

객체 지향 프로그래밍4\_상속1



# 상속 (Inheritance)

## ■ 상속

- 기존의 클래스를 재사용해서 새로운 클래스를 작성.
- 자손은 조상의 모든 멤버를 상속받는다.(생성자, 초기화블럭, private 멤버 제외)
- 자손의 멤버개수는 조상보다 적을 수 없다.(같거나 많다.)

```
Class Sub extends Super{  
    // ...  
}
```

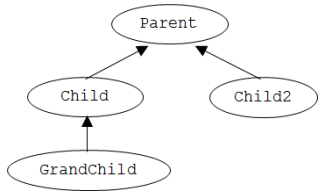
```
class Point {  
    int x;  
    int y;  
}
```

```
class Point3D {  
    int x;  
    int y;  
    int z;  
}
```

```
class Point3D extends Point {  
    int z;  
}
```

## ■ 클래스간의 관계 – 상속관계(inheritance)

- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리.
- 조상의 변경은 자손에 영향을 미치지만,  
자손의 변경은 조상에 아무런 영향을 미치지 않는다.



```
class Parent {}  
class Child extends Parent {}  
class Child2 extends Parent {}  
class GrandChild extends Child {}
```

## ■ 클래스간의 관계 – 포함관계(composite)

- 포함(composite)이란? 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
- 작은 단위의 클래스를 먼저 만들고, 이 들을 포함하는 클래스를 만든다.

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

```
class Circle {  
    int x; // 원점의 x좌표  
    int y; // 원점의 y좌표  
    int r; // 반지름(radius)  
}  
  
class Circle {  
    Point c = new Point(); // 원심  
    int r; // 반지름(radius)  
}
```



# 상속 관계 상속 vs. 포함

- 가능한 한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게 한다.
- '-is-a'와 '-has-a'를 가지고 문장을 만들어 본다.

상속관계 - '~은 ~이다.(is-a)'

포함관계 - '~은 ~을 가지고 있다.(has-a)'

원(Circle)은 점(Point)이다. - Circle **is a** Point.

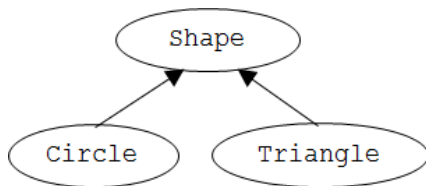
원(Circle)은 점(Point)을 가지고 있다. - Circle **has a** Point.

```
class Shape {
    String color = "blue";
    void draw() {
        // 도형을 그린다.
    }
}
```

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

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```



```
class Circle extends Shape {
    Point center;
    int r;

    Circle() {
        this(new Point(0,0),100);
    }

    Circle(Point center, int r) {
        this.center = center;
        this.r = r;
    }
}
```

```
class Triangle extends Shape {
    Point[] p;

    Triangle(Point[] p) {
        this.p = p;
    }

    Triangle(Point p1, Point p2, Point p3) {
        p = new Point[]{p1,p2,p3};
    }
}
```

```
Circle c1 = new Circle();
Circle c2 = new Circle(new Point(150,150),50);

Point[] p = {new Point(100,100),
             new Point(140,50),
             new Point(200,100)};

Triangle t1 = new Triangle(p);
```

