REPORT

Lab9

자바프로그래밍2

제출일 2023. 12.03

소속 컴퓨터공학과

학번 32183520

이름 이 주성

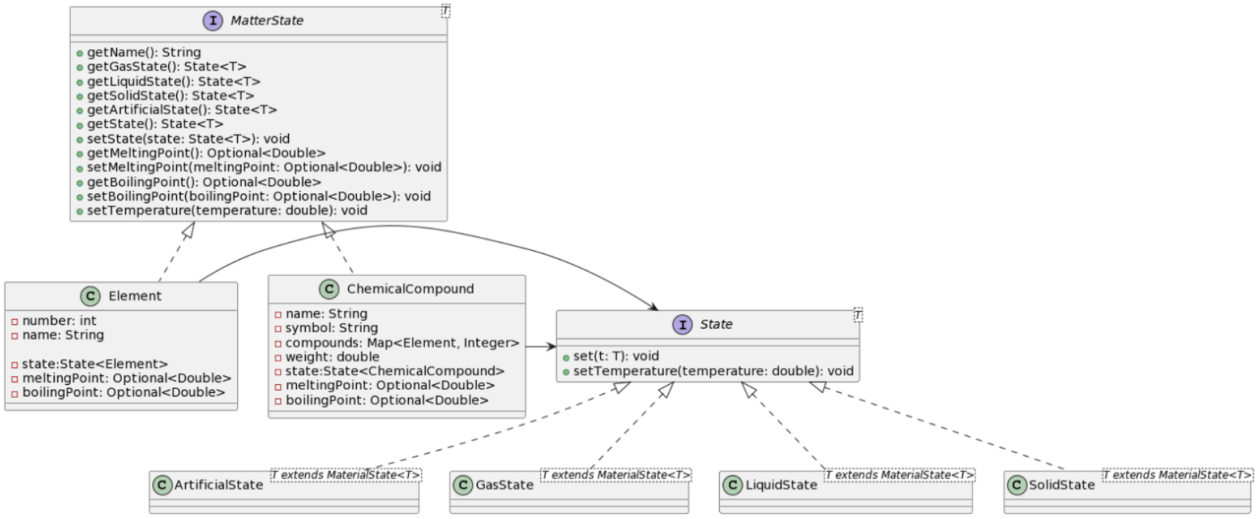
과제 목표

1. 주어진 JSON 파일을 Element, ChemicalCompound 리스트로 import
    - boilingPoint, meltingPoint는 Optional<Double>로 변환해서 넣고 null인 경우엔 Optional.empty()로 넣기
    - state 값을 State 객체로 변환해서 넣기
2. 사용자가 setTemperature로 온도 변화를 주면 객체는 State 패턴에 따라 객체 내부 상태가 변경
    - 온도 변화에 따른 상태 변화

