REPORT

자바프로그래밍**2 1**분반

Lab 3.

제출일**: 2023.10.02**

이름**:** 이 주 성

학번**: 32183520**

Subject 인터페이스

- **Observer Pattern**을 위한 **Subject** 인터페이스
- 이 인터페이스를 구현한 클래스는 이벤트 발생시 **notifyListeners()**로 등록된 모든 리스터들에게 알림

메소드

- **addListener:** 리스너 등록
- **removeListener:** 리스너 삭제
- **notifyListeners:** 등록된 리스너들에게 알림

PeriodicTable

- **Subject**을 구현
- 원소가 추가될 때마다 등록된 관찰자들에게 **update**

```
@Override
public void addListener(PeriodicTableListener listener) { this.listeners.add(listener); }

1 usage
@Override
public void removeListener(PeriodicTableListener listener) {
    int i = this.listeners.indexOf(listener);
    if (i >= 0) {
        this.listeners.remove(i);
    }
}

1 usage
@Override
public void notifyListeners(PeriodicElement element) {
    for (PeriodicTableListener listener : this.listeners) {
        listener.update(element);
    }
}
```

PeriodicTableListener 인터페이스 - Observer(Listener)

- 등록한 **PeriodicElement**가 등록될 때 호출되는 **update()**

6가지 Listeners

1. NameListener

- 공백을 기준으로 관심 있는 원소 이름 입력받기

```
public static String[] getStringArray() {
    String[] inputs;

    while (true) {
        try {
            inputs = br.readLine().split( regex: " ");

            if (inputs.length == 0) {
                System.out.println("다시 입력해주세요: ");
            } else {
                return inputs;
            }

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- 소문자 대문자 구분 없이 탐지할 수 있도록 **toLowerCase()** 사용 **<** 개인적으로 추가한 코드 **>**
- 입력받은 **Name** 리스트와 일치하는 지 확인 후 맞다면 **notification()**

```
@Override
public void update(PeriodicElement element) {
    // 소문자, 대분자 구분없이 탐지
    boolean isContain = Arrays.stream(inputs)
            .anyMatch(s →
                    s.toLowerCase().equals(element.getName().toLowerCase()));
    if (isContain) {
        notification(element);
    }
}

1 usage
private void notification(PeriodicElement element) {
    System.out.println("[NameListener] Notification: " + element.getName());
}
```

2. NumberListener

- **1~118** 사이의 숫자 배열을 공백을 기준으로 입력받음
- **Arrays.stream()**을 사용해 **String[]**을 **int[]**로 변환해 저장
- **1 ~118** 사이의 숫자가 아니면 다시 입력하도록 **<** 개인적으로 추가한 코드 **>**

```
while (true) {
    try {
        // 공백을 기준으로 입력받아 int[]로 변환
        numbers = Arrays.stream(br.readLine().split( regex: " ")) Stream<String>
                .mapToInt(Integer::parseInt) IntStream
                .toArray();

        boolean flag = false;

        for (int num : numbers) {
            if (num < min || num > max) {
                flag = true;
                break;
            }
        }

        if (flag) {
            System.out.print("1~118 사이의 번호만 입력해주세요. 다시 입력해주세요: ");
            continue;
        }

        return numbers;
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

3. SymbolListener

- **NameListener**과 동일

4. GroupListener

- **1~18** 사이의 숫자를 입력받음
- 그 사이의 숫자가 아닐 경우 다시 입력하도록 **<** 개인적으로 추가한 코드 **>**
- **null**일 경우 **0**으로 반환 후 나중에 후처리 **<** 개인적으로 추가한 코드 **>**

```
while (true) {
    try {
        String input = br.readLine();

        // null일 경우 0 반환
        if (input.isEmpty()) {
            num = 0;
            break;
        }
        num = Integer.parseInt(input);

        if (num ≥ min && num ≤ max) {
            break;
        } else {
            System.out.print(min + "과 " + max + " 사이의 값을 입력해주세요: ");
        }

    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

- **0**인 경우**,** 즉 **null**인 경우 **1~18** 범위가 아닌 값들에 반응

```java
@Override
public void update(PeriodicElement element) {
    // null인 경우
    if (group == 0) {
        if (element.getGroup() > 18 || element.getGroup() < 1) {
            peList.add(element);
            notification(element);
        }
    }

    if (element.getGroup() == group) {
        peList.add(element);
        notification(element);
    }
}