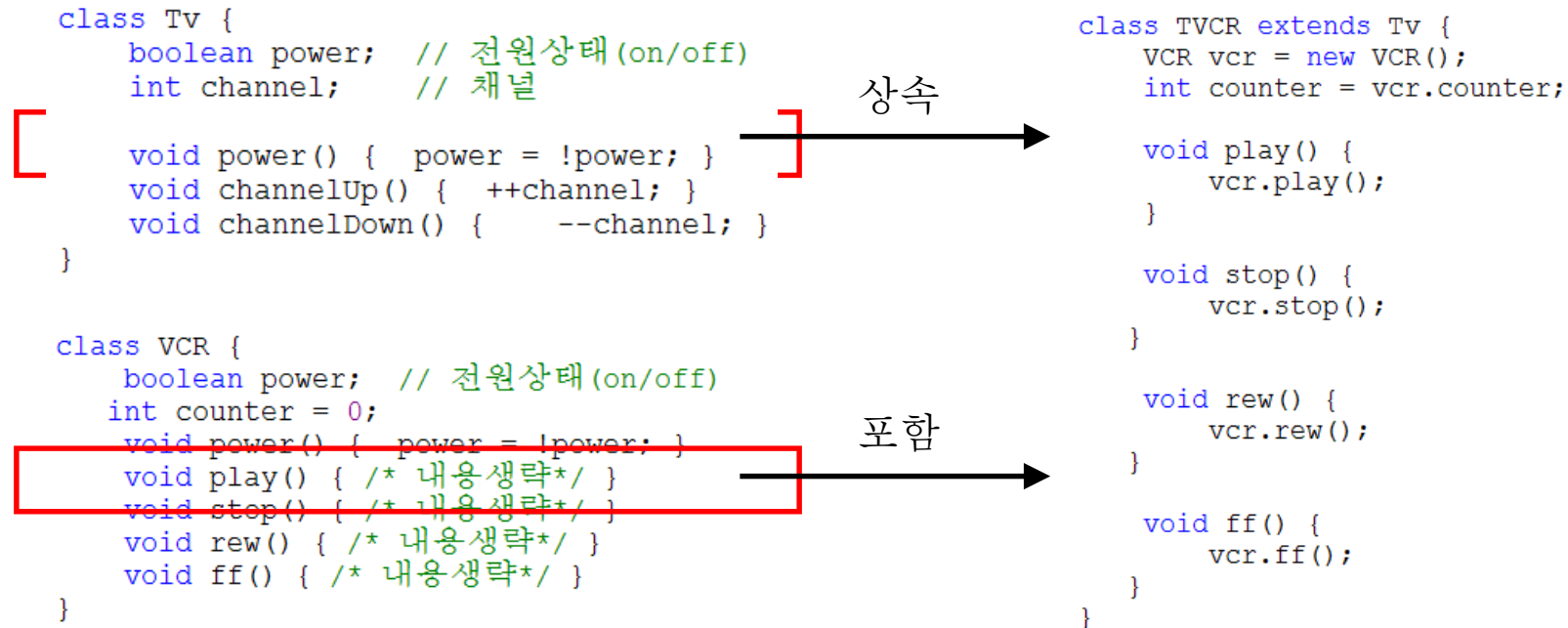


# 상속 단일 상속

## ■ 단일상속(single inheritance)

- Java는 단일상속만을 허용한다.(C++은 다중상속 허용)
- 비중이 높은 클래스 하나만 상속관계로, 나머지는 포함관계로 한다.

```
Class TVCR extends TV, VCR { // error
    // ...
}
```



## ■ Object클래스 - 모든 클래스의 최고 조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.
- 상속계층도의 최상위에는 Object클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다  
toString(), equals(Object obj), hashCode(), ...

```
class Tv {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```

→

```
class Tv extends Object {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```

docs.oracle.com/javase/7/docs/api/index.html

Java™ Platform  
Standard Ed. 7

1

All Classes

Packages

java.applet

java.awt

java.awt.color

java.awt.datatransfer

java.awt.dnd

java.awt.event

java.awt.font

java.awt.geom

java.awt.im

java.awt.im.spi

java.awt.image

java.awt.image.renderable

java.awt.print

java.beans

java.beans.beancontext

java.io

java.lang

java.lang.annotation

java.lang.instrument

java.lang.invoke

java.lang.management

SecurityManager

Short

StackTraceElement

StrictMath

String

StringBuffer

StringTokenizer

2

OverviewPackageClassUseTreeDeprecatedIndexHelp

Prev ClassNext ClassFramesNo Frames

Summary: Nested | Field | Constr | MethodDetail: Field | Constr | Method

java.lang

3

Class System

java.lang.Object

java.lang.System

public final class System

extends Object

The System class contains several useful class fields and methods. It

Among the facilities provided by the System class are standard input, standard output, standard error output; environment variables; a means of loading files and libraries; and a utility for getting the system name.

