

# 제5장 상속과 다형성

2018년1학기

컴퓨터공학과 김 명 교수

# 상속 ( Inheritance )

2

## □ 상속

- 상위 클래스의 특성 (필드, 메소드)을 하위 클래스에 물려주는 것
- 슈퍼 클래스 (superclass)
  - 특성을 물려주는 상위 클래스
- 서브 클래스 (subclass)
  - 특성을 물려 받는 하위 클래스
  - 슈퍼 클래스에 자신만의 특성(필드, 메소드)을 **추가**
  - 슈퍼 클래스의 특성(메소드)을 **수정** : 오버라이딩 (overriding)

## □ 상속을 하는 이유는?

- 동일한 특성을 재정의하지 않고 서브 클래스를 정의할 수 있음

# 5개의 클래스

3

Person

말하기  
먹기  
걷기  
잠자기

Student

말하기  
먹기  
걷기  
잠자기  
공부하기

StudentWorker

말하기  
먹기  
걷기  
잠자기  
공부하기  
일하기

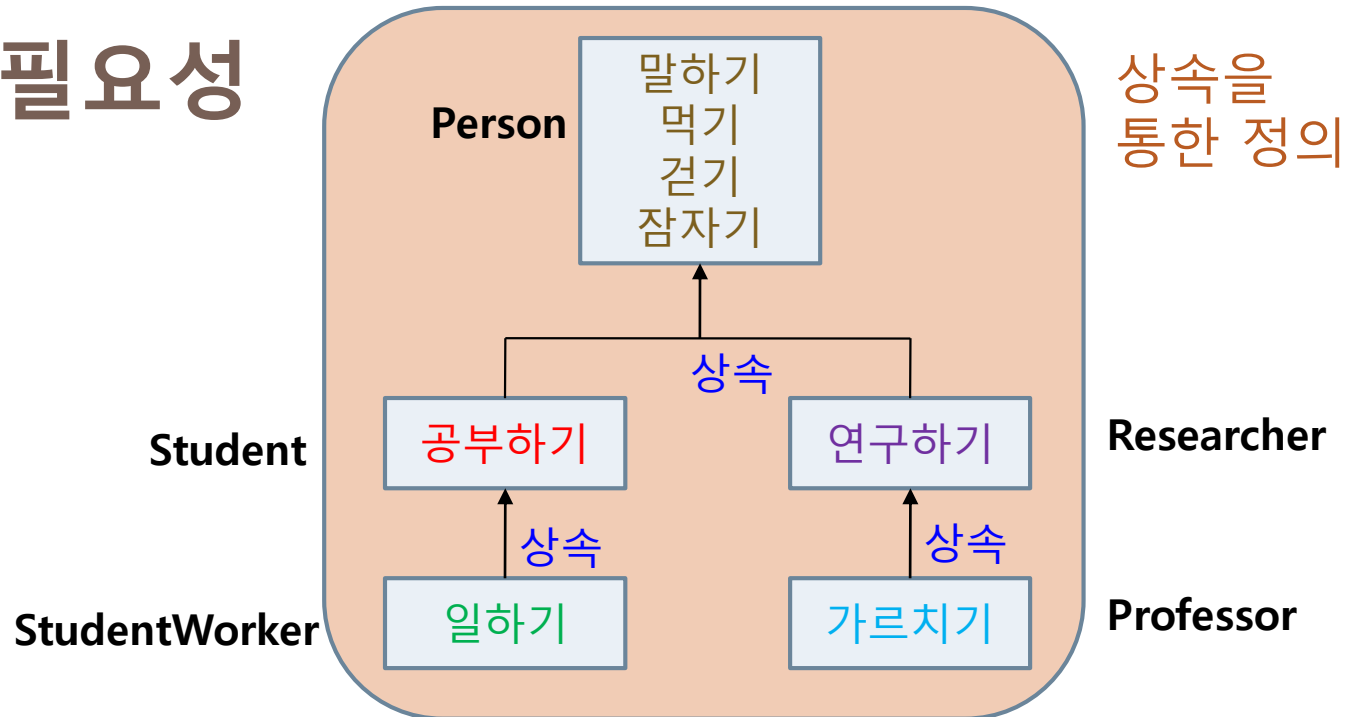
Researcher

말하기  
먹기  
걷기  
잠자기  
연구하기

Professor

말하기  
먹기  
걷기  
잠자기  
연구하기  
가르치기

## 상속의 필요성



# 클래스 상속과 객체

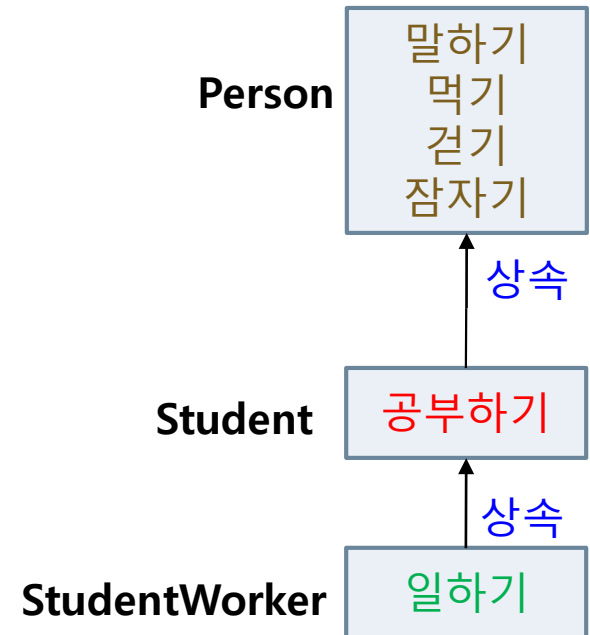
4

## □ 상속 선언

```
public class Person {  
    // ....  
}  
public class Student extends Person {  
    // ....  
}  
public class StudentWorker extends Student {  
    // ....  
}
```

## □ 자바 상속의 특징

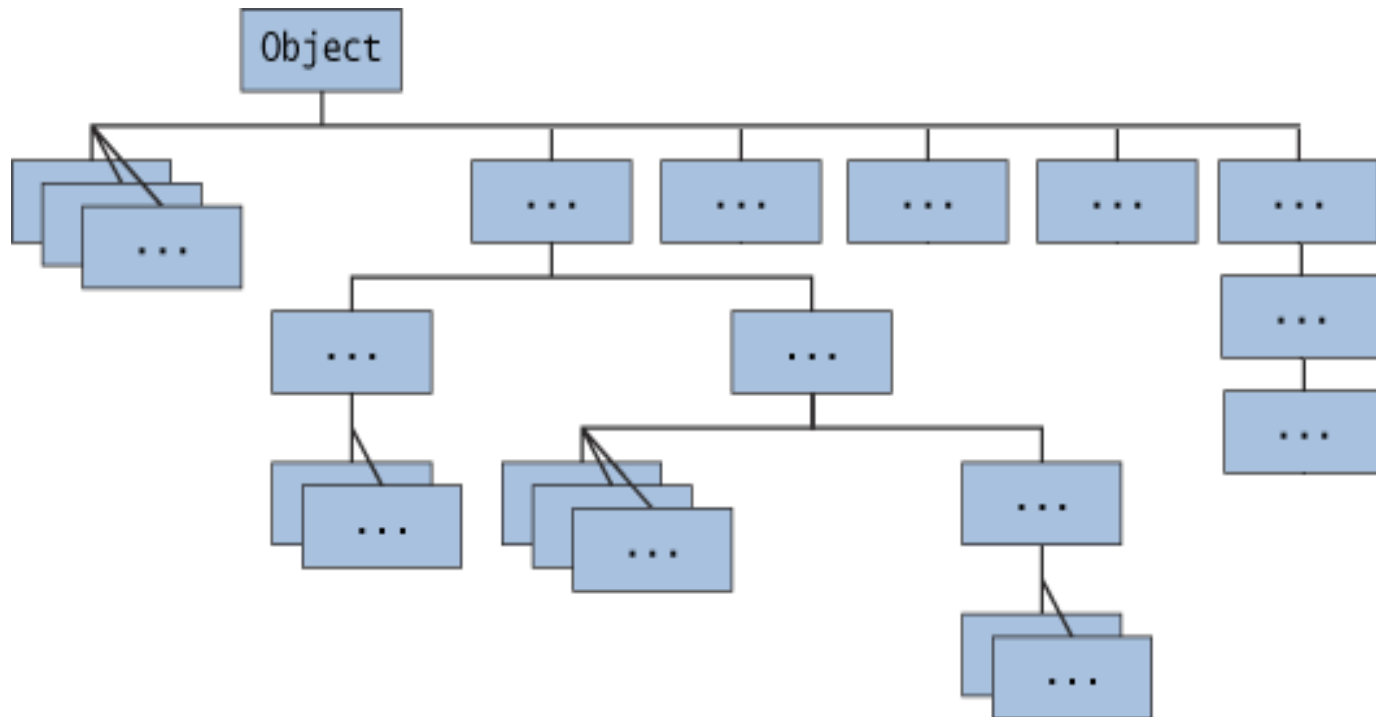
- 다중 상속을 지원하지 않는다.
- 상속의 횟수에 제한을 두지 않는다.
- 계층구조의 최상위에 있는 클래스는 java.lang.Object 클래스이다.



# 자바 클래스 계층 구조

5

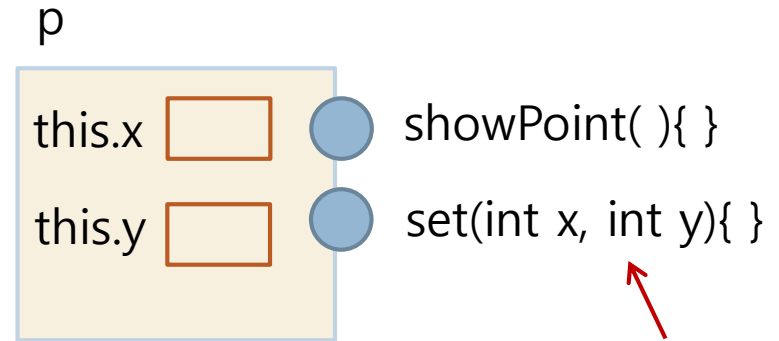
모든 클래스는 `java.lang.Object` 클래스를 자동으로 상속받는다.



