객체지향개발론및실습

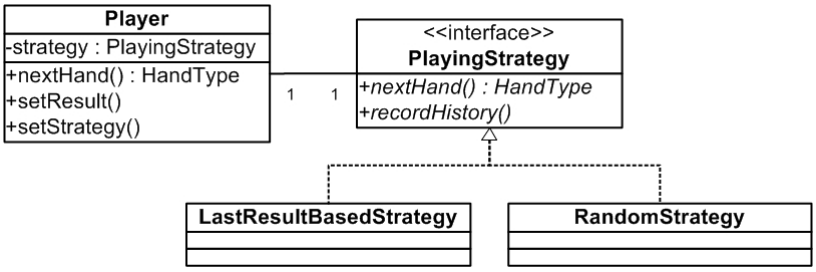
Laboratory 1. Strategy 패턴: 가위바위보 게임

1. 목적

- 알고리즘의 군을 정의하고 캡슐화해주며 서로 언제든지 바꿀 수 있도록 해주는 Strategy 패턴을 실습해본다.

2. 개요

- 사용자와 컴퓨터간에 가위바위보를 하는 게임을 개발하고자 한다.
- 컴퓨터가 가위바위보를 제시하는 다양한 알고리즘을 만들어 난이도에 따라 동적으로 바꾸고자 한다. 실습에서는 두 가지 알고리즘을 개발한다. 하나는 무조건 랜덤으로 가위바위보를 결정하는 전략과 다른 하나는 기존에 이긴 경우를 고려하여 다음에 낼 손을 결정하는 전략이다.
- 이 실습에서는 전략 패턴을 이용하여 컴퓨터가 제시하는 가위바위보 알고리즘을 구현하고자 한다.
- 구현하고자 하는 소프트웨어의 클래스 관계도는 다음과 같다.



<그림 2.1> 가위바위보 게임 클래스 관계도

3. 가위바위보 게임

3.1 ResultType 열거형 클래스

- 게임의 결과를 나타내는 정보를 정의하는 열거형 타입
- 열거형: WON, DRAWN, LOST

3.2 HandType 열거형 클래스

- 가위바위보 게임에서 손의 모양 정보를 정의하는 열거형 타입
- 열거형: GAWI, BAWI, BO
- 연산
  - `public String toString()`
    - 사후조건: 열거형 각 값에 해당하는 한글 문자열(“가위”, “바위”, “보”)를 반환함
  - `public static HandType valueOf(int n)`
    - 사전조건: `n`은 0, 1 또는 2 중 하나이어야 함. 사전조건을 검사하지 않아도 됨
    - 사후조건: 주어진 `n`에 해당되는 열거형 값을 반환함. 즉, 0이면 GAWI, 1이면 BAWI, 2이면 BO를 반환하여야 함

  - `public HandType winValueOf()`
    - 사후조건: 현재 값을 이길 수 있는 열거형 값을 반환함. 즉, GAWI이면 BAWI, BAWI이면 BO, BO이면 GAWI를 반환하여야 함

3.3 Player 클래스

- 가위바위보 게임에서 각 사용자 또는 컴퓨터 역할을 수행하는 클래스임. 설계의 단순화를 위해 상속 계층구조로 사용자와 컴퓨터를 모델링하지 않음
- 구성요소
  - 승, 무, 패 횟수 정보 유지: `wonCount: int`, `drawnCount: int`, `lossCount: int`
  - 게임 수: `gameCount: int`, 초기값: 0
    꼭 이 값을 별도로 유지할 필요가 있을까요?
  - 전략: `strategy: PlayingStrategy`
- 연산
  - 생성자: `public Player(PlayingStrategy strategy)`
    - 사후조건: 주어진 전략으로 내부 상태 strategy를 초기화함
  - `public void setResult(ResultType resultType)`
    - 사후조건: 주어진 값을 이용하여 적절하게 `gameInfo` 배열을 갱신함
  - `public void setStrategy(PlayingStrategy strategy)`
    - 사후조건: 주어진 전략으로 내부 상태 strategy를 변경함
  - `public HandType nextHand()`
    - 사후조건: strategy의 nextHand를 호출하여 다음 가위바위보 값을 반환함
  - `public String toString()`
    - 사후조건: 현재 게임 진행 상태를 보여줌(게임수, 승수, 패수)

