상속과 인터페이스

from “열혈강의 - 객체중심 Java”

기본적인 객체지향 프로그래밍 다음으로 배워야 하는 내용(1)

- 상속(Inheritance)
  - 특정 클래스를 구성할 때 기존 클래스의 데이터(속성)와 메소드를 상위(부모) 클래스로부터 그대로 물려받아서 중복적인 코드를 줄인다는 장점과 하나의 타입으로 여러 종류의 객체를 의미하는 추상화된 방식의 프로그래밍을 가능하게 하는 객체지향 기법
  - 용도
    - 이미 작성된 코드를 물려받는다
      - 계산기 -> 공학용 계산기
  - 상속과 관련된 용어
    - 부모클래스, 수퍼 클래스, Parent
    - 자식클래스, 서브 클래스, Child
    - 자식 클래스는 부모클래스의 모든 것을 물려받는다
      - 따라서 자식 클래스의 기능이 부모보다 더 많거나 같다

기본적인 객체지향 프로그래밍 다음으로 배워야 하는 내용 (2)

- 상속(Inheritance) (계속)
  - extends : 상속 표현
    - public class Wolf extends Animal {
  - 단일 상속 : Java는 오직 하나의 부모 클래스만을 가질 수 있음
    - 부모 클래스가 다른 클래스의 서브 클래스인 것은 가능
    - 그러나 부모클래스가 둘 이상이 될 수는 없음 (C++에서는 가능!)
  - 오버라이드(override) : 자식 클래스에서 부모클래스로부터 물려받은 메소드를 다시 정의
    - super : 부모 클래스를 부르는 키워드
      - 오버라이드 과정 중에서 부모 클래스에서 정의한 메소드를 먼저 수행한 후 기능 추가하는 방식
  - Private은 접근 제한의 의미

상속을 이용하면 빠른 개발이 가능할까?

- 변수의 타입과 실제 객체가 일치하지 않아도 된다는 장점
  - 구체적인 종류가 모두 결정되기 전에 그것들을 추상화한 클래스를 선언하면 모든 세부적인 사항이 결정되기 전에도 이용 가능
  - 다형성(polymorphism)
- 부모 클래스로 매개변수 선언이 가능
- 부모 클래스로 배열 선언 가능
- 상속은 개발 시간을 단축시킴
  - 새로운 기능이나 새로운 클래스를 개발하는 개발자
  - 기존 코드를 유지 보수하는 개발자

상속을 이용하면 if~else를 없앨 수 있음

- 오버라이드를 이용하면 간단한 코드로 if~else를 없앨 수 있음
- 다른 사람들이 반드시 오버라이드를 하도록 강제하는 방법 : 추상 클래스

추상 클래스

- 객체가 아닌, 타입으로만 존재하고, 추상메소드를 갖는 존재
  - 객체 생성이 목적이 아니고, 변수의 타입으로 선언하거나, 상속을 위해서만 존재하는 클래스
- 객체를 못 만들 뿐이지 상속의 기능은 그대로 있음
  - 추상 메소드도 있지만 구체적인 메소드도 있을 수 있음
- 추상 메소드는 부모의 부채
  - 따라서 반드시 구현해야 함
- 추상 메소드가 하나라도 있으면 추상 클래스가 되어야 함

상속을 꼭 써야 할까 ?

- 일반 상속 ? 추상 클래스 ?
  - 오버라이드가 필요 없을 때는 일반 상속을 사용
  - 부모클래스도 객체화될 때는 일반 상속을 사용
- 반드시 ~의 일종(is-a) 관계가 성립할 때에만 상속을 이용
- 단순 기능을 하나로 모으려고 상속을 이용하는 것은 위험
- 하위 클래스에서 오버라이드를 많이 할수록 설계가 잘못되지 않았는지 의심해야 합니다.

java.lang.Object

- 주요 메소드들
  - protected Object clone()
  - boolean equals(Object obj)
  - protected void finalize()
    - 절대로 override하지 말 것
  - Class <?> getClass()
  - int hashCode()
  - void notify()
  - void notifyAll()
  - String toString()
  - void wait()
  - void wait(long timeout)
  - void wait(long timeout, int nanos)

인터페이스란?

- 인터페이스에 대한 질문
  - 뭐 할 때 쓰는 건가?
    - 클래스를 만들기에 앞서 합의한 기능들
  - 어떻게 쓰는 건가?
    - 약속대로 구현
    - 이용하는 측에서는 약속대로 이용
  - 새로운 기능인가? 새로운 문법인가?
    - 상속과 더불어 다형성을 구현하는 객체지향 기법
  - 어떤 이로움이 있는가?
    - C#, ActionScript등 현대 언어들이 공통적으로 지원

문법으로 알아보는 인터페이스

- 인터페이스는 실제 객체를 의미하지 않음
  - 인터페이스는 기능들의 스펙 모음에 불과
    - 실제 코드는 없음
    - 추상 메소드와 상수만을 가질 수 있음
- 인터페이스의 상수는 private로 만들 수 없음
- 인테페이스에는 추상 메소드만 존재
  - 이름이 ~able로 끝나는 경우가 대부분
  - 'has-a' 관계
- 인터페이스는 객체 타입으로만 사용하고 실제 객체로 이용되지는 않음
- 인터페이스를 구현한 객체를 만들 때 'implements 인터페이스 이름'
- 인터페이스는 하나의 클래스에 여러 개를 붙일 수도 있음
  - public class Phone3G **implements** VoiceCalls, VisualCalls {
- 다른 인터페이스들을 조합하여 새로운 인터페이스를 만들 수도 있음
  - public interface PerfectPhone **extends** Camera, MP3, DMP, VoiceCalls, VisualCalls {

인터페이스 vs. 추상 클래스

- 공통점
  - 둘다 추상 메소드를 가지고 있음
  - 둘다 객체 생성은 불가능하고 타입으로만 사용됨
- 차이점
  - 인터페이스는 스펙이나 기능을 정의하고자 쓰지만, 추상 클래스는 '상속+약간의 강제성'이 목적
  - 인터페이스에는 상수와 추상 메소드만 존재하지만, 추상 클래스는 실제 변수나 메소드도 있을 수 있음
    - 인터페이스는 부채만 있지만, 추상 클래스는 재산도 있음
  - 인터페이스는 다중 상속(?)이 가능하지만, 추상 클래스는 단일 상속만 가능
- 추상클래스(상속)이 더 나은 상황
  - 거의 모든 기능은 그대로 물려주면서 부분적으로 한, 두가지 기능이 다를 때
- 인터페이스가 더 나은 상황
  - 추상 클래스(상속)가 적합한 때를 제외한 나머지 상황