# (예제1) Point 클래스

6

```
public class Point {  
    int x, y;  
    void set (int x, int y){  
        this.x = x;  
        this.y = y;  
    }  
    void showPoint(){  
        System.out.println("(" + x + ", " + y + ")");  
    }  
}  
  
public class PointDemo2 {  
    public static void main(String[] args) {  
        Point p = new Point();  
        p.set(10, 20);  
        p.showPoint();  
    }  
}
```



p.set(10, 20); 호출

Problems @ Javado

<terminated> PointDemo.

(10, 20)

# (상속 예제1) Point → ColorPoint

7

```
public class Point {  
    int x, y;  
    void set (int x, int y){  
        this.x = x;  
        this.y = y;  
    }  
    void showPoint(){  
        System.out.println("(" + x + "," + y + ")");  
    }  
}
```

상속

```
public class ColorPoint extends Point{  
    String color; // 점의 색  
    void setColor(String color){  
        this.color = color;  
    }  
    void showColorPoint(){  
        System.out.print(color); // 점의 색 출력  
        showPoint(); // Point 클래스에서 정의된 showPoint() 호출  
    }  
}
```

# (상속 예제1) ColorPoint 클래스

8

- Point 클래스를 상속받지 않고 독립적으로 선언된 ColorPoint 클래스의 모양

Point  
클래스

```
public class ColorPoint {  
    int x, y;  
    void set (int x, int y){  
        this.x = x;  
        this.y = y;  
    }  
    void showPoint(){  
        System.out.println("(" + x + ", " + y + ")");  
    }  
    String color; // 점의 색  
    void setColor(String color){  
        this.color = color;  
    }  
    void showColorPoint(){  
        System.out.print(color); // 점의 색 출력  
        showPoint(); // Point 클래스에서 정의된 showPoint() 호출  
    }  
}
```



# (상속 예제1) Point → ColorPoint 상속

9

```
public class PointDemo2 {  
    public static void main(String[] args) {  
        Point p = new Point();  
        p.set(10, 20);  
        p.showPoint();  
        ColorPoint cp = new ColorPoint();  
        cp.set(3, 4); // 좌표 지정  
        cp.setColor("red"); // 색 지정  
        cp.showColorPoint(); // 점 출력  
    }  
}
```

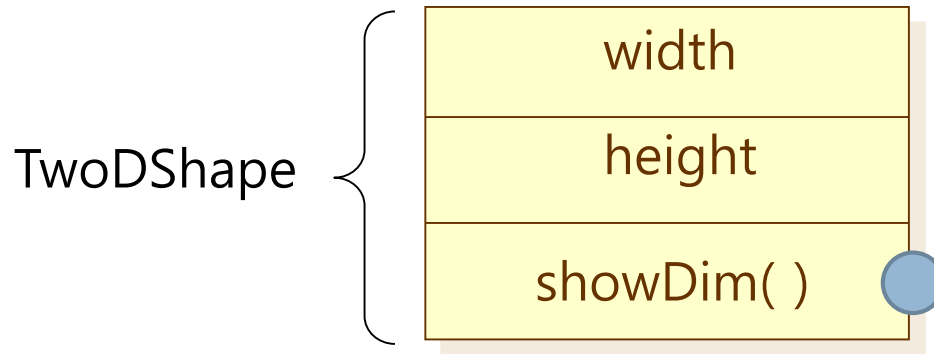
(10, 20)  
red(3, 4)

실행결과

## (예제2) TwoDShape (2차원 도형) 클래스

10

```
public class TwoDShape {  
    public double width;  
    public double height;  
    public void showDim() {  
        System.out.println("Width and height are " + width  
            + " and " + height);  
    }  
}
```



# (예제2) TwoDShape 클래스 사용 예제

11

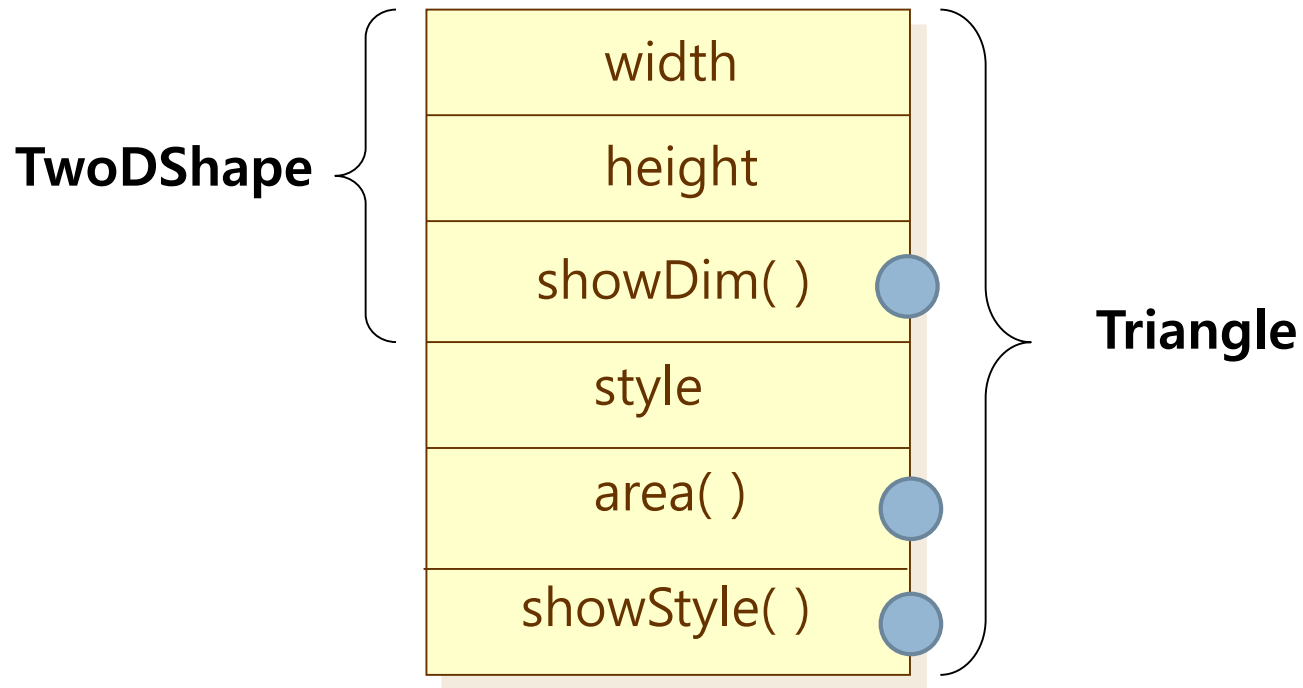
```
public class Shapes {  
    public static void main(String[] args) {  
        TwoDShape shape = new TwoDShape();  
        shape.width = 10;  
        shape.height = 20;  
        shape.showDim();  
    }  
}
```

실행결과

Width and height are 10.0 and 20.0

# (상속 예제2) TwoDShape → Triangle

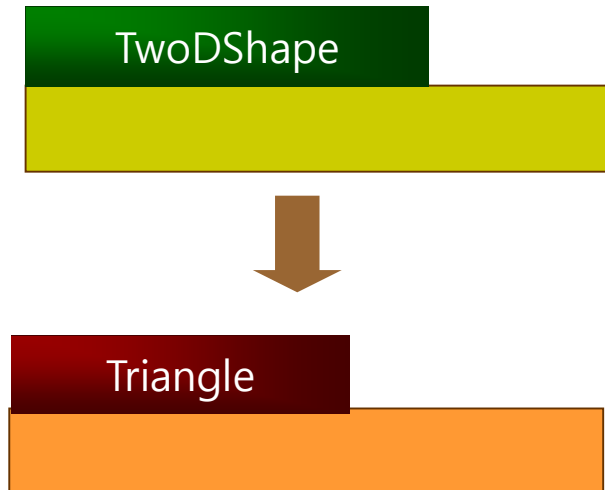
12



# (상속 예제2) Triangle 클래스

13

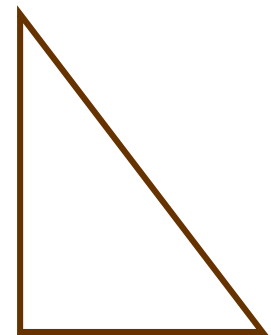
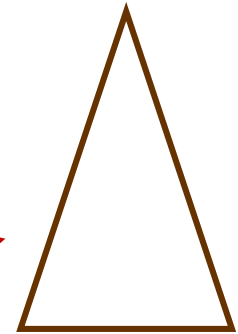
```
public class Triangle extends TwoDShape {  
    public String style;  
    public double area(){  
        return width * height / 2;  
    }  
    public void showStyle(){  
        System.out.println("Triangle si " + style);  
    }  
}
```



# (상속 예제2) Triangle 클래스의 사용 (1/2)

14

```
public class Shapes {  
    public static void main(String[] args) {  
        Triangle t1 = new Triangle();  
        Triangle t2 = new Triangle();  
        t1.width = 4.0;  
        t1.height = 4.0;  
        t1.style = "isosceles";  
  
        t2.width = 8.0;  
        t2.height = 12.0;  
        t2.style = "right";  
    }  
}
```



# (상속 예제2) Triangle 클래스의 사용 (2/2)

15

```
System.out.println("Info for t1:");  
t1.showStyle();  
t1.showDim();  
System.out.println("Area is " + t1.area());  
System.out.println();
```

Info for t1:  
Triangle is isosceles  
Width and height are 4 and 4  
Area is 8

```
System.out.println("Info for t2:");  
t2.showStyle();  
t2.showDim();  
System.out.println("Area is " + t2.area());
```

