

chapter7 객체지향(2)

ch7-1,2 상속

기존의 클래스로 새로운 클래스를 작성 하는 것 (코드의재사용)
두 클래스를 부모와 자식으로 관계를 맺어주는 것

```
class Parent { }  
class Child extends Parent { }
```

자손은 조상의 모든 멤버를 상속 받는다 (생성자, 초기화블럭 제외)
자손의 변경은 조상에 영향을 미치지 않는다

ch7-3,4 포함 관계

포함이란? 다른 클래스를 멤버로 참조변수를 선언하는 것
클래스간의 관계 결정

ch7-5 단일 상속

- java는 단일 상속만을 허용한다. (다중 상속시 충돌위험)
- 비중이 높은 클래스 하나만 상속관계로 하고 나머지는 포함관계로 메서드를 호출하여 사용하면 다중상속 효과

```
class Tv {  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { ++channel; }  
    void channelDown() { --channel; }  
}  
  
class DVD {  
    boolean power;  
  
    void power() { power = !power; }  
    void play() { /* 내용 생략 */ }  
    void stop() { /* 내용 생략 */ }  
    void rew() { /* 내용 생략 */ }  
    void ff() { /* 내용 생략 */ }  
}  
  
class TvDVD extends Tv {  
    DVD dvd = new DVD();  
  
    void play() {  
        dvd.play();  
    }  
  
    void stop() {  
        dvd.stop();  
    }  
  
    void rew() {  
        dvd.rew();  
    }  
  
    void ff() {  
        dvd.ff();  
    }  
}
```

ch7-6 Object 클래스 - 모든 클래스의 조상

- 부모가 없는 클래스는 자동적으로 Object클래스를 상속 받게 된다. (컴파일러가 자동으로)
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속 받는다. (9장)

ch7-7 오버라이딩

- 상속받은 조상의 메서드를 자신에 맞게 변경하는 것
(내용만 변경 가능, 선언부는 변경 불가능)

```
class MyPoint3 {  
    int x;  
    int y;  
    String getLocation() {  
        return "x:"+x+", y:"+y;  
    }  
}  
  
class MyPoint3D extends MyPoint3 {  
    int z;  
  
    // 조상의 getLocation()을 오버라이딩  
    String getLocation() {  
        return "x:"+x+", y:"+y+", z:"+z;  
    }  
}
```

상속받았지만 덮어 씌워 내용변경

ch7-8 오버라이딩의 조건

1. 선언부가 조상 클래스의 메서드와 일치해야 한다.
(반환타입, 메서드이름, 매개변수 목록)
2. 접근 제어자를 조상 클래스의 메서드보다 좁은 범위로 변경 할 수 있다.
3. 예외는 조상 클래스의 메서드 보다 많이 선언할 수 없다. (같거나 적어야 한다)

ch7-9 오버로딩 vs 오버라이딩

오버로딩 - 기존에 없는 새로운 메서드를 정의하는 것
(이름은 같음, 상속과 관계X)

오버라이딩 - 상속받은 메서드의 내용을 변경하는 것

ch7-10 참조변수 super (=this)

- 객체 자신을 가리키는 참조변수.
- 인스턴스 메서드(생성자)내에만 존재(static에서 사용불가)
- 조상의 멤버를 자신의 멤버와 구별 할 때 사용
(*this는 lv, iv구별에 사용)

ch7-11 super() - 조상의 생성자

- 조상의 생성자를 호출할 때 사용
(상속시 생성자, 초기화 블록은 상속이 안된다.)
- 조상의 멤버는 조상의 생성자를 호출해서 초기화
(자손의 생성자는 자신이 생성한 멤버만 초기화 해야한다.
조상의 멤버 초기화 할 수 있지만 표준이 아님!)

```
class Point {  
    int x, y;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
class Point3D extends Point {  
    int z;  
  
    Point3D(int x, int y, int z) {  
        this.x = x; // 조상의 멤버를 초기화  
        this.y = y; // 조상의 멤버를 초기화  
        this.z = z;  
    }  
}  
  
Point3D(int x, int y, int z) {  
    super(x, y); // 조상클래스의 생성자 Point(int x, int y)를 호출  
    this.z = z; // 자신의 멤버를 초기화  
}
```

