

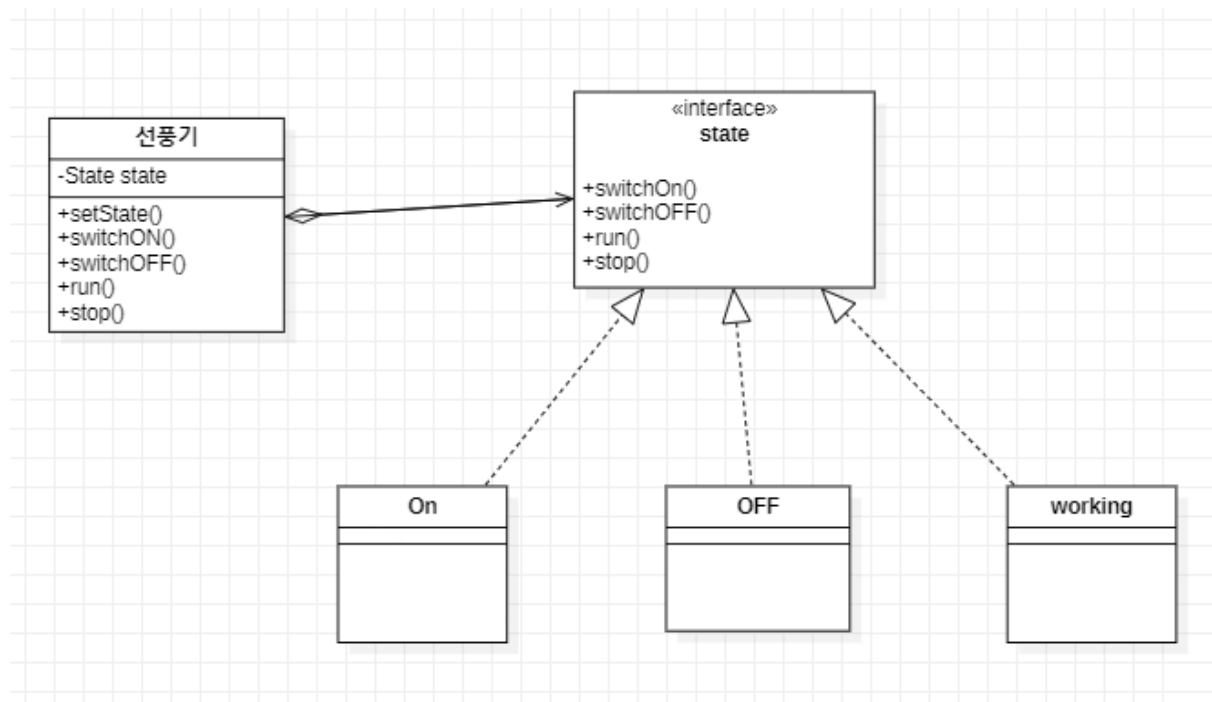
# 소프트웨어 분석 설계

## 9주차

20170677 오윙택

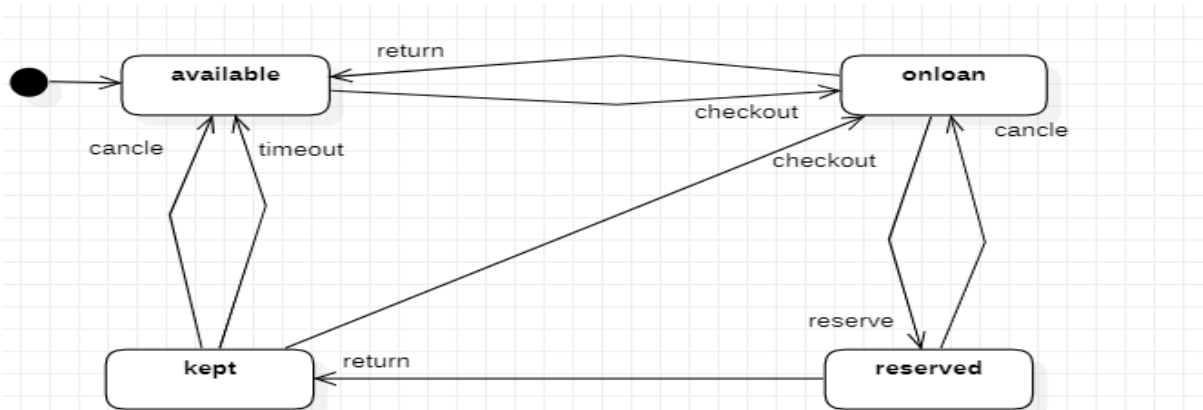
1. 연습문제 2번 :

(1) 선풍기 상태도(그림7-1)를 가지고 state pattern 클래스다이어그램을 그리시오. (p.439)



## 2. 연습문제 4번:

(1) 설명에서 상태를 찾자 ◇ 상태다이어그램을 그리자



도서 대출 시스템...?