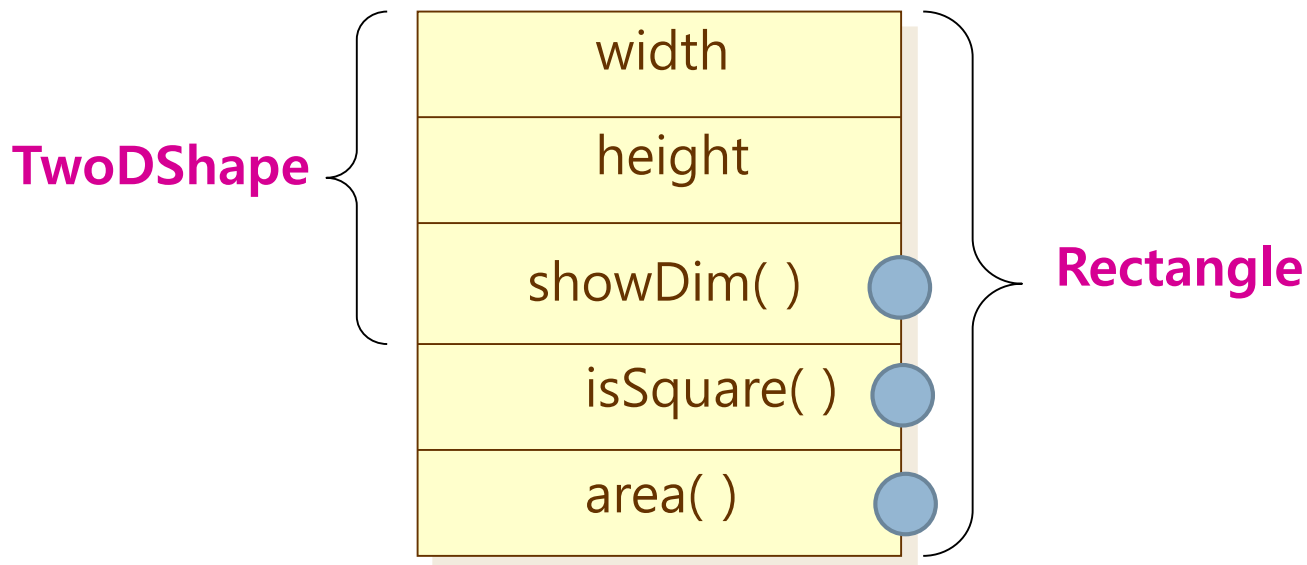
Info for t2:  
Triangle is right  
Width and height are 8 and 12  
Area is 48

```
}
```

```
}
```

# (상속 예제3) TwoDShape → Rectangle

16





## (상속 예제3) TwoDShape → Rectangle

17

```
public class Rectangle extends TwoDShape {  
    public boolean isSquare(){  
        if (width == height) return true;  
        return false;  
    }  
  
    public double area(){  
        return width * height;  
    }  
}
```

# (정리) 서브 클래스의 객체와 멤버

18

- 서브 클래스의 객체와 멤버
  - ▣ 서브 클래스의 객체는 슈퍼 클래스의 멤버도 포함
  - ▣ 슈퍼 클래스의 private 멤버는 상속되지 않음
    - 서브 클래스에서 직접 접근 불가
    - 슈퍼 클래스의 메소드를 통해서만 접근 가능
  - ▣ 서브 클래스 객체에 슈퍼 클래스 멤버가 포함되므로  
슈퍼 클래스 멤버의 접근은 서브 클래스 멤버 접근과 동일

```

public class A {
    public int p;
    private int n;
    public void setN(int n){
        this.n = n;
    }
    public int getN(){
        return n;
    }
}

```

```

public static void main(String[] args) {
    A a = new A();
    B b = new B();
}

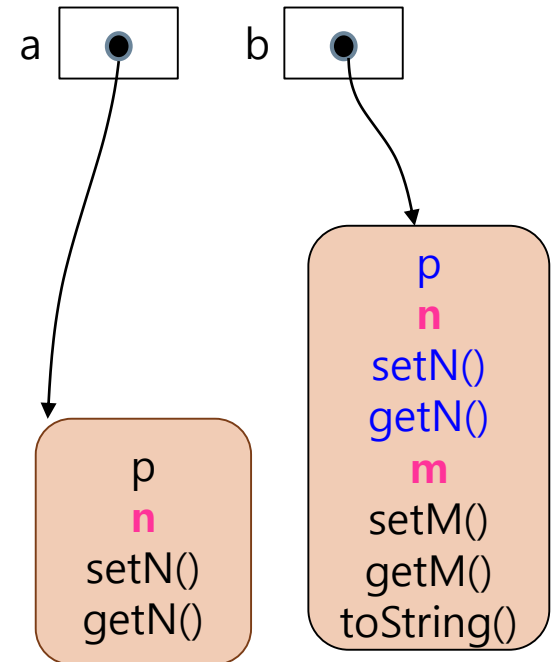
```

상속

```

public class B extends A {
    private int m;
    public void setM (int m){
        this.m = m;
    }
    public int getM(){
        return m;
    }
    public String toString(){
        String s = getN() + " " + getM();
        return s;
    }
}

```



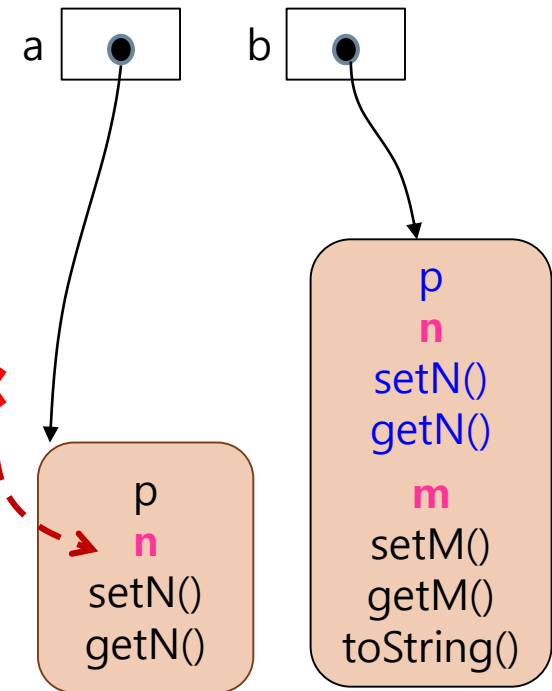
main( ) 실행 중 생성된 인스턴스

슈퍼 클래스 A와 서브 클래스 B 그리고 인스턴스 관계

# 서브 클래스의 객체 멤버 접근

20

```
public class ABDemo {  
    public static void main(String[] args) {  
        A a = new A();  
        B b = new B();  
  
        a.p = 5;  
        a.n = 5; // n 은 private 멤버, 컴파일 오류 발생  
  
        b.p = 5;  
        b.n = 5; // n 은 private 멤버, 컴파일 오류 발생  
        b.setN(10);  
        int i = b.getN(); // i는 10  
  
        b.m = 20; // m은 private 멤버, 컴파일 오류 발생  
        b.setM(20);  
        System.out.println(b.toString());  
    }  
}
```



실행결과

10 20

# 상속과 접근 지정자

21

- 자바의 접근 지정자
  - ▣ public, protected, default, private
    - 상속 관계에서 주의할 접근 지정자는 private와 protected
- private 멤버
  - ▣ 슈퍼 클래스의 private 멤버는 서브 클래스를 포함한 다른 모든 클래스에서 접근할 수 없다.
- protected 멤버
  - ▣ 같은 패키지 내의 모든 클래스에서 접근할 수 있다.
  - ▣ 동일 패키지 여부와 상관없이 서브 클래스에서 슈퍼 클래스의 멤버에 접근할 수 있다.

# 슈퍼 클래스 멤버의 접근 지정자

22

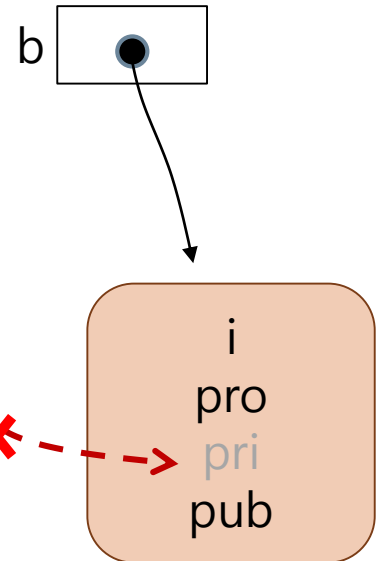
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
같은 패키지의 서브 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O
다른 패키지의 서브 클래스	X	X	O	O

# 동일 패키지 안의 클래스 상속

23

```
public class A {  
    int i;  
    protected int pro;  
    private int pri;  
    public int pub;  
}  
  
public class B extends A {  
    void set(){  
        i = 1;  
        pro = 2;  
        pri = 3; // private 멤버 접근 불가, 컴파일 오류  
        pub = 4;  
    }  
}  
  
public class ABDemo {  
    public static void main(String[] args) {  
        B b = new B();  
        b.set();  
    }  
}
```

패키지 PackageA



# 다른 패키지의 클래스를 상속

24

```
public class A {  
    int i;  
    protected int pro;  
    private int pri;  
    public int pub;  
}
```

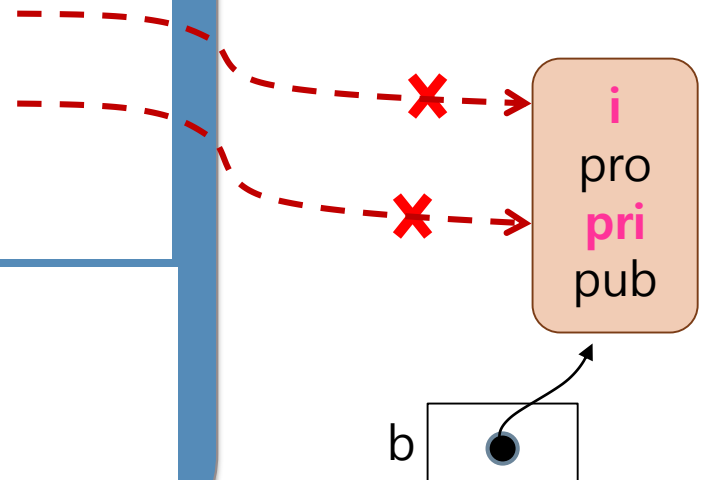
패키지 PackageA



```
public class B extends A{  
    void set(){  
        i = 1; // default 멤버, 오류  
        pro = 2;  
        pri = 3; // private 멤버, 오류  
        pub = 4;  
    }  
}
```

