Java Programming II

Lab1

514770-1
Fall 2023
9/12/2023

Kyoung Shin Park
Computer Engineering
Dankook University

DRY (Don’t Repeat Yourself) Principle

- In the book “The Pragmatic Programmer”, DRY is defined as “Every piece of **knowledge** must have a single, unambiguous, authoritative representation within a system."
  - Knowledge – a precise functionality or an algorithm
- Violations of DRY
  - WET, "We enjoy typing," or “Waste everyone’s time”.
- How to Achieve DRY
  - To avoid violating the DRY principle, divide your system into pieces. Divide your code and logic into **smaller reusable units** and use that code by calling it where you want.
- DRY Benefits
  - Less code is good: It saves time and effort, is easy to maintain, and also reduces the chances of bugs.

KISS (Keep It Simple Stupid) Principle

- "Keep It Simple Stupid", "Keep It Short and Simple"
- The KISS principle is descriptive to **keep the code simple and clear**, making it **easy to understand**.
- Violations of KISS
  - "Why they have written these unnecessary lines and conditions when we could do the same thing in just 2-3 lines?"
- How to Achieve KISS
  - To avoid violating the KISS principle, try to **write simple code**. Whenever you find lengthy code, divide that into multiple methods — **refactor**.
- KISS Benefits
  - If the code is written simply, then there will not be any difficulty in understanding that code, and also will be easy to modify.

YAGNI (You Aren't Gonna Need It) Principle

- YAGNI says "don't implement something until it is necessary." YAGNI tells us to cut off any unnecessary part while KISS advises to make the rest as simple as possible.
- Violations of YAGNI
  - "over engineering" - a feature for every possible case, functions with a lot of input parameters, multiple if-else branches, rich and detailed interfaces, all those could be a smell of over engineering.
- How to Achieve YAGNI
  - Always **implement things when you actually need them**, never when you just foresee that you need them.
- YAGNI Benefits
  - Software developers don't have enough information to make the call on extra features, the time spent could be used elsewhere more productively. Extra features mean extra development time, testing time, documentation time, code review time.