Since:

JDK1.0

Field Summary

Fields

Modifier and Type	Field and Description
static PrintStream	err
	The "standard" error output stream.
static InputStream	in



객체 지향 프로그래밍4\_상속2 (오버라이딩)



# 상속 (Inheritance)

## ■ 상속

- 기존의 클래스를 재사용해서 새로운 클래스를 작성.
- 자손은 조상의 모든 멤버를 상속받는다.(생성자, 초기화블럭, private 멤버 제외)
- 자손의 멤버개수는 조상보다 적을 수 없다.(같거나 많다.)

```
Class Sub extends Super{  
    // ...  
}
```

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

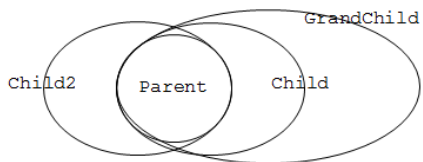
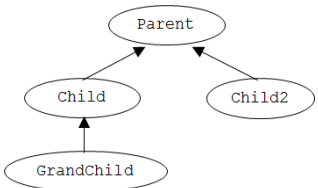
```
class Point3D {  
    int x;  
    int y;  
    int z;  
}
```

```
class Point3D extends Point {  
    int z;  
}
```

## ■ 클래스간의 관계 – 상속관계(inheritance)

- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리.
- 조상의 변경은 자손에 영향을 미치지만,  
자손의 변경은 조상에 아무런 영향을 미치지 않는다.

```
class Parent {}  
class Child extends Parent {}  
class Child2 extends Parent {}  
class GrandChild extends Child {}
```



## ■ 클래스간의 관계 – 포함관계(composite)

- 포함(composite)이란? 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
- 작은 단위의 클래스를 먼저 만들고, 이 들을 포함하는 클래스를 만든다.

```
class Point {  
    int x;  
    int y;  
}  
  
class Circle {  
    int x; // 원점의 x좌표  
    int y; // 원점의 y좌표  
    int r; // 반지름(radius)  
    Point c = new Point(); // 원심  
}
```



# 상속 관계 상속 vs. 포함

- 가능한 한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게 한다.
- '-is-a'와 '-has-a'를 가지고 문장을 만들어 본다.

상속관계 - '~은 ~이다.(is-a)'

포함관계 - '~은 ~을 가지고 있다.(has-a)'

원(Circle)은 점(Point)이다. - Circle **is a** Point.

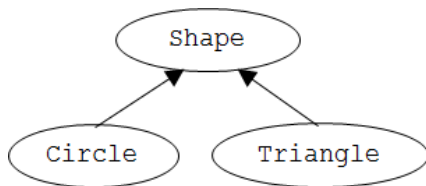
원(Circle)은 점(Point)을 가지고 있다. - Circle **has a** Point.

```
class Shape {
    String color = "blue";
    void draw() {
        // 도형을 그린다.
    }
}
```

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

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```



```
class Circle extends Shape {
    Point center;
    int r;

    Circle() {
        this(new Point(0,0),100);
    }

    Circle(Point center, int r) {
        this.center = center;
        this.r = r;
    }
}
```

```
class Triangle extends Shape {
    Point[] p;

    Triangle(Point[] p) {
        this.p = p;
    }

    Triangle(Point p1, Point p2, Point p3) {
        p = new Point[]{p1,p2,p3};
    }
}
```

```
Circle c1 = new Circle();
Circle c2 = new Circle(new Point(150,150),50);

Point[] p = {new Point(100,100),
             new Point(140,50),
             new Point(200,100)};

Triangle t1 = new Triangle(p);
```





# 상속 단일 상속

## ■ 단일상속(single inheritance)

- Java는 단일상속만을 허용한다.(C++은 다중상속 허용)
- 비중이 높은 클래스 하나만 상속관계로, 나머지는 포함관계로 한다.

```
Class TVCR extends TV, VCR { // error
    // ...
}
```

```
class Tv {
    boolean power; // 전원상태(on/off)
    int channel; // 채널

    void power() { power = !power; }
    void channelUp() { ++channel; }
    void channelDown() { --channel; }
}
```