나는 책을 언제 받아 갈 예정이니 도서관에서는 책을 보관하고 있어라

-> 이런 상황은 일반적으로 생기지 않음

why? 예약이라는 상황은 이미 대출중인 도서를 빌려가려고 할때 다음에 내가 빌려갈테니 보관해줘 라는 약속인 경우가 대부분

하지만 가능하다

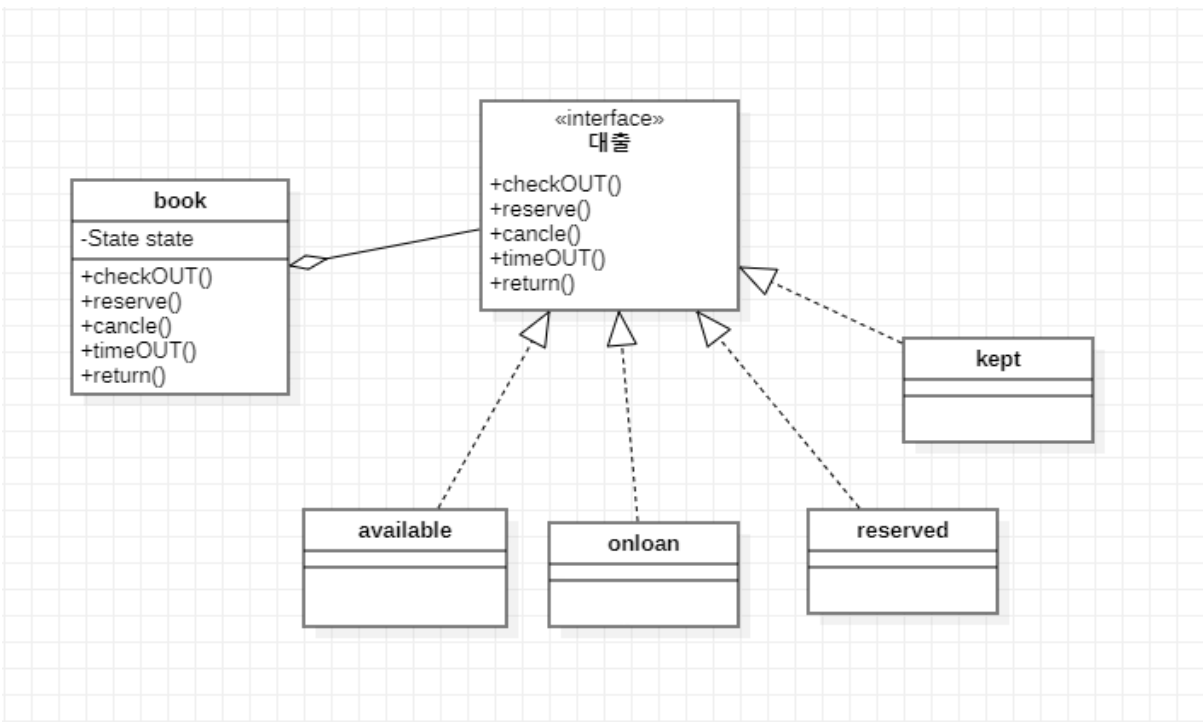
-> 로직을 수정해야하는 잘못된 상태머신인가?