3.4 PlayingStrategy 인터페이스

- 컴퓨터 플레이어가 다음 손 모양을 결정하기 위한 알고리즘을 구현하는 클래스들이 공통적으로 가져야 하는 인터페이스를 정의함
- 연산
  - `HandType nextHand()`: 다음 가위바위보 값을 결정하여 주는 메소드
  - `void recordHistory(ResultType currResult)`: 다음 가위바위보 값을 결정하기 위해 보조하는 메소드로 지난 게임의 결과(승, 무, 패 여부)를 인자로 전달받음

3.5 RandomStrategy 클래스

- 무조건 랜덤으로 다음에 낼 손을 결정하는 전략을 구현하는 클래스로서, `PlayingStrategy` 인터페이스를 구현한다.
- 구성요소
  - 랜덤값을 생성하기 위한 요소: `randomGen: java.util.Random`
- 연산
  - `public HandType nextHand()`
    - 사후조건: 랜덤 수를 생성하여 그 값에 따라 다음 낼 손을 생성하여 반환함.
      `HandType`의 `valueOf(int)` 메소드를 활용하면 편리함
  - `public void recordHistory(ResultType currResult)`
    - 빈 메소드로 구현함. 이 전략에서 필요없음

3.6 LastResultBasedStrategy 클래스

- 지난 게임의 결과가 사용자 결정에 많은 영향을 준다는 가정하에 만든 전략임.
  이 전략 클래스도 PlayingStrategy 인터페이스를 구현하도록 정의해야 함. 이 클래스의 알고리즘은 지난 게임의 결과에 따라 다음과 같이 행동하도록 구현해야 함.
  - 이긴 경우: 같은 손을 다시 내지 않을 것이라고 가정하고 그것을 제외한 나머지 두 종류 중 하나를 랜덤하게 결정함
  - 비긴 경우: 이긴 경우와 동일하게 처리함
  - 진 경우: 상대방이 같은 손을 낼 것이라고 가정하고 그것을 이길 수 있는 손으로 결정함
- 구성요소 (5점)
  - 랜덤값을 생성하기 위한 요소: `randomGen: java.util.Random`, 명백한 초기화를 함
  - 사용한 이전 손 모양: `prevHand: HandType`
  - 이전 게임의 결과: `prevResult: ResultType`
  - `prevHand`와 `prevResult`의 초기값은 스스로 고민하여 결정해야 함
- 연산
  - `public HandType nextHand()`
    - 사후조건: `prevHand`와 `prevResult`를 이용하여 이 전략에 맞게 손을 결정하여 반환하여야 하며, 최종 반환된 값은 다음 게임을 위해 `prevHand`에 대입되어야 함
  - `public void recordHistory(ResultType currResult)`
    - `currResult`: 지난 게임의 승,무,패 결과
    - 사전조건: `result`가 **null**이 아니어야 함. 이 사전조건은 고려할 필요없음
    - 사후조건: `prevResult`에 주어진 인자를 대입함

4. 실습내용

- 위에 제시된 6개 클래스 또는 인터페이스를 모두 구현하고 다음 테스트 프로그램을 통해 구현 결과를 확인하시오. 처음 테스트하는 과정에서는 반복 횟수 10 이하로 설정하여 결과를 확인해 보는 것이 필요합니다.

```
public class Test {
    public static void main(String[] args) {
        Player player1 = new Player(new RandomStrategy());
        Player player2 = new Player(new LastResultBasedStrategy());
        for(int i=0; i<100; i++){
            HandType h1 = player1.nextHand();
            HandType h2 = player2.nextHand();
            System.out.printf("사용자1: %s VS 사용자2: %s", h1, h2);
            if(h1==h2){
                System.out.println(" > 결과: 무승부");
                player1.setResult(ResultType.DRAWN);
                player2.setResult(ResultType.DRAWN);
            }
            else if(h1.winValueOf()==h2){
                System.out.println(" > 결과: 사용자2 승");
                player1.setResult(ResultType.LOST);
                player2.setResult(ResultType.WON);
            }
            else{
                System.out.println(" > 결과: 사용자1 승");
                player1.setResult(ResultType.WON);
                player2.setResult(ResultType.LOST);
            }
        }
```

```
        System.out.println(player1);
        System.out.println(player2);
    }
}
```

5. 숙제

- 실습에 제시된 2개의 전략 외에 개인적으로 새로운 알고리즘을 구현하여 제출하시오. 이 때 그와 같은 알고리즘을 고안한 배경을 상세히 설명한 자료와 함께 제출하시오.