상속

```
class TVCR extends Tv {
    VCR vcr = new VCR();
    int counter = vcr.counter;

    void play() {
        vcr.play();
    }

    void stop() {
        vcr.stop();
    }

    void rew() {
        vcr.rew();
    }

    void ff() {
        vcr.ff();
    }
}
```

```
class VCR {
    boolean power; // 전원상태(on/off)
    int counter = 0;
    void power() { power = !power; }
    void play() { /* 내용생략*/ }
    void stop() { /* 내용생략*/ }
    void rew() { /* 내용생략*/ }
    void ff() { /* 내용생략*/ }
}
```

포함

## ■ Object클래스 – 모든 클래스의 최고 조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.
- 상속계층도의 최상위에는 Object클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다  
toString(), equals(Object obj), hashCode(), ...

```
class Tv {
    // ...
}
```

```
class CaptionTv extends Tv {
    // ...
}
```

```
class Tv extends Object {
    // ...
}
```

```
class CaptionTv extends Tv {
    // ...
}
```



# 오버라이딩 Overriding

- 오버라이딩 : 조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞게 재정의하는 것.

```
class Point {  
    int x;  
    int y;  
  
    String getLocation() {  
        return "x : " + x + ", y : " + y;  
    }  
}
```

```
class Point3D extends Point {  
    int z;  
    String getLocation() { // 오버라이딩  
        return "x : " + x + ", y : " + y + ", z : " + z;  
    }  
}
```

- 오버라이딩의 조건

- 1. 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)
- 2. 접근제어자를 좁은 범위로 변경할 수 없다.
  - 조상의 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로만 변경 가능.
- 3. 조상클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

```
class Parent {  
    void parentMethod() throws IOException, SQLException {  
        // ...  
    }  
}  
  
class Child extends Parent {  
    void parentMethod() throws IOException {  
        //..  
    }  
}  
  
class Child2 extends Parent {  
    void parentMethod() throws Exception {  
        //..  
    }  
}
```



# Overriding Vs Overloading

오버로딩(over loading) - 기존에 없는 새로운 메서드를 정의하는 것(new)

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

```
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!!! 중복정의임  
}
```



## super : 참조변수

### ■ this : 인스턴스 자신을 가리키는 참조변수. 인스턴스의 주소가 저장되어있음.

- 모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재

### ■ super : this와 같음. 조상의 멤버와 자신의 멤버를 구별하는 데 사용.

```
class Parent {
    int x=10;
}

class Child extends Parent {
    int x=20;
    void method() {
        System.out.println("x=" + x);
        System.out.println("this.x=" + this.x);
        System.out.println("super.x="+ super.x);
    }
}

class Parent {
    int x=10;
}

class Child extends Parent {
    void method() {
        System.out.println("x=" + x);
        System.out.println("this.x=" + this.x);
        System.out.println("super.x="+ super.x);
    }
}

public static void main(String args[]) {
    Child c = new Child();
    c.method();
}

class Point {
    int x;
    int y;

    String getLocation() {
        return "x :" + x + ", y :" + y;
    }
}

class Point3D extends Point {
    int z;
    String getLocation() { // 오버라이딩
        // return "x :" + x + ", y :" + y + ", z :" + z;
        return super.getLocation() + ", z :" + z; // 조상의 메서드 호출
    }
}
```



## super() - 조상의 생성자

- 자손클래스의 인스턴스를 생성하면, 자손의 멤버와 조상의 멤버가 합쳐진 하나의 인스턴스가 생성된다.
- 조상의 멤버들도 초기화되어야 하기 때문에 자손의 생성자의 첫 문장에서 조상의 생성자를 호출해야 한다.
- Object클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자 호출이 없다면 컴파일러는 자동으로 "super();"를 첫 줄에 삽입한다.

```
class Point {  
    int x;  
    int y;  
  
    Point() {  
        this(0,0);  
    }  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
class Point extends Object {  
    int x;  
    int y;  
  
    Point() {  
        this(0,0);  
    }  
  
    Point(int x, int y) {  
        super(); // Object();  
        this.x = x;  
        this.y = y;  
    }  
}
```



```

class Point {
    int x;
    int y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    String getLocation() {
        return "x :" + x + ", y :" + y;
    }
}

