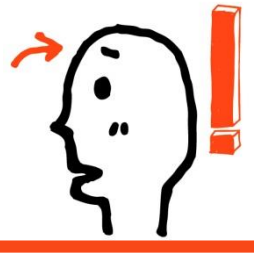
Android Java

14장 클래스의 상속 2: 오버라이딩

클래스의 상속 2: 오버라이딩

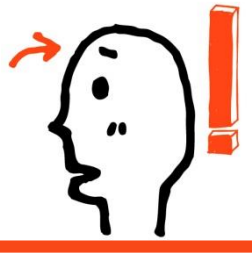


01 상속을 위한 관계

02 하위 클래스에서 메소드를 다시 정의한다면?

03 참조변수의 인스턴스 참조와 instanceof 연산자

상속을 위한 기본 조건인 IS-A 관계의 성립



✓ 상속을 위한 기본 조건

- 상속관계에 있는 두 클래스 사이에는 IS-A 관계가 성립해야 한다.
- IS-A 관계가 성립하지 않는 경우에는 상속의 타당성을 면밀히 검토해야 한다.
- IS-A 이외에 HAS-A 관계도 상속으로 표현 가능하다. 그러나 HAS-A를 대신해서 Composition 관계를 유지하는 것이 보다 적절한 경우가 많다.

- 전화기 → 무선 전화기
- 컴퓨터 → 노트북 컴퓨터

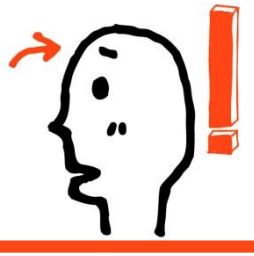
무선 전화기는 전화기를 상속한다!

노트북 컴퓨터는 컴퓨터를 상속한다!

- 무선 전화기는 일종의 전화기입니다.
- 노트북 컴퓨터는 일종의 컴퓨터입니다.

- 무선 전화기 is a 전화기.
- 노트북 컴퓨터 is a 컴퓨터.

IS-A 기반 상속의 예



```
class Computer
{
    String owner;
    public Computer(String name){   }
    public void calculate() {  }
}
```

일반적으로 IS-A 관계가 성립되면, 불필요한 상속관계는 형성될 수 있으나, 잘못된 상속관계가 형성된다고는 이야기하지 않는다.

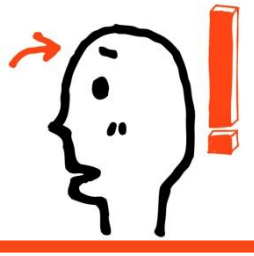
노트북 컴퓨터는 컴퓨터이다!

```
class NotebookComp extends Computer
{
    int battary;
    public NotebookComp(String name, int initChag) {  }
    public void charging() {  }
    public void movingCal() {  }
}
```

타블렛은 노트북 컴퓨터이다!

```
class TabletNotebook extends NotebookComp
{
    String regstPenModel;
    public TabletNotebook(String name, int initChag, String pen){ }
    public void write(String penInfo){ }
}
```

HAS-A 관계에 상속을 적용한 경우



```
class Gun
{
    int bullet;     // 장전된 총알의 수
    public Gun(int bnum) { bullet=bnum; }
    public void shut()
    {
        System.out.println("BBANG!");
        bullet--;
    }
}
```

```
class Police extends Gun
{
    int handcuffs;      // 소유한 수갑의 수
    public Police(int bnum, int bcuff)
    {
        super(bnum);
        handcuffs=bcuff;
    }
    public void putHandcuff()
    {
        System.out.println("SNAP!");
        handcuffs--;
    }
}
```

경찰은 총을 소유하고 있다!

경찰 has a 총!

상속은 강한 연결고리를 형성한다.

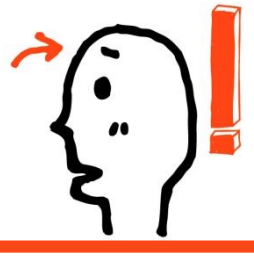
때문에 총을 소유하지 않는 경찰,

또는 총이 아닌 경찰봉을 소유하

는 경찰 등 다양한 표현에 한계를

보인다는 단점이 있다!

HAS-A 관계에 복합관계를 적용한 경우

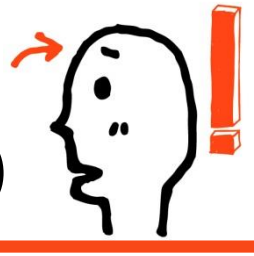


```
class Gun
{
    int bullet;     // 장전된 총알의 수
    public Gun(int bnum){bullet=bnum;}
    public void shut()
    {
        System.out.println("BBANG!");
        bullet--;
    }
}
```

복합관계는 강한 연결고리를 형성하지 않는다. 따라서 소유하던 대상을 소유하지 않을 수도 있고, 소유의 대상을 늘리는 것도 상속보다 훨씬 간단하다. 때문에 HAS-A 관계는 복합 관계로 표현한다.

```
class Police
{
    int handcuffs;      // 소유한 수갑의 수
    Gun pistol;         // 소유하고 있는 권총
    public Police(int bnum, int bcuff)
    {
        handcuffs=bcuff;
        if(bnum!=0)
            pistol=new Gun(bnum);
        else
            pistol=null;
    }
    public void putHandcuff()
    {
        System.out.println("SNAP!");
        handcuffs--;
    }
    public void shut()
    {
        if(pistol==null)
            System.out.println("Hut BBANG!");
        else
            pistol.shut();
    }
}
```

