2AA4-Assignment1 Report

Step 1:

Q1. Why is it preferable to use libraries such as Apache CLI or Log4J to support things like parsing the command line or printing log statements instead of writing code from scratch?

Using libraries such as Apache CLI or Log4J increases code usability and efficiency, as it focuses more on the central problem by offloading tasks such as the command line parsing to these libraries. In this way, it also enhances code maintainability and structure, since these libraries are all well designed. Also debugging, more features etc.

Q2. What are the advantages/inconveniences of using a logging approach?

Advantages:

Logs provide a detailed history about the problem, this is good for troubleshooting and debugging. Logs also have many features, that it supports display information, error messages and a lot more. Logs also do not interfere with other component of the code. Log messages could also be saved persistently, providing a history record for future access.

Inconveniences:

Log messages require some learning and understanding than just printing messages using system.out.print. Log also requires dependency and libraries. So in other environment, we might have to make some modifications in order to make the code work.

Step 2:

Q1: How did you identify these abstractions?

I identified these abstractions through planning in advance with UML diagrams. After focusing on the tasks need to perform and the provided starter code, I was able to disintegrate the main class into several classes: Maze, Explorer, and Path.

- The Maze class is responsible for handling take in file from command line prompts(-i flag), while storing the map in a data structure which allows future access.
- Explorer class has a explore method which traverse through the maze to find if there is a path to get from beginning to end. It will then return the path as a string; ex: “FFF”.
- Path is a class that handles with the user input path (-p flag). This class takes in a string representing the path, and will compute the path to see if it is valid. This class needs to be implemented after the MVP, when adding additional features. This class may need to extend from explorer class.

Q2: What makes you think they are the right ones? Justify your design choices.

I think the abstractions are the right ones because they all follow those fundamental OOP principles, such as encapsulation, single-responsibility. I designed those abstractions such that each class handles only one responsibility. For example, the maze class is only responsible for reading in and storing the maze, while the algorithm are done in other classes. This allows the separation of concerns, improving code structure and avoid potential bugs. I also applied encapsulation, so that all these three classes could have an instance of their objected and be accessed from main. The user in main do not necessarily need to know how each classes work, they only need to use the methods from the classes.

Step 3:

Q1: How did you identify these features?

I identified these features first by simplifying the problem. The easiest maze we could have in this case is linear. That the root to the end is just going straight. From then on, I developed a series of functionalities the program should perform; take in maze file, explore maze, check beginning and end position, output path.

Q2: How will you ensure that they model visible value for the end user?

Those features described above all have user facing value. The Maze class has features such as getStart() and getEnd(), ensuring user know the start and finish point. Path class provides a root for easy understanding for user(encapsulation). I also uses the Kanban Board for iterative development. This allows us to separate features into smaller tasks, and the user could see each incremental through Kanban board.

Step 4:

Q1: What made your MVP viable? Justify why your choices make your mvp release a viable one.

My MVP is viable since it is able to meet the basic requirement of what is being asked. The core functionalities are all well implemented; the program is able to recognize a -i flag for file input. It reads the file and is able to output a path with the basic algorithm. The program also uses logging and System.out.print to interact with user.

Q2: What made your MVP minimal? Justify why this version is minimal.

My MVP is minimum because it only targets specific type of maze; the direct maze. The program haven't yet implemented complex algorithm like right-hand exploration just yet. It is not able to handle other maze type involving dead end, turns, etc. The maze could still have other more features, such as validating path, that the user enters a -p flag and a path, the program validates it the path works on the maze. Many part of the code could still need optimization.

Step 5:

Question1: How did you encapsulate (information hiding) your maze exploration? How does it interact with your maze representation?

The maze exploration logic is well encapsulated in different classes, such that they abstract decision making and movements. In maze class, we have methods such as check wall and check passage, which encapsulates the internal algorithm while only returning a Boolean value. Similarly, the Position class encapsulates the details about the explorer's current statues(ie: direction, x and y coordinates). And the Explorer class encapsulates methods such as the right hand exploration algorithm.

Question2: How is your code design when assessed from a SOLID point of view?

SOLID consists of Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle and Dependency Inversion Principle. For single responsibility, the maze class only handles responsibilities of reading and storing the maze, the position class only handles the direction and current location. For open closed principle, the explorer class are open for extension for new exploration methods other than the right hand exploration. However, it is well implemented now so it is closed for modification. Interface segregation: The Maze class only provides relevant methods to exploration logic, avoiding unnecessary dependencies. Dependency Inversion: The exploration algorithm depends on abstractions (Maze and Position) rather than concrete implementations, making the system flexible.

Question3: How will your code support new algorithms?

Expanding from what is talked in question 2 about the open-closed principle. When I need to implement new algorithms in the future, I could simply extend form the Explorer class(ie class NewExplorer extends Explorer). And override the method of explore with the new algorithm implementation. This makes the code flexible while avoiding bugs from existing code.

Question4: How did iterative and incremental principles manifest in your project?

The development of the product was a in iterative and incremental approach. I started with the beginning code that was given. First by replacing print statements to logs and separating the code from a huge main class into different classes. Later on, I was able to complete a MVP that works for a specific type of maze. I continue the incremental process until the final product is done.

Question5: What was the typical timespan of an iteration?

The typical timespan of an interaction is 1-3 weeks. Similarly for this product, the first week was setting up everything: the dependencies, logs and different classes. Next week I worked on achieving a MVP. The final week, I incremented to completing the final product which contains all the business logic and features.