no

예제는 책에 관한 상태다이어그램

내가 생각한건 도서관 대출 시스템 다이어그램

(2) State 패턴으로 클래스다이어그램을 그리자

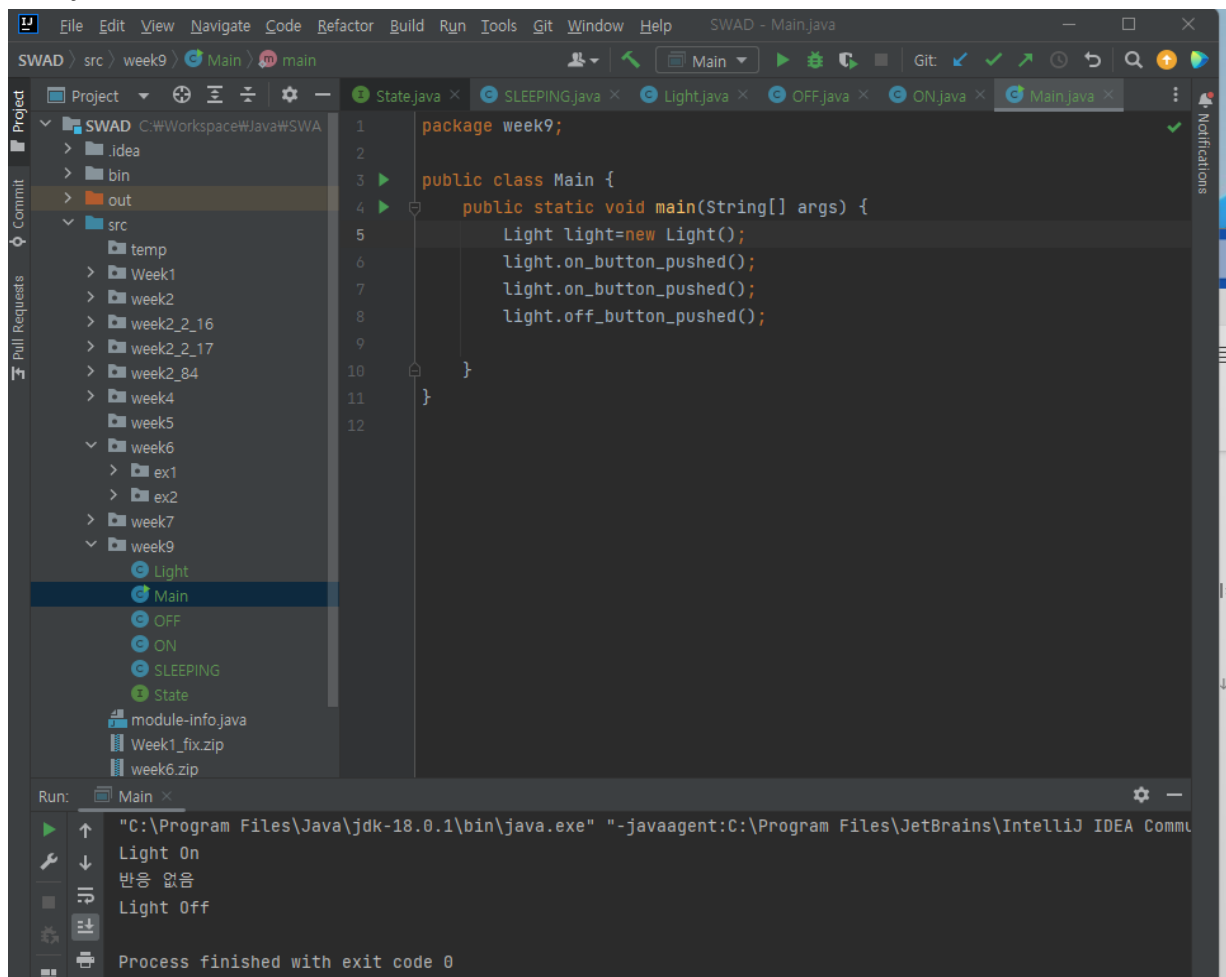


### 3. 연습문제 3 번:

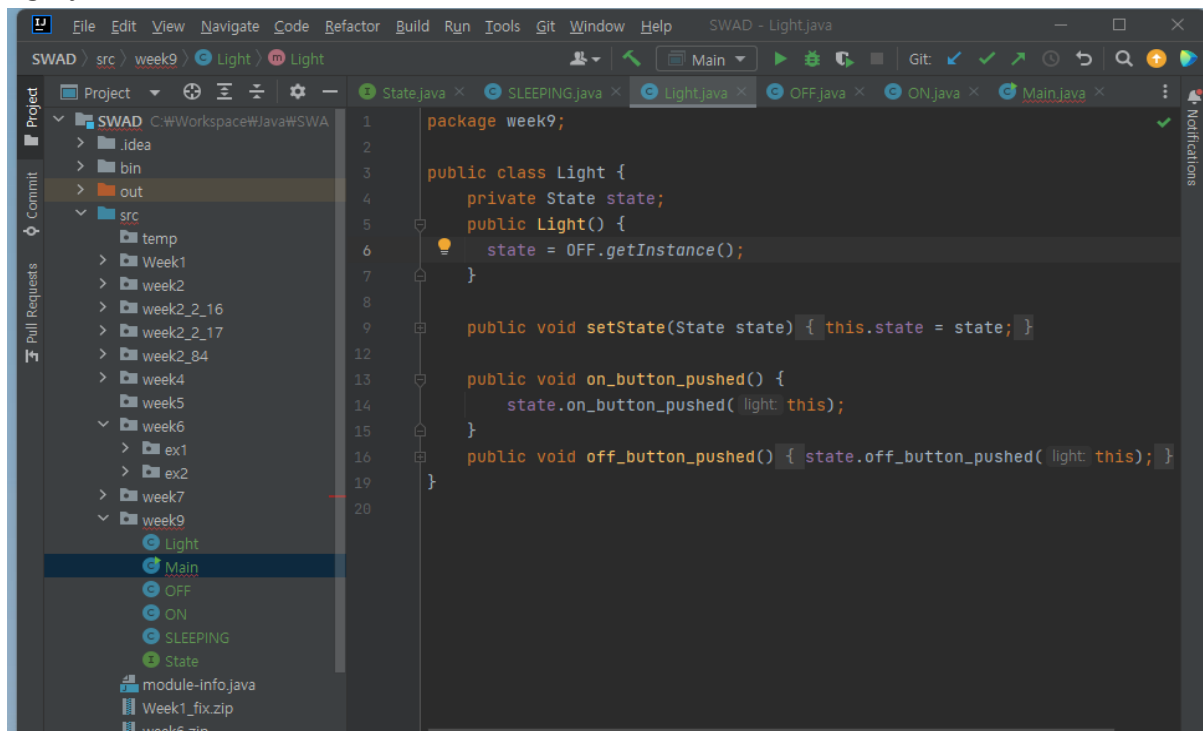
(1) Code7-234567 를 코딩하고 실행시켜본 후 그림 7-3 의 다이어그램을 참조하여 SLEEPING 상태를 추가하여 확장 구현하시오.

```
public class Main {  
    public static void main(String[] args){  
        Light light=new Light();  
        light.on_button_pushed();  
        light.on_button_pushed();  
        light.off_button_pushed();  
    }  
}
```