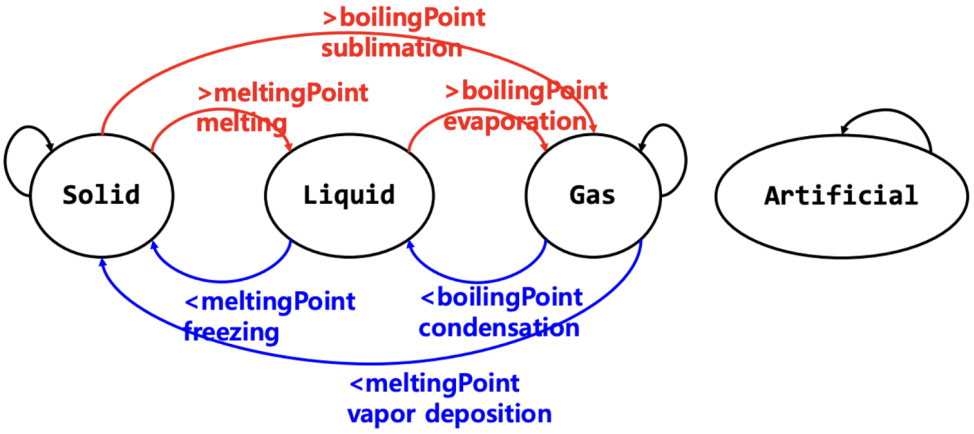
State 패턴이란**?**

- 객체가 내부 상태 변화에 따라 행동을 변경할 수 있게 하는 행동 디자인 패턴
- 객체가 내부 상태가 변경될 때 동적으로 행동을 변경할 수 있도록 해주고 객체는 다른 상태 객체에게 책임을 위임해서 동작을 변화시킬 수 있다.
- 각 상태의 동작을 별도의 클래스로 캡슐화해 깨끗하고 모듈화된 코드 구현 가능
- State 패턴을 도입하면 많은 조건문과 case문을 삭제할 수 있다.

클래스 다이어그램



가능한 **State** 정의



State<T>

- 상태를 나타냄

```java
public interface State<T> {
   // 상태의 원소 설정
   void set(T element);
```

```
    // 상태의 온도 설정
    // 녹는점이나 끓는 점과 같은 온도 변화가 생길 경우 이 메서드 사용
    void setTemperature(double temperature);
}
```

MatterState<T>

- 물질 상태를 나타냄

```
public interface MatterState<T> {
    // 이름 반환
    String getName();

    // 각각의 상태에 대한 State를 반환
    State<T> getArtificialState();
    State<T> getGasState();
    State<T> getLiquidState();
    State<T> getSolidState();
    State<T> getState();

    // 현재 상태 설정
    void setState(State<T> state);

    // 녹는 점 값 getter & setter
    Optional<Double> getMeltingPoint();
    void setMeltingPoint(Optional<Double> meltingPoint);

    // 끓는 점 값 getter & setter
    Optional<Double> getBoilingPoint();
    void setBoilingPoint(Optional<Double> boilingPoint);

    // 현재 상태의 물질의 온도 설정
    void setTemperature(double temperature);
}
```

GasState

- 현재 온도가 끓는점보다 낮아지면 액체화

```
// MatterState<T>의 하위 타입인 T 제네릭
public class GasState<T extends MatterState<T>> implements State<T> {
    private T t;

    @Override
    public void set(T t) {
        this.t = t;
    }

    @Override
    public void setTemperature(double temperature) {
        // 기체 -> 액체화 확인
        if (t.getBoilingPoint().isPresent()) {
            // 현재 온도가 끓는 점보다 낮다면 액체화
            if (temperature < t.getBoilingPoint().get()) {
                System.out.println("Gas is condensing.");
                // 액체 상태로 변경
                t.setState(t.getLiquidState());
                System.out.println(t.getName() + " state changed: " +
t.getState());
            }
        }
    }

    @Override
    public String toString() {
        return "gas";
    }
}
```

LiquidState

- 현재 온도가 끓는점보다 같거나 높으면 기체화
- 현재 온도가 녹는 점보다 낮으면 고체화

```
@Override
public void setTemperature(double temperature) {
    // 변경 플래그
    boolean changeState = false;

    // 액체 -> 고체화 확인
    if (t.getMeltingPoint().isPresent()) {
```

```
        // 현재 온도가 녹는 점보다 낮다면 고체화
        if (temperature < t.getMeltingPoint().get()) {
            System.out.println("Liquid is solidifying.");
            // 고체 상태로 변경
            t.setState(t.getSolidState());
            changeState = true;
        }
    }

    // 액체 -> 기체화 확인
    if (!changeState && t.getBoilingPoint().isPresent()) {
        // 현재 온도가 끓는 점보다 같거나 높으면 기체화
        if (temperature >= t.getBoilingPoint().get()) {
            System.out.println("Liquid is vaporizing.");
            // 기체 상태로 변경
            t.setState(t.getGasState());
            changeState = true;
        }
    }

    // 변화했으면 출력
    if (changeState) {
        System.out.println(t.getName() + " state changed: " +
t.getState());
    }
}
```

