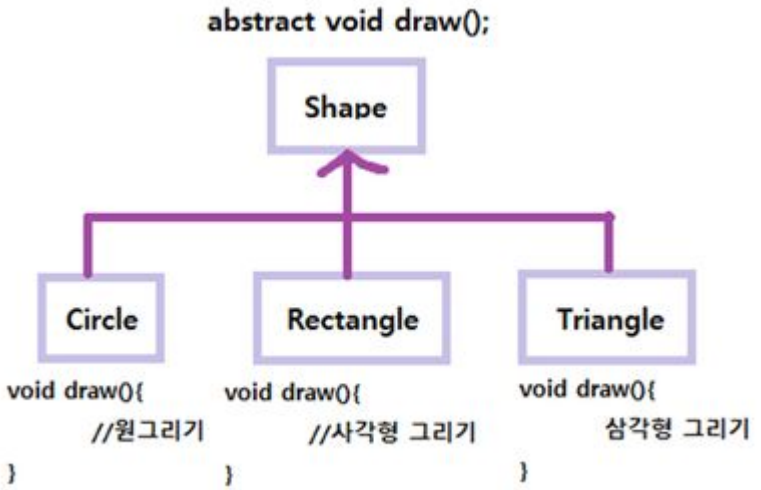
인터페이스와 추상클래스

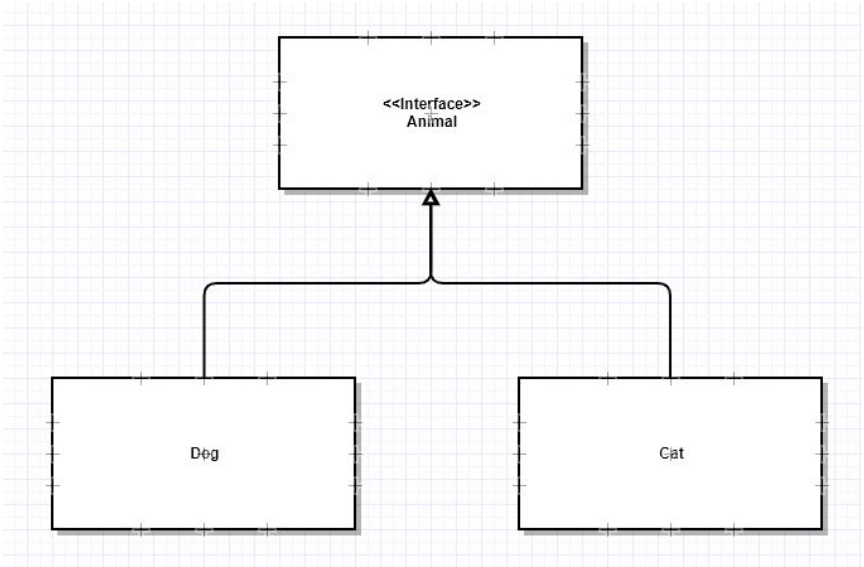
추상클래스

```
abstract class Shap{
    abstract void draw();

    void sketch() {
        System.out.println("sketch start");
    }
}

class Circle extends Shap{

    @Override
    void draw() {
        System.out.println("draw circle");
    }

}

class Square extends Shap{

    @Override
    void draw() {
        System.out.println("draw Square");
    }
}
```



인터페이스

```
interface Animal {
    public void cry();
    public void sleep();
}

class Dog implements Animal{

    @Override
    public void cry() {
        System.out.println("왈!! 왈왈!! 왈왈~!!! 왈월워훵왈!!! 멍멍!");
    }
    @Override
    public void sleep() {
        System.out.println("자신의 집에서 잠을 잡니다.");
    }

}

class Cat implements Animal{

    @Override
    public void cry() {
        System.out.println("야옹~~ 옹옹오오오오오옹~~~~~~~ 야오우오우오우옹~~~~~");
    }
    @Override
    public void sleep() {
        System.out.println("집사 얼굴 위에서 잠을 잡니다.");
    }

}
```



