REPORT

HW4

자바프로그래밍2

제출일 2023. 10.14

소속 컴퓨터공학과

학번 32183520

이름 이 주성

Decorator Pattern

데코레이터 패턴은 기능을 확장할 때 상속이나 subclassing을 사용하지 않고 동적으로 추가적인 기능을 객체에 부여할 수 있는 패턴이다. 새로운 기능이 필요할 때마다 새로운 데코레이터를 만들고 여러 데코레이터를 중첩해 기능을 확장할 수 있다. 이렇게 하면 기존 클래스를 수정하지 않고도 새로운 기능을 추가할 수 있기 때문에 OCP원칙을 준수할 수 있다.

과제에 데코레이터 패턴을 적용해보면 원소의 속성은 number, name 말고도 group, type, period 등 많이 있다. 현재 number, name 속성만 가진 Element 리스트가 있다면 여기에 데코레이터 패턴을 적용해 필요한 속성을 붙여줄 것이다.

이렇게 데코레이터 패턴을 적용하면 기존의 Element 클래스를 변경하지 않고 속성을 원하는만큼 추가할 수 있게 된다.

Component

- **IElement**
    - 공통 인터페이스
    - Decorator 및 Concrete Component로 구현됨

```
public interface IElement {
   String getDescription();
}
```

Concrete Component

- **Element**
    - IElement 인터페이스 구현
    - 기본 동작인 number, name 출력하는 getDescription() 매소드 제공

```
// Element를 구현한 Concrete Component
```

```java
public class Element implements IElement {
    // 원소의 번호와 이름만 보유
    // 나머지 속성은 데코레이터로 추가할 예정
    private int number;
    private String name;

    // 생성자: number, name 파라미터
    public Element(int number, String name) {
        this.number = number;
        this.name = name;
    }

    // number,name 형식으로 반환
    @Override
    public String getDescription() {
        return number + "," + name;
    }

}
```

Decorator

- **ElementDecorator**
    - IElement 인터페이스 구현
    - IElement 참조 보유

```java
public abstract class ElementDecorator implements IElement {
    // Component인 IElement 참조
    protected IElement decoratedElement;

    // Decorator로 래핑할 Component를 파라미터로 받아 생성
    public ElementDecorator(IElement element) {
        this.decoratedElement = element;
    }
}
```

Concrete Decorators

- **SymbolDecorator**

- symbol 정보와 장식할 IElement를 파라미터로 받아 생성자에 넣어줌

```
public class SymbolDecorator extends ElementDecorator {
    private String symbol;

    // symbol 정보와 IElement를 받아 생성
    public SymbolDecorator(String symbol, IElement element) {
        super(element);
        this.symbol = symbol;
    }

    // 오버라이딩으로 기능(설명) 확장
    @Override
    public String getDescription() {
        return this.decoratedElement.getDescription() + "," +
symbol;
    }
}
```

- 나머지 **GroupDecorator, PeriodDecorator, PhaseDecorator, TypeDecorator, WeightDecorator** 데코레이터 모두 동일한 형식

MainTest

1. PeriodicElement.csv 파일을 load해서 PeriodicElement 리스트로 저장

```
// PeriodicElements.csv 파일을 load해서 PeriodicElement 리스트로
저장
List<PeriodicElement> peList =
PeriodicElementImporter.loadCSV("PeriodicElements.csv");
```

2. PeriodicElement 리스트를 하나씩 돌면서 번호와 이름만 담긴 Element 리스트로 변환

```
// peList를 for 문으로 돌면서 번호와 이름으로 Element를 생성한 후
elementList에 add
List<Element> eList = new ArrayList<>();
for (PeriodicElement pe : peList) {
   eList.add(new Element(pe.getNumber(), pe.getName()));
}
```

3. Element 리스트를 하나씩 돌면서 데코레이터들로 장식 후 IElement 리스트로 추가

```
// Element를 하나씩 뽑아서 데코레이터들로 장식 후 IElement 리스트에
추가
List<IElement> weightAndSymbolDecoList = new ArrayList<>();
List<IElement> allDecoList = new ArrayList<>();
for (int i = 0; i < eList.size(); i++) {
   PeriodicElement pe = peList.get(i);
   Element e = eList.get(i);

   // WeightDecorator와 SymbolDecorator로 장식 후 리스트에 추가
   IElement ie1 = new SymbolDecorator(pe.getSymbol(), new
WeightDecorator(pe.getWeight(), e));
   weightAndSymbolDecoList.add(ie1);

   // 모든 데코레이터를 사용해 장식 후 리스트에 추가
   // 순서는 SymbolDecorator, WeightDecorator, Periodecorator,
GroupDecorator, PhaseDecorator, TypeDecorator 순
   IElement ie2 = new TypeDecorator(pe.getType(),
           new PhaseDecorator(pe.getPhase(),
                   new GroupDecorator(pe.getGroup(),
                           new PeriodDecorator(pe.getPeriod(),
                                   new
WeightDecorator(pe.getWeight(),
                                           new
SymbolDecorator(pe.getSymbol(), e))))));
   allDecoList.add(ie2);
}
```

