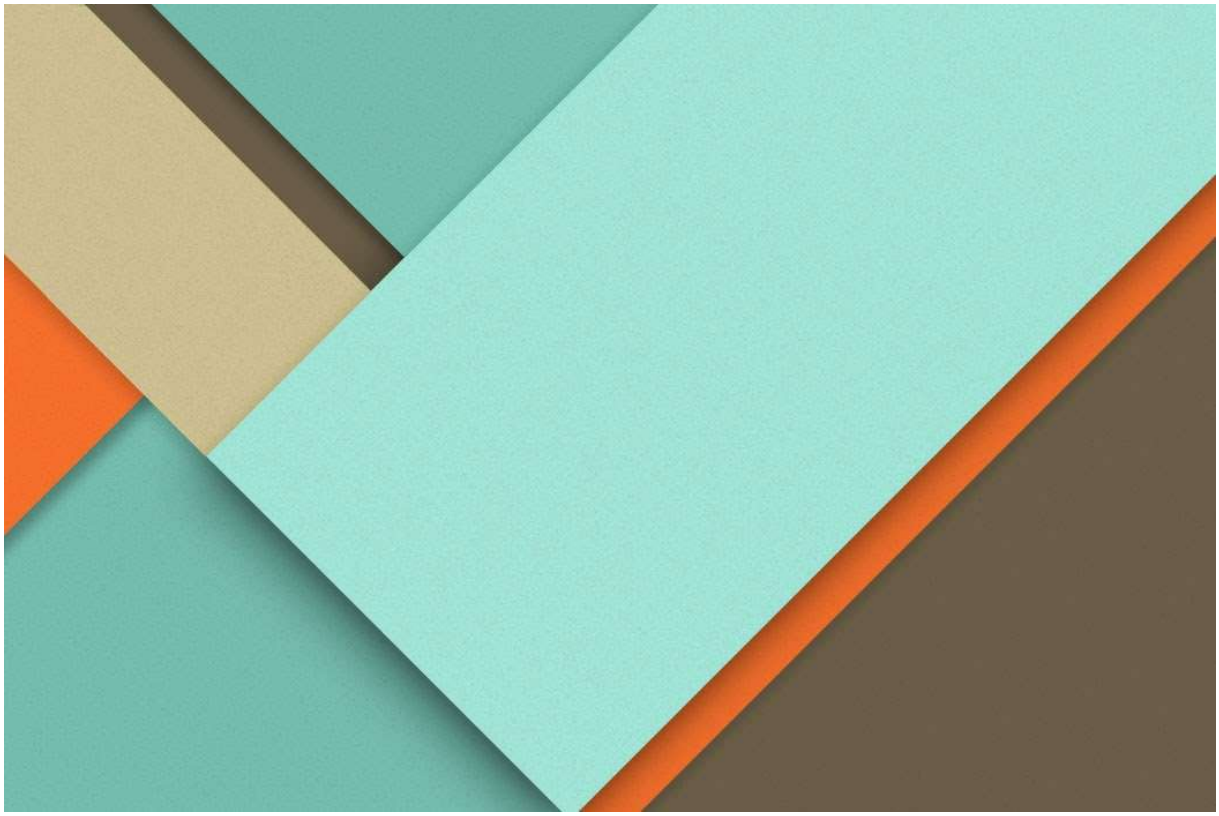


Design Patterns

CSC207 - A3 Threemusketeers

2021/11/20

Team: voidNull

Group Memebers:

Zezhu Yu,

Hongsheng Zhong,

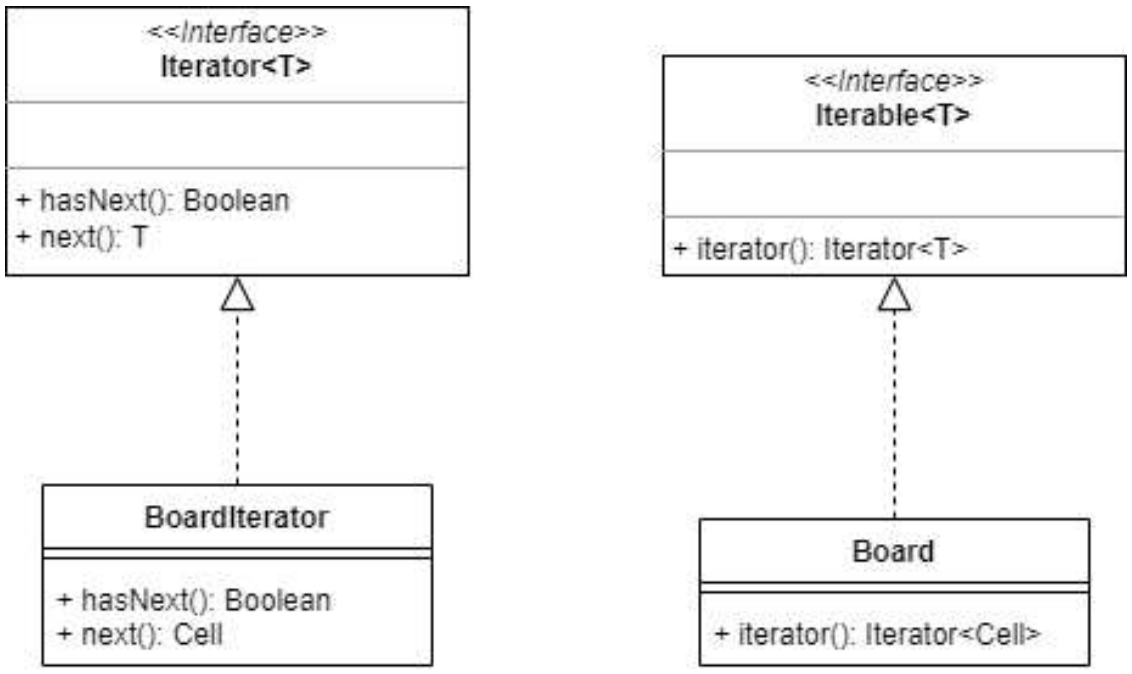
Philip Huang,

Pengcheng Wang

Table of Contents

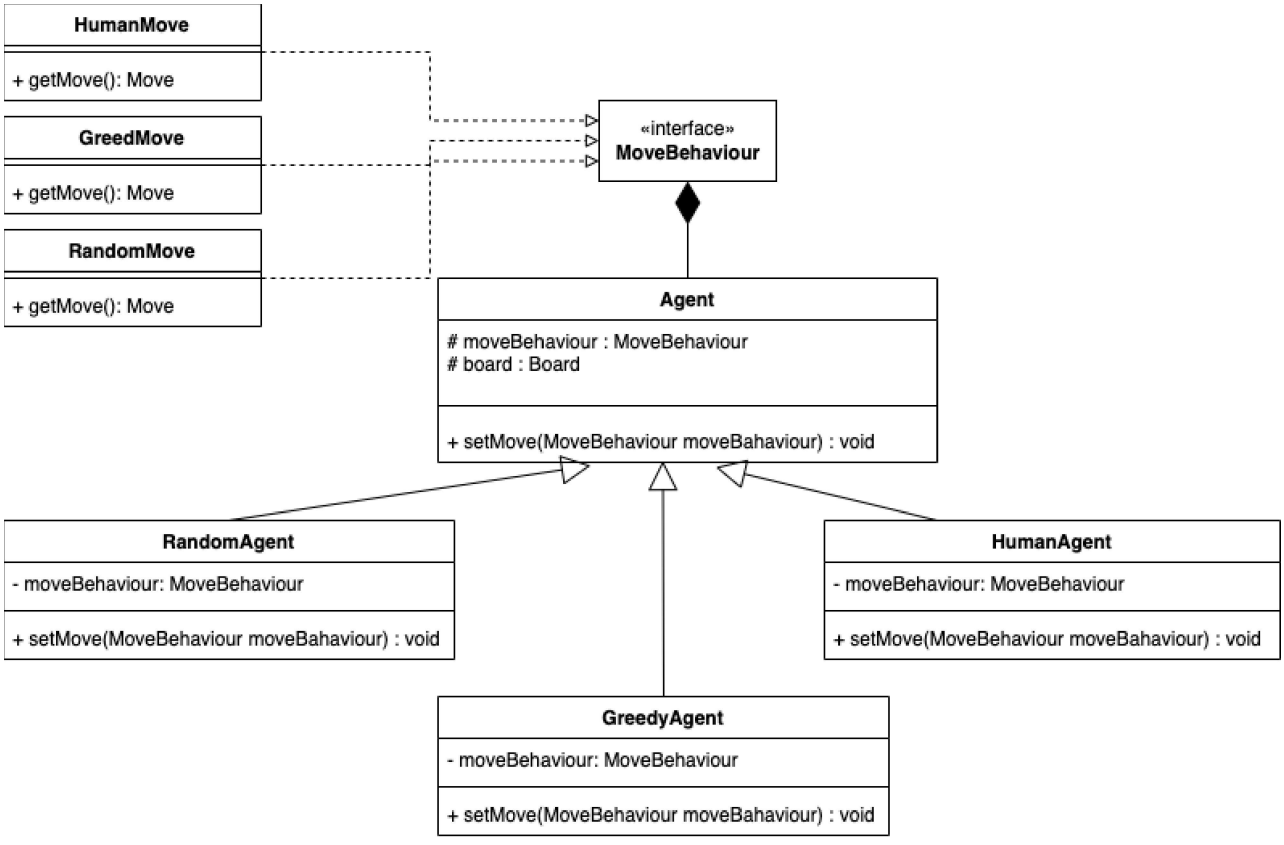
Iterator Pattern

The Iterator Pattern defines how to iterate an iterable object. In the Three Musketeers game, we have a lot of work on iterating the board to find cells (for example, getCells, getGuardCells, getMusketeerCells and so on). Iterating a nested list of cells creates huge blocks of code and wastes a lot of the computer's resources. Therefore, the Iterator Pattern helps improve our work on iterating cells of a board.