메소드 오버라이딩(슈퍼클래스 메소드 무시하기)

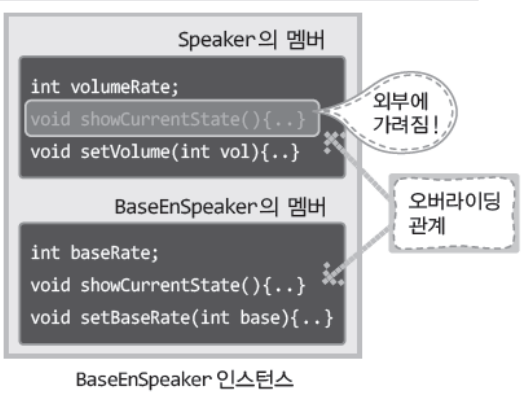


• 상위 클래스에 정의된 메소드의 이름, 반환형, 매개변수 선언까지 완전히 동일한 메소드를 하위 클래스에서 다시 정의하는 것!

• 하위 클래스에 정의된 메소드에 의해 상위 클래스의 메소드는 가리워진다.

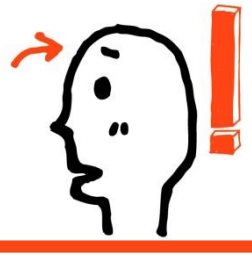
```
class Speaker
{
    private int volumeRate;
    public void showCurrentState()
    {
        System.out.println("볼륨 크기 : "+ volumeRate);
    }
    public void setVolume(int vol)
    {
        volumeRate=vol;
    }
}
```

```
class BaseEnSpeaker extends Speaker
{
    private int baseRate;
    public void showCurrentState()
    {
        super.showCurrentState();
        System.out.println("베이스 크기 : "+baseRate);
    }
    public void setBaseRate(int base)
    {
        baseRate=base;
    }
}
```



하위 클래스에서 메소드를 다시 정의한다면?

상위 클래스의 참조변수로 하위 클래스의 인스턴스 참조



- 중 저음 보강 스피커는 (일종의) 스피커이다. (O)
- BaseEnSpeaker is a Speaker. (O)

- 스피커는 (일종의) 중 저음 보강 스피커이다. (X)
- Speaker is a BaseEnSpeaker. (X)

자바 컴파일러의 실제 관점

```
public static void main(String[] args)
{
    Speaker bs=new BaseEnSpeaker();
    bs.setVolume(10);
    bs.setBaseRate(20);   // 컴파일 에러
    bs.showCurrentState();
}
```

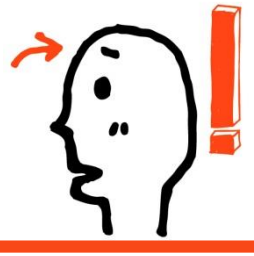
BaseEnSpeaker도 Speark의 인스턴스이므로 성립한다!

bs가 참조하는것은 Speaker의 인스턴스로 인식하기 때문에 BaseEnSpeaker의 멤버에 접근 불가!

위의 내용을 정확히 이해하는 것이 중요하다. 특히 상위 클래스의 참조변수가 하위 클래스의 인스턴스를 참조할 수 있는 이유를 잘 이해하자!

하위 클래스에서 메소드를 다시 정의한다면?

참조변수의 참조 가능성에 대한 일반화



```
class AAA { . . . }
class BBB extends AAA { . . . }
class CCC extends BBB { . . . }
```

아래의 문제 제시를 위한 클래스의 상속관계

```
AAA ref1 = new BBB();
AAA ref2 = new CCC();
BBB ref3 = new CCC();
```

```
CCC ref1 = . . .    // 컴파일 완료
BBB ref2 = ref1;
AAA ref3 = ref1;
```

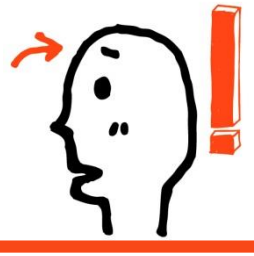
```
AAA ref1 = new CCC();
BBB ref2 = ref1;
CCC ref3 = ref1;
```

참조변수의 자료형에 따라서 대입연산의 허용여부가 결정된다.

이 사실을 바탕으로 왼쪽 문장 중에서 컴파일 에러가 발생하는 문장들을 모두 고르면?

하위 클래스에서 메소드를 다시 정의한다면?

업캐스팅과 다운캐스팅



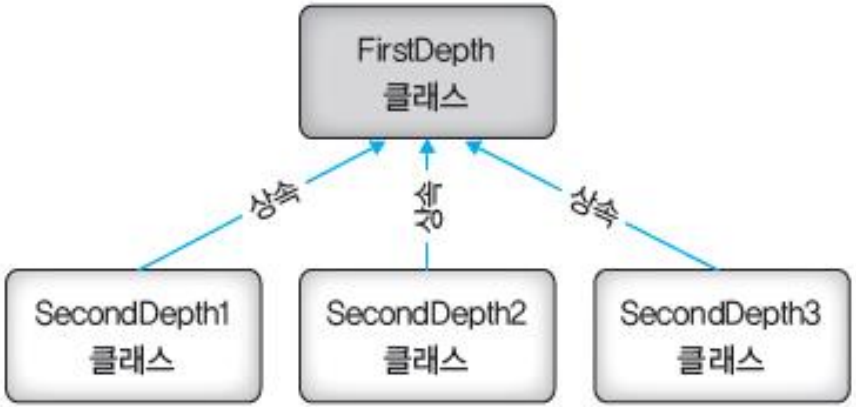
업캐스팅과 다운캐스팅

- **업캐스팅(up casting)** : 자식 클래스 객체를 부모 클래스 객체로 형 변환하는 것

  (객체 내에 있는 모든 멤버를 접근할 수 없고 슈퍼 클래스의 멤버만 접근 가능)