패키지 PackageB

```
public class ABDemo2 {  
    public static void main(String[] args) {  
        B b = new B();  
        b.set();  
    }  
}
```

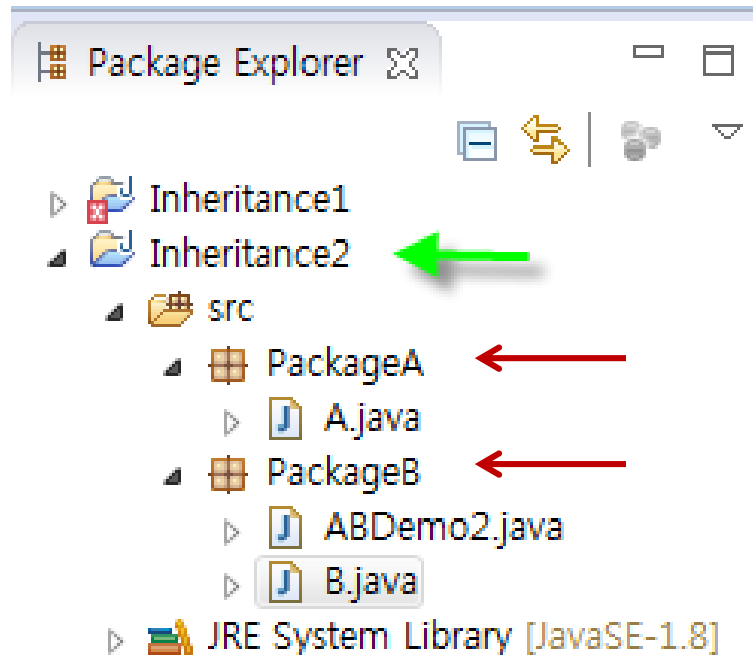




# (참고1) 패키지 사용하려면...

25

- 프로젝트를 생성하고, class 를 정의하기 전에 패키지를 만든다. 만들어 놓은 패키지 속에 class 들을 만들면 된다.




# (참고1) 패키지 사용하려면...

26


- 패키지 속에 클래스를 생성하면, 패키지명이 자동 생성된다.

```
package PackageA;
public class A {
    int i;
    protected int pro;
    private int pri;
    public int pub;
}
```



- 패키지 밖의 클래스를 상속할 때는 어떤 패키지의 어떤 클래스를 상속하는가를 import 문을 통해 써 주어야 한다.


```
package PackageB;
import PackageA.A; // 클래스 A 가 있는 곳 명시
public class B extends A{
    void set(){
        i = 1; // default 멤버, 오류
        pro = 2;
        pri = 3; // private 멤버, 오류
        pub = 4;
    }
}
```



# (참고1) 패키지 사용하려면...

27

- 클래스 B 는 클래스 A 를 상속받기 때문에 import PackageA.A 를 써 주어야 하지만, ABDemo2 에서는 클래스 A의 이름을 언급하지 않으므로 그렇게 할 필요가 없다.

```
package PackageB;   
public class ABDemo2 {  
    public static void main(String[] args) {  
        B b = new B();  
        b.set();  
    }  
}
```

# 상속 관계에 있는 클래스 간 멤버 접근

28

```
class Person {  
    int age;  
    public String name;  
    protected int height;  
    private int weight;  
  
    public void setWeight(int weight) {  
        this.weight = weight;  
    }  
  
    public int getWeight() {  
        return weight;  
    }  
}
```

```
public class MM {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.set();  
    }  
}
```

상속

```
public class Student extends Person {  
    void set() {  
        age = 30;  
        name = "홍길동";  
        height = 175;  
        setWeight(99);  
    }  
}
```

# 슈퍼클래스와 서브 클래스의 생성자간의 호출 및 실행 관계

29

서브 클래스의 생성자가 먼저 호출되지만, 계속하여 슈퍼 클래스의 생성자를 호출하고, 최상위 슈퍼 클래스의 생성자가 실행되면서 아래로 최하위 서브 클래스의 생성자가 실행되는 과정을 거친다.

생성자 호출 ③

```
class A {  
    public A ( ) {  
        System.out.println("생성자A");  
    }  
}
```

생성자 실행 ④

리턴

생성자 호출 ②

```
class B extends A {  
    public B ( ) {  
        System.out.println("생성자B");  
    }  
}
```

생성자 실행 ⑤

생성자 호출 ①

```
class C extends B {  
    public C ( ) {  
        System.out.println("생성자C");  
    }  
}
```

생성자 실행 ⑥

예상 실행 결과는 ?

생성자A  
생성자B  
생성자C

```
public class ConstructorEx {  
    public static void main(String[] args) {  
        C c;  
        c = new C ( );  
    }  
}
```

# 서브 클래스와 슈퍼 클래스의 생성자 짝 맞추기

30

- 슈퍼 클래스와 서브 클래스 : 각각 여러 개의 생성자를 가질 수 있다.
- 슈퍼 클래스와 서브 클래스의 생성자 짝 맞추기
  - ▣ 서브클래스의 객체가 생성될 때, 실행 가능한 슈퍼 클래스와 서브 클래스의 생성자를 조합한다.
    - default : 경우 1, 3
    - 개발자 지정 : 경우 2, 4 (super( ) 키워드 사용)

경우	1	2	3	4
서브 클래스	기본 생성자	기본 생성자	매개 변수를 가진 생성자	매개 변수를 가진 생성자
슈퍼 클래스	기본 생성자	매개 변수를 가진 생성자	기본 생성자	매개 변수를 가진 생성자

## case 1: 슈퍼클래스(기본생성자),서브클래스(기본생성자)

31

서브 클래스의 생성자가  
기본생성자인 경우  
컴파일러는 자동으로  
슈퍼클래스의 기본생성  
자와 짝을 맺는다.

```
class A {  
    public A() {  
        System.out.println(" 생성자 A");  
    }  
    public A(int x) {  
        // .....  
    }  
}
```

```
class B extends A {  
    public B() {  
        System.out.println("생성자 B");  
    }  
}
```

실행결과

생성자 A  
생성자 B

```
public class ConstructorEx2 {  
    public static void main(String[] args) {  
        B b;  
        b = new B();  
    }  
}
```

## case 1: 슈퍼클래스(기본생성자),서브클래스(기본생성자)

32

컴파일러가  
public B()에 대한  
짝을 찾을 수 없음

```
class A {  
    public A(int x) {  
        System.out.println(" 생성자 A");  
    }  
}
```

```
class B extends A {  
    public B() {  
        System.out.println("생성자 B");  
    }  
}
```

```
public class ConstructorEx2 {  
    public static void main(String[] args) {  
        B b;  
        b = new B();  
    }  
}
```

컴파일러에 의해 "Implicit super constructor A() is undefined.  
Must explicitly invoke another constructor" 오류 메시지가 발생



## case 3: 서브 클래스에 매개변수 있는 생성자는 슈퍼클래스의 기본생성자와 짝을 이룸

33

```
class A {  
    public A() {  
        System.out.println(" 생성자 A");  
    }  
    public A(int x) {  
        System.out.println("매개변수 생성자 A");  
    }  
}
```

```
class B extends A {  
    public B() {  
        System.out.println("생성자 B");  
    }  
    public B(int x) {  
        System.out.println("매개변수 생성자 B");  
    }  
}
```

```
public class ConstructorEx2 {  
    public static void main(String[] args) {  
        B b;  
        b = new B(5);  
    }  
}
```

실행결과

생성자 A  
매개변수 생성자 B

# super()

34

- super()
  - ▣ 서브 클래스에서 명시적으로 슈퍼 클래스의 생성자를 선택 호출할 때 사용
  - ▣ 사용 방식
    - `super(parameter);`
    - 인자를 이용하여 슈퍼 클래스의 적당한 생성자 호출
    - 반드시 서브 클래스 생성자 코드의 제일 첫 라인에 와야 한다.

# super( )를 이용한 생성자 호출 사례

35

```
class A {  
    public A() {  
        System.out.println(" 생성자 A ");  
    }  
    public A(int x) {  
        System.out.println("매개변수 생성자 A " + x);  
    }  
}
```

```
class B extends A {  
    public B() {  
        System.out.println("생성자 B");  
    }  
    public B(int x) {  
        super(x);  
        System.out.println("매개변수 생성자 B " + x);  
    }  
}
```

```
public class ConstructorEx2 {  
    public static void main(String[] args) {  
        B b;  
        b = new B(5);  
    }  
}
```

매개변수 생성자 A 5  
매개변수 생성자 B 5

## (생성자 예제1)

### 생성자를 사용하는 Triangle 클래스 (1/2)

36

```
public class Triangle extends TwoDShape {
    public String style;
    public Triangle(String s, double w, double h) {
        width = w; // 슈퍼 클래스 변수
        height = h; // 슈퍼 클래스 변수
        style = s; // 서브 클래스 변수
    }
    public double area(){
        return width * height / 2;
    }
    public void showStyle(){
        System.out.println("Triangle si " + style);
    }
}
```

## (생성자 예제1)

### 생성자를 사용하는 Triangle 클래스 (2/2)

37

```
public class Shapes {  
    public static void main(String[] args) {  
        Triangle t1 = new Triangle("isosceles", 4.0, 4.0);  
        Triangle t2 = new Triangle("right", 8.0, 12.0);  
  
        System.out.println("Info for t1:");  
        t1.showStyle();  
        t1.showDim();  
        System.out.println("Area is " + t1.area());  
        System.out.println();  
  
        System.out.println("Info for t2:");  
        t2.showStyle();  
        t2.showDim();  
        System.out.println("Area is " + t2.area());  
    }  
}
```

## (생성자 예제1)

### 생성자를 사용하는 TwoDShape 클래스

38

```
public class TwoDShape {  
    public double width;  
    public double height;  
    public TwoDShape(){};  
    public TwoDShape(double w, double h) {  
        width = w;  
        height = h;  
    }  
    public void showDim() {  
        System.out.println("Width and height are " + width  
            + " and " + height);  
    }  
}
```

꼭 있어야 함


width
height
showDim( )