```

→

```

Point(int x, int y) {
    super(); // Object();
    this.x = x;
    this.y = y;
}

```

```

class Point3D extends Point {
    int z;

    Point3D(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    String getLocation() { // 오버라이딩
        return "x :" + x + ", y :" + y + ", z :" + z;
    }
}

```

→

```

Point3D(int x, int y, int z) {
    super(); // Point()를 호출
    this.x = x;
    this.y = y;
    this.z = z;
}

```

↓

```

Point3D(int x, int y, int z) {
    // 조상의 생성자 Point(int x, int y)를 호출
    super(x,y);
    this.z = z;
}

```

```

class PointTest {
    public static void main(String args[]) {
        Point3D p3 = new Point3D(1,2,3);
    }
}

```

```

----- javac -----
PointTest.java:24: cannot find symbol
symbol   : constructor Point()
location: class Point
    Point3D(int x, int y, int z) {

```

1 error



```

1  class PointTest2 {
2      public static void main(String args[]) {
3          Point3D p3 = new Point3D();
4      }
5  }
6  class Point {
7      int x=10;
8      int y=20;
9      Point(int x, int y) {
10         this.x = x;
11         this.y = y;
12     }
13 }
14 class Point3D extends Point {
15     int z=30;
16     Point3D() {
17         this(100, 200, 300);
18     }
19     Point3D(int x, int y, int z) {
20         super(x, y);
21         this.z = z;
22     }
23 }

```

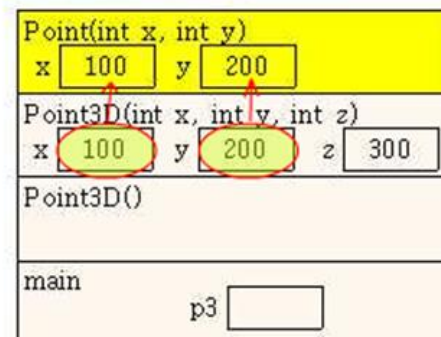
### Method Area

PointTest2  
클래스

Point  
클래스

Point3D  
클래스

### Call Stack



### Heap

	0x100
x	10
y	20
z	30



객체 지향 프로그래밍4\_상속2 (제어자 modifier)





# 제한자 (modifier, 제어자)

- 클래스, 변수, 메서드의 선언부에 사용되어 부가적인 의미를 부여한다.
- 제어자는 크게 접근 제어자와 그 외의 제어자로 나뉜다.
- 하나의 대상에 여러 개의 제어자를 조합해서 사용할 수 있으나, 접근제어자는 단 하나만 사용할 수 있다.

접근제어자 : **public, protected, default, private**

그 외 : **static, final, abstract** 등

## ■ 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 자손클래스에서 접근이 가능하다.  
**public** - 접근 제한이 전혀 없다.

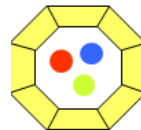
- 클래스에는 public, default 만 사용
- 일반적으로 생성자의 접근 제어자는 클래스의 접근 제어자와 일치한다.

제어자	같은 클래스	같은 패키지	자손클래스	전 체
public				
protected				
default				
private				

```
class AccessModifierTest {  
    int iv;           // 멤버변수 (인스턴스변수)  
    static int cv;    // 멤버변수 (클래스변수)  
  
    void method() {}  
}
```