SolidState, ArtificialState 동일한 구조

Element 구조

- State 패턴을 적용해 State 참조 값들을 가지고 있다.
- 각 State들은 클래스로 정의해뒀다.
- 초기에는 하나의 State 값을 가지고 있는데 이 값은 JSON으로 받은 값을 파싱할 때 넣어줄 것이다.
- setTemperature() 메서드에 의해 객체들의 상태가 변하도록 할 것이다.

```java
public class Element implements MatterState<Element> {
   // 구성 필드 ..

   private Optional<Double> meltingPoint;
   private Optional<Double> boilingPoint;

   private State<Element> gasSate;
   private State<Element> liquidState;
   private State<Element> solidState;
   private State<Element> artificialState;
   private State<Element> state = null;

   // constructor
   public Element(int number, String name, String symbol, double weight, int
period, int group, String type) {
      // 값 설정..

      // 초기화
      this.gasSate = new GasState<>();
      this.liquidState = new LiquidState<>();
      this.solidState = new SolidState<>();
      this.artificialState = new ArtificialState<>();
   }

   @Override
   public void setState(State<Element> state) {
      // 상태 설정
      this.state = state;
      // 설정한 상태에 현재 객체를 설정
      this.state.set(this);
   }

   @Override
   public void setTemperature(double temperature) {
      // 현재 객체의 온도 설정
      this.state.setTemperature(temperature);
   }

   // Override..

   // ToString..
   }
}
```

JSON 데이터를 **Element**로 파싱

파싱에 필요한 **Deserializer**는 **2**개

1. ElementDeserializer
    - State는 String 값으로 이름을 얻어서 이름에 따라 초기 상태를 지정해준다.

```
public class ElementDeserializer implements JsonDeserializer<Element> {
   @Override
   public Element deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context) {
       JsonObject jsonObject = json.getAsJsonObject();

       // JSON 데이터 파싱
       int number = jsonObject.get("number").getAsInt();
       String name = jsonObject.get("name").getAsString();
       String symbol = jsonObject.get("symbol").getAsString();
       double weight = jsonObject.get("weight").getAsDouble();
       int period = jsonObject.get("period").getAsInt();
       int group = jsonObject.get("group").getAsInt();
       String type = jsonObject.get("type").getAsString();
       Optional<Double> meltingPoint =
Optional.ofNullable(jsonObject.get("meltingPoint")).map(JsonElement::getAsDo
uble);
       Optional<Double> boilingPoint =
Optional.ofNullable(jsonObject.get("boilingPoint")).map(JsonElement::getAsDo
uble);

       // Element 생성 해준 후 녹는점, 끓는점 설정
       Element e = new Element(number, name, symbol, weight, period, group,
type);

//        System.out.println("boilingPoint = " + boilingPoint);
//        System.out.println("meltingPoint = " + meltingPoint);
       e.setMeltingPoint(meltingPoint);
       e.setBoilingPoint(boilingPoint);

       // state 이름을 얻어와 이름에 따라 상태 설정
       String stateName = jsonObject.get("state").getAsString();
       State<Element> state;
```

```
        switch (stateName.toLowerCase()) {
            case "gas":
                state = new GasState<>();
                break;
            case "liq":
                state = new LiquidState<>();
                break;
            case "solid":
                state = new SolidState<>();
                break;
            case "artificial":
                state = new ArtificialState<>();
                break;
            default:
                throw new IllegalArgumentException("Unknown state name: " +
stateName);
        }
        e.setState(state);

        // 완성된 Element 반환
        return e;
    }
}
```

2. Optional<Double>을 위한 DoubleOptionalDeserializer
    - null인 경우 Optional.empty()로 넣어준다.

```
public class DoubleOptionalDeserializer implements
JsonDeserializer<Optional<Double>> {
    @Override
    public Optional<Double> deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context) {
        if (json.isJsonPrimitive() && json.getAsJsonPrimitive().isNumber()) {
            return Optional.of(json.getAsDouble());
        } else {
            return Optional.empty();
        }
    }
}
```