4. IElement 리스트를 CSV 파일로 저장

```
// CSV 파일로 저장
PeriodicElementImporter.saveCSV("Elements1.csv",
weightAndSymbolDecoList, "#AtomicNumber,Element,AtomicMass,Symbol");
PeriodicElementImporter.saveCSV("Elements2.csv", allDecoList,
"#AtomicNumber,Element,Symbol,AtomicMass,Periodic,Group,Phase,Type");
```

실행결과

```
file import: PeriodicElements.csv
line contains #: #AtomicNumber,Element,Symbol,AtomicMass,Period,Group,Phase,
load successfully

file save: Elements1.csv
file save: Elements2.csv
```

Elements1.csv

```
#AtomicNumber,Element,AtomicMass,Symbol
1,Hydrogen,1.007,H
2,Helium,4.002,He
3,Lithium,6.941,Li
4,Beryllium,9.012,Be
5,Boron,10.811,B
6,Carbon,12.011,C
7,Nitrogen,14.007,N
8,Oxygen,15.999,O
```

Elements2.csv

```
#AtomicNumber,Element,Symbol,AtomicMass,Periodic,
1,Hydrogen,H,1.007,1,1,gas,Nonmetal
2,Helium,He,4.002,1,18,gas,Noble Gas
3,Lithium,Li,6.941,2,1,solid,Alkali Metal
4,Beryllium,Be,9.012,2,2,solid,Alkaline Earth Meta
5,Boron,B,10.811,2,13,solid,Metalloid
6,Carbon,C,12.011,2,14,solid,Nonmetal
7,Nitrogen,N,14.007,2,15,gas,Nonmetal
8,Oxygen,O,15.999,2,16,gas,Nonmetal
```

개인적으로 추가한 코드

필요한 데코레이터를 입력으로 받아 동적으로 원하는 속성 정보가 담긴 엑셀 파일을 저장할 수 있도록 기능을 추가해봤습니다.

1. 필요한 데코레이터 입력받기

InputStreamReader는 바이트 입력 스트림을 문자입력 스트림으로 변환해준다. 여기에 데코레이터 패턴을 적용해 BufferedReader 데코레이터로 기능을 확장한다. BufferReader는 InputStreamReader에서 읽은 문자 데이터를 버퍼링해 읽기 성능을 향상시킨다.

```
public static List<Decorator> getDecorators() {
    // InputStreamReader를 BufferedReader로 데코레이트
    // 사용자 입력으로 받은 문자 데이터를 버퍼링해 읽기 기능 향상
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

    // 입력받은 데코레이터 리스트
    List<Decorator> decorators = new ArrayList<>();

    //...
        // 공백을 기준으로 입력받은 문자열 분리
        StringTokenizer st = new StringTokenizer(br.readLine(), "
");

        while (st.hasMoreTokens()) {
            String d = st.nextToken();

            //...
        }
    //...

    return decorators;
}
```

2. 입력받은 데코레이터 리스트를 이용해 입력받은 순서대로 데코레이트를 전부 해준 후 csv 파일로 저장하기

```
private void custumList(List<Element> eList,
List<PeriodicElement> peList) {
    // 원하는 데코레이터 입력받기
    List<Decorator> decorators = UserInput.getDecorators();

    // 커스텀 원소 리스트
    List<IElement> customList = new ArrayList<>();

    // Element를 하나씩 뽑아서 데코레이터들로 장식 후 IElement 리스트에
추가
    for (int i = 0; i < eList.size(); i++) {
        PeriodicElement pe = peList.get(i);
        Element e = eList.get(i);

        // IElement 장식 과정
        IElement ie = new Element(pe.getNumber(), pe.getName());
        for (Decorator d : decorators) {
            switch (d) {
                case SymbolDecorator:
                    ie = new SymbolDecorator(pe.getSymbol(), ie);
                    break;
                case WeightDecorator:
                    ie = new WeightDecorator(pe.getWeight(), ie);
                    break;
                case TypeDecorator:
                    ie = new TypeDecorator(pe.getType(), ie);
                    break;
                //...
            }
        }

        // 장식한 IElement 리스트에 추가
        customList.add(ie);
```

```
    }


    // firstLine 생성
    String firstLine = decorators.stream()
            .map(decorator ->
decorator.toString().replace("Decorator", ""))
            .collect(Collectors.joining(","));

    // csv 파일로 저장
    PeriodicElementImporter.saveCSV("CustomList.csv", customList,
"#AtomicNumber,Element," + firstLine);

}
```

실행결과

```
데코레이터 종류:
1. SymbolDecorator
2. WeightDecorator
3. PeriodDecorator
4. GroupDecorator
5. PhaseDecorator
6. TypeDecorator
추가하고 싶은 순서대로 데코레이터의 번호를 공백을 기준으로 입력해주세요: 6 5 4
```

```
#AtomicNumber,Element,Type,Phase,Group
1,Hydrogen,Nonmetal,gas,1
2,Helium,Noble Gas,gas,18
3,Lithium,Alkali Metal,solid,1
4,Beryllium,Alkaline Earth Metal,solid,2
5,Boron,Metalloid,solid,13
6,Carbon,Nonmetal,solid,14
7,Nitrogen,Nonmetal,gas,15
```