2 usages
private void notification(PeriodicElement element) {
    System.out.println("[GroupListener] Notification: " + element.getName());
}
```

5. PeriodListener

- **GroupListener**과 동일

6. PhaseListener

- 사용자 입력을 받은 후 **enum** 타입으로 등록해둔 **Phase** 인지 확인
- 아니면 다시 입력하도록

```
public static String getPhase() {
    String input = null;

    while (true) {
        try {
            input = br.readLine();

            if (!input.isEmpty()) {
                for (Phase p : Phase.values()) {
                    if (p.toString().equals(input)) {
                        return input;
                    }
                }
                System.out.print("gas, liq, solid, artificial 중 하나를 입력해주세요
            } else {
                System.out.print("다시 입력해주세요: ");
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- 입력받은 **phase**와 동일한 원소 등록시 **update()**

```
@Override
public void update(PeriodicElement element) {
    if (input.equals(element.getPhase())) {
        notification(element);
    }
}


1 usage
private void notification(PeriodicElement element) {
    System.out.println("[PhaseListener] Notification: " + element.getName());
}
```

MainTest

1. Create the priodic table

2. Observer Setting

```
// NumberListener
System.out.print("[NumberListener] 관심있는 원소의 번호를 등록하세요. \" \" 공백을 기준으로 구분해주세요. (1~118): ");
int[] numbers = UserInput.getIntegerArrayBetween(1, 118);
NumberListener numL = new NumberListener(numbers);

// SymbolListener
System.out.print("[SymbolListener] 관심있는 원소의 기호를 등록하세요. \" \" 공백을 기준으로 구분해주세요. : ");
String[] symbols = UserInput.getStringArray();
SymbolListener symL = new SymbolListener(symbols);

// NameListener
System.out.print("[NameListener] 관심있는 원소의 이름를 등록하세요. \" \" 공백을 기준으로 구분해주세요. : ");
String[] names = UserInput.getStringArray();
NameListener nameL = new NameListener(names);
```

```
// PeriodListener
System.out.print("[PeriodListener] 관심있는 주기 번호를 등록하세요 (1~7): ");
int period = UserInput.getIntegerBetween(1, 7);
PeriodListener periodL = new PeriodListener(period);

// GroupListener
System.out.print("[GroupListener] 관심있는 그룹 번호를 등록하세요 (null or 1~18): ");
int group = UserInput.getNullOrIntegerBetween(1, 18);
GroupListener groupL = new GroupListener(group);

// PhaseListener
System.out.print("[PhaseListener] 관심있는 Phase를 등록하세요 (gas/liq/solid/artificial): ");
String phase = UserInput.getPhase();
PhaseListener phaseL = new PhaseListener(phase);
```

3. Add Listeners

```
// Add Listeners
t.addListener(numL);
t.addListener(symL);
t.addListener(nameL);
t.addListener(periodL);
t.addListener(groupL);
t.addListener(phaseL);
```

4. Add elements to the periodic table every 0.5 second

```
for (PeriodicElement element: list) {
    try {
        Thread.sleep( millis: 500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("========== Add element : " + element.getName() + "

    // Add element to the periodic table
    t.addElement(element);
}
```

5. removeListener - NumberListener

```
// removeListener ...
System.out.println("removerListener: NumberListener");
t.removeListener(numL);
```

6. 4번 과정 반복

결과

```
[NumberListener] 관심있는 원소의 번호를 등록하세요. " " 공백을 기준으로 구분해주세요. (1~118): 1 23 123 43 21
1~118 사이의 번호만 입력해주세요. 다시 입력해주세요: 1 34 21 52 35
[SymbolListener] 관심있는 원소의 기호를 등록하세요. " " 공백을 기준으로 구분해주세요. : H O Be
[NameListener] 관심있는 원소의 이름를 등록하세요. " " 공백을 기준으로 구분해주세요. : Hydrogen
[PeriodListener] 관심있는 주기 번호를 등록하세요 (1~7): 3
[GroupListener] 관심있는 그룹 번호를 등록하세요 (null or 1~18):
[PhaseListener] 관심있는 Phase를 등록하세요 (gas/liq/solid/artificial): solid
Start adding elements to the periodic table
========== Add element : Hydrogen ==========
[NumberListener] Notification: Hydrogen
[SymbolListener] Notification: Hydrogen
[NameListener] Notification: Hydrogen
========== Add element : Helium ==========
========== Add element : Lithium ==========
========== Add element : Beryllium ==========
[SymbolListener] Notification: Beryllium
========== Add element : Boron ==========
========== Add element : Carbon ==========
========== Add element : Nitrogen ==========
========== Add element : Oxygen ==========
```

...