- **다운캐스팅(down casting)** : 부모 클래스 객체를 자식 클래스 객체로 형 변환하는 것

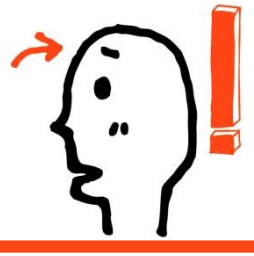
- 업캐스팅의 경우에는 형 변환 연산자를 생략해도 무방
  (상속이 트리 구조로 구현되어 있기 때문)



△그림 6-10 간단한 상속 예

하위 클래스에서 메소드를 다시 정의한다면?

업캐스팅과 다운캐스팅



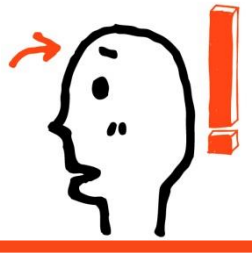
업캐스팅과 다운캐스팅

- 자바는 다중 상속을 지원하지 않으므로 부모는 무조건 하나
- **묵시적 형변환** : 업캐스팅의 경우에는 형 변환 연산자를 생략해도 상관이 없음
- **명시적 형변환** : 부모 클래스의 시각에서는 여러 개의 자식 클래스가 있을 수 있으므로 다운캐스팅 시에는 반드시 형 변환 연산자를 사용해서 어떤 데이터형으로 형 변환할 것인지 명확하게 적어줘야 함

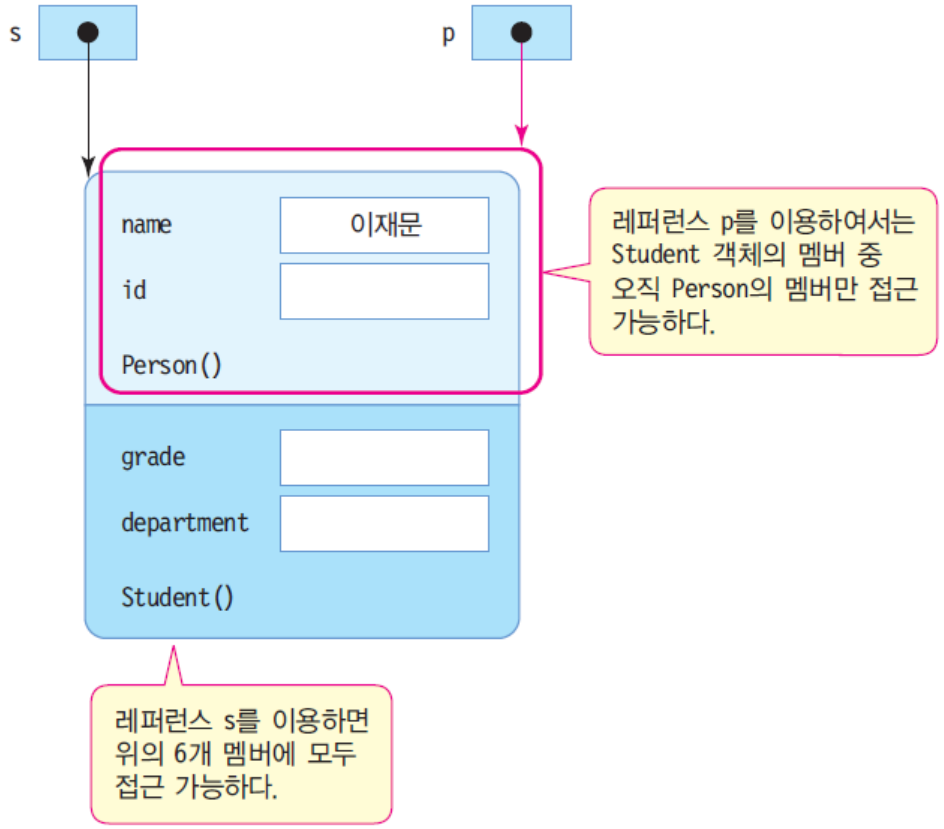
```
업캐스팅 사용예 : SecondDepth1 obj1 = new SecondDepth1();
                  FirstDepth obj2 = obj1;       // 업캐스팅으로 인한 묵시적 형 변환
다운캐스팅 사용예 : FirstDepth obj1 = new FirstDepth1();
                   SecondDepth2 obj2 = (SecondDepth2)obj1       // 명시적 형 변환
```

하위 클래스에서 메소드를 다시 정의한다면?