Main.java



## Light.java



```
package week9;

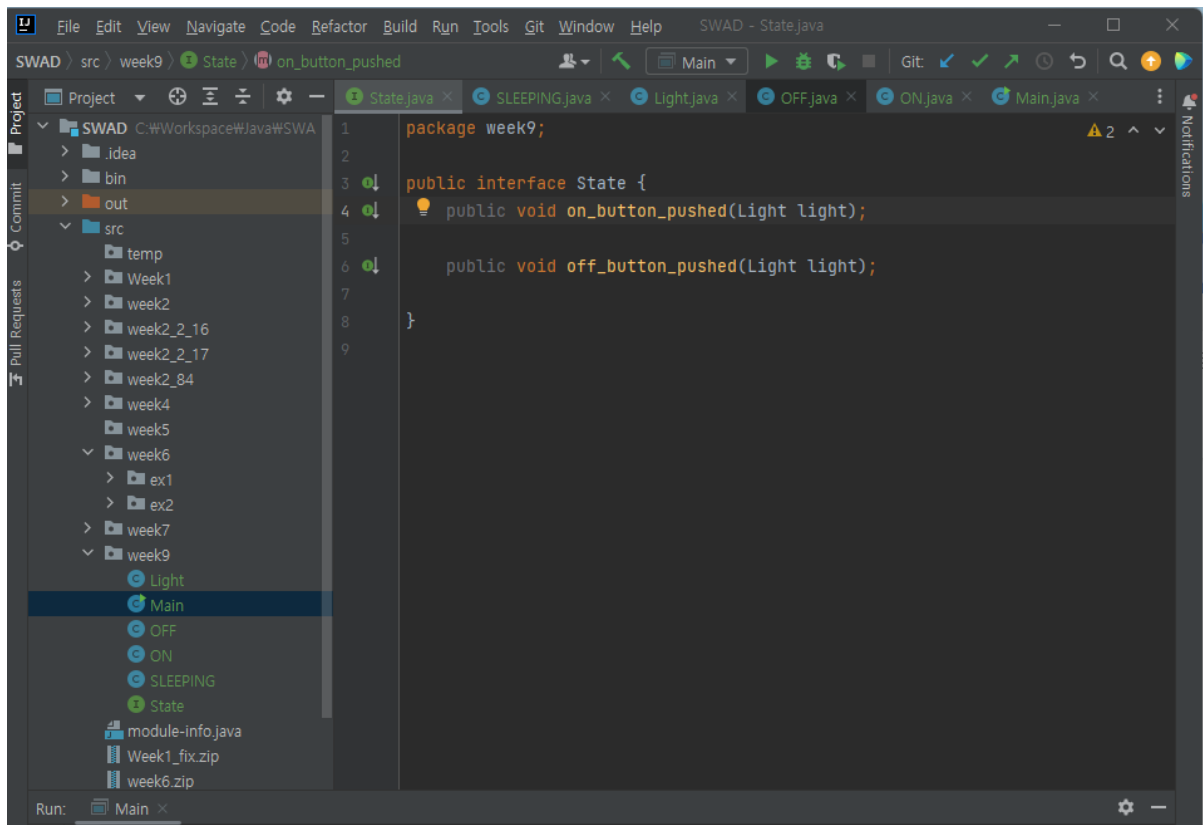
public class Light {
    private State state;
    public Light() {
        state = OFF.getInstance();
    }

    public void setState(State state) { this.state = state; }

    public void on_button_pushed() {
        state.on_button_pushed( light this);
    }

    public void off_button_pushed() { state.off_button_pushed( light this); }
}
```

