스트레티지 패턴

- 행위를 클래스로 캡슐화해 동적으로 행위를 자유롭게 바꿀 수 있게 해주는 패턴이다.

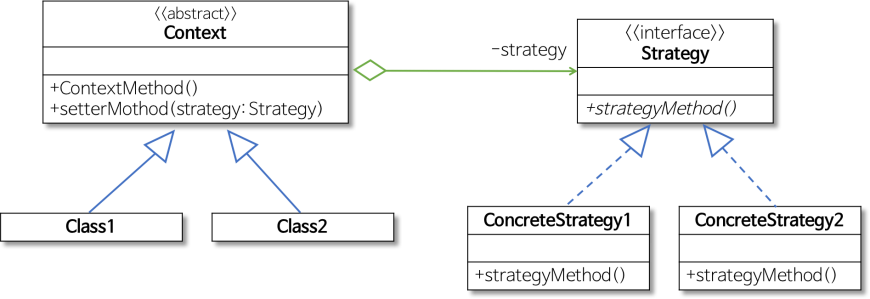
- 같은 문제를 해결하는 여러 알고리즘이 클래스별로 캡슐화되어 있고, 이들이 필요할 때 교체할 수 있또록 함으로 동일한 문제를 다른 알고리즘으로 해결할 수 있게 하는 디자인 패턴이다.

- 행위 패턴중 하나이다.

- 전략을 쉽게 바꿀 수 있도록 해주는 디자인 패턴이다.

- 전략 : 어떤 목적을 달성하기 위해 일을 수행하는 방식, 비즈니스 규칙, 문제를 해결하는 알고리즘 등

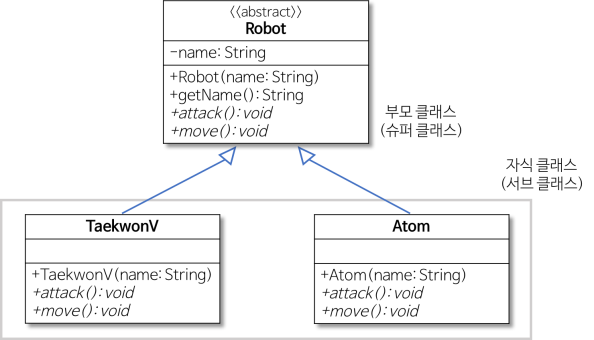
- 특히 게임 프로그래밍에서 게임 캐릭터가 자신이 처한 상황에 따라 공격이나 행동하는 방식을 바꾸고 싶을 떄 스트레티지 패턴은 매우 유용하다.

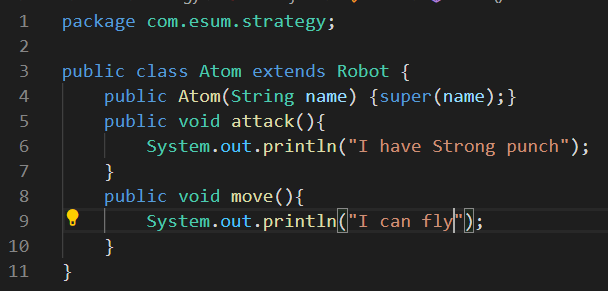
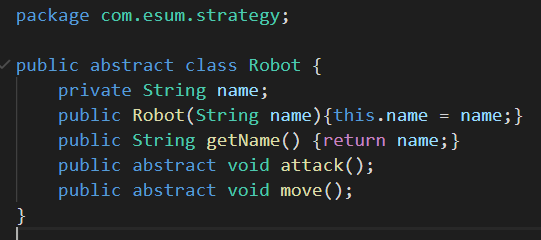


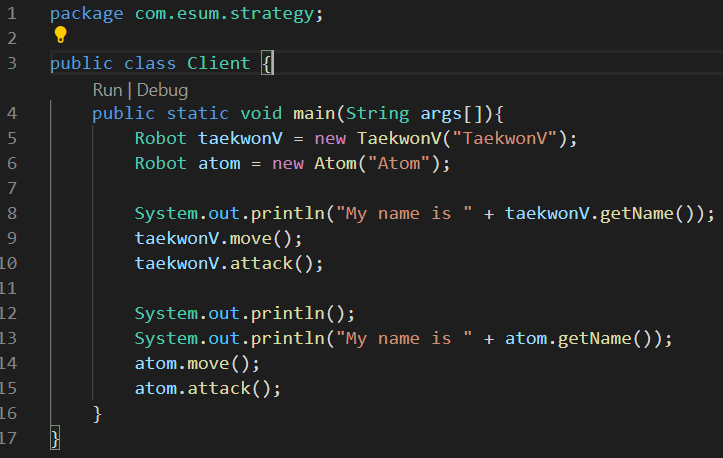
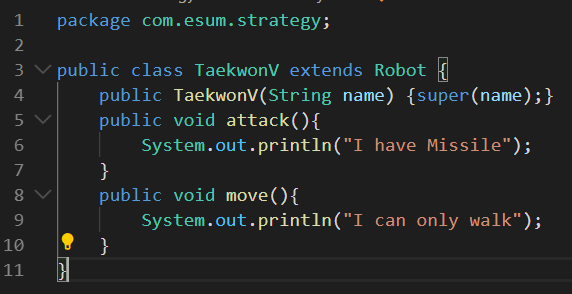
* 역할이 수행하는 작업
  + Strategy
    - 인터페이스나 추상 클래스로 외부에서 동일한 방식으로 알고리즘을 호추랗는 방법을 명시
  + ConcreateStrategy
    - 스트래티지 패턴에서 명시한 알고리즘을 실제로 구현한 클래스
  + Context
    - 스트래티지 패턴을 이용하는 역할을 수행
    - 필요에 따라 동적으로 구체적인 전략을 바꿀 수있도록 setter메서드를 제공한다.
* 참고
  + 행위패턴
    - 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴
    - 한 객체가 혼자 수행할 수 없느 작업을 여러 개의 객체로 어떻게 분배하는지, 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.
  + 집약관계
    - 참조 값을 인자로 받아 필드를 세팅하는 겨우
    - 전체 객체의 라이프 타임과 부분객체의 라이프 타임은 독립적이다.
    - 즉 전체 객체가 메모리에서 사라진다 해도 부분 객체는 사라지지 않는다.