+추가조건

- 생성자의 첫 줄에는 반드시 다른 생성자 (this(), super())를 호출해야 한다. 안하면 컴파일러가 생성자의 첫줄에 super();를 삽입 (때문에 모든 생성자에 기본생성자 생성 필수)

ch7-12 패키지

- 서로 관련된 클래스의 묶음
- 클래스는 클래스파일, 패키지는 폴더.
- 클래스의 실제 이름은 패키지를 포함.(java.lang.String)

ch7-13 패키지 선언

패키지네 소스파일의 첫 번째 문장으로 단 한번 선언
패키지 선언이 없으면 (default package)

ch7-14 클래스패스

- 클래스 파일(*.class)의 위치를 알려주는 경로
- 환경변수 classpath로 관리, 경로간의 구분은 ‘.’를 사용 classpath(환경변수)에 패키지의 루트를 등록해줘야함

ch7-15 import문 (ctrl+shift+o)

- 자바 클래스를 사용할 때 패키지이름을 생략 할 수 있다.
- 컴파일러에게 클래스가 속한 패키지를 알려준다.
- java.lang패키지는 자바의 기본 핵심 패키지이기 때문에 import없이 사용 가능하다. (ex-java.lang.toString)

```
import 패키지명.클래스명;
또는
import 패키지명.*;
```

import문은 패키지문과 클래스선언의 사이에 선언한다.

ch7-16 static import문

static멤버를 사용할 때 클래스 이름을 생략 할 수 있게 해 준다. (15장에서 자세히..)

```
import static java.lang.Integer.*; // Integer클래스의 모든 static메서드
import static java.lang.Math.random; // Math.random()만 필요 인라인
import static java.lang.System.out; // System.out을 out만으로 참조가능
```



코드의 간결화를 위해 쓴다.

ch7-17-제어자

클래스와 클래스의 멤버(변수,메서드)에 부가적 의미부여

접근 제어자	public, protected, (default), private
그 외	static, final, abstract, native, transient, synchronized, volatile, strictfp
-하나의 대상에 여러 제어자를 같이 사용가능	

ch7-18 static - 클래스의, 공통적인

제어자	대상	의 미
static	멤버변수	- 모든 인스턴스에 공통적으로 사용되는 클래스 변수가 된다. - 클래스 변수는 인스턴스를 생성하지 않고도 사용 가능하다. - 클래스가 메모리에 로드될 때 생성된다.
	메서드	- 인스턴스를 생성하지 않고도 호출이 가능한 static 메서드가 된다. - static메서드 내에서는 인스턴스멤버들을 직접 사용할 수 없다.

ch7-19 final - 마지막의, 변경될 수 없는

제어자	대상	의 미
final	클래스	변경될 수 없는 클래스, 확장될 수 없는 클래스가 된다. 그래서 final로 지정된 클래스는 다른 클래스의 조상이 될 수 없다.
	메서드	변경될 수 없는 메서드, final로 지정된 메서드는 오버라이딩을 통해 재정의 될 수 없다.
	멤버변수	변수 앞에 final이 붙으면, 값을 변경할 수 없는 상수가 된다.
	지역변수	

ch7-20 abstract - 추상의, 미완성의

제어자	대상	의 미
abstract	클래스	클래스 내에 추상 메서드가 선언되어 있음을 의미한다.
	메서드	선언부만 작성하고 구현부는 작성하지 않은 추상 메서드임을 알린다.

