Assignment 2

Exercise 1

Task 1

Where are getters/setters or public variables used in your source code? Refactor nine of these cases, so that they are not necessary anymore and describe your refactoring/redesign.

Description of the refactorings

- [x] Board: remove method getBoard

This getter is unused, thus it can safely be refactored. Was done in ea632f1, before copying the checkers game.

- [x] Board: remove method getAllPieces

This getter is unused, thus it can safely be refactored. Was done in ea632f1, before copying the checkers game.

- [x] Board: move check if it's the last row to Row enum

The row enum can determine itself if it's the last row. This way we can hide that the coordinates are implemented with an enum.

- [x] Board: move check if it's the firstrow to Row enum

The row enum can determine itself if it's the first row. This way we can hide that the coordinates are implemented with an enum.

- [x] Move:getCoordinatesBetween: calculate RowIndexBetween in Row

The Row enum can determine itself which row is between itself and another row.

- [x] Move:getCoordinatesBetween: calculate ColIndexBetween in Column

The Column enum can determine itself which column is between itself and another column.

- [x] Move:isJumpMove: calculate row distance in Row enum.

We can calculate the distance between two Rows in the Row enum.

- [x] Move:isJumpMove: calculate the column distance in the Column enum.

We can calculate the distance between two Columns in the Column enum.

- [x] MoveLength: Use the methods in Row and Column to calculate the difference

Exercise 2

Task 1

Google (used to?) ask their employees to spend 20% of their time at Google on a project that their job description does not cover. As a result of the 20% Project, Google now has services such as Gmail and AdSense. This is your occasion to have similar freedom. You can decide what to do next to your game:5 It can be an extension/improvement from any perspective, such as improved code quality or novel features. Define your own requirements and get them approved by your tutor (especially in terms of load). Afterwards you must implement the requirements.

Description of the requirements

We decided to implement the following novel feature in our checkers game: Before every jump move the current player has the option to toss a coin. If the coin lands on heads the current player loses its piece and his move is skipped. If its heads, the players jump move is executed, and he gets to move again.

Additionally, we will fix the bug that the inputs with square brackets ([a3]x[b4]) are accepted, and that inputs with an uppercase X (a3Xb4, [a3]X[b4]) are accepted too. We implement that by removing square brackets from the input. Like that, even incomplete inputs ([a3x[b4) are allowed. The uppercase X is implemented by replacing uppercase x with lowercase x in the input. Because the change was very small, we didn't create a design document. All the changes are made in Move::parse in commit 58cb81b.

To improve the readability of the code, we will replace all type inference using the "var" keyword with explicit type definition. cfbbac9

To improve the package structure, we moved the Domain Object Model (Board, BoardCoordinates, Move, Piece, Player) to the separate package dom. 85d69d6

To improve the package structure, we moved the util classes (BoardPrinter, Console) to the separate package util. 5aa0115

Task 2

During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the produced documents)

Responsibility Driven Design

CRC Cards

That it's possible to write unittests, we extract the random generator to the separate class CoinTosser. Then we can mock the result.