## (생성자 예제2)

### super 를 사용하는 Triangle 클래스

39

```
public class Triangle extends TwoDShape{
    public String style;
    public Triangle(String s, double w, double h) {
        width = w;    // 슈퍼 클래스 변수
        height = h;   // 슈퍼 클래스 변수
        style = s;    // 서브 클래스 변수
    }
    public double area() {
        return width * height / 2;
    }
    public void showStyle() {
        System.out.println("Triangle is " + style);
    }
}
```



# 객체의 타입 변환 : 업캐스팅

40

- 업캐스팅 (upcasting)
  - ▣ 프로그램에서 이루어지는 자동 타입 변환
  - ▣ 서브 클래스의 레퍼런스 값을 슈퍼 클래스의 레퍼런스에 대입
    - 슈퍼 클래스의 레퍼런스가 서브 클래스 객체를 가리키게 되는 현상
    - 슈퍼 클래스의 레퍼런스는 서브 클래스 객체 내에 있는 모든 멤버들을 접근할 수 없고, 슈퍼 클래스에서 정의된 멤버들만 접근 가능

```
class Person {  
}  
  
class Student extends Person {  
}  
  
Student s = new Student();  
Person p = s; // 업캐스팅, 자동타입변환
```

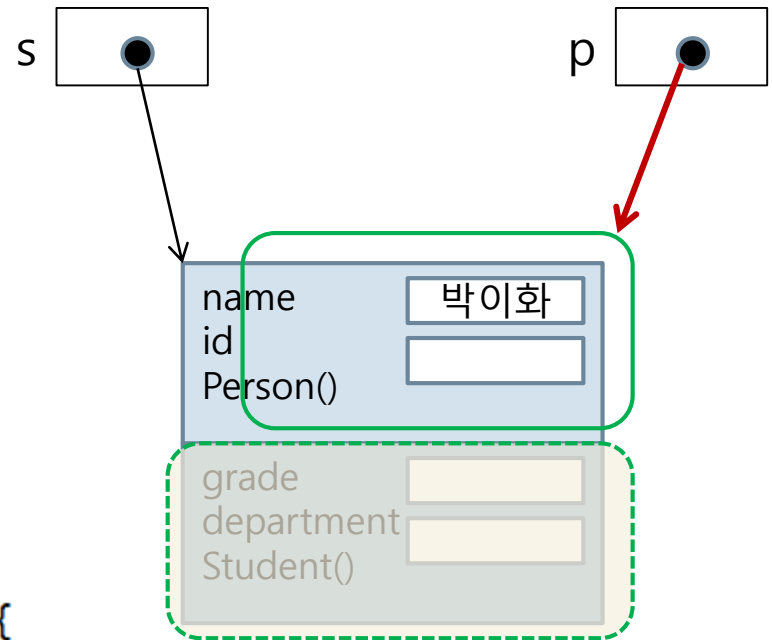


# 업캐스팅 사례

```
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("박이화");
        p = s; // 업캐스팅 발생
        System.out.println(p.name); // 오류 없음
        p.grade = "A"; // 컴파일 오류
        p.department = "CSE"; // 컴파일 오류
    }
}
```



레퍼런스 P를 이용하면  
Student 객체의 속성 중에서  
오직 Person에서 정의된 속  
성들만 접근할 수 있다.

# 객체의 타입 변환 : 다운캐스팅

42

- 다운캐스팅 (downcasting)
  - ▣ 슈퍼 클래스 레퍼런스를 서브 클래스 레퍼런스에 대입하는 것
  - ▣ 업캐스팅된 것을 다시 원래대로 되돌리는 것
  - ▣ 다운캐스팅할 타입을 명시적으로 지정해야 함

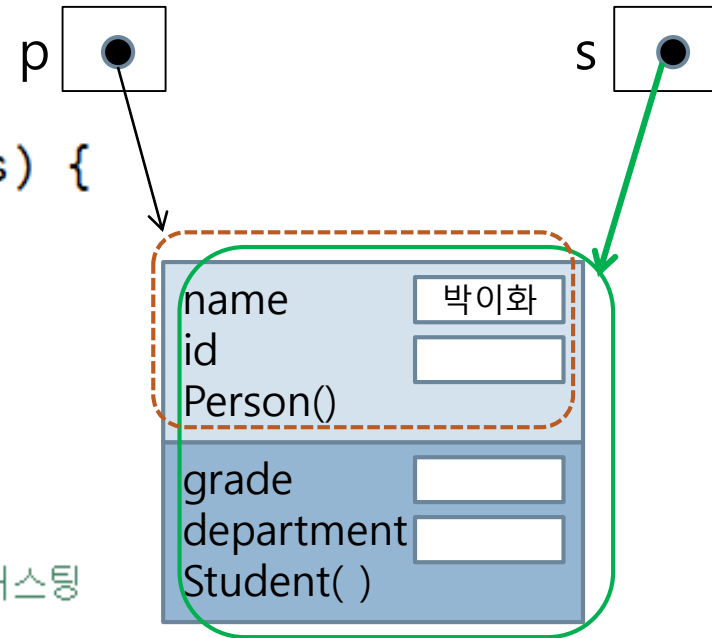
```
class Person {  
}  
class Student extends Person {  
}
```

```
Student s = (Student) p; // 다운캐스팅, 강제타입변환
```

# 다운캐스팅 사례

43

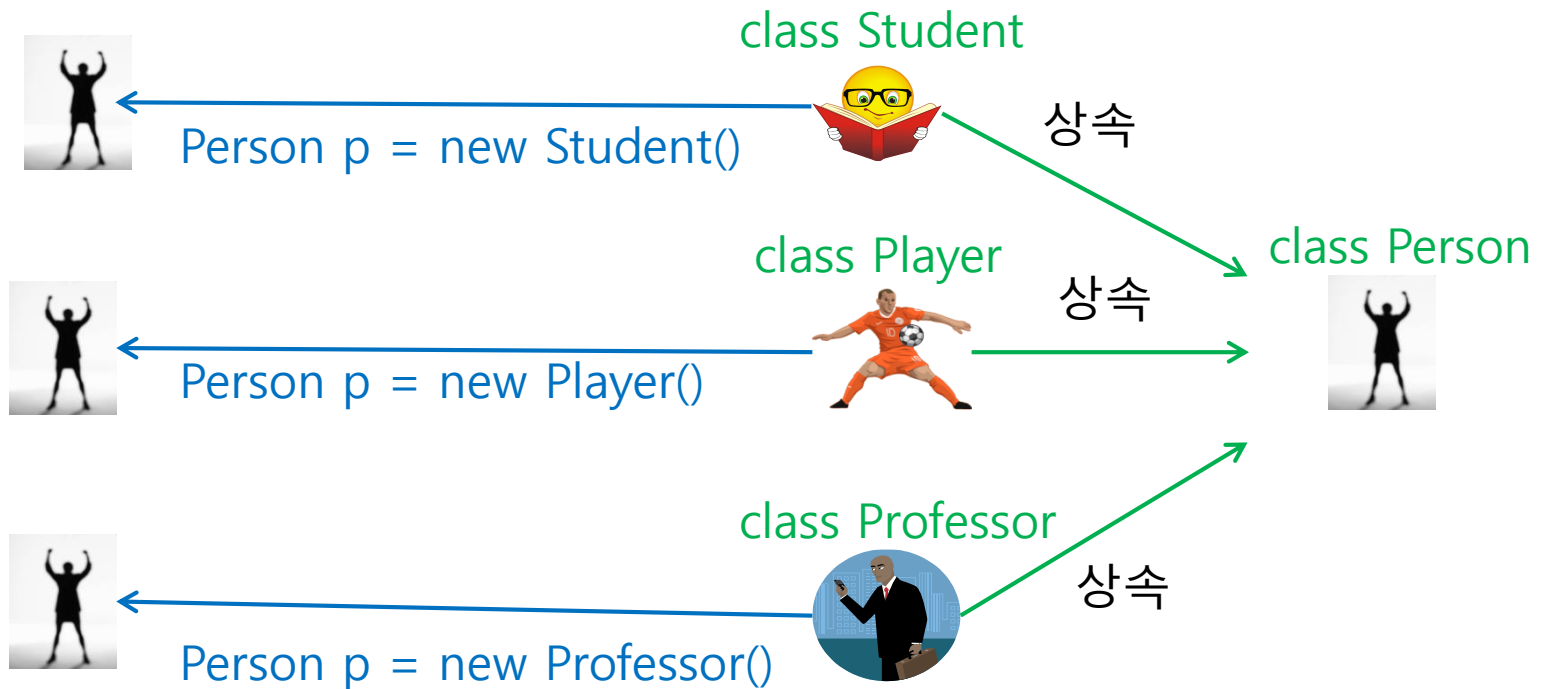
```
public class UpcastingEx {  
    public static void main(String[] args) {  
        Person p;  
        Student s = new Student("박이화");  
  
        p = s; // 업캐스팅  
        System.out.println(p.name);  
  
        Student s2 = (Student) p; // 다운캐스팅  
  
        s2.grade = "A";  
        s2.department = "CSE";  
        System.out.println(s2.name);  
    }  
}
```



# 객체 구별?

44

Person p가 가리키는 실제 객체는 무엇인가?