업캐스팅 사례



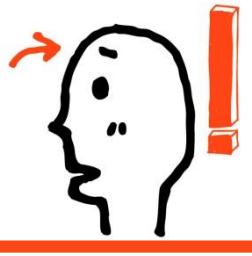
```
class  Person {
    String name;
    String id;

    public Person(String name) {
        this.name = name;
    }
}

class  Student extends Person {
    String grade;
    String department;

    public Student(String name) {
        super(name);
    }
}

public class UpcastingEx {
    public static void main(String[] args) {
        Person  p;
        Student s = new Student("이재문");
        p = s; // 업캐스팅 발생

        System.out.println(p.name); // 오류 없음

        p.grade = "A"; // 컴파일 오류
        p.department = "Com"; // 컴파일 오류
    }
}
```



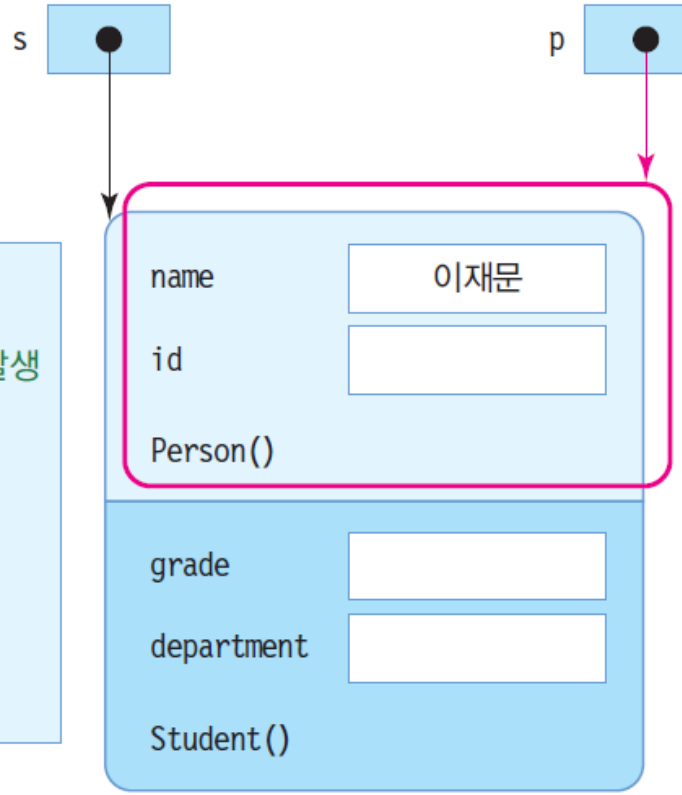
이재문

하위 클래스에서 메소드를 다시 정의한다면?

다운캐스팅 사례



```
public class DowncastingEx {
    public static void main(String[] args) {
        Person p = new Student("이재문"); // 업캐스팅 발생
        Student s;

        s = (Student)p; // 다운캐스팅

        System.out.println(s.name); // 오류 없음
        s.grade = "A"; // 오류 없음
    }
}
```

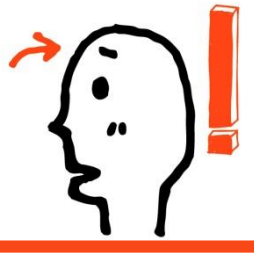


실행 결과

```
이재문
```

하위 클래스에서 메소드를 다시 정의한다면?

메소드 오버라이딩 사례



```
class DObject {
    public DObject next;

    public DObject() {next = null;}
    public void draw() {
        System.out.println("DObject draw");
    }
}
```

Line, Rect, Circle 클래스는 모두 DObject를 상속받고 draw() 메소드를 오버라이딩함

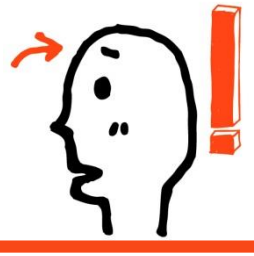
```
class Line extends DObject {
    public void draw() {
        System.out.println("Line");
    }
}
```

```
class Rect extends DObject {
    public void draw() {
        System.out.println("Rect");
    }
}
```

```
class Circle extends DObject {
    public void draw() {
        System.out.println("Circle");
    }
}
```

하위 클래스에서 메소드를 다시 정의한다면?

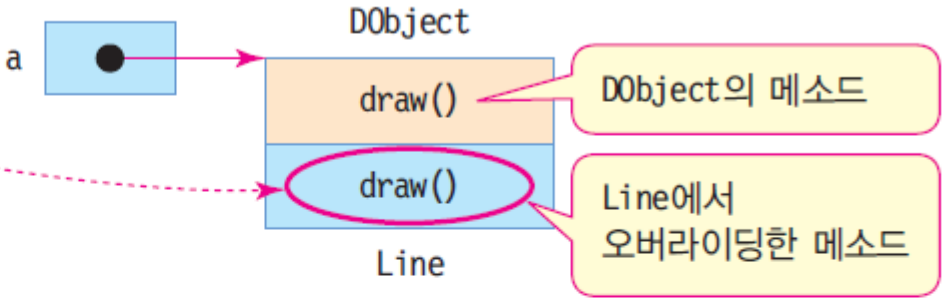
서브 클래스 객체와 오버라이딩된 메소드 호출



(1) 서브 클래스 레퍼런스로 오버라이딩된 메소드 호출

```
Line a = new Line();
a.draw();
```

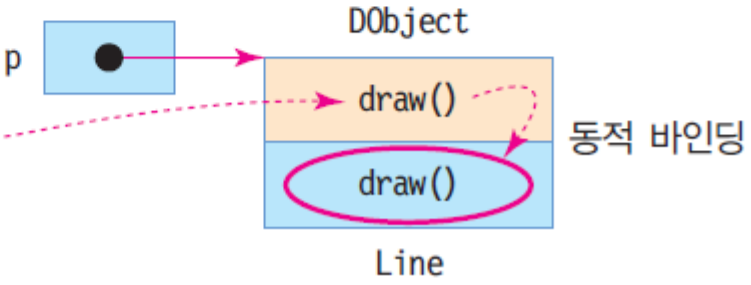
실행 결과: Line이 출력됨



(2) 업캐스팅에 의해 슈퍼 클래스 레퍼런스로 오버라이딩된 메소드 호출(동적 바인딩)

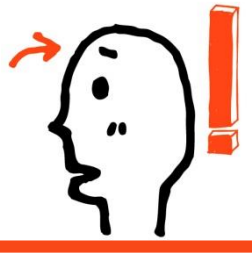
```
DObject p = new Line();
p.draw();
```

실행 결과: Line이 출력됨



하위 클래스에서 메소드를 다시 정의한다면?