## State.interface

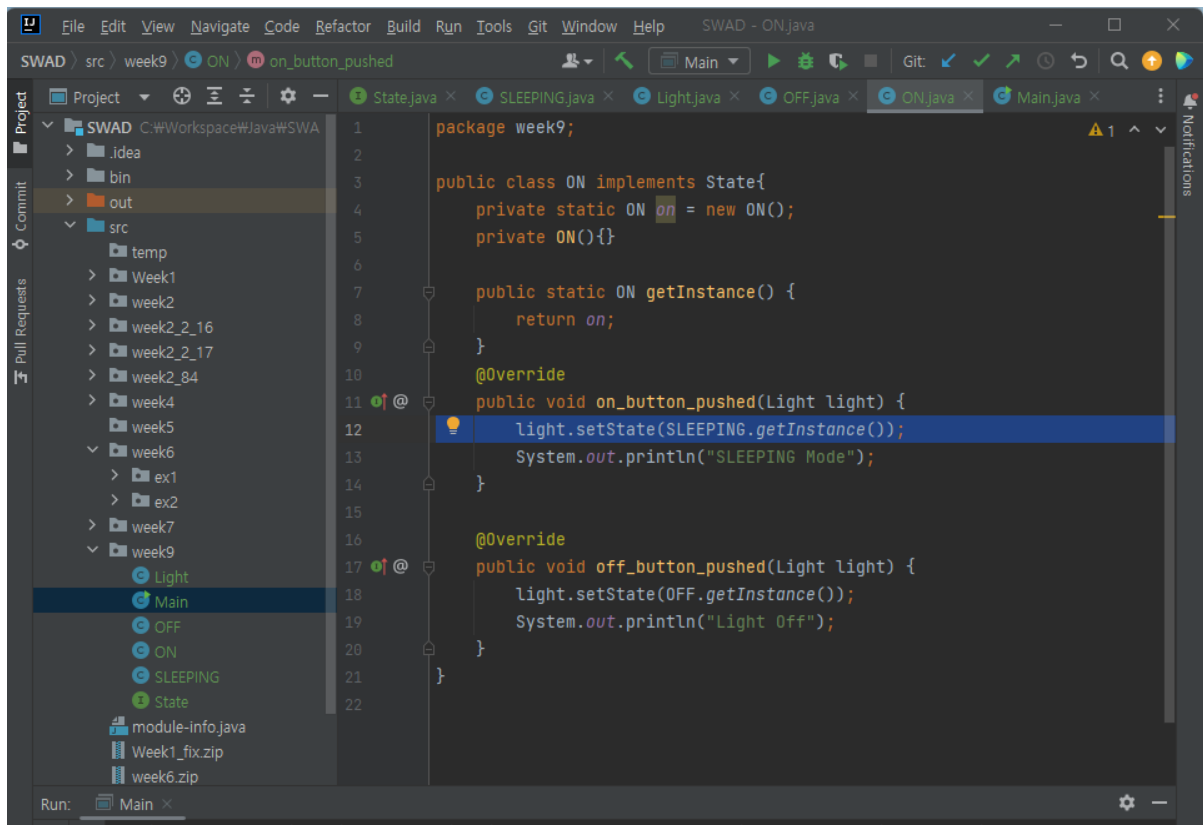


```
package week9;

public interface State {
    public void on_button_pushed(Light light);

    public void off_button_pushed(Light light);
}
```

## ON.JAVA



```
package week9;

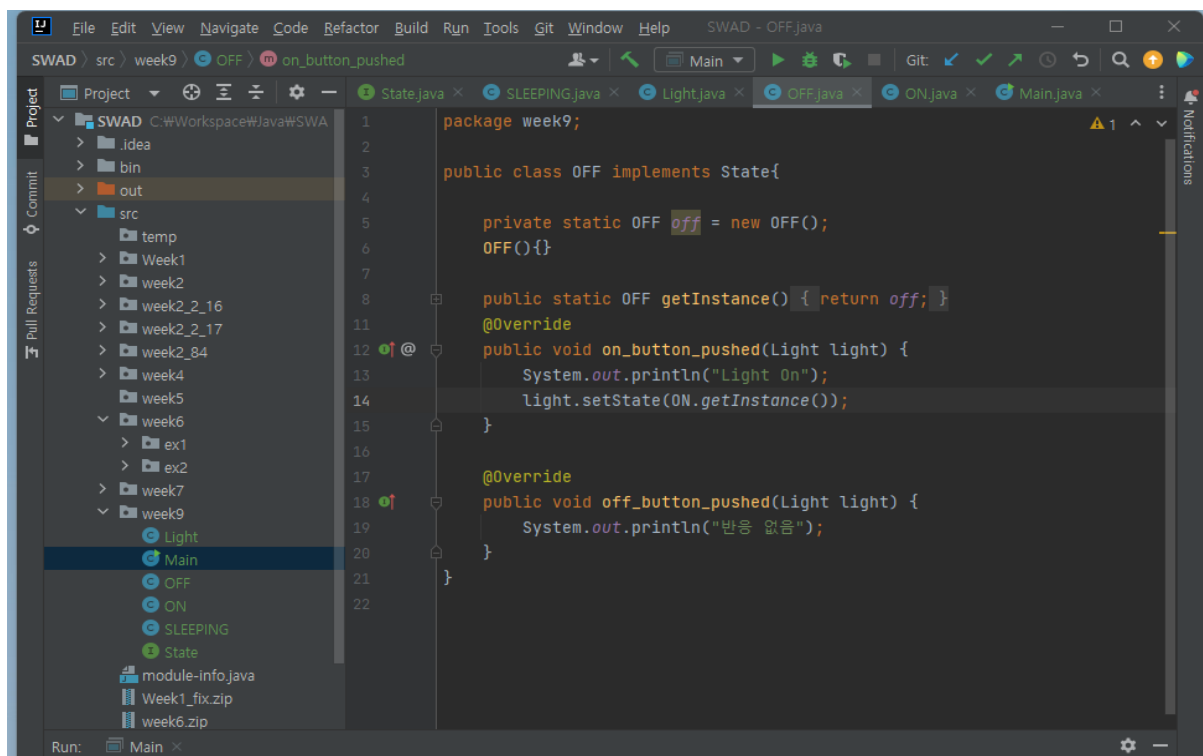
public class ON implements State{
    private static ON on = new ON();
    private ON(){}

    public static ON getInstance() {
        return on;
    }

    @Override
    public void on_button_pushed(Light light) {
        light.setState(SLEEPING.getInstance());
        System.out.println("SLEEPING Mode");
    }

    @Override
    public void off_button_pushed(Light light) {
        light.setState(OFF.getInstance());
        System.out.println("Light Off");
    }
}
```