# instanceof 연산자와 객체 구별

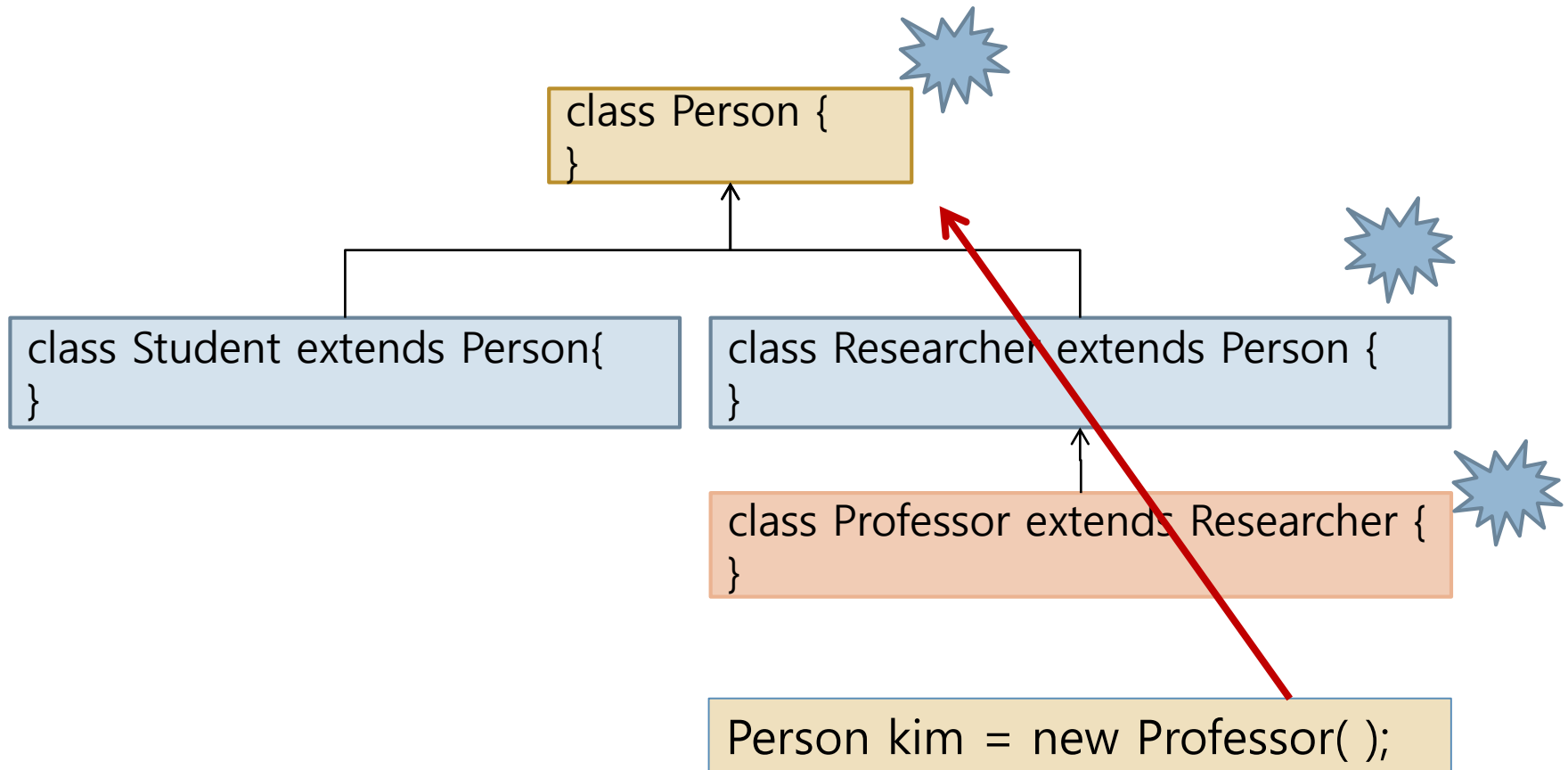
45

- 업캐스팅된 레퍼런스로 객체의 원래 타입을 구분하기 어려움
  - ▣ 하나의 슈퍼 클래스는 여러 서브 클래스에 상속될 수 있기 때문!!!
- instanceof 연산자
  - ▣ instanceof를 이용하여 레퍼런스가 가리키는 객체의 원래 타입을 식별함
  - ▣ 사용법 :

객체 레퍼런스 instanceof 클래스 타입 --> true/false의 불린 값

# instanceof 사용 예제

46



# instanceof를 이용한 객체 구별

47

```
public class InstanceofDemo {  
    public static void main(String[] args) {  
        Person jee = new Student();  
        Person kim = new Professor();  
        Person lee = new Researcher();  
  
        if (jee instanceof Student)  
            System.out.println("jee : Student");  
        if (jee instanceof Researcher)  
            System.out.println("jee : Researcher");  
        if (kim instanceof Student)  
            System.out.println("kim : Student");  
        if (kim instanceof Professor)  
            System.out.println("kim : Professor");  
        if (kim instanceof Researcher)  
            System.out.println("kim : Researcher");  
        if (kim instanceof Person)  
            System.out.println("kim : Person");  
        if (lee instanceof Professor)  
            System.out.println("lee : Professor");  
        if ("java" instanceof String)  
            System.out.println("\"java\": String");  
    }  
}
```

## 실행결과

```
jee : Student  
kim : Professor  
kim : Researcher  
kim : Person  
"java": String
```

# 메소드 오버라이딩

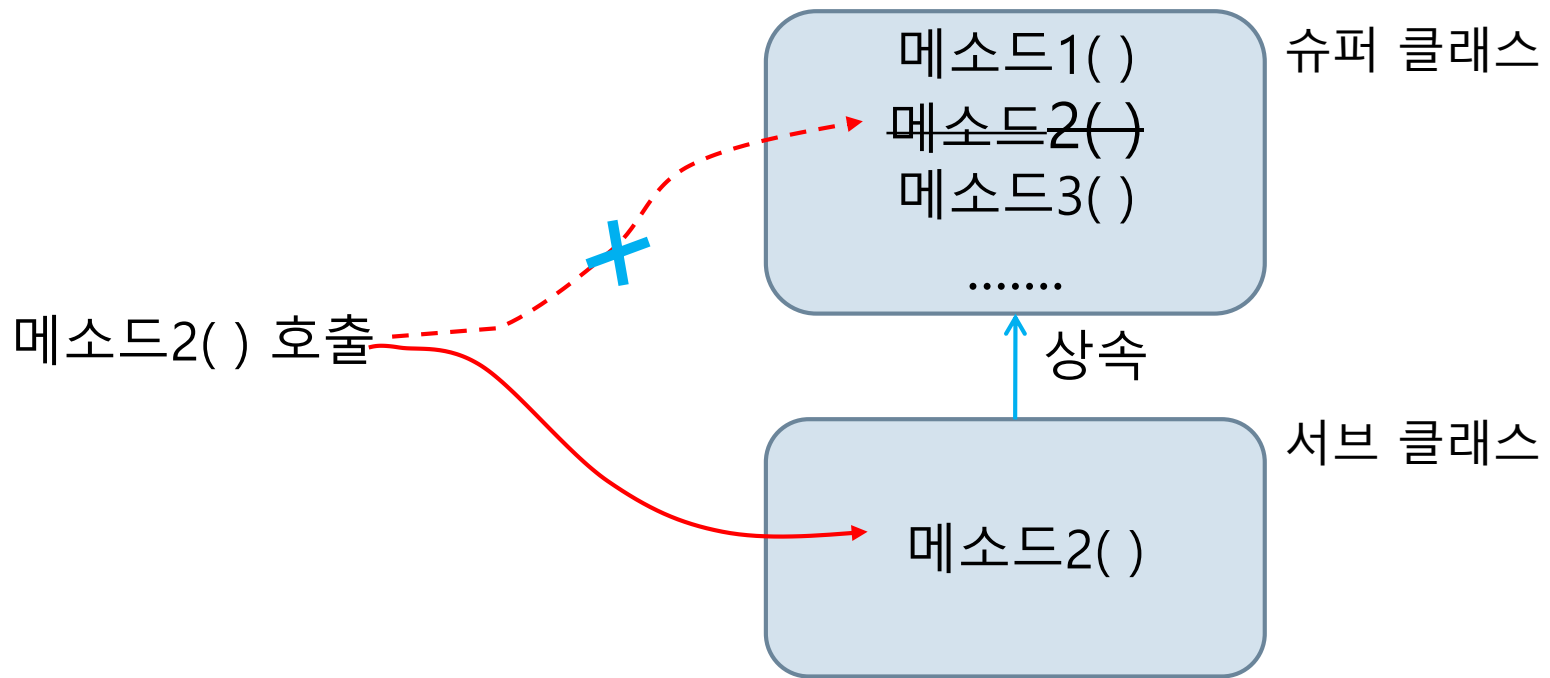
48

- 메소드 오버라이딩(Method Overriding)
  - ▣ 슈퍼 클래스의 메소드를 서브 클래스에서 재정의하는 것
    - 슈퍼 클래스의 메소드 이름, 메소드 인자 타입 및 개수, 리턴 타입 등 모든 것을 동일하게 정의한다.  
( 이 중에서 하나라도 다르면 메소드 오버라이딩이 실패한 것임 )
  - ▣ 슈퍼 클래스의 "메소드 무시하기"로 번역되기도 함
  - ▣ 동적 바인딩 발생
    - 오버라이딩된 메소드가 무조건 실행되도록 동적 바인딩이 된다.



# 메소드 오버라이딩: 슈퍼 클래스의 메소드를 무시하고 서브 클래스에서 새로 작성된 메소드가 실행됨

49



# 메소드 오버라이딩 사례

50

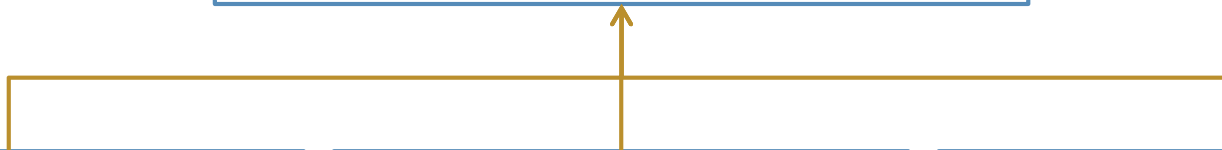
```
class DObject {  
    public DObject next;  
  
    public DObject( ) { next = null; }  
    public void draw( ) {  
        System.out.println("DObject draw");  
    }  
}
```

Line, Rect, Circle 클래스는  
모두 DObject를 상속받음.

