
- It is valid to have door *without* a corresponding key, or vice versa
- It is valid to have keys at the start and/or end locations.
- It is valid (though obviously redundant) to have multiple "`E ...`" lines indicate the same grid square.
- It is valid for a square (or any subset of the maze) to be completely enclosed in walls.
- It is valid for a maze to be impossible to complete.
- It is valid for a maze to be instantly completable, if an end location coincides with the start location.

There are some non-fatal errors though:

- If any row or column specified in the file is outside the bounds of the maze.
- If there is no "`S ...`" line (starting location), or more than one "`S ...`" lines.
- If there is no "`E ...`" line (ending location).
- If there are multiple walls or doors, or a wall and a door, at the exact same place (where "place" includes the orientation — horizontal or vertical). This includes attempts to place walls or doors on the perimeter of the maze, given that there are automatically walls there to begin with.

In these cases, the game must be allowed to proceed, but the user must be warned of the specific issues before it starts. The game should then either ignore the offending line(s), or use default value(s), as appropriate.

Any other deviation from the file format is a fatal error (especially if something is unparseable). In that case, the game must not proceed.

Here's an example file:

```
4 4
WV 0 1
WV 0 3
WV 1 2
WV 1 3
WV 2 1
```

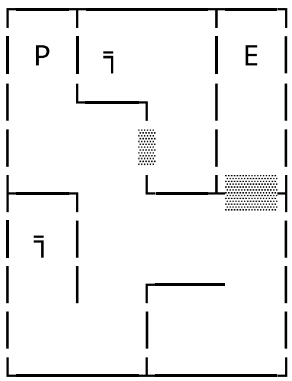
```
WV 3 2
WH 1 1
WH 2 0
WH 2 2
WH 2 3
WH 3 2
S 0 0
M 0 0 Good morning!
M 0 0 Which way?
E 0 3
M 0 3 Finished - well done!
DV 1 2 1
DH 2 3 2
K 0 1 2
K 2 0 1
M 3 2 You stare blankly at a wall.
```


4.3 Display

The maze game will be terminal-based, making use of two tricks: box-drawing characters, and ANSI escape codes (both of which are described here).

Box-drawing characters

With box-drawing characters, your game should be able to create a maze like this (depending of course on the input file):



This is omitting the start position and any messages that might be shown, as these do not need to be shown within the maze itself. We're also omitting the colours of the doors and keys, but we'll get back to that later.

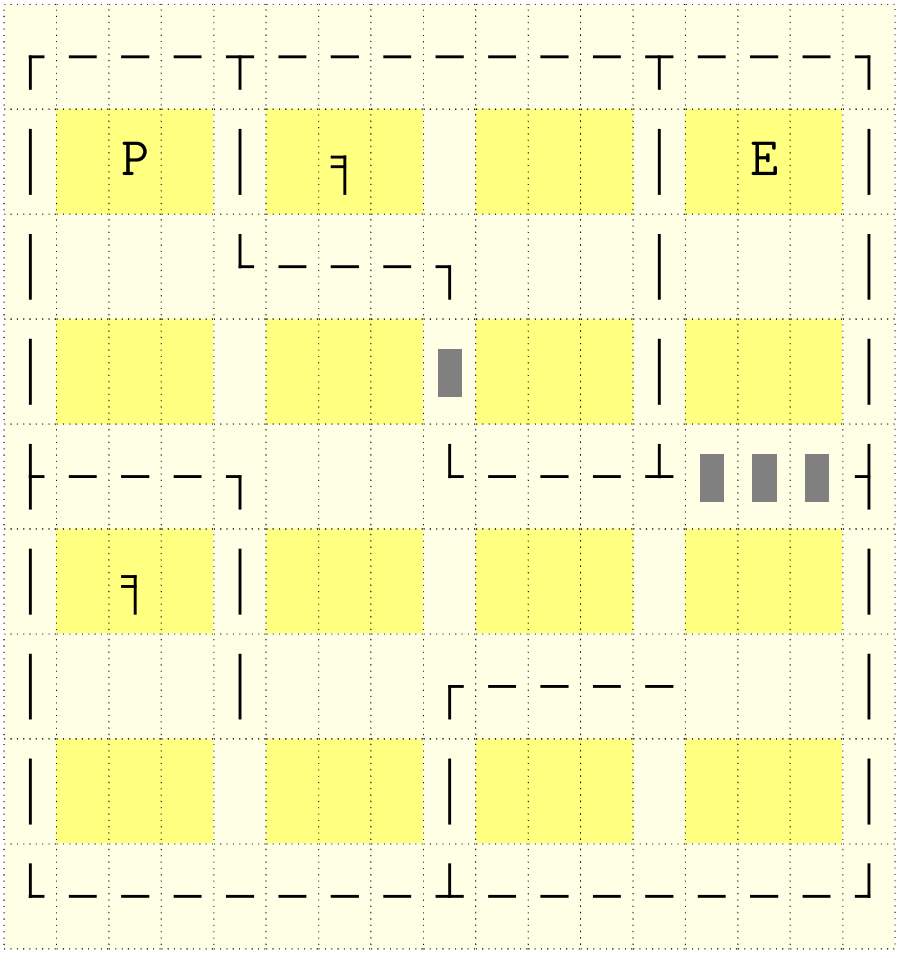
The above consists of individual characters that make up horizontal and vertical lines, joins and blocks. The minimally-useful set of such characters (and their Unicode codes) are: –

