2023-03-15

상속의 정의와 장점

기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것.

두 클래스를 부모과 자식으로 관계를 맺어주는 것.

자식은 부모의 모든 멤버를 상속받는다.(생성자,초기화블럭 제외)

자식의 멤버개수는 부모보다 적을 수 없다. (같거나 많다.)

```
class 자식클래스 extends 부모클래스 {
  // ...
}
```

클래스간의 관계 - 상속관계(inheritance)

공통부분은 부모에서 관리하고 개별부분은 자식에서 관리한다.

부모의 변경은 자식에게 영향을 미치지만, 자식의 변경은 부모에게 영향을 끼치지않는다.

클래스간의 관계 - 포함관계(composite)

▼ 포함이란?

- 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
- 작은 단위의 클래스를 먼저 만든 뒤 조합하여 하나의 커다란 클래스를 만든다.

클래스간의 관계 결정하기 - 상속 vs 포함

- 가능한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게한다.
- is-a 와 has-a를 가지고 문장을 만들어서 결정한다

```
원(Circle)은 점(Point)이다. - Circle is a Poin.
원(Circle)은 점(Point)을 가지고 있다. - Circle has a Point

상속관계 - '~은 ~이다.(is-a)'
포함관계 - '~은 ~을 가지고 있다.(has-a)'

class Circle extends Point{
  int r; // 반지름(radius)
}
```

```
class Circle {
  Point c = new Point(); // 원점
  int r; // 반지름
}

class Point {
  int x;
  int y;
}
```

단일상속(single inheritance)

Java는 단일상속만을 허용한다.

비중이 높은 클래스 하나만 상속관계로, 나머지는 포함관계로 한다.

```
class TVCR extends TV, VCR {
  // 이와 같은 표현은 Java에서 허용하지 않는다.
}
```

Object클래스 - 모든 클래스의 최고 부모

- 부모가 없는 클래스는 자동적으로 Object클래스를 상속받는다
- 상속계층도의 최상위는 Object클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다.

  toString(),equals(Object obj), hashCode(), …

오버라이딩(overriding)

부모클래스에게 상속받은 메서드의 내용을 상속받은 클래스에 맞게 변경하는 것.

- 조건
  - 선언부가 같아야한다.(이름, 매개변수, 리턴타입)
  - 접근제어자를 좁은 범위로 변경할 수 없다.
    - 부모 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로만 변경가능
  - 부모클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

```
class Parent {
  void parentMethod() { }
```

```
}
class Child extends Parent {
  void parentMethod() { } // 오버라이딩
  void parentMethod(int i) { } // 오버로딩

  void childMethod() { }
  void childMethod(int i) { } // 오버로딩
  void childMethod() { } // 에러!!! 중복정의
```

오버로딩 - 기존에 없는 새로운 메서드를 정의하는 것(new)

오버라이딩 - 상속받은 메서드의 내용을 변경하는 것(change, modify)

참조변수

this - 인스턴스 자신을 가리키는 참조변수. 인스턴스에 주소저장. 모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재

super - this와 같음. 부모의 멤버와 자신의 멤버를 구별하는 데 사용

Object클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자(같은 클래스의 다른 생성자 또는 부모생성자)를 호출해야한다. 그렇지 않으면 컴파일러가 자동으로 super();을 생성자의 첫줄에 삽입.

```
Date d1 = new Date();
Object d2 = new Date();
둘의 차이점은 접근할 수 있는 클래스의 범위가 다르다.
// Object:부모 클래스에서 정의한 멤버에만 접근 가능. 자식 클래스에서 정의한 멤버에는 접근불가
```

다형성

여러가지 형태를 가질 수 있는 능력

하나의 참조변수로 여러타입의 객체를 참조할 수 있는 것이다. 즉 부모타입의 참조변수로 자식타입의 객체를 다룰 수 있는것이 다형성이다.

- 참조변수의 형변환
  - 서로 상속관계에 있는 타입간의 형변환만 가능하다
  - 자식타입에서 부모타입으로 형변환 하는 경우, 형변환 생략가능.

```
자식타입 -> 부모타입 (Up-casting) : 형변환 생략가능
자식타입 <- 부모타입 (Down-casting) : 형변환 생략불가
```

참조형 매개변수는 메서드 호출시, 자신과 같은 타입 또는 자식타입의 인스턴스를 넘겨줄 수 있다.

제어자(modifier)

- 접근제어자
  - public, protected, default(제어자 생략시를 말함), private
    - 클래스 : public, (default)
    - 멤버변수, 메서드, 생성자메서드 : public, protected, default, private
- 활용제어자
  - static, final, abstract …
    - static : 정적, 고정.

      멤버변수, 메서드, 블럭
    - final : 마지막의, 변경할 수 없는.

