©2017, sangjin@koreatech.ac.kr

객체지향개발론및실습

Laboratory 6. Adapter, Template Method 패턴

1. 목적

- 한 클래스의 인터페이스를 클라이언트가 기대하는 인터페이스로 바꾸어 주는 Adapter 패턴을 실습 해본다.
- 알고리즘의 구조를 변경하지 않고 하위 클래스에서 알고리즘의 일부 단계를 재정의할 수 있도록 해주는 Template Method 패턴을 실습 해본다.

2. 실습 1. FullNameAdapter

- 사용자 정보를 다음과 같이 유지하는 User 클래스가 있다고 가정하자.

```
public class User {
    private String firstName;
    private String lastName;
    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
     return lastName;
    }
}
```

- User 클래스는 직접적으로 수정이 가능하지 않은 상태라고 가정하자.
- User 객체를 활용하는 클라이언트에서는 getFullName이라는 메소드의 사용이 필요하다.

```
public interface FullName {
    String getFullName();
}
```

- 위 **interface**를 구현하는 User 클래스를 위한 FullNameAdapter를 객체 어댑터 방식과 클래스 어댑터 방식으로 각각 구현하시오. 각각의 클래스 이름을 FullNameAdapter1, FullNameAdapter2라고 하시오. 이 때 getFullName은 성이 "김", 이름이 "상진"일 때 "김 상진"을 반환해야 함
- User를 상속받는 SuperUser 클래스가 있다고 하자. 그러면 위 두 개의 어댑터 중에 어떤 어댑터는 수정없이 SuperUser에 대해서 사용 가능한가?

3. 실습 2. 고객 정보 출력

- 다음과 같은 Customer 클래스와 고객 정보를 출력해주는 SimpleReportGenerator라는 클래스가 있다고 가정하자.

```
public class Customer {
    private String name;
    private int point;
```

```
    public Customer(String name, int point) {
        this.name = name;
        this.point = point;
    }
    public String getName() {
        return name;
    }
    public int getPoint() {
        return point;
    }
}

public class SimpleReportGenerator {
    public String generate(List<Customer> customers){
        String report = String.format("고객 수: %d명%n", customers.size());
        report = customers.stream()
            .map(c->String.format("%s : %d%n", c.getName(), c.getPoint()))
            .reduce(report,String::concat);
        return report;
    }
}
```

- 만약 고객 점수가 200점 이상인 고객 대상으로 다음 보고서를 생성하는 ComplexReportGenerator 클래스를 완성하시오.

```
고객 수: 2명
200: 김유신
250: 장보고
합계: 450
```

```
public class ComplexReportGenerator {
    public String generate(List<Customer> customers){
    }
}
```

- Template Method 패턴을 사용하여 코드 중복을 최소화 하시오. ReportGenerator 추상 클래스를 정의하고, 각 클래스가 ReportGenerator를 상속받도록 하시오.

4. 실습 3. Card Game

- 모든 카드 게임은 동일하게 52장의 카드를 가지고 진행한다. 종류에 따라 각 사용자에게 나누어주는 카드의 개수가 다르지만 이 실습에서 생각하는 모든 카드 게임은 동일한 수의 카드를 각 사용자에게 나누어준다.
- 이를 위해 CardGame이라는 추상 클래스를 정의하고, 초기 게임 설정 과정을 template method 패턴을 사용하여 다음과 같이 만들고자 한다.

```
public abstract class CardGame {
    protected ArrayList<Card> originalDeck = new ArrayList<>();
    protected Queue<Card> remainingDeck = new ArrayDeque<>();
    protected ArrayList<ArrayList<Card>> userCards;
    public final void init(){
        CardFace[] cardFaces = CardFace.values();
        for(int i=0; i<originalDeck.length; i++){
            originalDeck.add(new Card( i % 13 + 1, cardFaces[i/13]));
        }
        shuffle();
        remainingDeck.addAll(originalDeck);
        dealCards();
```

```
    }
    protected void shuffle(){
    }
    protected abstract void dealCards();
}
```

- 여기서 remainingDeck는 사용자에게 주고 남은 카드들임.
- shuffle 메소드는 hook 메소드로 하위 클래스에서 필요하면 shuffle 방법을 새롭게 구현할 수 있음
- dealCards는 추상 메소드로 각 하위 클래스에서 반드시 재정의해야 하는 메소드임
- CardGame의 shuffle 메소드를 구현하시오. 라이브러리 함수를 이용하여도 됨
- CardGame을 상속받는 BlackJackGame을 정의하시오. 이 때 생성자는 플레이어 수를 받아야 하며, dealCards에서는 해당 수만큼의 플레이어가 분배할 카드를 나누어 주어야 함. 참고로 BlackJack은 각 플레이어에게 2장의 카드를 나누어 줌