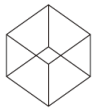
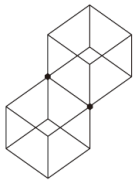
복습
Strategy/Command/Observer



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는



OCP를 만족하지 않는 설계

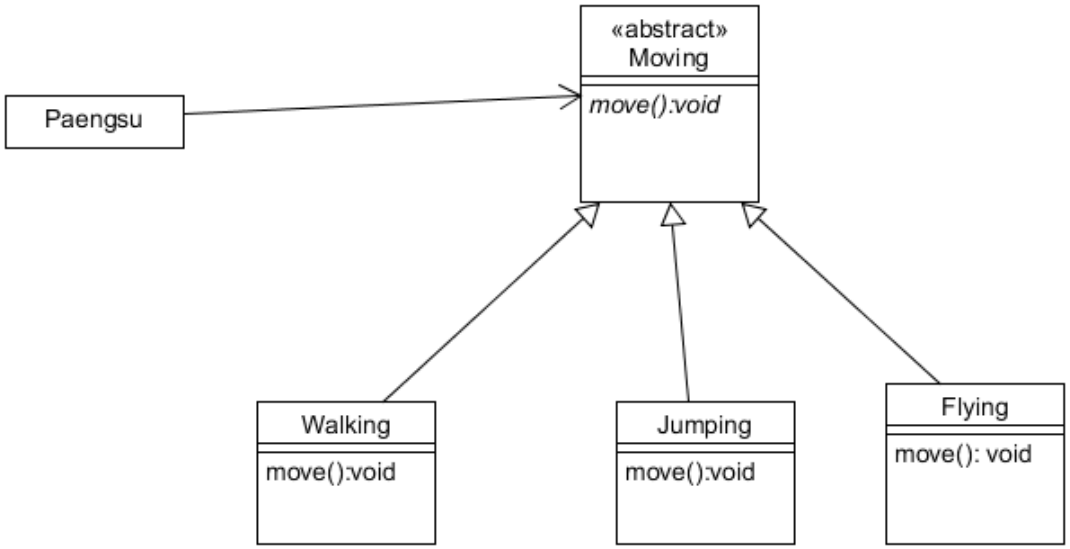
```
public class Paengsu {
    public void modeSelected(Mode mode) {
        switch (mode) {
            case WALK:
                walk();
                break;
            case JUMP:
                jump();
                break;
        }

    }
    private void walk() {
        System.out.println("Walking");
    }

    private void jump() {
        System.out.println("Jumping");
    }
}
```

OCP를 만족하는 설계

1. 변화되는 것을 식별
2. 변화되는 것을 클래스로 캡슐화
3. 변화되는 것을 아울릴 수 있는 개념을 인터페이스나 추상 클래스로 모델링하고 이를 사용하도록 설계



OCP를 만족하는 설계

- 스트래티지 패턴 적용
- E-class 코드 참조

Command 패턴

```
public class Bird {
    public void sing() {
        System.out.println("Bird is singing");
    }
}


public class MenuItem {
    public void buttonPressed(Bird b) {
        b.sing();
    }
}
```

Command 패턴을 통한 OCP 만족

- E-class에 Command 패턴 적용한 코드 참조
- 설명은 동영상 참조

Strategy 패턴과 Command 패턴 비교

- Strategy 패턴에서 변화되는 것은 동일한 목적을 가지는 알고리즘
- Strategy 패턴의 알고리즘을 실행하는 주체가 동일
- Command 패턴에서 변화되는 것은 요청(request)
- 각각의 요청은 목적이 다름
- Command 패턴에서 실제 request를 실행하는 주체는 요청마다 다를 수 있음(power를 실행하는 요청은 Tv, sing은 Bird)

Observer 패턴

- 관찰 대상과 관찰자와 일대 다 관계
- 관찰자의 추가에 따라 OCP 만족
- 관찰자는 관찰 대상에 대한 연관관계 설정 필요
- 자세한 사항은 동영상 참조