# 접근 제어자(access modifier)를 이용한 캡슐화



## 접근 제어자를 사용하는 이유

- 외부로부터 데이터를 보호하기 위해서
- 외부에는 불필요한, 내부적으로만 사용되는, 부분을 감추기 위해서

```
class Time {  
    private int hour;  
    private int minute;  
    private int second;  
  
    Time(int hour, int minute, int second) {  
        setHour(hour);  
        setMinute(minute);  
        setSecond(second);  
    }  
  
    public int getHour() {        return hour; }  
  
    public void setHour(int hour) {  
        if (hour < 0 || hour > 23) return;  
        this.hour = hour;  
    }  
  
    ... 중간 생략 ...  
  
    public String toString() {  
        return hour + ":" + minute + ":" + second;  
    }  
}
```

```
public static void main(String[] args) {  
    Time t = new Time(12, 35, 30);  
    // System.out.println(t.toString());  
    System.out.println(t);  
    // t.hour = 13; 에러!!!  
  
    // 현재시간보다 1시간 후로 변경한다.  
    t.setHour(t.getHour()+1);  
    System.out.println(t);  
}
```

```
----- java -----  
12:35:30  
13:35:30
```

출력 완료 (0초 경과)



# static, abstract

## ■ **static** : 클래스의, 공통의

제어자	대상	의 미
static	멤버변수	<ul style="list-style-type: none"><li>- 모든 인스턴스에 공통적으로 사용되는 클래스변수가 된다.</li><li>- 클래스변수는 인스턴스를 생성하지 않고도 사용 가능하다.</li><li>- 클래스가 메모리에 로드될 때 생성된다.</li></ul>
	메서드	<ul style="list-style-type: none"><li>- 인스턴스를 생성하지 않고도 호출이 가능한 static 메서드가 된다.</li><li>- static메서드 내에서는 인스턴스멤버들을 직접 사용할 수 없다.</li></ul>

```
class StaticTest {  
    static int width = 200;  
    static int height = 120;  
  
    static { // 클래스 초기화 블록  
        // static변수의 복잡한 초기화 수행  
    }  
  
    static int max(int a, int b) {  
        return a > b ? a : b;  
    }  
}
```

## ■ **abstract** : 추상의, 미상의.

제어자	대상	의 미
abstract	클래스	클래스 내에 추상메서드가 선언되어 있음을 의미한다.
	메서드	선언부만 작성하고 구현부는 작성하지 않은 추상메서드임을 알린다.

```
abstract class AbstractTest { // 추상클래스  
    abstract void move(); // 추상메서드  
}
```

**[참고]** 추상메서드가 없는 클래스도 abstract를 붙여서 추상클래스로 선언하는 것이 가능하기는 하지만 그렇게 해야 할 이유는 없다.



# final

## ■ **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	

[참고] 대표적인 final클래스로는 String과 Math가 있다.

## ■ final 변수는 상수이므로 보통은 선언과 초기화를 동시에 하지만, 인스턴스변수의 경우 생성자에서 초기화 가능.

```
class Card {
    final int NUMBER;           // 상수지만 선언과 함께 초기화 하지 않고
    final String KIND;          // 생성자에서 단 한번만 초기화할 수 있다.
    static int width = 100;
    static int height = 250;

    Card(String kind, int num) {
        KIND = kind;
        NUMBER = num;
    }

    Card() {
        this("HEART", 1);
    }

    public String toString() {
        return "" + KIND + " " + NUMBER;
    }
}
```

```
final class FinalTest {
    final int MAX_SIZE = 10; // 멤버변수

    final void getMaxSize() {
        final LV = MAX_SIZE; // 지역변수
        return MAX_SIZE;
    }
}

class Child extends FinalTest {
    void getMaxSize() {} // 오버라이딩
}
```

```
public static void main(String args[]) {
    Card c = new Card("HEART", 10);
    // c.NUMBER = 5;     예러!!!
    System.out.println(c.KIND);
    System.out.println(c.NUMBER);
}
```



# 종합

## ■ 제어자 종합

대 상	사용가능한 제어자
클래스	public, (default), final, abstract
메서드	모든 접근 제어자, final, abstract, static
멤버변수	모든 접근 제어자, final, static
지역변수	final

### 1. 메서드에 **static**과 **abstract**를 함께 사용 불가.

- static메서드는 몸통(body, 구현부)이 있는 메서드에만 사용할 수 있기 때문이다.