미완성 클래스는 객체생성불가, 추상클래스를 상속받아 완전한 클래스를 만든후에 객체생성 가능

ch7-21 접근제어자

private	같은 클래스 내에서만 접근이 가능하다.
(default)	같은 패키지 내에서만 접근이 가능하다.
protected	같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다.
public	접근 제한이 전혀 없다.
접근제한없음 > 같은 패키지+자손 > 같은 패키지 > 같은 클래스	
public > protected > (default) > private	

class앞에는 public과 (default) 만 가능

멤버 앞에는 4가지 다 가능

ch7-22 캡슐화와 접근 제어자

접근 제어자 사용 이유

- 외부로부터 데이터를 보호 목적
- 외부에서는 불필요한, 내부적으로만 사용하는 부분을 감추기 위해서 사용한다.
- 매개변수가 있는 메서드는 반드시 유효성검사를 한다.

ch7-23 다형성

- 조상타입 참조변수로 자손 타입 객체를 다루는 것 리모콘을 변경하는 것
- 객체와 참조변수 타입이 일치할 때, 않을 때 차이??
- >접근할 수 있는 멤버가 제한된다..?(조상것만 사용가능)
- 사용할 수 있는 멤버의 개수가 달라진다.

ch7-24,25 참조변수의 형변환

- 사용할 수 있는 멤버의 개수를 조절하는 것 (기본형의 형변환은 값이 달라진다.)
- 조상, 자손 관계의 참조변수는 서로 형변환 가능

ch7-24,25 참조변수의 형변환(2)

- 객체가 없어도 형변환은 문제가 없다. (실행시 NullPointerException발생)
- 참조 변수가 가리키는 실제 객체가 중요!! (컴파일은 문제없지만 실행시 에러발생..!)
- 실제 객체가 가진 멤버의 갯수 내에서만 형변환 해야함.

ch7-26 instanceof 연산자

- 참조변수의 형변환 가능여부 확인에 사용.
- 가능하면 true반환 (A instanceof B)
- (지금 들어오는 것(A)는 B의 자손이니?)

ch7-27 매개변수의 다형성

- 다형성의 장점 1.다형적 매개변수 2.하나의 배열로 여러종류 객체 다루기

다형성 정리

- Tv t = new Smart Tv();
- 참조변수의 형변환 - 리모콘 바꾸기
- instance of 연산자 - 형변환 가능여부

1.다형적 매개변수

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


->따라서 조상타입으로 매개변수타입을 지정하면 자손들의 인스턴스를 한번에 넘겨줄 수 있다!

ch7-29,30 여러 종류의 객체를 배열로 다루기

-조상타입의 배열에 자손들의 객체를 담을 수 있다.

(하나의 배열에는 같은 타입의 객체만 저장할 수 있지만 다형성을 이용하면 하나의 배열에 여러 종류 객체를 저장할 수 있다.)

```
Product p1 = new Tv();
Product p2 = new Computer();
Product p3 = new Audio();
```



```
Product p[] = new Product[3];
p[0] = new Tv();
p[1] = new Computer();
p[2] = new Audio();
```

가변배열기능을 가지고 있는 Vector 클래스는 Object배열을 멤버로 가지고 있어 모든종류의 객체저장 가능하고 배열의 길이를 직접관리하지 않아도 알아서 길이를 조절해준다. add()로 추가만 하면 됨

ch7-31 추상클래스

- 미완성 설계도. 미완성 메서드를 갖고 있는 클래스

(멤버가 부족하거나 메서드의 구현부가 없음)

-다른 클래스 작성에 도움을 주기위한 것. 객체 생성 불가

-상속을 통해 추상메서드를 완성하고 객체를 생성해야한다

-추상클래스를 참조변수타입으로 지정하여 사용할 수 있다. 미완성이지만 상속을 통해 추상클래스가 가지고 있는 메서드 호출 가능

ch7-31,32,33 추상메서드

-미완성 메서드. 구현부가 없는 메서드

(주석을 통해 어떤 기능을 가지고 있는 메서드인지 설명)

abstract 리턴타입 메서드이름();

-추상클래스에 추상메서드를 2개 상속 받았으면 자손클래스에서 2개 다 구현해야 완성된다.(보이지않지만 상속받음) 1개만 구현하는 경우 미완성이므로 class앞에 abstract를 붙여야 한다.