ElementJSONImporter 구현

- 위의 2개의 Desrializer를 이용해 Importer 구현

```
@Override
public List<Element> importFile(String filepath) {
    // 반환할 Element 객체 리스트
    List<Element> list = new ArrayList<>();

    // Optional<Double>을 직접 타입으로 사용할 수 없기 따로 타입으로 선언
    Type doubleOptionalType = new TypeToken<Optional<Double>>() {}.getType();

    // Deserializer 생성
    Gson gson = new GsonBuilder()
            .registerTypeAdapter(Element.class, new ElementDeserializer())
            .registerTypeAdapter(doubleOptionalType, new
DoubleOptionalDeserializer())
            .create();

    try (BufferedReader br = new BufferedReader(new FileReader(filepath))) {
        // JSON 문자열을 List<Element>으로 역직렬화.
        Type listType = new TypeToken<List<Element>>() {}.getType();
        List<Element> elist = gson.fromJson(br, listType);

        // 반환할 리스트에 전부 추가
        list.addAll(elist);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }

    // 리스트 반환
    return list;
}
```

ChemicalCompound의 구조와 파싱 과정은 **Element**와 거의 동일하기에 코드는 생략하였습니다**.**

ChemicalCompound 구조

- State 패턴을 적용해 State 참조 값들을 가지고 있다.
- 각 State들은 클래스로 정의해뒀다.
- 초기에는 하나의 State 값을 가지고 있는데 이 값은 JSON으로 받은 값을 파싱할 때 넣어줄 것이다.
- setTemperature() 메서드에 의해 객체들의 상태가 변하도록 할 것이다.

JSON 데이터를 ChemicalCompound로 파싱

파싱에 필요한 **Deserializer**는 **2**개

1., ChemicalCompoundElementDeserializer

- State는 String 값으로 이름을 얻어서 이름에 따라 초기 상태를 지정해준다.

2. Optional<Double>을 위한 DoubleOptionalDeserializer

- null인 경우 Optional.empty()로 넣어준다.

ElementJSONImporter 구현

- 위의 2개의 Desrializer를 이용해 Importer 구현

```
    // Deserializer 생성
    Gson gson = new GsonBuilder()
            .registerTypeAdapter(Element.class, new ElementDeserializer())
            .registerTypeAdapter(ChemicalCompound.class, new
ChemicalCompoundDeserializer())
            .registerTypeAdapter(doubleOptionalType, new
DoubleOptionalDeserializer())
```

```
            .create();
```

MainTest

1. JSON 데이터를 Element 리스트로 파싱
2. 온도를 22도에서 10씩 올려가며 5000도까지 온도 변화 설정
3. 온도는 22도에서 10씩 내려가며 -300도까지 온도 변화 설정
4. JSON 데이터를 ChemicalCompound 리스트로 파싱
5. 온도를 22도에서 10씩 올려가며 1500도까지 온도 변화 설정
6. 온도는 22도에서 10씩 내려가며 -300도까지 온도 변화 설정

Hydrogen과 **Water**를 예시로 진행하였습니다**.**

```
Element state at temperature = -248.0



Element state at temperature = -258.0
Gas is condensing.
Hydrogen state changed: liq



Element state at temperature = -268.0
Liquid is solidifying.
Hydrogen state changed: solid
```

```
ChemicalCompound state at temperature = 92.0


ChemicalCompound state at temperature = 102.0
Liquid is vaporizing.
Water state changed: gas


ChemicalCompound state at temperature = 112.0
```

```
ChemicalCompound state at temperature = 22.0
Gas is condensing.
Water state changed: liq


ChemicalCompound state at temperature = 12.0


ChemicalCompound state at temperature = 2.0


ChemicalCompound state at temperature = -8.0
Liquid is solidifying.
Water state changed: solid
```