


Software construction - Assignment 3

First of all, we ask you to create a new folder for this assignment in your GitHub repository. Before starting the assignment, copy and paste the code developed for assignment 2.

Please note that the folder used in the previous assignment must still contain the original code.

Ülkü

Part 1: Choose **two** design patterns among those that you saw in class: **Singleton, Observer, Adapter and Iterator**. For each chosen design pattern, you must add a corresponding implementation in your code.  **Janosch**

Use a copy of the code of assignment 2 as starting point.

Then, per each chosen design pattern, complete the following points:

1. Write a natural language description of **why** and **how** the pattern is implemented in your code.
2. Make a class diagram of **how** the pattern is structured statically in your code.
3. Make a sequence diagram of **how** the pattern works dynamically in your code.

Address all of these 3 points in a **ANSWERS.md** file and add it to the assignment folder.

Part 2: Consider the Board class implemented in your chess game. Draw its sequence diagram, considering all the method calls that can make it transition to a different state.

Write a description of the sequence diagram (if needed) to further describe some of your choices.

Address these points in the **ANSWERS.md** file in the assignment folder.

Part 3: In the final part of the assignment we ask you to choose **one** of the following three functionalities and implement it in your code. Use the code from assignment 2 as starting point.

Specify which functionality you decided to implement in the ANSWERS.md file.

1) A chess game with non-standard pieces. In this version of the game, you should implement the following pieces:

Superqueen: the queen becomes a superqueen. The superqueen class must implement the following method:

- *Teleport:* allows the superqueen to teleport in any free square on the map. It must be used instead of the method move of the other pieces

Archbishop: the bishops become archbishops. An Archbishop class must implement the method:

- *Gallop:* allows the bishop to move like a normal bishop or a knight. It must be used instead of the method move of the other pieces.

In this implementation, you must use the **Adapter** design pattern to allow the game to work with the new pieces without significantly changing the code that you already implemented in assignment 2. Ideally, you should not need any modification of the previous code.

The input used for the previous assignment can remain the same. The superqueen can still be referred to as 'Q' and the archbishops as 'B'. The same applies for the output.

Please state again the notation accepted as input in the ANSWERS.md file.

Use the ANSWERS.md file to describe your implementation choices.

2) Implement a Scoreboard and a ChessPieceIterator class:

- Implement a Scoreboard class using the **Singleton** design pattern. The Scoreboard class must keep track of the score of each player. Each player scores a point when he/she eats an opponent piece and the Scoreboard must be updated accordingly.

Note: each piece is worth 1 point, excluding the queen who's worth 5 points.

Furthermore, the output of the game must be modified to show the players' score after each turn. For example, you should output a String similar to this: "Player 1, score: x - Player 2, score: y", where x and y are the scores of the players.

- Implement a class ChessPieceIterator which contains a list of all the pieces on the board at the beginning of the game.

This class must implement the Iterator design pattern (therefore implementing the next() and hasNext() methods) and should be used to print a list of all the chess pieces when the program starts.

For example, the output format might look like "White pawn position a2" , "White pawn position b2" and so on.

As long as the list of printed pieces is clear, you are free to decide which information show to the user.

Please state again the notation accepted as input in the ANSWERS.md file.

Use the ANSWERS.md file to describe your implementation choices.

Jonas

3) Implement a Scoreboard using the **Observer** pattern. The Scoreboard class must keep track of the score of each player. Each player scores a point when he/she eats an opponent piece and the Scoreboard must be updated accordingly.

Note: each piece is worth 1 point, excluding the queen who's worth 5 points.

Furthermore, the output of the game must be modified to show the players' score after each turn. For example, you should output a String similar to this: "Player 1, score: x - Player 2, score: y ", where x and y are the scores of the players.

Every time a piece is eaten the Board (or an equivalent class in your implementation) must notify the Scoreboard that something has changed. The Scoreboard must update itself accordingly.

Please state again the notation accepted as input in the ANSWERS.md file.

Use the ANSWERS.md file to describe your implementation choices.

It is possible that your previous design does not perfectly allow the addition of the functionality you chose. In case it is necessary for you to further modify the code written in assignment 2, please clearly describe your modifications in the ANSWERS.md file.

Please clearly separate the ANSWERS.md file based on the 3 parts that compose this assignment.

Deadline: Friday November 8 2019 at 18:00.

Please remember to make use of the ANSWERS.md file to inform us about choices you took and you would like us to consider during our correction.

We will only evaluate the code present on the Master branch at the time of the deadline.