메소드 오버라이딩 만들기



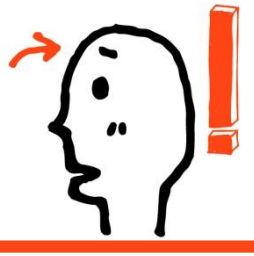
```
class DObject {
   public DObject next;

   public DObject() { next = null;}
    public void draw() {
      System.out.println("DObject draw");
   }
}

class Line extends DObject {
   public void draw() { // 메소드 오버라이딩
       System.out.println("Line");
   }
}

class Rect extends DObject {
   public void draw() { // 메소드 오버라이딩
       System.out.println("Rect");
   }
}

class Circle extends DObject {
   public void draw() { // 메소드 오버라이딩
       System.out.println("Circle");
   }
}
```

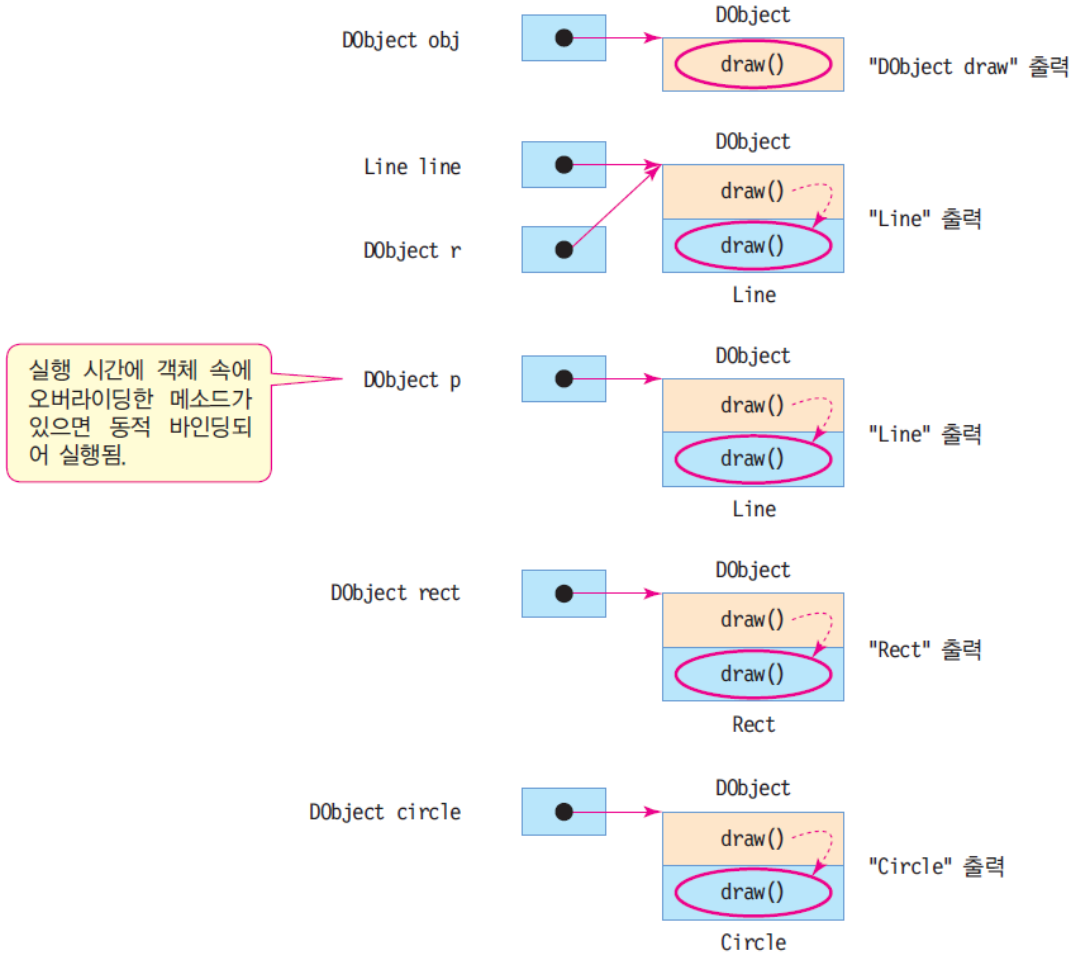
```
public class MethodOverringEx {
   public static void main(String[] args) {
      DObject obj = new DObject();
      Line line = new Line();
      DObject p = new Line();
      DObject r = line;

      obj.draw(); // DObject.draw() 메소드 실행. "DObject draw" 출력
      line.draw(); // Line.draw() 메소드 실행. "Line" 출력
      p.draw(); // 오버라이딩된 메소드 Line.draw() 실행, "Line" 출력
      r.draw(); // 오버라이딩된 메소드 Line.draw() 실행, "Line" 출력

      DObject rect = new Rect();
      DObject circle = new Circle();
      rect.draw(); // 오버라이딩된 메소드 Rect.draw() 실행, "Rect" 출력
      circle.draw(); // 오버라이딩된 메소드 Circle.draw() 실행, "Circle" 출력
   }
}
```

```
DObject draw
Line
Line
Line
Rect
Circle
```

하위 클래스에서 메소드를 다시 정의한다면?

