REPORT

자바 프로그래밍2 1분반 Lab5

제출일 : 2023.10.30
소속 : 소프트웨어학과
학번 : 32201817
이름 : 박정운

1. Element

```
public abstract class Element {
    protected int number;
    protected String name;
    protected String symbol;
    protected double weight;
    //builder를 이용한 생성자 -> Element return
    protected Element(ElementBuilder builder){
        this.number = builder.number;
        this.name = builder.name;
        this.symbol = builder.symbol;
        this.weight = builder.weight;
    }
    //getter
    public int getNumber(){ return number; }
    public String getName(){ return name; }
    public String getSymbol(){ return symbol; }
    public double getDouble(){ return weight; }
```

Abstract class인 Element는 다음과 같이 구성됩니다. number, name, symbol, weight 멤버 필드를 가지며 이 필드를 반환하는 각각의 getter가 있습니다. Element의 constructor는 inner class인 ElementBuilder에 의해 setting 됩니다.

2. ElementBuilder

```
//추상 inner class - ElementBuilder
public abstract static class ElementBuilder{
    protected int number;
    protected String name;
    protected String symbol;
    protected double weight;

    //필수 요소(number, name)을 포함한 생성자
    public ElementBuilder(int number, String name){
        this.number =  number;
        this.name = name;
    }

    //setter -> ElementBuilder return
    public ElementBuilder setSymbol(String symbol){
        this.symbol = symbol;
        return this;
    }
    public ElementBuilder setWeight(double weight){
        this.weight = weight;
        return this;
    }

    //abstract method(하위 Builder에 의해 구현) - build()
    public abstract Element build();
}
```

ElementBuilder class는 Element 내부에 static으로 구현된 inner class이며, Element와 동일한 멤버 필드를 가집니다. Element는 기본적으로 number, name을 필수값으로 가지고 있으며 나머지 필드들은 선택적으로 입력되므로 생성자에서 number, name을 setting 합니다. 그리고 추가적인 필드들은 setter를 이용해 setting한 후 ElementBuilder를 return 합니다.

3. GasElement

```java
public class GasElement extends Element {
    private Phase phase;
    //gas Builder를 통한 생성자
    protected GasElement(GasElementBuilder builder) {
        super(builder);
        this.phase = builder.phase;
    }
    //getter
    public Phase getPhase(){ return phase;}
    //GasElementBuilder(extends ElementBuilder)
    public static class GasElementBuilder extends ElementBuilder{
        private Phase phase;
        //필수값(number, name)을 포함한 builder 생성자
        public GasElementBuilder(int number, String name) {
            super(number, name);
        }
        //phase set -> elementBuilder return
        public GasElementBuilder setPhase(Phase phase){
            this.phase = phase;
            return this;
        }
        //return GasElement
        @Override
        public Element build() {
            return new GasElement(this);
        }
    }
}
```

GasElement와 GasElementBuilder는 각각 Element와 ElementBuilder를 상속받아 구체화 한 구현체입니다. Element엔 없는 phase 필드를 추가하고, 이에 대한 getter(GasElement), setter(GasElementBuilder) 를 구현합니다. 그리고 이를 build하여 new GasElement(GasElementBuilder)로 return 합니다. LiquidElement, SolidElement, ArtificialElement 또한 GasElement와 완전히 동일합니다.

4. PhaseElementFactory

```
public class PhaseElementFactory {
    // getInstanceby phase -> Element
    public static Element getInstance(int number, String name, String symbol, double weight, Phase phase) {
        //phase를 기준으로 element 생성
        switch (phase) {
            //Builder pattern을 활용하여 Element 하위 객체 생성 후 return
            case gas:
                return new GasElement.GasElementBuilder(number, name)
                                        .setPhase(Phase.gas)
                                        .setSymbol(symbol)
                                        .setWeight(weight)
                                        .build();
            case solid:
                return new SolidElement.SolidElementBuilder(number, name)
                                        .setPhase(Phase.solid)
                                        .setSymbol(symbol)
                                        .setWeight(weight)
                                        .build();
            case liq:
                return new LiquidElement.LiquidElementBuilder(number, name)
                                        .setPhase(Phase.liq)
                                        .setSymbol(symbol)
                                        .setWeight(weight)
                                        .build();
            case artificial:
                return new ArtificialElement.ArtificialElementBuilder(number, name)
                                        .setPhase(Phase.artificial)
                                        .setSymbol(symbol)
                                        .setWeight(weight)
                                        .build();

            default:
                return null;
        }
    }

}
```

PhaseElementFactory 클래스는 Phase 별로 구분하여 Element 구현 객체를 return 하는 팩토리 클래스입니다. Number, name, symbol, weight, phase 값을 입력받고 이를 phase 별로 분류하여 gas가 입력되면 gasElement를, solid가 입력되면 solidElement를, liq가 입력되면 liquidElement를, artificial이 입력되면 artificialElement 객체를 builder 패턴을 이용해 생성한 후 return 해줍니다.

5. UserInput

```
public class UserInput {
    public static String getString(){
        //scanner를 통한 phase(String) input
        Scanner sc = new Scanner(System.in);
        System.out.print(s:"phase 값을 입력하시오 : ");
        return sc.nextLine();
    }
}
```

UserInput 클래스는 Scanner 객체를 이용하여 Phase 값을 String으로 입력받고 return 합니다.

6. MainTest

```
public static void MainTest(){
    //import List<PeriodicElement> from CSV
    List<PeriodicElement> list = PeriodicElementImporter.loadCSV(filename:"PeriodicElements.csv");

    // create list of Element using PhaseElementFactory
    List<Element> elementList = new ArrayList<>();
    //Factory Method의 getInstance를 이용해 PeriodicElement -> Element(number, name, symbol, weight, phase)
    for(PeriodicElement pe : list){
        Element e = PhaseElementFactory.getInstance(
            pe.getNumber(), pe.getName(), pe.getSymbol(), pe.getMass(), Phase.valueOf(pe.getPhase()));
        elementList.add(e);
    }

    //UserInput을 통해 받아온 값으로 Phase init
    Phase phase= Phase.valueOf(UserInput.getString());

    //phase에 해당하는 element List 생성
    Element[] found = elementList.stream().filter(e -> e.
        getClass().equals(getClass(phase))).toArray(Element[]::new);

    //List found의 element.name 출력
    for(Element e : found){
        System.out.println("name: "+ e.getName());
    }
}
```

MainTest에서는 CSV 파일을 통해 List<PeriodicElement>를 생성합니다. 그리고 반복문을 통해 PE를 탐색하면서 PhaseElementFactory.getInstance()를 이용해 Element 객체를 생성한 후 elementList에 add 해줍니다. UserInput을 통해 사용자로부터 Phase 값을 입력받으면 해당 Phase에 맞는 객체들만 List로 filtering 한 후 객체들의 이름을 차례로 출력합니다.

Result

```
jeongwoon@jeongwoonui-MacBookPro HW5 %  /usr/bin/env /Library/Java/JavaVirtualMach
gwoon/Documents/univ/3-2/javaProgramming/HW5/bin App
file import: PeriodicElements.csv
line contains #: #AtomicNumber,Element,Symbol,AtomicMass,Period,Group,Phase,Type
load successfully
phase 값을 입력하시오 : liq
name: Bromine
name: Mercury
jeongwoon@jeongwoonui-MacBookPro HW5 %
```