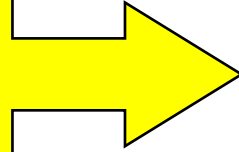


Java

제 10 강

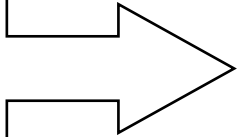
상속 & 객체지향개념 II-1

- 1. 상속
- 2. 오버라이딩
- 3. package와 import



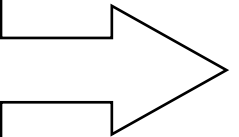
객체지향개념 II-1

- 4. 제어자
- 5. 다형성



객체지향개념 II-2

- 6. 추상클래스
- 7. 인터페이스



객체지향개념 II-3

1. 상속(inheritance)

1.1 상속의 정의와 장점

1.2 클래스간의 관계

1.3 클래스간의 관계결정하기

1.4 단일 상속(single inheritance)

1.5 Object클래스

2. 오버라이딩(overriding)

2.1 오버라이딩이란?

2.2 오버라이딩의 조건

2.3 오버로딩 vs. 오버라이딩

2.4 super

2.5 super()

3. package와 import

3.1 패키지(package)

3.2 패키지의 선언

3.3 클래스패스 설정

3.4 import문

3.5 import문의 선언

1. 상속(inheritance)

1.1 상속(inheritance)의 정의와 장점

▶ 상속이란?

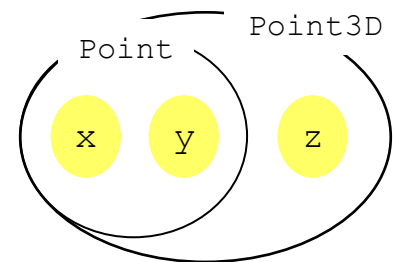
- 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것.
- 두 클래스를 조상과 자손으로 관계를 맺어주는 것.
- 자손은 조상의 모든 멤버를 상속받는다.(생성자, 초기화블럭 제외)
- 자손의 멤버개수는 조상보다 적을 수 없다.(같거나 많다.)

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

```
class 자손클래스 extends 조상클래스 {  
    // ...  
}
```

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

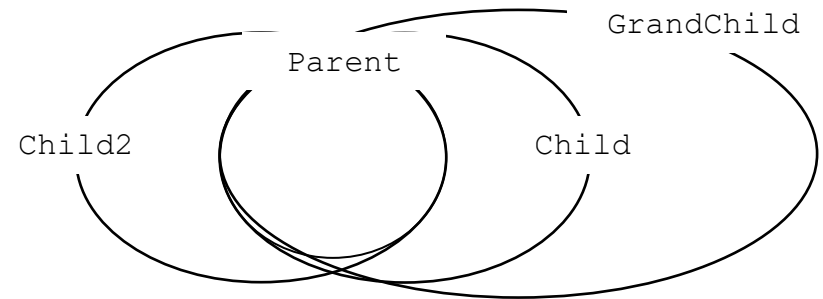
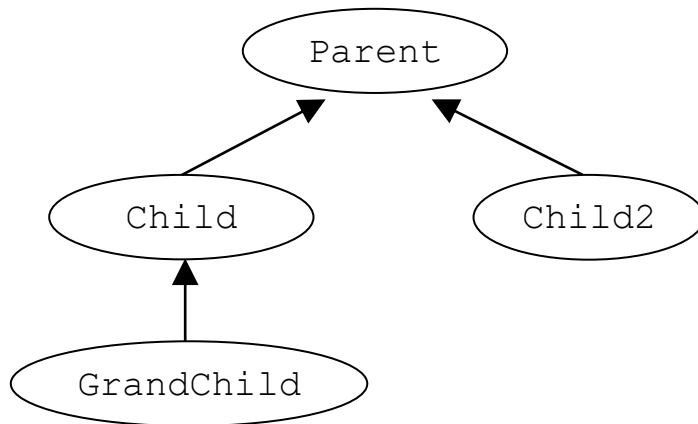
```
class Point3D extends Point {  
    int z;  
}
```



1.2 클래스간의 관계 – 상속관계(inheritance)

- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리한다.
- 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 {}
```



1.2 클래스간의 관계 – 포함관계(composite)

▶ 포함(composite)이란?

- 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
- 작은 단위의 클래스를 먼저 만들고, 이 들을 조합해서 하나의 커다란 클래스를 만든다.