### 2. 클래스에 **abstract**와 **final**을 동시에 사용 불가.

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

### 3. **abstract**메서드의 접근제어자가 **private**일 수 없다.

- abstract메서드는 자손클래스에서 구현해주어야 하는데 접근 제어자가 private이면, 자손클래스에서 접근할 수 없기 때문이다.

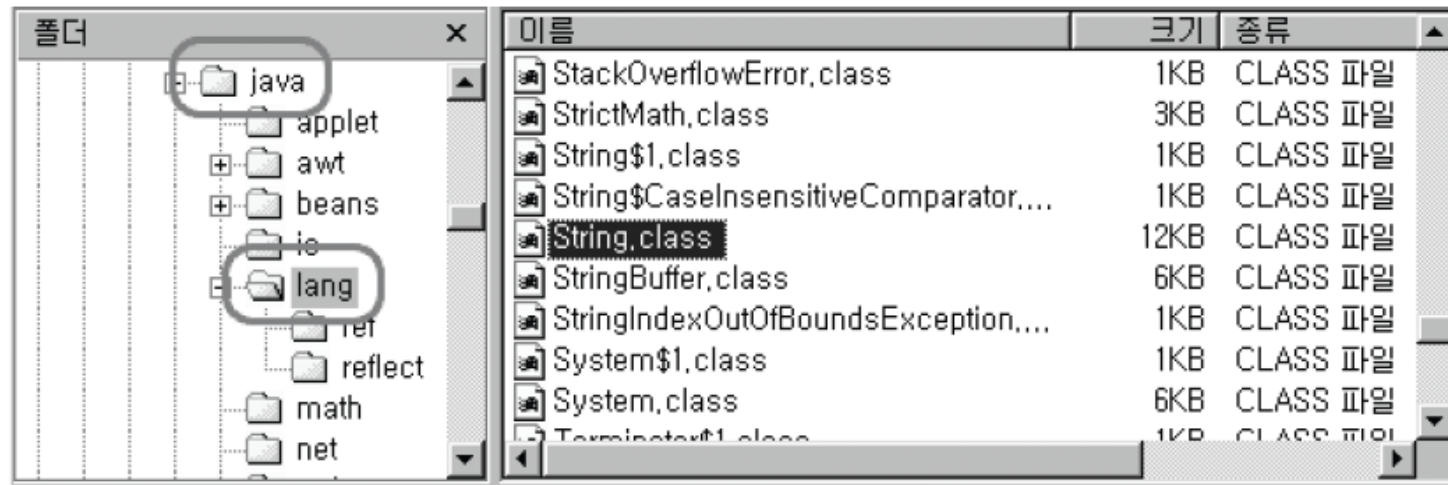
### 4. 메서드에 **private**과 **final**을 같이 사용할 필요는 없다.

- 접근 제어자가 private인 메서드는 오버라이딩될 수 없으므로 이 둘 중 하나만 사용해도 의미가 충분하다.



# 패키지(package)

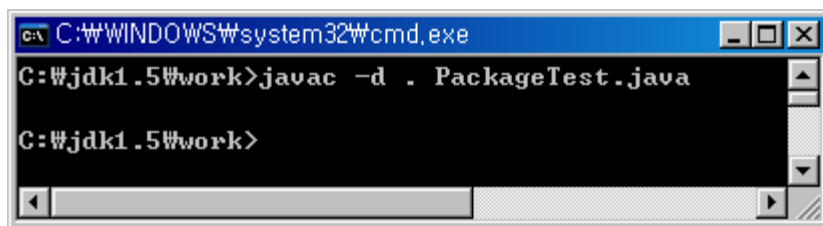
- 서로 관련된 클래스와 인터페이스의 묶음.
- 클래스가 물리적으로 클래스파일(\*.class)인 것처럼, 패키지는 물리적으로 폴더이다. 패키지는 서브패키지를 가질 수 있으며, '.'으로 구분한다.
- 클래스의 실제 이름(full name)은 패키지명이 포함된 것이다.
  - (String클래스의 full name은 java.lang.String)
- - rt.jar는 Java API의 기본 클래스들을 압축한 파일. (JDK설치경로\jre\lib에 위치)