-추상메서드 호출이 가능하다(호출할때는 선언부만 필요)

ch7-34 추상클래스의 작성

-여러 클래스에 공통적으로 사용될 수 있는 기능을 바로 추상클래스로 작성하거나 기존 클래스의 공통 부분을 뽑아서 추상클래스로 만든다.

(선언부는 일치하고 내용이 불일치 할 경우)

장점 - 설계도를 쉽게 작성 할 수 있고 중복코드 제거가능
코드관리(변경) 용이

ch7-34 추상클래스의 작성

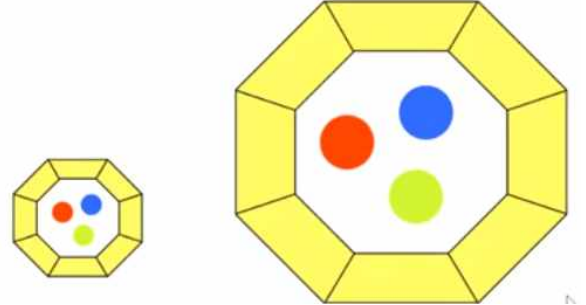
-추상클래스를 단계별로 만들면서 클래스를 구체화

-추상화된 코드는 구체화된 코드보다 유연하다

ch7-35 인터페이스

-추상메서드의 집합 (프로그래밍 관점)

-구현된 것이 전혀 없는 설계도.껍데기(모든멤버가 public)



데이터 보호를 위해 메서드를 이용하여 변수에 접근

인터페이스와 추상클래스의 차이

- 추상클래스는 일반 메서드인데 추상 메서드를 가지고 있는 클래스 이고, 인터페이스는 추상메서드 밖에 없다.

(생성자, 멤버변수를 가질 수 없다. 상수가능)

```
interface 인터페이스이름 {
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수목록);
}
```

인터페이스에서 상수는 public static final이 생략 가능하고, 추상메서드는 public abstract 생략가능하다.

- 두 객체간의 '연결,소통'을 돕는 '중간역할'이다.

껍데기를 통해서 어떤 기능을 쉽게 조작하기 위해 중간역할의 인터페이스가 필요하다.

ch7-36 인터페이스의 상속

-인터페이스의 조상은 인터페이스만 가능(Object가 아님)

-다중 상속이 가능(추상 메서드는 충돌해도 문제없음 내용이 없기 때문에! 어느쪽을 받던 노상관노노)

ch7-37 인터페이스의 구현

-인터페이스에 정의된 추상 메서드를 완성하는 것

```
class 클래스이름 implements 인터페이스이름 {
    // 인터페이스에 정의된 추상메서드를 모두 구현해야 한다.
}
```

클래스가 인터페이스를 구현(완성)했다.

ch7-38 인터페이스를 이용한 다형성

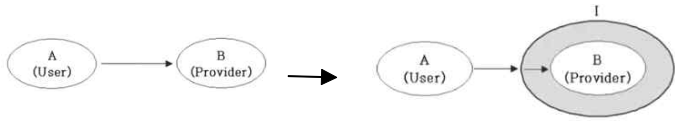
-인터페이스도 구현 클래스의 부모가 될 수 있다.

-인터페이스 구현을 통해서 다중상속의 문제를 해결하면서도 다중상속과 같은 효과를 냄

-인터페이스를 메서드의 리턴타입으로 지정할 수 있다.

ch7-39 인터페이스의 장점

- 선언과 구현을 분리시킬 수 있게 한다.
기능 변경에 유리하고 유연한 코드가 된다. (느슨한 결합)
객체지향을 쓰는 이유는 변경에 유리하기 때문이다.