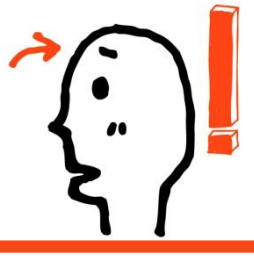
예제실행 과정





하위 클래스에서 메소드를 다시 정의한다면?

오버라이딩 관계에서의 메소드 호출



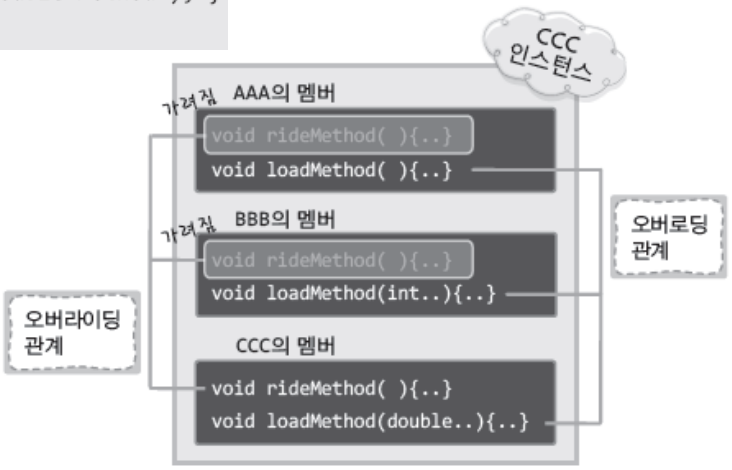
```
class AAA
{
    public void rideMethod() { System.out.println("AAA's Method"); }
    public void loadMethod() { System.out.println("void Method"); }
}
class BBB extends AAA
{
    public void rideMethod() { System.out.println("BBB's Method"); }
    public void loadMethod(int num) { System.out.println("int Method"); }
}
class CCC extends BBB
{
    public void rideMethod() { System.out.println("CCC's Method"); }
    public void loadMethod(double num) { System.out.println("double Method"); }
}
```

참조변수의 자료형에 상관없이 오버라이딩 된 메소드는 외부로부터 가려지므로, 마지막으로 오버라이딩 한 메소드가 호출된다!

```
public static void main(String[] args)
{
    AAA ref1=new CCC();
    BBB ref2=new CCC();
    CCC ref3=new CCC();

    ref1.rideMethod();
    ref2.rideMethod();
    ref3.rideMethod();

    ref3.loadMethod();
    ref3.loadMethod(1);
    ref3.loadMethod(1.2);
}
```

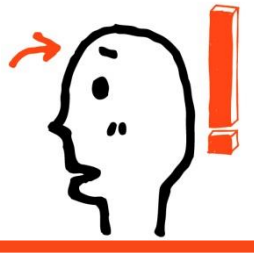
실행결과

```
CCC's Method
CCC's Method
CCC's Method
void Method
int Method
double Method
```

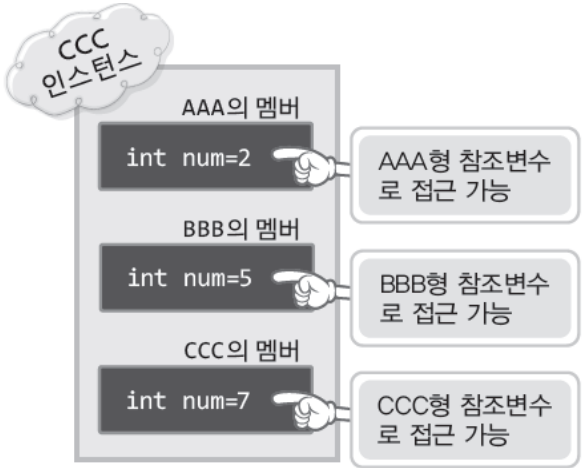


하위 클래스에서 메소드를 다시 정의한다면?

인스턴스 변수도 오버라이딩 되나요?



```
class AAA
{
    public int num=2;
}
class BBB extends AAA
{
    public int num=5;
}
class CCC extends BBB
{
    public int num=7;
}
```



인스턴스 변수는 오버라이딩 관계에 놓이지 않는다.
따라서 참조변수의 자료형에 따라서 접근대상이 결정된다.

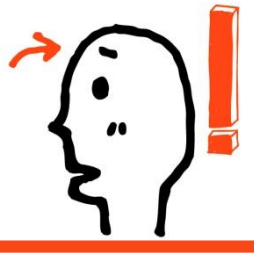
```
public static void main(String[] args)
{
    CCC ref1=new CCC();
    BBB ref2=ref1;
    AAA ref3=ref2;

    System.out.println("CCC's ref : "+ref1.num);
    System.out.println("BBB's ref : "+ref2.num);
    System.out.println("AAA's ref : "+ref3.num);
}
```