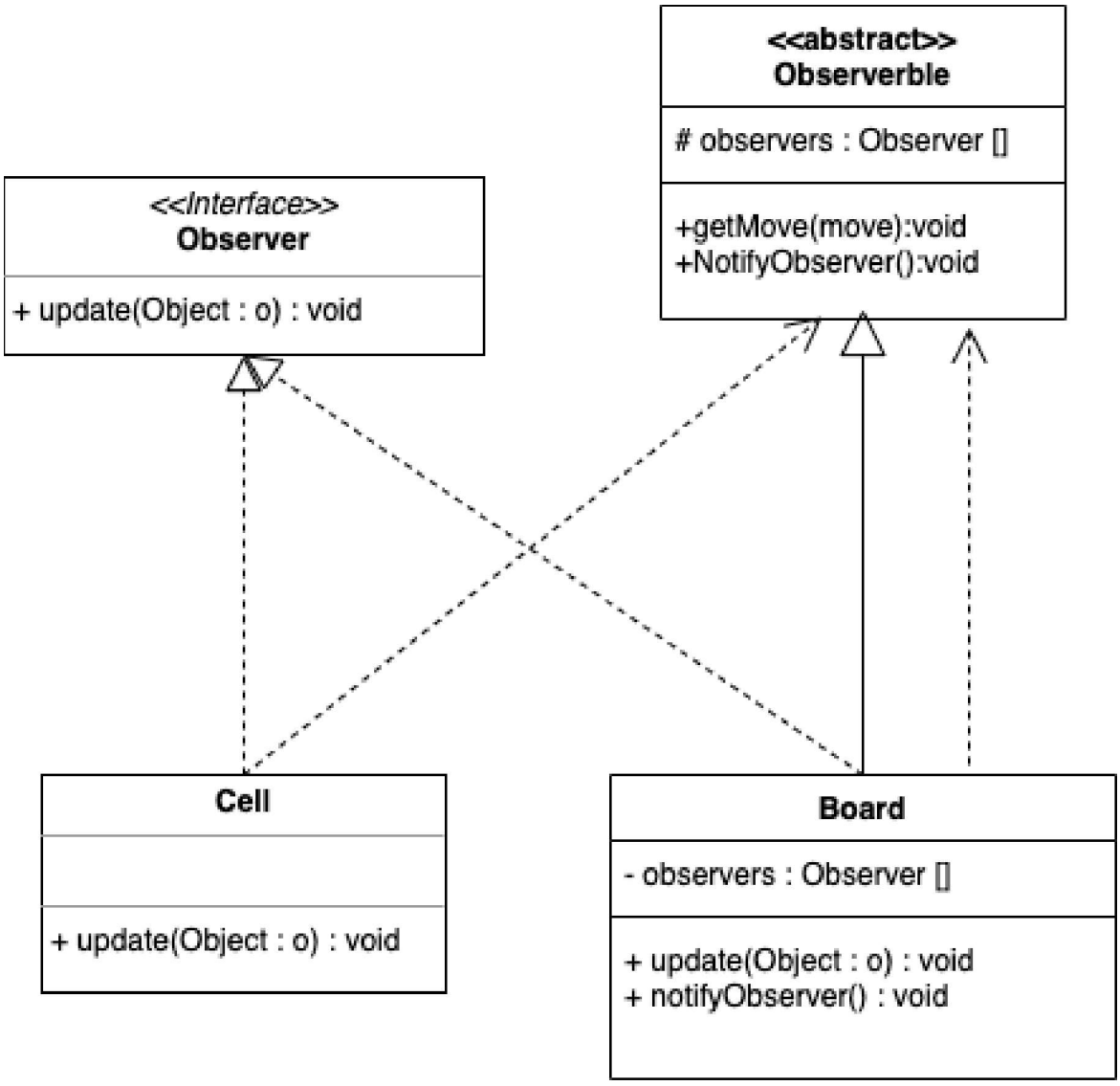
Stratergy Pattern

The strategy pattern used when the threemusketeer.java calls the agent for a move.The agent class composite the interface MoveBehaviour and the child class call the getMove method in the MoveBehaviour respectively. The interface MoveBehaviour then will generate a move for the given agent. The pattern follows Open Closed Principle and Interface Segregation Principle. which open for extension and design different interfaces for different behaviour.