```
class Circle {  
    int x; // 원점의 x좌표  
    int y; // 원점의 y좌표  
    int r; // 반지름(radius)  
}
```

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

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

```
class Car {  
    Engine e = new Engine(); // 엔진  
    Door[] d = new Door[4]; // 문, 문의 개수를 넷으로 가정하고 배열로 처리했다.  
    //...  
}
```

1.3 클래스간의 관계결정하기 – 상속 vs. 포함

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

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

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(radius)  
}
```

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


1.3 클래스간의 관계결정하기 – 예제설명

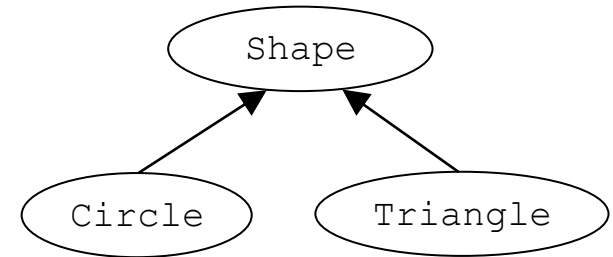
- 원(Circle)은 도형(Shape)이다.(A Circle is a Shape.) : 상속관계
- 원(Circle)은 점(Point)를 가지고 있다.(A Circle has a Point.) : 포함관계

```
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);
```

1.3 클래스간의 관계결정하기 – 예제설명2

```
class Deck {  
    final int CARD_NUM = 52;    // 카드의 개수  
    Card c[] = new Card[CARD_NUM];  
  
    Deck () {    // Deck의 카드를 초기화한다.  
        int i=0;  
  
        for(int k=Card.KIND_MAX; k > 0; k--) {  
            for(int n=1; n < Card.NUM_MAX + 1 ; n++) {  
                c[i++] = new Card(k, n);  
            }  
        }  
    }  
  
    Card pick(int index) {    // 지정된 위치(index)에 있는 카드 하나를 선택한다.  
        return c[index%CARD_NUM];  
    }  
  
    Card pick() {    // Deck에서 카드 하나를 선택한다.  
        int index = (int) (Math.random() * CARD_NUM);  
        return pick(index);  
    }  
  
    void shuffle() {    // 카드의 순서를 섞는다.  
        for(int n=0; n < 1000; n++) {  
            int i = (int) (Math.random() * CARD_NUM);  
            Card temp = c[0];  
            c[0] = c[i];  
            c[i] = temp;  
        }  
    }  
} // Deck클래스의 끝
```

```
public static void main(String[] args) {  
    Deck d = new Deck();  
    Card c = d.pick();  
  
    d.shuffle();  
    Card c2 = d.pick(55);  
}
```

1.4 단일상속(single inheritance)

- Java는 단일상속만을 허용한다.(C++은 다중상속 허용)

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

- 비중이 높은 클래스 하나만 상속관계로, 나머지는 포함관계로 한다.

```
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();
    }
}
```