## OFF.JAVA



```
package week9;

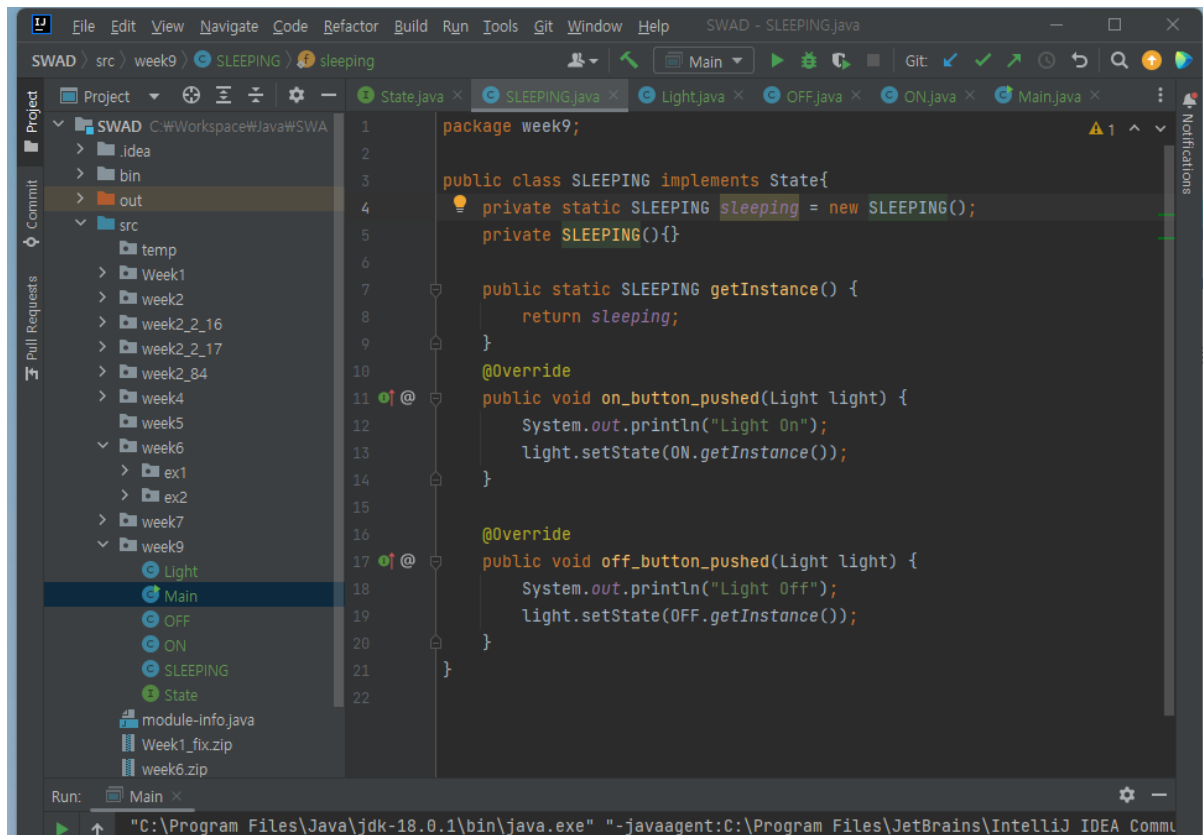
public class OFF implements State{
    private static OFF off = new OFF();
    OFF(){}

    public static OFF getInstance() { return off; }

    @Override
    public void on_button_pushed(Light light) {
        System.out.println("Light On");
        light.setState(ON.getInstance());
    }

    @Override
    public void off_button_pushed(Light light) {
        System.out.println("반응 없음");
    }
}
```

