Strategy Pattern

알고리즘(전략)을 전부 교체해서 수정하기 쉽도록 만들기

Architecture, Modeling, DesignPattern, Behavioral

- 시나리오 : 가위바위보 게임

Strategy

```
public class Hand {
    public static final int HANDVALUE_ROCK = 0;
    public static final int HANDVALUE_SCISSORS = 1;
    public static final int HANDVALUE_PAPER = 2;
    private static final Hand[] hand = {
        new Hand(HANDVALUE_ROCK),
        new Hand(HANDVALUE_SCISSORS),
        new Hand(HANDVALUE_PAPER)
    };
    private static final String[] name = {
     "주먹","가위","보"
    };
    private int handValue;

    private Hand(int handValue) {
        this.handValue = handValue;
    }
    public static Hand getHand(int handValue) {
        return hand[handValue];
    }

    public boolean isStgrongerThan(Hand h) {
        return fight(h) == 1;
    }
    public boolean isWeakerThan(Hand h) {
        return fight(h) == -1;
    }
    private int fight(Hand h) {
        if(this == h)
            return 0;
        else if((this.handValue + 1) % 3 == h.handValue)
            return 1;
        else
            return -1;
    }
    @Override
    public String toString() {
```

```
        return name[handValue];
    }
}
```

```
public interface Strategy {
    Hand nextHand();
    void study(boolean win);
}
```

```
/* 이긴손 다시 내기 */
public class WinningStrategy implements Strategy {

    private Random random;
    private boolean won = false;
    private Hand prevHand;
    public WinningStrategy() {
        this.random = new Random();
    }
    @Override
    public Hand nextHand() {
        if(!won)
            prevHand = Hand.getHand(random.nextInt(3));
        return prevHand;
    }

    @Override
    public void study(boolean win) {
        won = win;
    }
}
```

```
/* 과거 승패 이력으로 확률적으로 바꾸기 */
public class ProbStrategy implements Strategy {
    private Random random;
    private int prevHandValue;
    private int currentHandValue;
    private int[][] history = {
            {1, 1, 1},
            {1, 1, 1},
            {1, 1, 1}
    };
    public ProbStrategy() {
        this.random = new Random();
```

```
    }

    @Override
    public Hand nextHand() {
        int bet = random.nextInt(getSum(currentHandValue));
        int handValue = 0;
        if(bet < history[currentHandValue][0])
            handValue = 0;
        else if(bet < history[currentHandValue][0] +
history[currentHandValue][1])
            handValue = 1;
        else
            handValue = 2;
        prevHandValue = currentHandValue;
        currentHandValue = handValue;
        return Hand.getHand(handValue);
    }

    private int getSum(int hv) {
        return IntStream.range(0, 3).map(i -> history[hv][i]).sum();
    }

    @Override
    public void study(boolean win) {
        if(win)
            history[prevHandValue][currentHandValue]++;
        else {
            history[prevHandValue][(currentHandValue + 1) % 3]++;
            history[prevHandValue][(currentHandValue + 2) % 3]++;
        }
    }
}
```

Context

Strategy를 이용하는 역할.

```
public class Player {
    private String name;
    private Strategy strategy; // 위임!

    private int winCount;
    private int loseCount;
    private int gameCount;
    public Player(String name, Strategy strategy) {
        super();
        this.name = name;
        this.strategy = strategy;
```

```
    }
    public Hand nextHand() {
        return strategy.nextHand();
    }
    public void win() {
        strategy.study(true);
        winCount++;
        gameCount++;
    }
    public void lose() {
        strategy.study(true);
        loseCount++;
        gameCount++;
    }
    public void even() {
        gameCount++;
    }

    @Override
    public String toString() {
        return "Player [name=" + name + ", winCount=" + winCount + ",
loseCount=" + loseCount
                + ", gameCount=" + gameCount + "]";
    }
}
```

Client

```
public class Main {
    public static void main(String[] args) {
        Player player1 = new Player("철수", new WinningStrategy());
        Player player2 = new Player("영희", new ProbStrategy());
        IntStream.range(0, 10000).forEach(i -> {
            Hand nextHand1 = player1.nextHand();
            Hand nextHand2 = player2.nextHand();
            if(nextHand1.isStgrongerThan(nextHand2)) {
                System.out.println("Winner : " + player1.toString());
                player1.win();
                player2.lose();
            }
            else if(nextHand2.isStgrongerThan(nextHand1)) {
                System.out.println("Winner : " + player2.toString());
                player1.lose();
                player2.win();
            }
            else {
                System.out.println("Even!");
```

```
                player1.even();
                player2.even();
            }
        });
        System.out.println("\nTotal result:");
        System.out.println(player1.toString());
        System.out.println(player2.toString());
    }
}

/*
...
Winner : Player [name=영희, winCount=7843, loseCount=2135, gameCount=9997]
Winner : Player [name=영희, winCount=7844, loseCount=2135, gameCount=9998]
Winner : Player [name=영희, winCount=7845, loseCount=2135, gameCount=9999]

Total result:
Player [name=철수, winCount=2135, loseCount=7846, gameCount=10000]
Player [name=영희, winCount=7846, loseCount=2135, gameCount=10000]
*/
```

From:
https://wiki.ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://wiki.ledyx.xyz/doku.php?id=design_pattern:strategy_pattern&rev=1509274534

Last update: **2026/10/24 07:03**