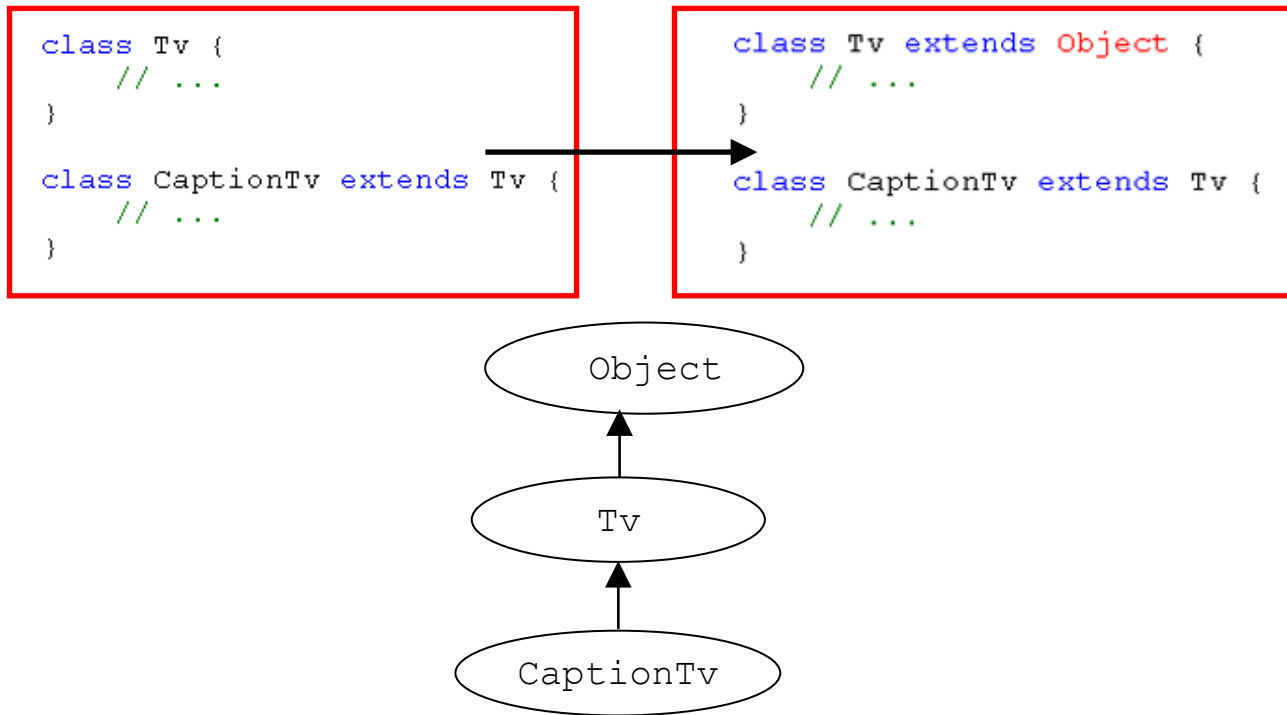
```
class VCR {
    boolean power; // 전원상태(on/off)
    int counter = 0;

    void power() { power = !power; }
    void play() { /* 내용생략*/ }
    void stop() { /* 내용생략*/ }
    void rew() { /* 내용생략*/ }
    void ff() { /* 내용생략*/ }
}
```

포함

1.5 Object클래스 – 모든 클래스의 최고조상

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



2. 오버라이딩(overriding)

2.1 오버라이딩(overriding)이란?

“조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞게 변경하는 것을 오버라이딩이라고 한다.”

* override - vt. '~위에 덮어쓰다(overwrite)., '~에 우선하다!

```
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;  
    }  
}
```

2.2 오버라이딩의 조건

1. 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)
2. 접근제어자를 좁은 범위로 변경할 수 없다.
 - 조상의 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로만 변경할 수 있다.
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 {  
        //..  
    }  
}
```

2.3 오버로딩 vs. 오버라이딩

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  
}
```


2.4 super – 참조변수(1/2)

- ▶ 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();  
}
```

2.4 super – 참조변수(2/2)

- ▶ 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 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
    }
}
```

2.5 super() – 조상의 생성자(1/3)

- 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(같은 클래스의 다른 생성자 또는 조상의 생성자)를 호출해야한다.

그렇지 않으면 컴파일러가 자동적으로 'super();'를 생성자의 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;  
    }  
}
```

2.5 super() – 조상의 생성자(2/3)

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

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

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

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

```
}
```

```
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
```

```
    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;  
    }
```

2.5 super() – 조상의 생성자(3/3)

The screenshot shows the Flash IDE with a Java project. The main window displays the source code for `PointTest2.java`. The `main` method calls `new Point3D()`. The `Point3D` class extends `Point` and its constructor calls `super(100, 200, 300)`. The `Point` class has a constructor `Point(int x, int y)` which is currently executing, as indicated by the yellow highlight on line 9. The `Method Area` shows three classes: `PointTest2`, `Point`, and `Point3D`. The `Call Stack` shows the execution flow: `main` calls `Point3D()`, which calls `Point3D(int x, int y, int z)`, which calls `Point(int x, int y)`. The `Heap` shows the memory layout for the `Point` object at address `0x100`, with `x` at 10, `y` at 20, and `z` at 30.

PointTest2.java