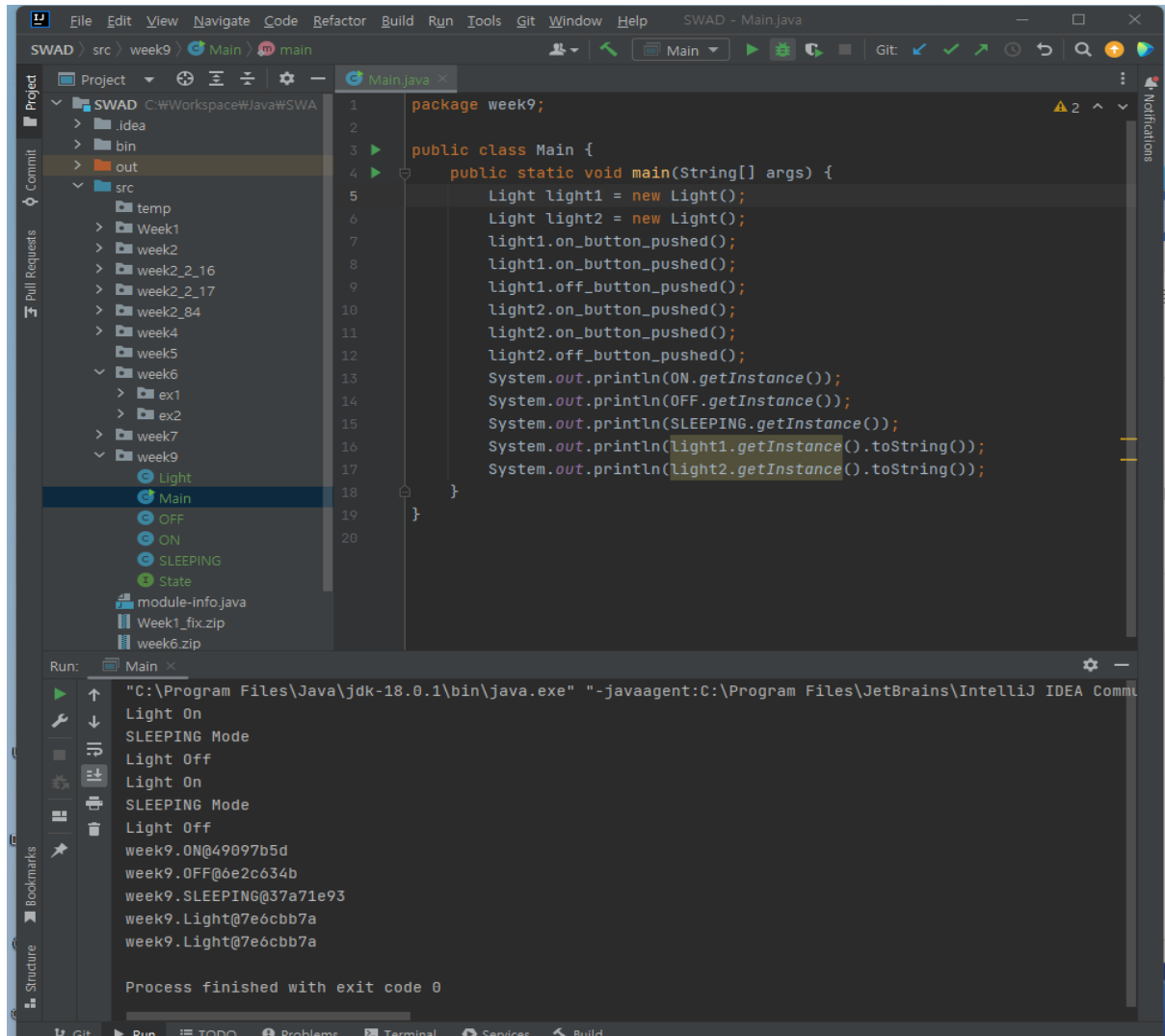
## SLEEPING.JAVA



State 라는 인터페이스를 구체화한 ON, OFF, SLEEPING 이라는 상태 인스턴스를 1 개씩 싱글턴으로 사용하고 있고, Main.java 에서 메소드 Call 을 통해 Light 의 State 를 변경하면 각 인스턴스들이 참조되면서 ON - OFF - SLEEPING 이라는 상태 모드가 변경된다.