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

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

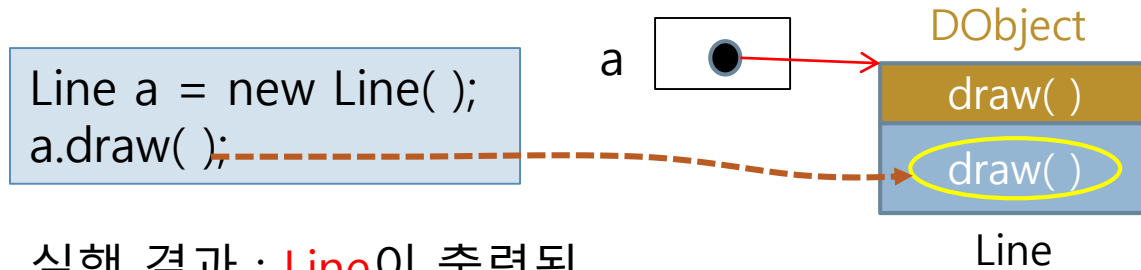
```
class Circle extends DObject {  
    public void draw( ) {  
        System.out.println("Circle");  
    }  
}
```



# 서브 클래스 객체와 오버라이딩된 메소드 호출

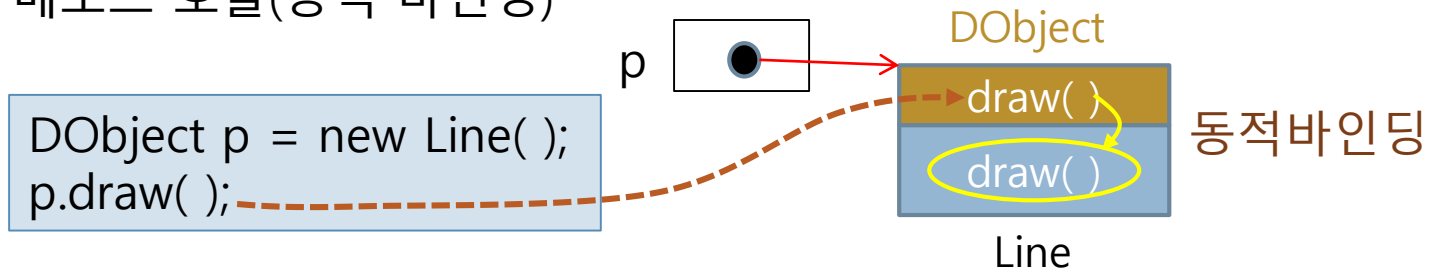
51

(1) 서브 클래스 레퍼런스로 오버라이딩된 메소드 호출



실행 결과 : **Line**이 출력됨

(2) 업캐스팅에 의해 슈퍼클래스 레퍼런스로 오버라이딩된 메소드 호출(동적 바인딩)



실행 결과 : **Line**이 출력됨

# 메소드 오버라이딩 만들기

52

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

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

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

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

# 메소드 오버라이딩 만들기

53

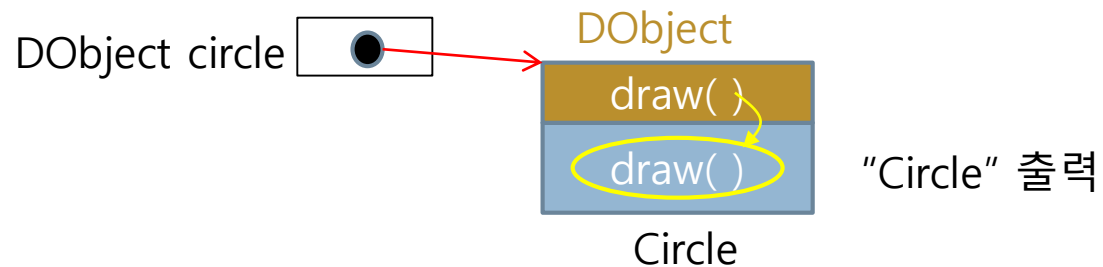
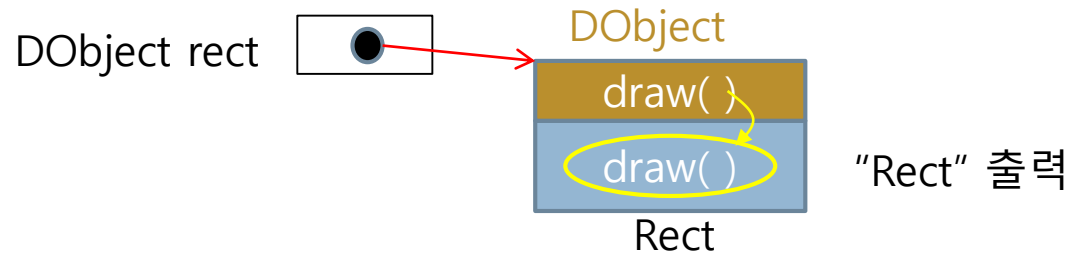
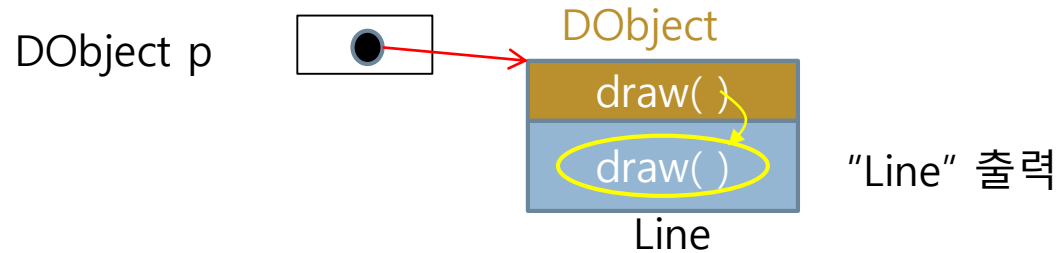
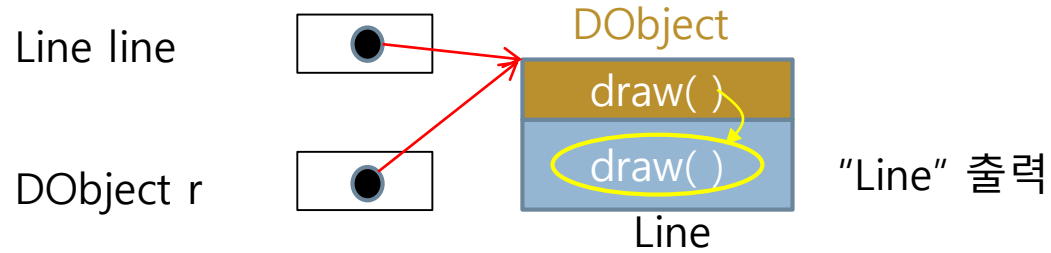
```
public class MethodOverridingDemo {  
    public static void main(String[] args) {  
        DObject obj = new DObject();  
        Line line = new Line();  
        DObject p = new Line();  
        DObject r = line;  
  
        obj.draw(); // DObject.draw()  
        line.draw(); // Line.draw()  
        p.draw(); // Line.draw()  
        r.draw(); // Line.draw()  
  
        DObject rect = new Rect();  
        DObject circle = new Circle();  
        rect.draw(); // Rect.draw()  
        circle.draw(); // Circle.draw()  
    }  
}
```

## 실행결과

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

## 예제 실행 과정

실행 시간에 객체 속에  
오버라이딩한 메소드가  
있으면 동적 바인딩되어  
그 메소드가 실행됨



# 메소드 오버라이딩 조건

55

1. 반드시 슈퍼 클래스 메소드와 동일한 이름, 동일한 호출 인자, 반환 타입을 가져야 한다.
2. 오버라이딩된 메소드의 접근 지정자는 슈퍼 클래스의 메소드 접근 지정자 보다 좁아질 수 없다.  
  
public > protected > private 순으로 지정 범위가 좁아진다.
3. 반환 타입만 달라도 오류
4. static, private 또는 final 메소드는 오버라이딩 될 수 없다.

# 메소드 오버라이딩 조건

56

```
class Person {  
    String name;  
    String phone;  
    static int ID;  
    public void setName(String s) {  
        name = s;  
    }  
    public String getPhone() {  
        return phone;  
    }  
    public static int getID() {  
        return ID;  
    }  
}
```

---

```
class Professor extends Person {  
    protected void setName(String s) { } // 2번 조건 위배  
    public String getPhone() { } // 1번 조건 성공  
        return phone;  
    }  
    public void getPhone(){ } // 3번 조건 위배  
    public int getID() { } // 4번 조건 위배  
}
```



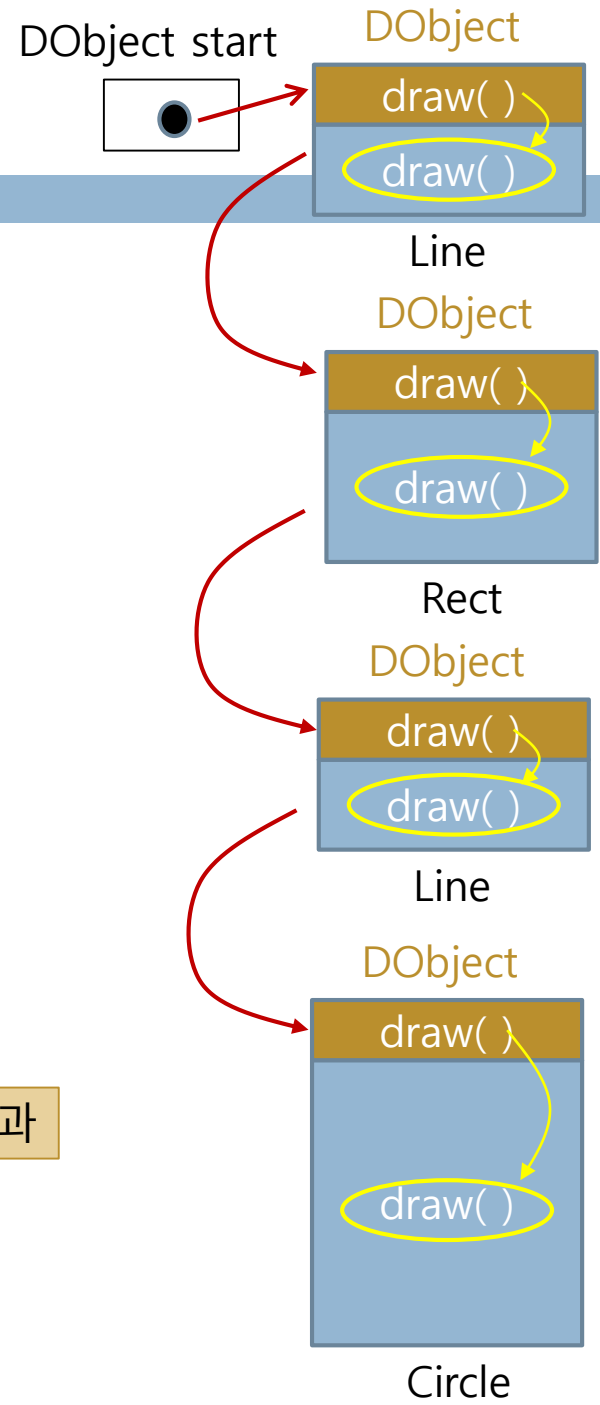
# 오버라이딩 활용

57

```
public class MethodOverridingDemo {  
    public static void main(String[] args) {  
        DObject start, n, obj;  
        // linked list 로 도형 생성하여 연결하기  
        start = new Line(); // Line 객체 연결  
        n = start;  
        obj = new Rect();  
        n.next = obj; // Rect 객체 연결  
        n = obj;  
        obj = new Line();  
        n.next = obj; // Line 객체 연결  
        n = obj;  
        obj = new Circle();  
        n.next = obj; // Circle 객체 연결  
        while (start != null){ // 모든 도형 출력  
            start.draw();  
            start = start.next;  
        }  
    }  
}
```