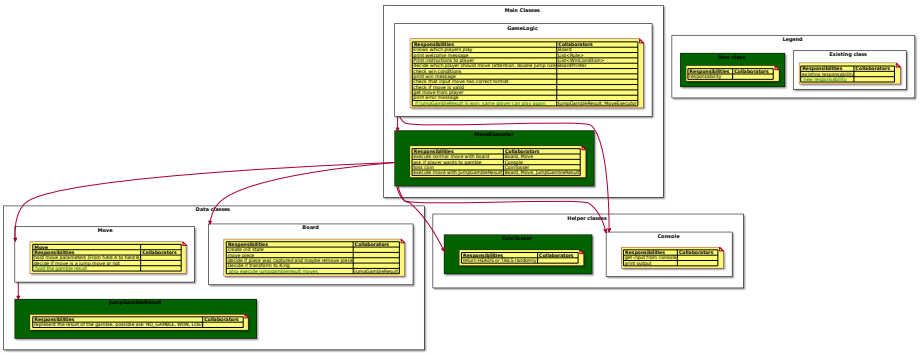
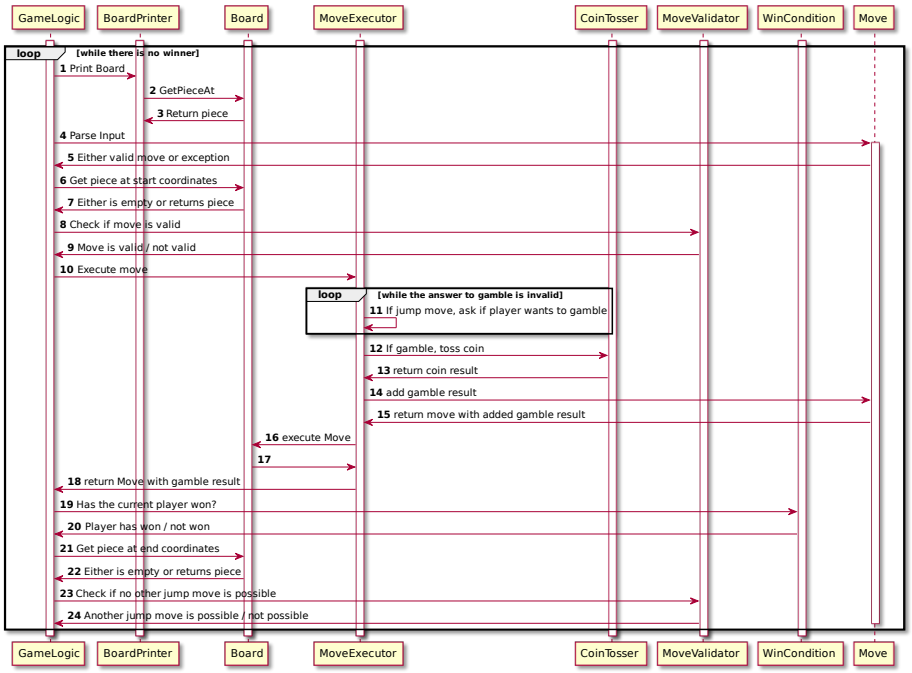
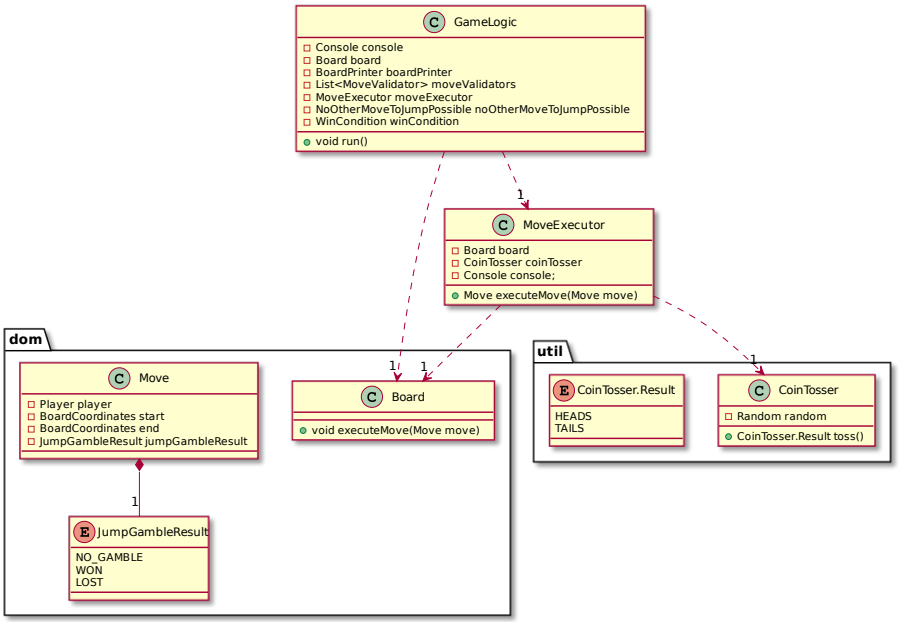


Figure 1: Gamble CRC Cards

UML





Exercise 3

Task 1

Write a natural language description of why and how the pattern is implemented in your code.