(\u2500), │ (\u2502), ┌ (\u250c), ┐ (\u2510), └ (\u2514), ┘ (\u2518), ├ (\u251c), ┤ (\u2524), ┬ (\u252c), ┴ (\u2534), ┼ (\u253c), ▒ (\u2592) and ╕ (\u2555). They are designed to line up exactly to one another in a terminal, to produce continuous lines (though ╕ just happens to look a bit like a key). You can copy and paste them directly into your source code (except for C++), or alternatively use the Unicode codes. (See the full range of such characters on Wikipedia: wikipedia.org/wiki/Box-drawing_character.)

The above example is, of course, a 4×4 maze with two keys, two doors, the player's icon in the upper left, and an end goal in the upper right. In plain-text terms, though, it's a 17×9 grid of characters. Each maze square occupies 3 adjacent characters, mostly spaces, but other symbols as needed. Every 2nd row of characters (starting on the 1st row) is set aside for horizontal walls/doors (or space where they might go). Every 4th column of characters (starting at the 1st column) is set aside for vertical walls/doors.

Where the wall-rows and wall-columns meet, we get join characters like ┌, ┴, ┼, etc. The particular join character used should be based on the adjoining walls/doors.

Here's a detailed representation of the above, to avoid ambiguity:



Each small, dotted rectangle indicates a single character. The darker yellow indicates the actual maze grid squares, while the remaining character positions are the walls and doors (or positions for them).

ANSI escape codes

Doors (▒) and keys (╕) should be drawn in their particular colours, and the screen should be cleared before the maze is drawn.

ANSI escape codes let us do this. They consist of the "escape character", followed by "[" (ordinary left square bracket), followed sometimes by an integer, followed by a letter. The "escape character" is non-printable, but can be represented in source code as "`\033`". In particular:

- "`\033[31m`" sets the text colour to red, and other numbers from 32–36 select other colours: green, yellow/orange, blue, magenta and cyan;
- "`\033[m`" resets the colour to the default terminal colour;
- "`\033[2J`" clears the screen.
- "`\033[H`" re-positions the cursor (where the next text will appear) in the top-left. This may overlap in function with "`\033[2J`".

For instance:

```
// Display "Hello" in red, then "world" in green
System.out.println("\033[31mHello \033[32mworld\033[m");
```

You're free to use colours for other aspects of the game too, if you like, or experiment with other ANSI codes.

> **Note**
>
> Windows 10+ supports ANSI escape codes in "Windows Terminal", and in version 1909 they are (should be!) enabled by default. If you're using an older version, you may notice that printing an ANSI escape code will just print the literal code, without the desired effect.
>
> In that case, it may be possible to enable ANSI, or use a third-party terminal program (e.g., ConEmu).

However, if you can't make it work at all, don't worry! Just make a note of this, and use alternate characters to represent different "colours" of doors and keys.

4.3.1 Multiple items at the same location

Each grid square consists of only three characters, whereas it could contain any number of keys, as well as (possibly) an end location.

You cannot, of course, draw six keys using only three characters. Instead, just draw one (or perhaps up to three if you like).

4.4 Game Mechanics

Once the game starts, the player must be able to move around the maze by typing "`n`", "`s`", "`e`" or "`w`" at a prompt, to be displayed below the maze. On each move, the screen should be cleared, and the maze re-displayed, showing the character's updated position.

For an invalid move (e.g., if the player tries to move into a wall), an error message should be shown. You don't necessarily need to re-display the maze in this case.

If the player's current grid square contains a message (or messages), they should be displayed under the maze (above the prompt).

If the player's current grid square contains a key (or keys), the player immediately "picks up" these keys, and they disappear from the maze. The player's acquired keys should be displayed under the maze (above the prompt/messages).

If the player attempts to move through a door (▒), this will succeed if the player holds the corresponding key, and fail otherwise, with an error message. In either case, nothing else happens. The key remains with the player, and the door remains in place.

If the player reaches an end square (`E`), they win, and the game ends.

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.