예시

* 로봇만들기





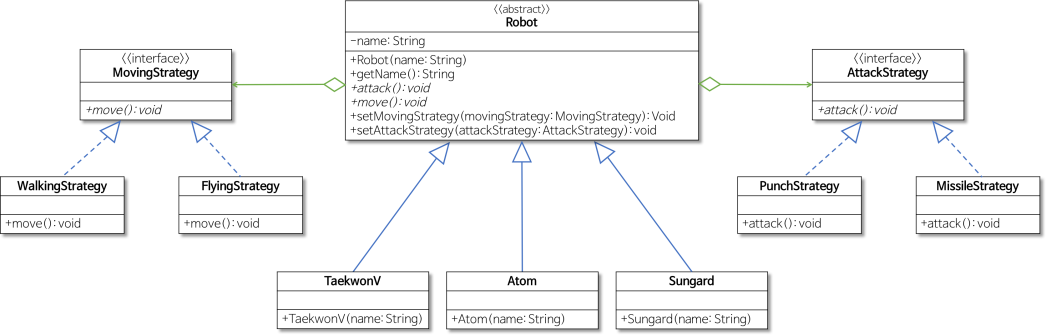


문제점

* 기존 로봇의 공격과 이동 방법을 수정하는 경우
  + Atom이 날 수는 없고 오직 걷게만 만들고 싶다면?
  + TaekwonV를 날게 하려면
  + 새로운 기능으로 변경하려고 기존 코드의 내용을 수정해야 하므로 ocp에 위배된다(기존 소스코드는 그대로 놓은상태에서 기능만 추가하는 방식)
  + 또한 TaekwonV와 Atom의 move() 메서드의 내용이 중복된다. 이런 중복 상황은 많은 문제를 야기하는 원인이 된다.
  + 만약 걷는 방식에 문제가 있거나 새로운 방식으로 수정하려면 모든 중복 코드를 일관성 있게 변경해야 한다.

해결책

* 문제를 해결하기 위해 무엇이 변화되었는지 찾은 후에 이를 클래스로 캡슐화 해야한다.
  + 로봇 예제에서 변화되면서 문제를 발생시키는 요인은 로봇의 이동방식과 공격방식의 변화이다.
  + 이를 캡슐화하려면 외부에서 구체적인 이동방식과 공격 방식을 담은 구체적인 클래스들을 은닉해야한다.
    - 공격과 이동을 위한 인터페이스를 각각 만들고 이들을 실제 실현한 클래스를 만들어야 한다.



* + - Robot 클래스가 이동 기능과 공격 기능을 이용하는 클라이언트 역할을 수행
      * 구체적인 이동, 공격 방식이 MovingStrategy와 AttackStrategy 인터페이스에 의해 캡슐화되어 있다.
      * 이 인터페이스들이 일종의 방화벽 역할을 수행해 Robot클래스의 변경을 차단해준다.
    - 스트레티지 패턴을 이용하면 새로운 기능의 추가(새로운 이동, 공격 기능)가 기존의 코드에 영향을 미치지 못하게 하므로 OCP를 만족하는 설계가 된다.
      * 이렇게 변경된 새로운 구조에서는 외부에서는 로봇 객체의 이동, 공격 방식을 임의대로 바꾸도록 해주는 setter 메서드가 필요하다.
      * setMovingStrategy, setAttackStrategy
      * 이렇게 변경이 가능한 이유는 상속 대신 집약 관계를 이용했기 떄문이다.

정리하면 공격에 관한 인터페이스를 생성, 미사일공격, 펀치공격에 대한 클래스 생성 후 공격에 관련된 인터페이스를 implements, 이동도 마찬가지로 이동에 관한 인터페이스를 생성, 날아다니는 클래스, 걷는 클래스를 생성후 이동에 관한 인터페이스를 implements