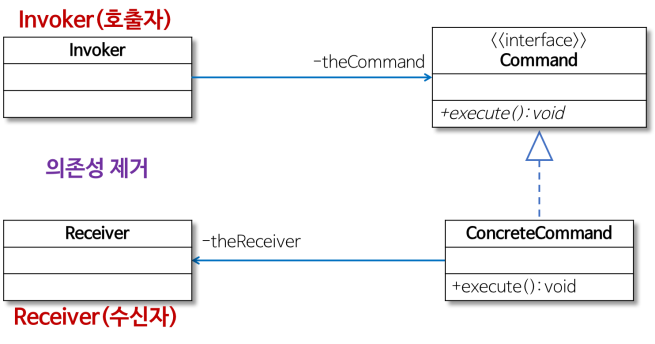
커맨드 패턴

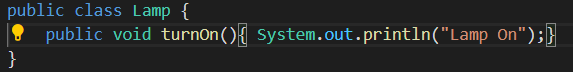
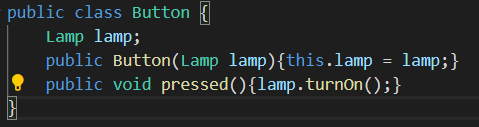
* 실행될 기능을 캡슐화 함으로 주어진 여러 기능을 실행할 수 잇는 재사용 성이 높은 클래스를 설계하는 패턴
  + 즉 이벤트가 발생했을 떄 실행될 기능이 다양하면서도 변경이 필요한 경우에 이벤트를 발생시키는 클래스를 변경하지 않고 재사용하고자 할 때 유용하다.
  + 행위 패턴 중 하나.

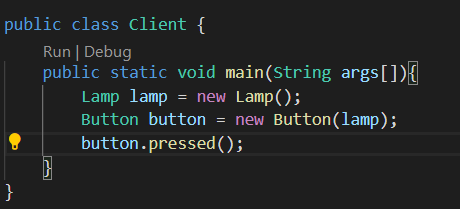


* 실행될 기능을 캡슐화함으로써 기능의 실행을 요구하는 호출자 클래스와 실제 기능을 실행하는 수신자 클래스 사이의 의존성을 제거한다.
* 따라서 실행될 기능의 변경에도 호출자 클래스를 수정 없이 그대로 사용할 수 있도록 해준다.
* 역할이 수행하는 작업
  + Command
    - 실행될 기능에 대한 인터페이스
    - 실행될 기능을 excute 메서드로 선언함
  + ConcreteCommand
    - 실제로 실행되는 기능을 구현
    - 즉, Command라는 인터페이스를 구현
  + Invoker
    - 기능의 실행을 요청하는 호출자 클래스
  + Receiver
    - ConcreteCommand에서 excute 메서드를 구현할 때 필요한 클래스
    - 즉, ConcreteCommand의 기능을 실행하기 위해 사용하는 수신자 클래스
* 참고
  + 행위 패턴
    - 객체나 클래스 사이의 알고리즘이나 책임 분베애 관련된 패턴
    - 한 객체가 혼자 수행할 수 없는 여러 개의 객체로 어떻게 분배하는지 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.
* 예제 만능버튼 만들기



* 버튼이 눌리면 램프의 불이 켜지는 프로그램

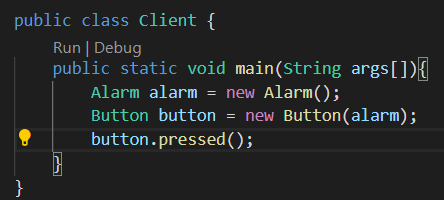
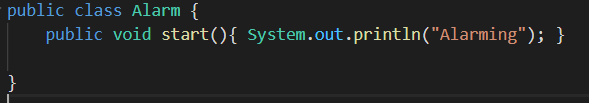


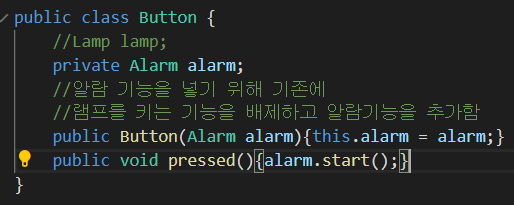


* Button 클래스의 생성자를 이용해 불을 켤 Lamp 객체를 전달한다.
* Button 클래스의 pressed()가 호출되면 생성자를 통해 전달받은 Lamp객체의 turnOn()를 호출해 불을 켠다.

문제점

* 버튼을 눌렀을 때 다른 기능을 실행한 경우
  + 버튼을 눌렀을 때 알람이 시작되기 하려면

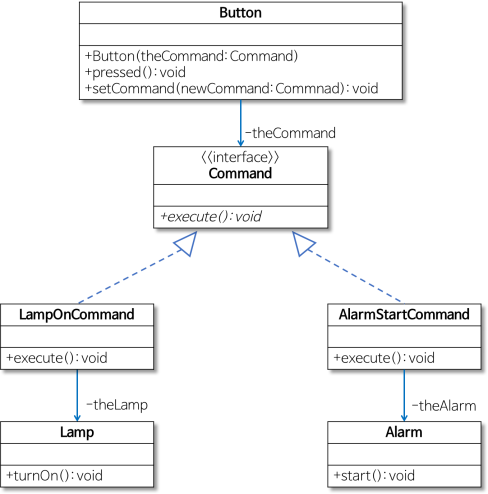




* + 새로운 기능으로 변경하려고 기존 코드의 내용을 수정해야 하므로 ocp에 위배된다.
  + Button 클래스의 pressed() 전체를 변경해야 한다.
* 버튼을 누르는 동작에 따라 다른 기능을 실행하는 경우
  + 버튼을 처음 눌렀을 때는 램프를 켜고 두번쨰 눌렀을 떄는 알람을 동작하게 하려면
  + 필요한 기능을 새로 추가할 떄마다 Button 클래스의 코드를 수정해야 하므로 재사용하기가 어렵다.
* 해결책

문제를 해결하기 위해서는 구체적인 기능을 직접 구현하는 대신에 실행될 기능을 캡슐화해야 한다.

* + 즉 Button클래스의 pressed() 메소드에서 구체적인 기능(램프켜기, 알람동작 등) 을 직접 구현하는 대신에 버튼을 눌렀을 떄 실행될 기능을 Button클래스 외부에서 제공받아 캡슐화해 pressed()메소드에서 호출한다.
  + 이를 통해 Button 클래스 코드를 수정하지 않고 그대로 사용할 수 있다.



* Button 클래스는 미리 약속된 Command 인터페이스의 excute메소드를 호출한다.
  + 램프를 켜는 경우에는 theLamp.turnOn()메소드를 호출하고
  + 알람이 동작하는 경우에는 theAlarm.start()메소드를 호출하도록 pressed메서드를 수정한다.
  + 마찬가지로 AlarmStartCommand클래스는 Command인터페이스의 excute메서드를 구현해 alarm 클래스의 start메서드를 호출한다.
* Command 인터페이스를 구현하는 LampOnCommand와 AlarmStartCommand객체를 Button객체에 설정한다.
* Button 클래스의 pressed 메소드에서 Command인터페이스의 excute 메서드를 호출한다.
* 즉, 버튼을 눌렀을 때 필요한 임의의 기능은 Command 인터페이스를 구현한 클래스의 객체를 Button객체에서 설정할 수 있다.
* 이렇게 Command 패턴을 이용하면 Button 클래스의 코드를 변경하지 않으면서 다양한 동작을 구현할 수 있게 된다.
* 