- 개발시간을 단축 할 수 있다.
A가 B를 사용하기 위해서는 B가 완성될 때까지 A는 기다려야 하지만 오른쪽 그림은 interface접데기만 있으면 호출, 사용이 가능하다.

- 표준화가 가능하다
ex) JDBC (인터페이스 집합)

- 서로 관계없는 클래스들의 관계를 맺어준다.

ch7-40,41 디폴트 메서드와 static 메서드

- 인터페이스에 디폴트 메서드, static메서드 추가(JDK1.8)
- 인터페이스에 새로운 메서드(추상)를 추가하기 어려움
(인터페이스를 구현했던 클래스들이 새로 추상메서드를 구현해야 하기 때문)
해결책 =>디폴트 메서드(인터페이스에 구현부가 있는 메서드를 추가 할 수 있도록 함)

디폴트 메서드가 기존메서드와 충돌 할 때의 해결책

1. 여러 인터페이스의 디폴트 메서드 간의 충돌
- 인터페이스를 구현한 클래스에서 디폴트 메서드를 오버라이딩해야 한다.
2. 디폴트 메서드와 조상 클래스의 메서드 간의 충돌
- 조상 클래스의 메서드가 상속되고, 디폴트 메서드는 무시된다.

-----객체지향 END

ch7-42 내부클래스

클래스 안의 클래스

- 장점: 1.내부클래스에서 외부 클래스의 멤버들을 쉽게 접근
(객체 생성 없이도 접근, private도 가능)
2.코드의 복잡성을 줄일 수 있다.(캡슐화)
3.내부 클래스의 보안성

ch7-43,44 내부 클래스의 종류와 특징

내부 클래스	특 징
인스턴스 클래스 (instance class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 인스턴스멤버처럼 다루어진다. 주로 외부 클래스의 인스턴스멤버들과 관련된 작업에 사용될 목적으로 선언된다.
스태틱 클래스 (static class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 static멤버처럼 다루어진다. 주로 외부 클래스의 static멤버, 특히 static메서드에서 사용될 목적으로 선언된다.
지역 클래스 (local class)	외부 클래스의 메서드나 초기화블럭 안에 선언하며, 선언된 영역 내부에서만 사용될 수 있다.
익명 클래스 (anonymous class)	클래스의 선언과 객체의 생성을 동시에 하는 이름없는 클래스(일회용)

ch7-45~48 내부클래스의 제어자와 접근성

- 원래 클래스 앞에는 public과 (default)만 붙일 수 있지만 내부 클래스의 제어자는 모든 제어자 사용가능
- static 내부 클래스만 static멤버를 가질 수 있다.
(final static은 상수이므로 허용)
- 메서드 내 지역클래스는 메서드의 값이 바뀌는 지역변수
는 접근불가 하지만 값이 바뀌지 않는 지역변수는 상수로 간주 하여 접근 가능하다
(지역 클래스가 메서드보다 오래 존재가능)
--> 상수는 constant pool에 따로 저장
메서드가 종료되어도 살아남아있다.

ch7-49 외부클래스 밖에서 내부클래스

내부클래스에 접근 하기위해 외부클래스 객체 생성 후 내부클래스에 접근한다.

static 내부클래스의 멤버는 외부클래스의 객체를 생성 하지 않아도 된다.

내부클래스만 객체생성

->외부클래스이름.내부클래스 변수 = new 외부클래스이름.내부클래스();

ch7-50 익명 클래스

- 이름 없는 일회용 클래스. 정의와 생성을 동시에

```
new 조상클래스이름() {  
    // 멤버 선언  
}  
  
또는  
  
new 구현인터페이스이름() {  
    // 멤버 선언  
}
```

```
Object iv = new Object(){ void method(){ } }; // 익명 클래스  
AWT (java 윈도우 프로그래밍)에서 많이 씀
```