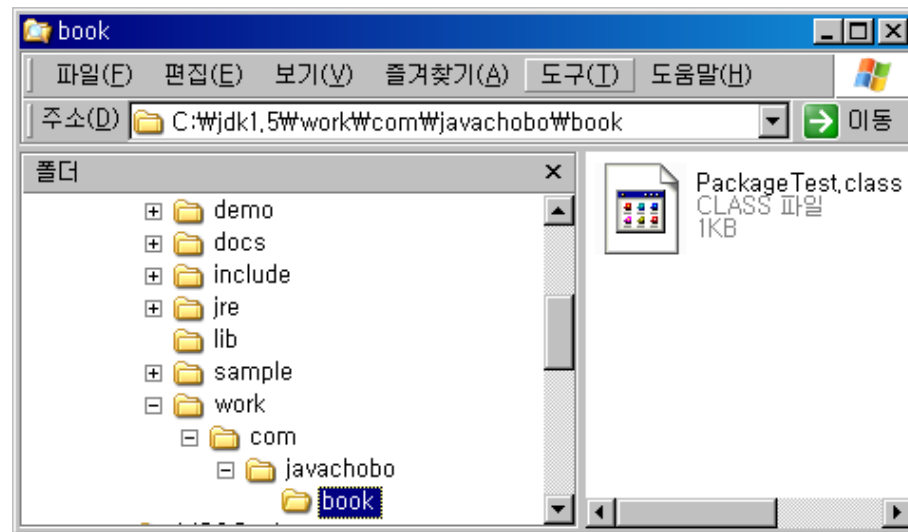
# 패키지 선언

- 패키지는 소스파일에 첫 번째 문장(주석 제외)으로 단 한번 선언한다.
- 하나의 소스파일에 둘 이상의 클래스가 포함된 경우, 모두 같은 패키지에 속하게 된다.(하나의 소스파일에 단 하나의 public클래스만 허용한다.)
- 모든 클래스는 하나의 패키지에 속하며, 패키지가 선언되지 않은 클래스는 자동적으로 이름없는(unnamed) 패키지에 속하게 된다.

```
1 // PackageTest.java
2 package com.javachobo.book;
3
4 public class PackageTest {
5     public static void main(String[] args) {
6         System.out.println("Hello World!");
7     }
8 }
9
10 public class PackageTest2 {}
```



```
C:\WINDOWS\system32\cmd.exe
C:\Wjdk1.5\work>javac -d . PackageTest.java
C:\Wjdk1.5\work>
```



# import문

- 사용할 클래스가 속한 패키지를 지정하는데 사용.
- import문을 사용하면 클래스를 사용할 때 패키지명을 생략할 수 있다.

```
class ImportTest {  
    java.util.Date today = new java.util.Date();  
    // ...  
}
```

```
import java.util.*;  
  
class ImportTest {  
    Date today = new Date()  
}
```

```
import java.sql.*; // java.sql.Date  
import java.util.*; // java.util.Date  
  
public class ImportTest {  
    public static void main(String[] args) {  
        java.util.Date today = new java.util.Date()  
    }  
}
```

- java.lang패키지의 클래스는 import하지 않고도 사용할 수 있다. String, Object, System, Thread ...

```
import java.lang.*;  
  
class ImportTest2  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Hello World!");  
    }  
}
```

```
public static void main(java.lang.String[] args)  
{  
    java.lang.System.out.println("Hello World!");  
}
```

## import문의 선언

- import문은 패키지문과 클래스선언의 사이에 선언한다.

```
import java.util.Calendar;  
import java.util.Date;  
import java.util.ArrayList;
```

```
import java.util.*;
```

```
1 package com.javachobo.book;  
2  
3 import java.text.SimpleDateFormat;  
4 import java.util.*;  
5  
6 public class PackageTest {  
7     public static void main(String[] args) {  
8         // java.util.Date today = new java.util.Date();  
9         Date today = new Date();  
10        SimpleDateFormat date = new SimpleDateFormat("yyyy/MM/dd");  
11    }  
}
```