```

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

- Point(int x, int y)
 - x: 100, y: 200
- Point3D(int x, int y, int z)
 - x: 100, y: 200, z: 300
- Point3D()
- main
 - p3

Heap

0x100

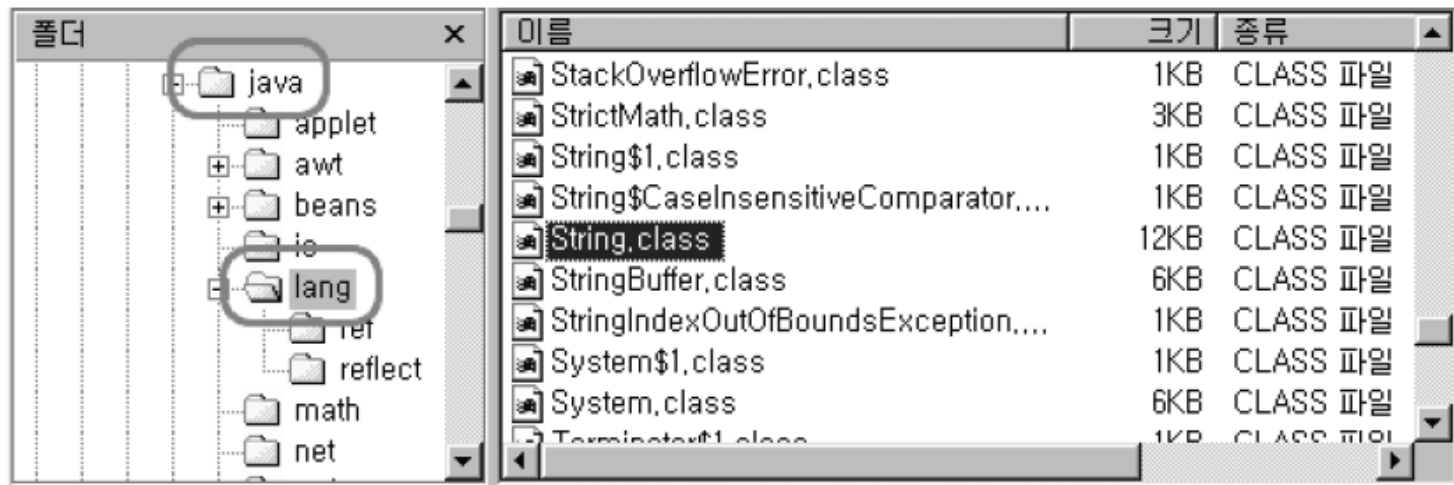
x	10
y	20
z	30

7. Point(int x, int y)는 호출 시 넘겨받은 값으로 인스턴스변수 x와 y의 값을 변경한다.
여기서 this.x는 인스턴스변수이고, x는 생성자Point(int x, int y)의 지역변수이다.

3. package와 import

3.1 패키지(package)

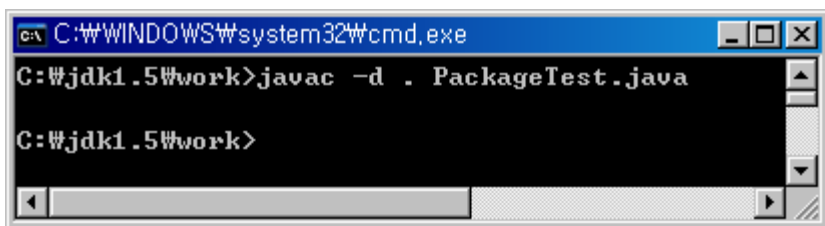
- 서로 관련된 클래스와 인터페이스의 묶음.
- 클래스가 물리적으로 클래스파일(*.class)인 것처럼, 패키지는 물리적으로 폴더이다. 패키지는 서브패키지를 가질 수 있으며, '.'으로 구분한다.
- 클래스의 실제 이름(full name)은 패키지명이 포함된 것이다.
(String클래스의 full name은 java.lang.String)
- rt.jar는 Java API의 기본 클래스들을 압축한 파일
(JDK설치경로\jre\lib에 위치)