실행결과

```
CCC's ref : 7
BBB's ref : 5
AAA's ref : 2
```

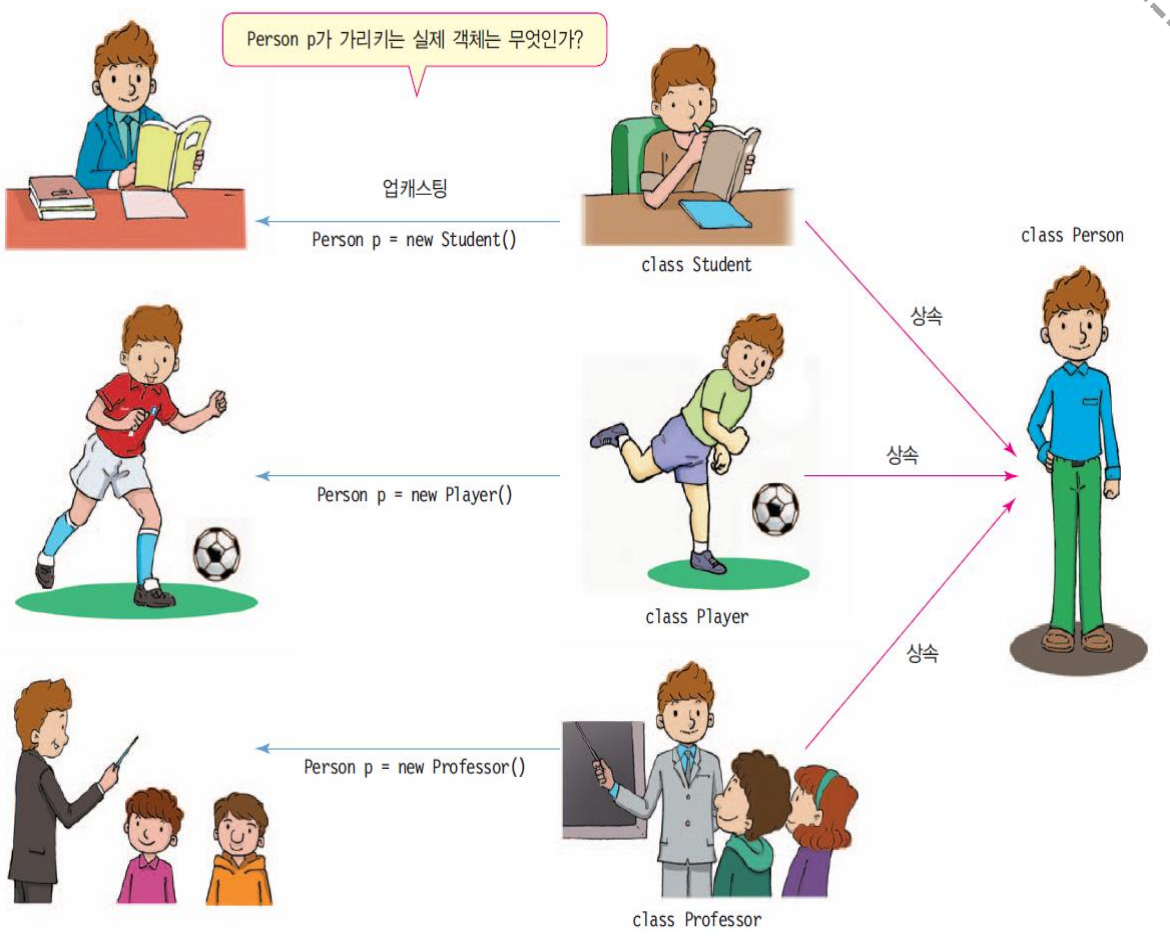
하위 클래스에서 메소드를 다시 정의한다면?

업캐스팅된 객체의 실제 타입은 무엇?



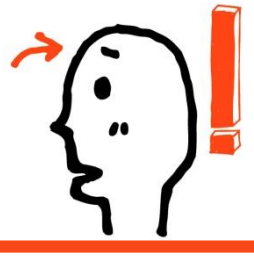
```
class Person {
.....
}
class Student extends Person {
.....
}
class Player extends Person {
.....
}
class Professor extends Person {
.....
}
Person p = new Person();
Person p = new Student();    // 업캐스팅
Person p = new Player();     // 업캐스팅
Person p = new Professor(); // 업캐스팅
```

```
void f(Person p) {
// p가 가리키는 객체가 Person 타입일 수도 있고,
// Student, Player, Professor 타입이 될 수도 있다.
.....
}
```



하위 클래스에서 메소드를 다시 정의한다면?

객체 형 변환하기



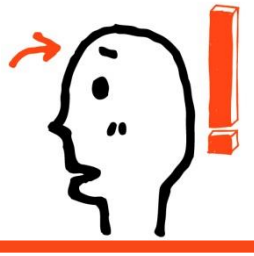
instanceof 키워드

- **instanceof** : 연산자의 일종, 인스턴스(객체)와 데이터형(클래스)을 비교하는 연산자
- instanceof 키워드는 2항 연산자의 형태로 연산자를 기준으로
  왼쪽에는 비교 대상 객체를, 오른쪽에는 비교할 클래스 이름을 위치
- 비교 객체의 원형 클래스가 피연산 클래스와 같거나 자식 클래스인 경우 true를 반환
- 레퍼런스가 가리키는 객체의 진짜 타입 식별

사용법 : [비교 대상 객체] instanceof [데이터명, 클래스 이름];

사용예 :
```
SecondDepth2 obj1 = new SecondDepth2();
obj1 instanceof SecondDepth2          // true
obj1 instanceof SecondDepth1          // false
obj1 instanceof FirstDepth1           // true
obj1 instanceof String                // false
```

객체 형 변환하기



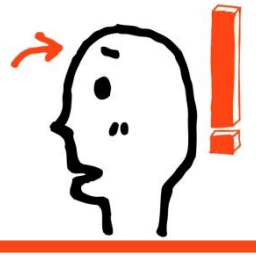
instanceof 키워드

코드 6-10 package com.gilbut.chapter6;

```
1   public class CastingExample
2   {
3      public static void main(String[] args)
4      {
5         CastingExample castingExam = new CastingExample();
6         String str = new String("String");
7         double d = 3.24D;
8         int i = 1;
9         long l = 324L;
10        float f = 3.14F;
11
12        System.out.println(castingExam.getType(str));
13        System.out.println(castingExam.getType(d));
14        System.out.println(castingExam.getType(i));
15        System.out.println(castingExam.getType(l));
16        System.out.println(castingExam.getType(f));
17     }
18
```

이처럼 형 변환 연산자 없이 묵시적 업캐스팅이 가능합니다.