인터페이스

```
public interface 인터페이스명 {

  //상수
  타입 상수명 = 값;


  //추상 메소드
  타입 메소드명(매개변수, ... );


  //디폴트 메소드
  default 타입 메소드명(매개변수, ... ){
    //구현부
  }

  //정적 메소드
  static 타입 메소드명(매개변수) {
    //구현부
  }

}
```

상수 : 인터페이스에서 값을 정해줄테니 함부로 바꾸지 말고 제공해주는 값만 참조해라 **(절대적)**

추상메소드 : 가이드만 줄테니 추상메소드를 오버라이팅해서 재구현해라. **(강제적)**

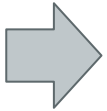
디폴트메소드 : 인터페이스에서 기본적으로 제공해주지만, 맘에 안들면 각자 구현해서 써라. **(선택적)**

정적메소드 : 인터페이스에서 제공해주는 것으로 무조건 사용 **(절대적)**

공통점

1. 둘 다 메서드의 선언만 있고 구현 내용이 없는 클래스이기 때문에 단독으로 객체를 생성할 수 없다.

2. 하위 클래스에게 메소드의 구현을 강제한다.



객체를 “추상화” 하는 방법

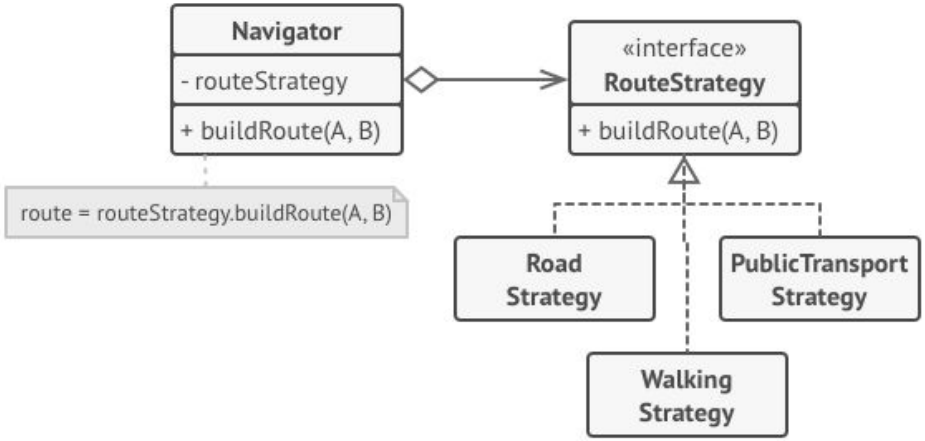
인터페이스 vs 추상 클래스 차이점

1. 추상 클래스(단일상속) / 인터페이스(다중상속)
2. 추상 클래스의 목적은 **상속을 받아서 기능을 확장시키는 것(부모의 유전자를 물려받는다)**
3. 인터페이스의 목적은 **구현하는 모든 클래스에 대한 특정한 메서드가 반드시 존재하도록 강제하는 역할**
4. 추상 클래스는 extends, 인터페이스는 implememts
5. Interface -> public abstract
   Abstract -> public, protected, private 메소드를 가질 수 있다

<추상클래스>
IS-KIND-OF
정의에 대한 추상화

<인터페이스>
IS-ABLE-TO
기능에 대한 추상화

[인터페이스 사례] 전략 패턴



네비게이션 시스템을 만든다고 가정

- 운전경로탐색 기능
- 도보경로탐색 기능을 추가
  .
- 자전거도로 경로탐색 기능을 추가

[추상클래스 사례] 템플릿 메서드 패턴

```
abstract public class AbstractTemplate {
    public final void templateMethod(){
        sequence1();
        sequence2();
        sequence3();
        sequence4();
    }

    abstract protected void sequence1();
    abstract protected void sequence2();
    abstract protected void sequence3();
    abstract protected void sequence4();
}
```