      클래스, 멤버변수, 메서드, 지역(매개)변수
    - abstract : 추상적인, 미완성의

      클래스, 메서드

접근 제어자(access modifier)

접근 제어자가 사용될 수 있는 곳 - 클래스, 멤버변수, 메서드, 생성자

private - 같은 클래스 내에서만 접근이 가능하다.

default - 같은 패키지 내에서만 접근이 가능하다.

protected - 같은 패키지 내에서, 그리고 다른 패키지의 자식클래스에서 접근이 가능하다.

public - 접근 제한이 전혀 없다.

- 일반적으로 생성자의 접근 제어자는 클래스의 접근 제어자와 일치한다.
- 생성자에 접근 제어자를 사용함으로써 인스턴스의 생성을 제한할 수 있다.

제어자의 조합

대상	사용가능한 제어자
클래스	public, (default), final, abstract

메서드	모든 접근 제어자, final, abstract,static
멤버변수	모든 접근 제어자, final, static
지역변수	final

1. 메서드에 static과 abstract를 함께 사용할 수 없다

- static메서드는 구현부(몸통)가 있는 메서드에만 사용할 수 있다.

2. 클래스에 abstract와 final을 동시에 사용할 수 없다.

- 클래스에 사용되는 final은 클래스를 확장할 수 없다는 의미이고, abstract는 상속을 통해서 완성되어야 한다는 의미이므로 서로 모순되기 때문이다.

3. abstract메서드의 접근제어자가 private일 수 없다.

- abstract메서드는 자식클래스에서 구현해주어야하는데 접근 제어자가 private이면, 자식클래스에서 접근할 수 없기 때문이다.

4. 메서드에 private과 final을 같이 사용할 필요는 없다.

- 접근 제어자가 private인 메서드는 오버라이딩 될 수 없기 때문이다. 이 둘 중 하나만 사용해도 의미가 충분하다.

추상메서드(abstract Method)

메서드의 헤더만 정하고 코드 블록을 생략한 메서드

abstract메서드를 1개 이상 정의한 클래스는 반드시 abstract로 정해야한다.

자손에 의해서 오버라이딩해야하는 메서드를 뜻한다.

추상클래스(abstract Class)

미완성클래스로 객체 생성은 불가하며 상속으로만 사용가능한 클래스

0개 이상의 abstract 메서드가 존재할 수 있다.

```
abstract class Player {
  int currentPos; // 현재 Play되고 있는 위치를 저장하기 위한 변수

  Player() { // 추상클래스도 생성자가 있어야 한다.
    currentPos = 0;
  }
  abstract void play(int pos); // 추상메서드
  abstract void stop(); // 추상메서드

  void play() {
    play(currentPos); // 추상메서드를 사용할 수 있다.
  }
}
```

- 선언부만 있고 구현부(몸통,body)가 없는 메서드

```
/* 주석을 통해 어떤 기능을 수행할 목적으로 작성하였는지 설명한다. */
abstract 리턴타입 메서드이름();

Ex)
/* 지정된 위치(pos)에서 재생을 시작하는 기능이 수행되도록 작성한다. */
abstract void play(int pos);
```

- 꼭 필요하지만 자식마다 다르게 구현될 것으로 예상되는 경우 사용
- 추상클래스를 상속받는 자손클래스에서 추상메서드의 구현부를 완성해야한다.

```
abstract class Player {
  ...
  abstract void play(int pos); // 추상메서드
  abstract void stop(); // 추상메서드
  ...
}

class AudioPlayer extends Player {
  void play(int pos) { /* 내용 생략 */ }
  void stop() { /* 내용 생략 */ }
}
abstract class AbstractPlayer extends Player {
  void play(int pos) { /* 내용 생략 */ }
}
```

- 추상클래스의 작성

여러 클래스에 공통적으로 사용될 수 있는 추상클래스를 바로 작성하거나 기존클래스의 공통 부분을 뽑아서 추상클래스 생성

```
abstract class Unit{
  int x, y;
  abstract void move(int x, int y);
  void stop() { /* 현재 위치에 정지 */ }
}

class Marine extends Unit { // 보병
  void move(int x, int y) { /* 지정된 위치로 이동 */ }
  void stimPack() { /* 스팀팩을 사용한다. */ }
}
class Tank extends Unit { // 탱크
  void move(int x, int y) { /* 지정된 위치로 이동 */ }
  void changeMode() { /* 공격모드를 변환한다. */ }
}
class DropShip extends Unit { // 수송선
```

```
  void move(int x, int y) { /* 지정된 위치로 이동 */ }
  void load() { /* 선택된 대상을 태운다. */ }
  void upload() { /* 선택된 대상을 내린다. */ }
}
```

객체생성이 목적이면 abstract은 NO!