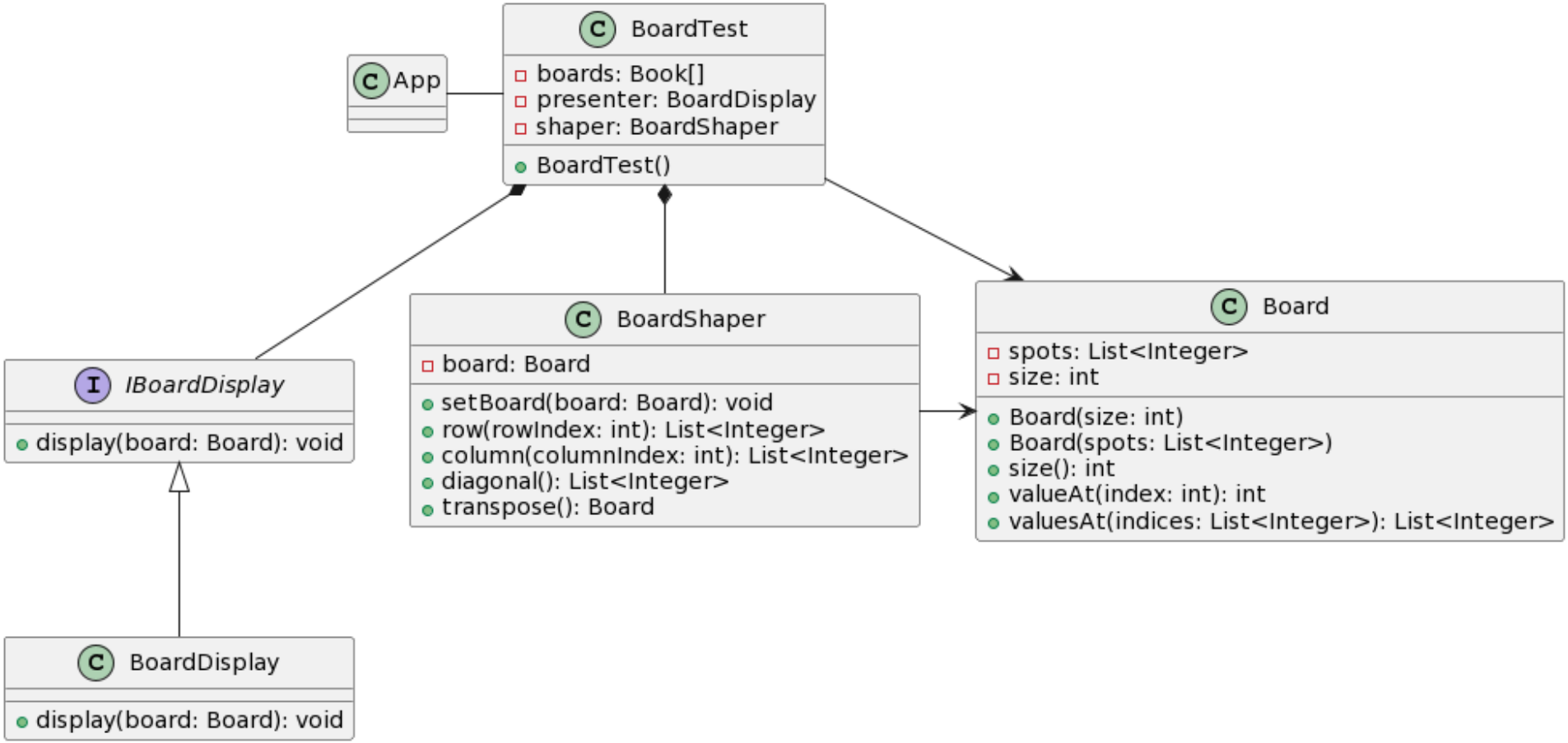
SOLID Principle

- **Single Responsibility Principle**
  - "A class should have one, and only one, reason to change."
- **Open/Closed Principle**
  - "Software entities (e.g. classes, modules, functions, etc) should be open for extension, but closed for modification."
- **Liskov Substitution Principle**
  - "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program."
- **Interface Segregation Principle**
  - "Clients should not be forced to depend upon interfaces that they do not use." Reduce fat interfaces into multiple smaller and more specific client specific interfaces.
- **Dependency Inversion Principle**
  - One should depend on abstractions (interfaces and abstract classes) instead of concrete implementations (classes).

Lab1

- Given a **3 x 3 Board** program (**Board3x3 and Board3x3Test** class), write a **n x n Board** program to display board elements, rows, and columns**.**
- Similar to 3x3 Board program, **BoardTest** class should display **2x2, 3x3, 4x4, 5x5 Board** elements, rows, and columns.
- Write the following classes
  - Board
  - IBoardDisplay
  - BoardDisplay
  - BoardShaper
  - BoardTest

Lab1


BoardTest
boards: Book[]
presenter: BoardDisplay
shaper: BoardShaper
BoardTest()
App
IBoardDisplay
display(board: Board): void
BoardShaper
board: Board
setBoard(board: Board): void
row(rowIndex: int): List<Integer>
column(columnIndex: int): List<Integer>
diagonal(): List<Integer>
transpose(): Board
Board
spots: List<Integer>
size: int
Board(size: int)
Board(spots: List<Integer>)
size(): int
valueAt(index: int): int
valuesAt(indices: List<Integer>): List<Integer>
BoardDisplay
display(board: Board): void

Lab1

- Board
  - List<Integer> spots; int size;
  - Board(int size)
    - Creates ArrayList<Integer> based on size, e.g. if size is 2 => 1,2,3,4
  - Board(List<Integer> spots)
    - Assigns spots and then calculate size, e.g. if 1,2,3,4 => size is 2
  - int size()
    - Returns size
  - int valueAt(int index)
    - Returns this.spots.get(index)
  - List<Integer> valuesAt(List<Integer> indices)
    - Returns List<Integer> values

Lab1

- BoardShaper
  - Board board;
  - setBoard(Board board)
    - Sets this.board to board
  - List<Integer> row(int rowIndex)
    - Returns List<Integer> row
  - List<Integer> column(int columnIndex)
    - Returns List<Integer> column
  - List<Integer> diagonal()
    - Returns List<Integer> diagonal
  - Board transpose()
    - Returns transpose

Lab1

- BoardDisplay implements IBoardDisplay
  - void display(Board board)
    - 1|2
      3|4
    - 1|2|3
      4|5|6
      7|8|9
    - 1|2|3|4
      5|6|7|8
      9|10|11|12
      13|14|15|16
    - 1|2|3|4|5
      6|7|8|9|10
      11|12|13|14|15
      16|17|18|19|20
      21|22|23|24|25

```
public class BoardTest {
  Board[] boards = {new Board(2), new Board(3), new Board(4), new Board(5)};
  IBoardDisplay presenter = new BoardDisplay();
  BoardShaper shaper = new BoardShaper();
  public BoardTest() {
    for (Board board: boards) {
      // board display
      presenter.display(board);
      // set board in a shaper
      shaper.setBoard(board);
      // boardReturnsRow
      for (int index = 1 ; index <= board.size(); index++) {
         System.out.println("row#" + index + " " + shaper.row(index));
      }
      // boardReturnsColumn
      for (int index = 1 ; index <= board.size(); index++) {
         System.out.println("column#" + index + " " + shaper.column(index));
      }
      // boardReturnsDiagonal
      System.out.println("diagonal " + shaper.diagonal());
      // boardReturnsTranspose
      Board transpose = shaper.transpose();
      System.out.println("transpose");
      presenter.display(transpose);
    }
  }
}
```

Submit to e-learning

- Add your code (e.g., additional method, class, routine, etc) in the Lab1 assignment.
- Submit the Lab1 assignment (including the report) to e-learning **(due by 9/18)**.