객체 형 변환하기

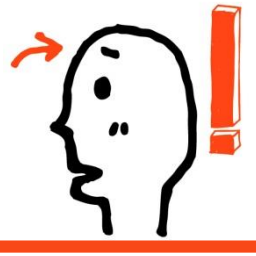


instanceof 키워드

```
19      public String getType(Object obj)
20      {
21          String rt = null;
22
23          if (obj instanceof String)
24              rt = “String object”;
25          else if (obj instanceof Double)
26              rt = “Double object”;
27          else if (obj instanceof Integer)
28              rt = “Integer object”;
29          else if (obj instanceof Long)
30              rt = “Long object”;
31          else
32              rt = “Unknown object”;
33
34          return rt;
35      }
36  }
```

instanceof 키워드는 객체의 클래스형을 비교한다는 것! 잊지 맙시다.

객체 형 변환하기



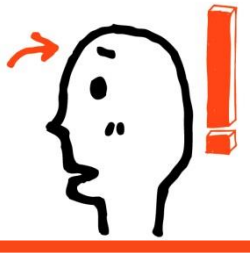
instanceof 키워드

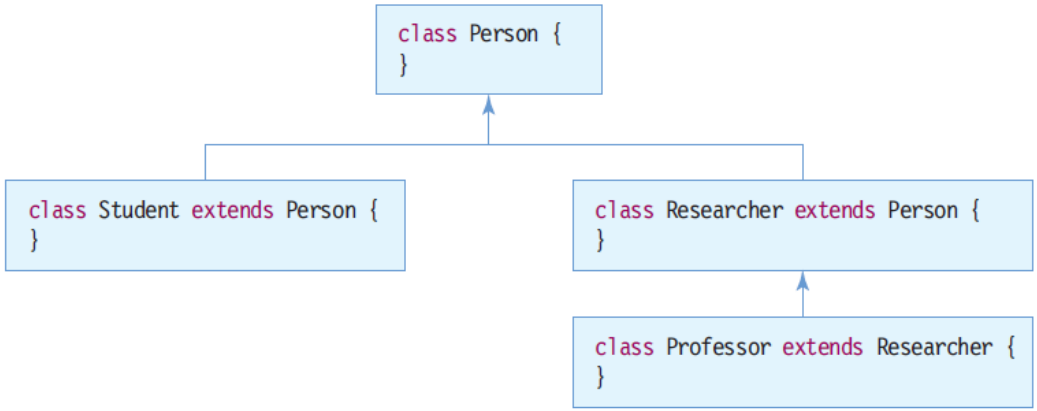
실행결과

```
String object
Double object
Integer object
Long object
Unknown object
```

- str, d, i, l 같은 변수들이 getType( ) 메소드의 매개변수로 들어가면서 자동으로 업캐스팅
- 업캐스팅의 경우는 형 변환 연산자가 생략 가능

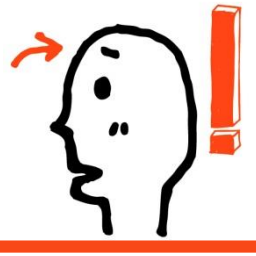
instanceof 사용 예






```
Person jee= new Student();
Person kim = new Professor();
Person lee = new Researcher();
if (jee instanceof Person)          // jee는 Person 타입이므로 true
if (jee instanceof Student)         // jee는 Student 타입이므로 true
if (kim instanceof Student)         // kim은 Student 타입이 아니므로 false
if (kim instanceof Professor)       // kim은 Professor 타입이므로 true
if (kim instanceof Researcher)      // kim은 Researcher 타입이기도 하므로 true
if (lee instanceof Professor)       // lee는 Professor 타입이 아니므로 false
if ("java" instanceof String)       // "java"는 String 타입의 인스턴스이므로 true
if (3 instanceof int)       // 문법 오류. instanceof는 객체에 대한 레퍼런스에만 사용
```

instanceof 연산자



- 형변환이 가능한지를 묻는 연산자이다.
- 형변환이 가능하면 true를 가능하지 않으면 false를 반환.

```
class Box
{
    public void simpleWrap() { . . . }
}

class PaperBox extends Box
{
    public void paperWrap() { . . . }
}

class GoldPaperBox extends PaperBox
{
    public void goldWrap() { . . . }
}
```

```
public static void wrapBox(Box box)
{
    if(box가 GoldPaperBox로 형변환 가능하다면)
        ((GoldPaperBox)box).goldWrap();
    else if(box가 PaperBox로 형변환 가능하다면)
        ((PaperBox)box).paperWrap();
    else
        box.simpleWrap();
}
```

```
public static void wrapBox(Box box)
{
    if(box instanceof GoldPaperBox)
        ((GoldPaperBox)box).goldWrap();
    else if(box instanceof PaperBox)
        ((PaperBox)box).paperWrap();
    else
        box.simpleWrap();
}
```

THANK YOU

실무에서 알아야 할 기술은 따로 있다! 자바를 다루는 기술