(2) Singleton 도 적용하여 작성하기 객체.toString()으로 객체번호 확인하여 하나의 객체만 활용함을 확인하기

변경된 Main.java



The screenshot displays the IntelliJ IDEA IDE with the 'Main.java' file open. The code defines a 'Main' class with a 'main' method that interacts with 'Light' objects. The 'Light' class is part of the 'week9' package. The 'main' method creates two 'Light' objects, 'light1' and 'light2', and performs a sequence of operations: turning on, turning off, and checking the state. The output of the program is shown in the 'Run' console at the bottom.

```
package week9;

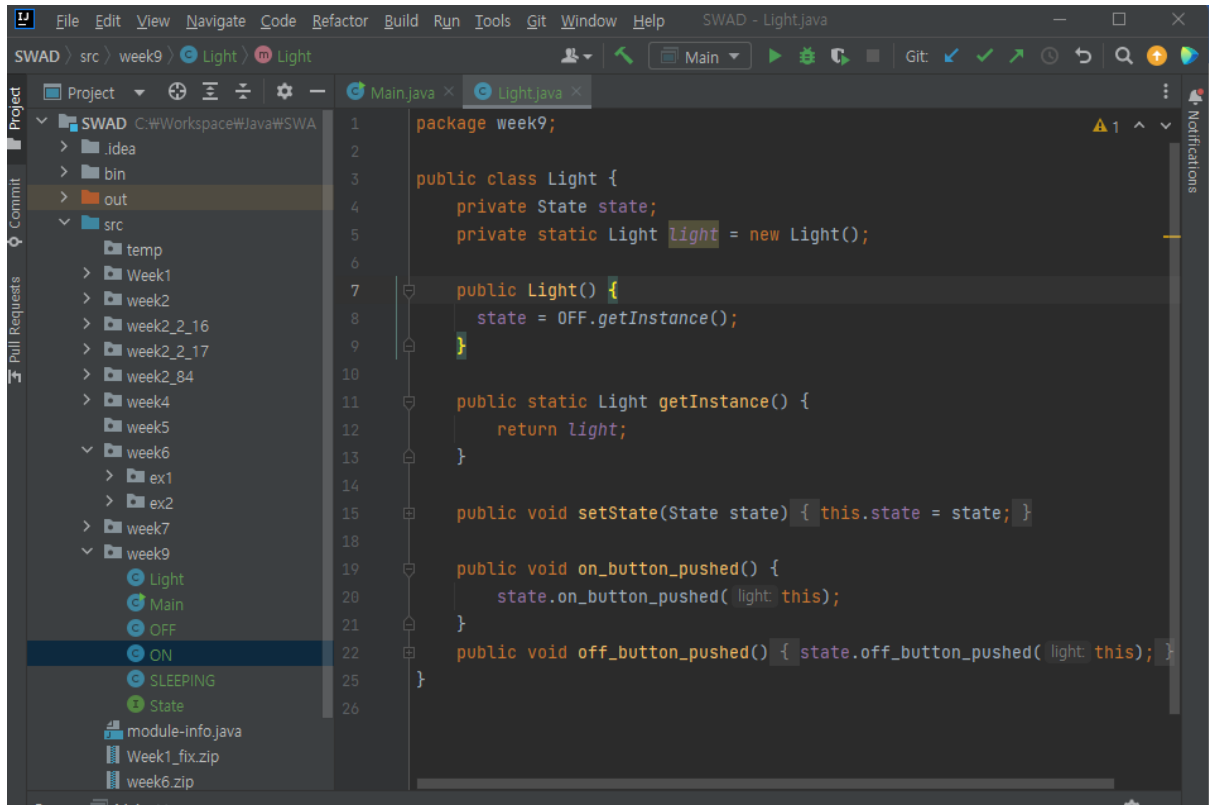
public class Main {
    public static void main(String[] args) {
        Light light1 = new Light();
        Light light2 = new Light();
        light1.on_button_pushed();
        light1.on_button_pushed();
        light1.off_button_pushed();
        light2.on_button_pushed();
        light2.on_button_pushed();
        light2.off_button_pushed();
        System.out.println(ON.getInstance());
        System.out.println(OFF.getInstance());
        System.out.println(SLEEPING.getInstance());
        System.out.println(light1.getInstance().toString());
        System.out.println(light2.getInstance().toString());
    }
}
```

Run: Main

```
"C:\Program Files\Java\jdk-18.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Commu
Light On
SLEEPING Mode
Light Off
Light On
SLEEPING Mode
Light Off
week9.ON@49097b5d
week9.OFF@6e2c634b
week9.SLEEPING@37a71e93
week9.Light@7e6cbb7a
week9.Light@7e6cbb7a

Process finished with exit code 0
```

변경된 Light.JAVA



이전 문제에서 ON, OFF, SLEEPING 이라는 상태 클래스들은 싱글톤이 적용되어 있었다. 하지만 Light 클래스는 싱글톤이 적용되지 않아 Light 개체를 새로 생성하면 Light1 개체와 Light2 개체가 상태 인스턴스들을 각각 독립적으로 갖게 된다. 때문에 Light 인스턴스 또한 싱글톤으로 구현해야 모든 상태를 전역적으로 관리할 수 있게 된다.

Light 클래스 내부에서 개체를 생성하고, 이를 getInstance 를 통해 호출할 수 있도록 구현하였다. 하지만 이러한 방식은 단점이 존재한다

두 Light 인스턴스가 상태를 공유하므로 한쪽의 상태가 변경되면 다른 한쪽의 인스턴스의 상태 또한 같이 변경된다는 점이다. Light1 은 불을 켜놓고, Light2 는 불을 SLEEPING 모드로 사용하고 싶어도 그것이 불가능해진다는 것이다.

이를 해결하기 위해선 이전의 코드와 같이 Light 개체는 독립적인 인스턴스를 갖는 것이 좋다. 때문에 싱글톤은 꼭 필요한 경우에만 적절하게 사용하는 것이 좋을 것 같다.