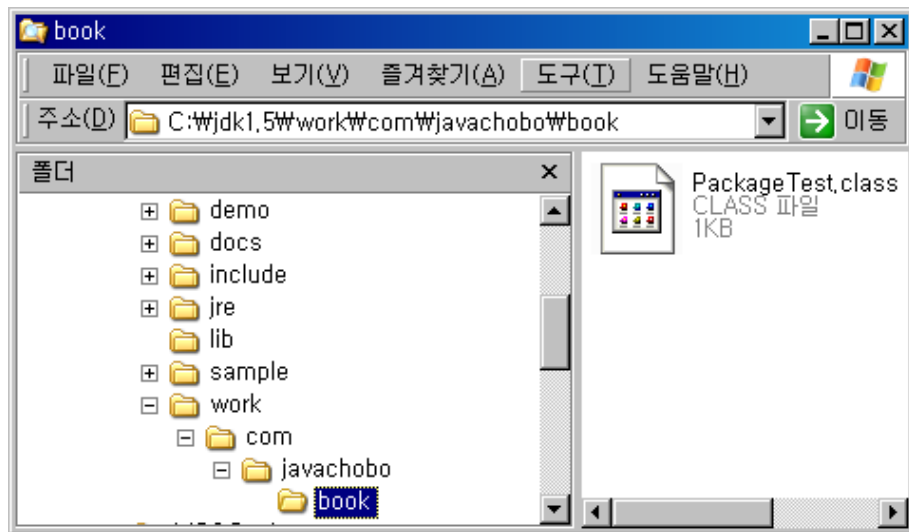
3.2 패키지의 선언

- 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 {}
```

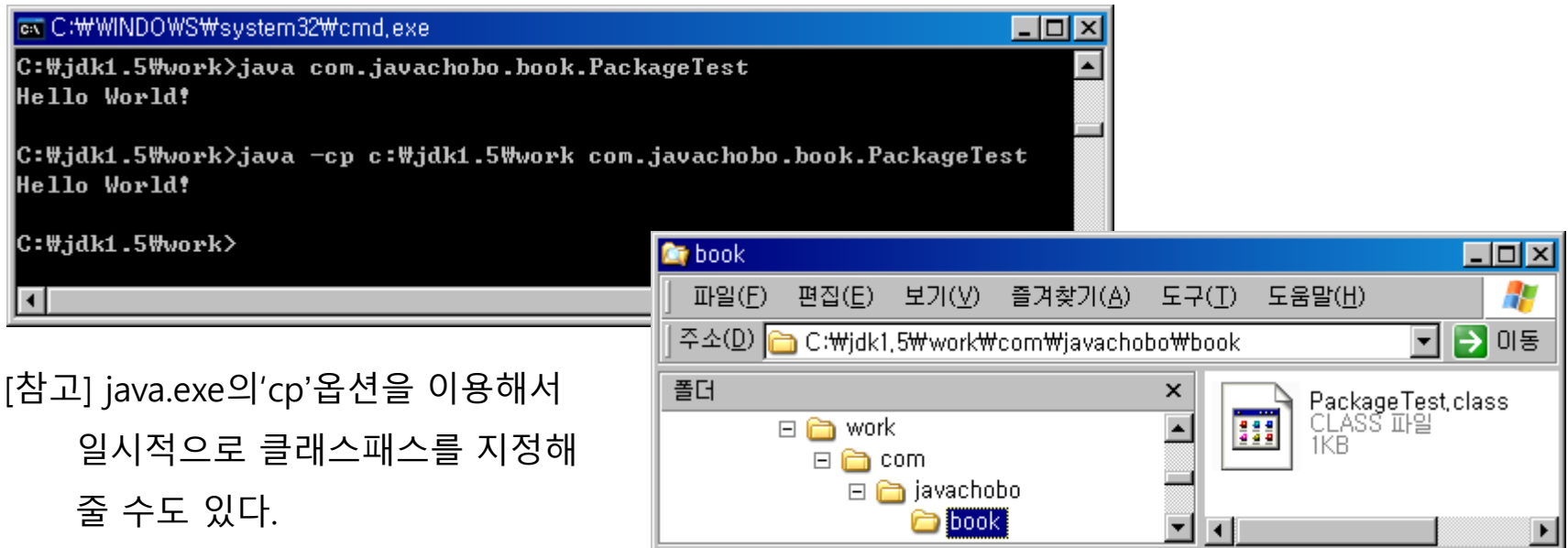