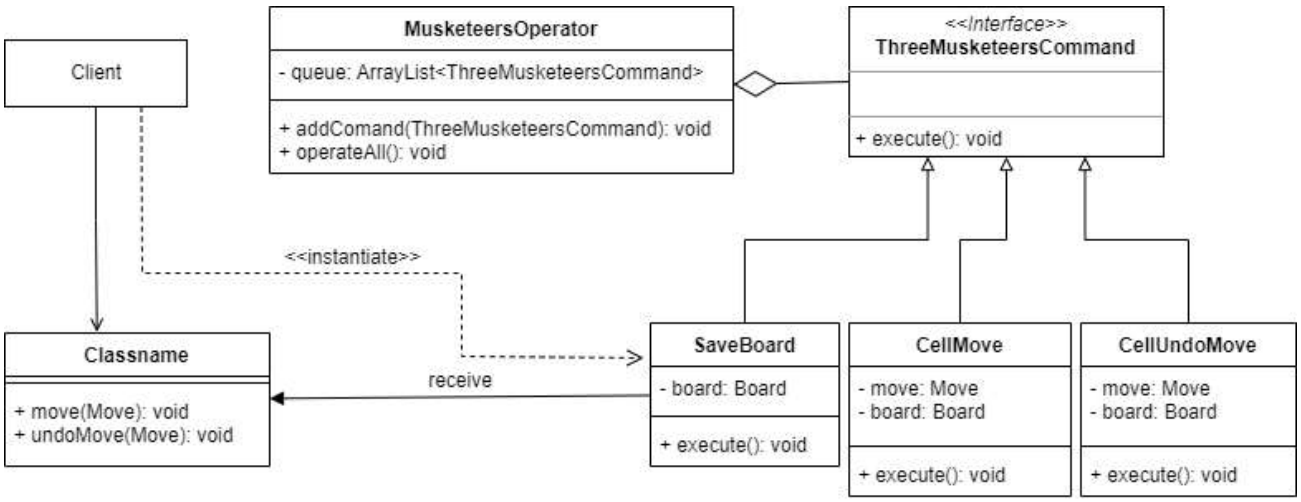
Observer Pattern

Reduces the coupling between the target and the observer, which are abstractly coupled. It conforms to the principle of dependency inversion. At the same time, a trigger mechanism is established between the target and the observer.



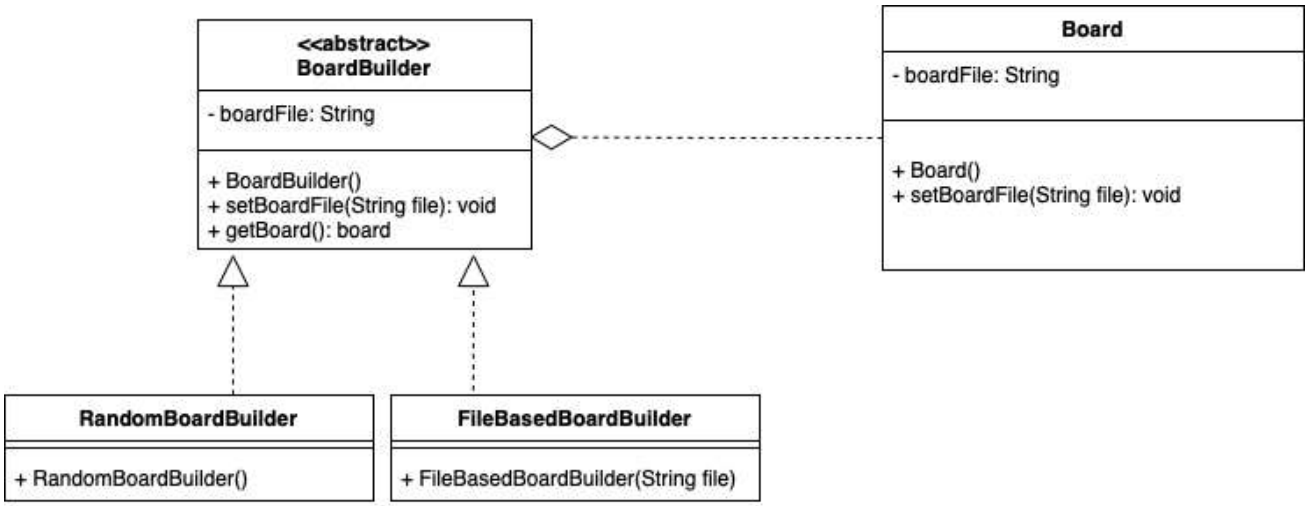
Command Pattern

The Command Pattern unifies all inputs/commands by an interface for execution, and stores them in the same collection. In the class ThreeMusketeers, it combines generating a game and asking player's input in the same class. This violates the Single Responsibility Principle of OOP design. To make the codes more orderly and increase the readability, we use the Command Pattern to manage the user's input/command.



Builder Pattern

The Builder Pattern is implemented by creating a Board class with the boardFile attributes and different Builder classes which contain a getBoard() method to create a Board object. By using Builder Pattern, we can create different boards based on the user's needs. For example, creating a board based on a file, or generating a new random board.



Simple Factor Pattern

Simple factory pattern will be invoked when Cell starts giving Pieces, CreateCell class will generate Pieces based on the board string. creating Pieces will not expose the creation logic. Both Demeter's Law and the open closure principle are followed, which helps to reduce the coupling between cells and pieces.