Description of why and how the pattern is implemented

Currently, the GameLogic decides when to print the board. But the board could also notify an Observer when it's state changes, and that observer could print the board. That way the GameLogic does not need to depend on BoardPrinter. And the players always want to see the changes made to the board.

Task 2

Make a sequence diagram of how the pattern works dynamically in your code

Sequence diagram

Task 3

Make a class diagram of how the pattern is structured statically in your code

Class diagram

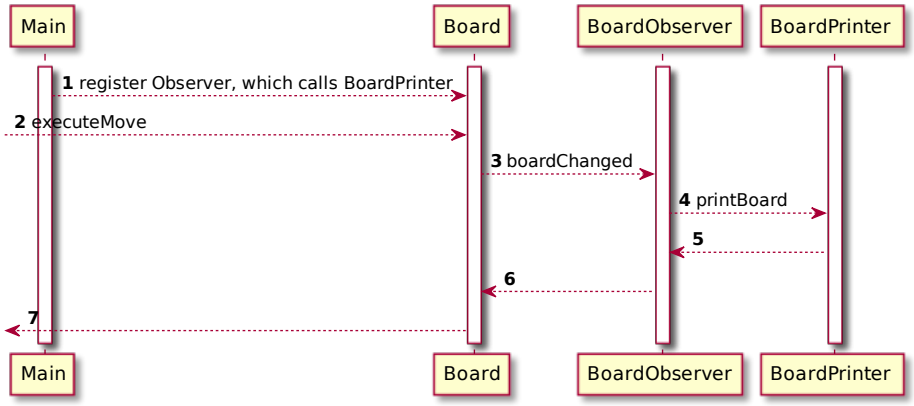


Figure 2: Observer sequence diagram

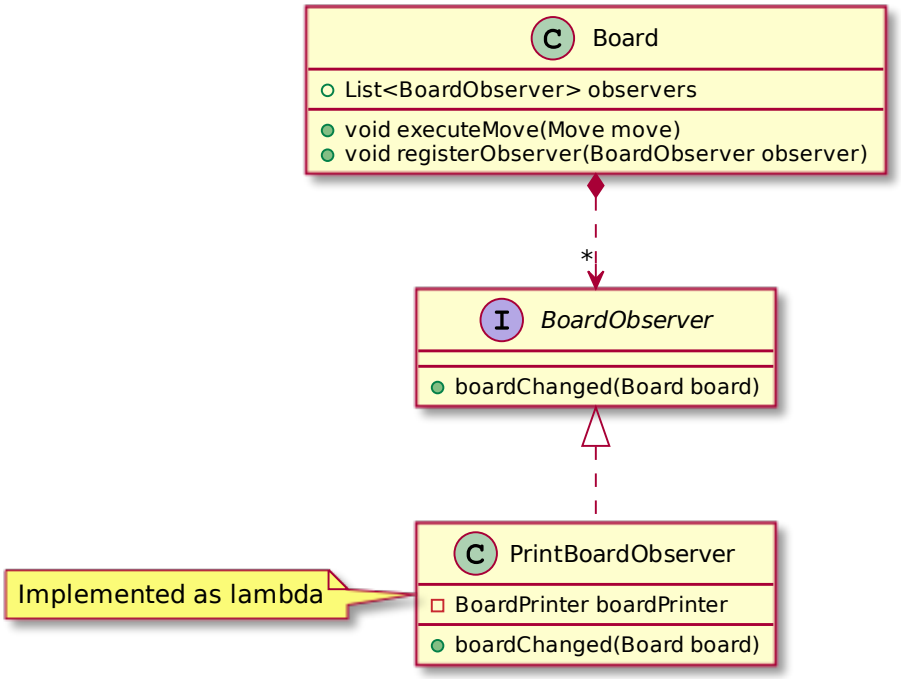


Figure 3: Observer class diagram