```
C:\WINDOWS\system32\cmd.exe
C:\j\jdk1.5\work>javac -d . PackageTest.java
C:\j\jdk1.5\work>
```



3.3 클래스패스(classpath) 설정(1/2)

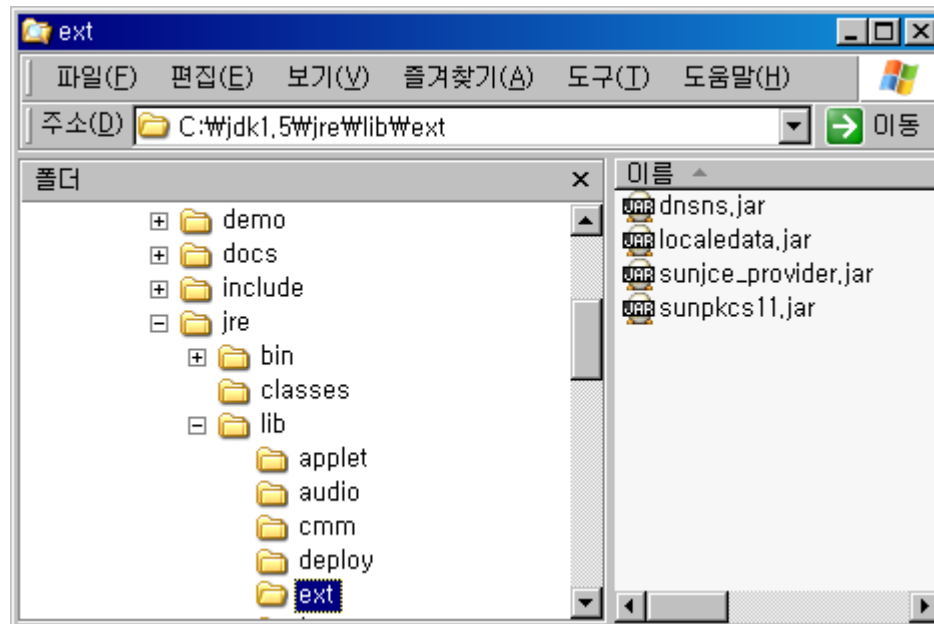
- 클래스패스(classpath)는 클래스파일(*.class)를 찾는 경로. 구분자는 ‘;’
- 클래스패스에 패키지가 포함된 폴더나 jar파일을(*.jar) 나열한다.
- 클래스패스가 없으면 자동적으로 현재 폴더가 포함되지만
클래스패스를 지정할 때는 현재 폴더(.)도 함께 추가해주어야 한다.



[참고] java.exe의 'cp' 옵션을 이용해서
일시적으로 클래스패스를 지정해
줄 수도 있다.

3.3 클래스패스(classpath) 설정(2/2)

- ▶ 클래스패스로 자동 포함된 폴더 for 클래스파일(*.class) : 수동생성 해야함.
 - JDK설치경로\jre\classes
- ▶ 클래스패스로 자동 포함된 폴더 for jar파일(*.jar) : JDK설치시 자동생성됨.
 - JDK설치경로\jre\lib\ext



3.4 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();  
}
```

- 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!");  
}
```

3.5 import문의 선언

- import문은 패키지명과 클래스선언의 사이에 선언한다.

일반적인 소스파일(*.java)의 구성은 다음의 순서로 되어 있다.

- ① package문
- ② import문
- ③ 클래스 선언

- import문을 선언하는 방법은 다음과 같다.

import 패키지명.클래스명;

또는

import 패키지명.*;

```
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    }
12 }
```

3.5 import문의 선언 - 선언예

- import문은 컴파일 시에 처리되므로 프로그램의 성능에 아무런 영향을 미치지 않는다.

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

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

- 다음의 두 코드는 서로 의미가 다르다.

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

```
import java.*;
```

- 이름이 같은 클래스가 속한 두 패키지를 import할 때는 클래스 앞에 패키지명을 붙여줘야 한다.

```
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();  
    }  
}
```

감사합니다.