실행결과

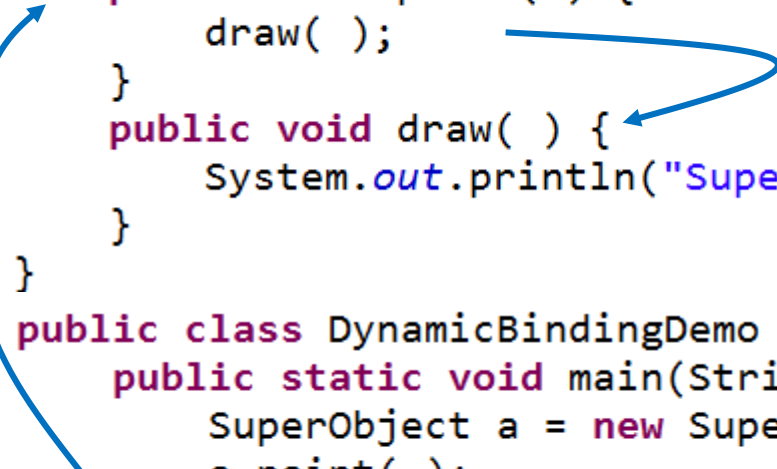
Line  
Rect  
Line  
Circle



# 동적 바인딩 (1/2)

58

```
public class SuperObject {  
    protected String name;  
    public void paint( ) {  
        draw( );  
    }  
    public void draw( ) {  
        System.out.println("Super Object");  
    }  
}  
  
public class DynamicBindingDemo {  
    public static void main(String[] args) {  
        SuperObject a = new SuperObject( );  
        a.paint( );  
    }  
}
```



실행결과

Super Object



# 동적 바인딩 (2/2)

59

실행결과

Sub Object

b



paint( )  
draw( )

SuperObject 부분

draw( )

SubObject 부분

동적 바인딩

```
public class SuperObject {  
    protected String name;  
    public void paint( ) {  
        draw( );  
    }  
    public void draw( ) {  
        System.out.println("Super Object");  
    }  
}  
public class SubObject extends SuperObject {  
    public void draw( ) {  
        System.out.println("Sub Object");  
    }  
}  
public class DynamicBindingDemo {  
    public static void main(String[] args) {  
        SuperObject b = new SubObject();  
        b.paint( );  
    }  
}
```

# super 키워드

60

- super는 서브클래스에서 슈퍼 클래스의 멤버를 접근할 때 사용되는 슈퍼클래스 타입의 레퍼런스
- 상속 관계에 있는 서브 클래스에서만 사용됨
- 오버라이딩된 슈퍼 클래스의 메소드 호출 시 사용

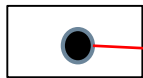
# super 키워드

61

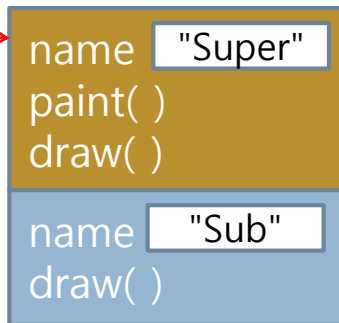
실행결과

Super  
Sub

b



SuperObject 부분



SubObject 부분

```
public class SuperObject {  
    protected String name;  
    public void paint( ) {  
        draw( );  
    }  
    public void draw( ) {  
        System.out.println(name);  
    }  
}
```

```
public class SubObject extends SuperObject {  
    protected String name;  
    public void draw( ) {  
        name = "Sub";  
        super.name = "Super";  
        super.draw();  
        System.out.println(name);  
    }  
}
```

```
public class DynamicBindingDemo {  
    public static void main(String[] args) {  
        SuperObject b = new SubObject();  
        b.paint( );  
    }  
}
```

# 메소드 오버라이딩 (1/2)

62

Person을 상속받는 Professor라는 새로운 클래스를 만들고 Professor 클래스에서 getPhone() 메소드를 재정의하시오. 그리고 이 메소드에서 슈퍼 클래스의 메소드를 호출하도록 작성하시오.

```
class Person {  
    String phone;  
    public void setPhone(String phone) {  
        this.phone = phone;  
    }  
    public String getPhone() {  
        return phone;  
    }  
}  
public class Professor extends Person {  
    public String getPhone(){  
        return "Professor : " + super.getPhone();  
    }  
}
```

super.getPhone( )은 아래  
p.getPhone( )과 달리 동적  
바인딩이 일어나지 않는다.

# 메소드 오버라이딩 (2/2)

63

```
public class Overriding {  
    public static void main(String[] args) {  
        Professor a = new Professor();  
  
        a.setPhone("011-123-1234");  
        System.out.println(a.getPhone());  
  
        Person p = a;  
        System.out.println(p.getPhone());  
    }  
}
```

## 실행결과

Professor : 011-123-1234  
Professor : 011-123-1234

동적 바인딩에 의해 Professor의  
getPhone( ) 호출.

# 오버라이딩 vs. 오버로딩

64

비교 요소	메소드 오버로딩	메소드 오버라이딩
정의	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 재작성
관계	동일한 클래스 내 혹은 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 정의하여 사용의 편리성 향상	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함
조건	메소드 이름은 반드시 동일함. 메소드의 인자의 개수나 인자의 타입이 달라야 성립	메소드의 이름, 인자의 타입, 인자의 개수, 인자의 리턴 타입 등이 모두 동일하여야 성립
바인딩	<b>정적 바인딩.</b> 컴파일 시에 중복된 메소드 중 호출되는 메소드 결정	<b>동적 바인딩.</b> 실행 시간에 오버라이딩된 메소드 찾아 호출



# 슈퍼 클래스 레퍼런스 ← 서브 클래스 레퍼런스 (1/5)

65

```
public class TwoDShape {  
    double width;  
    double height;  
  
    // Default constructor.  
    public TwoDShape() {  
        width = height = 0.0;  
    }  
  
    // Constructor for TwoDShape.  
    public TwoDShape(double w, double h) {  
        width = w;  
        height = h;  
    }  
  
    // Construct object with equal width and height.  
    public TwoDShape(double x) {  
        width = height = x;  
    }  
}
```

# 슈퍼 클래스 레퍼런스 ← 서브 클래스 레퍼런스 (2/5)

66



```
// Construct object from an object.  
public TwoDShape(TwoDShape ob) {  
    width = ob.width;  
    height = ob.height;  
}
```

```
public void showDim() {  
    System.out.println("Width and height are " + width + " and " + height);  
}  
}
```

# 슈퍼 클래스 레퍼런스 ← 서브 클래스 레퍼런스 (3/5)

67

```
class Triangle extends TwoDShape {
    String style;

    // A default constructor.
    public Triangle() {
        style = "null";
    }

    // Constructor for Triangle.
    public Triangle(String s, double w, double h) {
        super(w, h);
        style = s;
    }

    // Construct an isosceles triangle.
    public Triangle(double x) {
        super(x);
        style = "isosceles";
    }
}
```

# 슈퍼 클래스 레퍼런스 ← 서브 클래스 레퍼런스 (4/5)

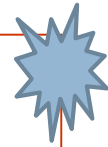
68

```
// Construct an object from an object.
```

```
public Triangle(Triangle ob) {  
    super(ob);  
    style = ob.style;  
}
```

```
public double area() {  
    return width * height / 2;  
}
```

```
public void showStyle() {  
    System.out.println("Triangle is " + style);  
}  
}
```



# 슈퍼 클래스 레퍼런스 ← 서브 클래스 레퍼런스 (5/5)

69

```
Info for t1:  
Triangle is right  
Width and height are 8.0 and 12.0  
Area is 48.0
```

```
Info for t2:  
Triangle is right  
Width and height are 8.0 and 12.0  
Area is 48.0
```

```
public class MM {  
    public static void main(String[] args) {  
        Triangle t1 = new Triangle("right", 8.0, 12.0);  
  
        // make a copy of t1  
        Triangle t2 = new Triangle(t1);  
        System.out.println("Info for t1: ");  
        t1.showStyle();  
        t1.showDim();  
        System.out.println("Area is " + t1.area());  
  
        System.out.println();  
  
        System.out.println("Info for t2: ");  
        t2.showStyle();  
        t2.showDim();  
        System.out.println("Area is " + t2.area());  
    }  
}
```

# 추상 메소드와 추상 클래스

70

```
public abstract class DObject {  
    public DObject next;  
    public DObject(){  
        next = null;  
    }  
    public abstract void draw();  
}
```

- 추상 메소드 (abstract method)
  - ▣ 선언되어 있으나 구현되어 있지 않은 메소드
  - ▣ 정의 : 접근 지정자 **abstract** 반환형 메소드이름( );
    - 추상 메소드는 서브 클래스에서 오버라이딩하여 구현한다.
- 추상 클래스 (abstract class)
  - ▣ 추상 메소드를 하나라도 가지고 있으면 추상 클래스이다.
    - 클래스 앞에 반드시 abstract라고 선언해야 한다.
  - ▣ 추상 메소드를 가지고 있지는 않지만 클래스 앞에 abstract로 선언한 경우

# 추상 클래스의 특성

71

- 추상 클래스의 객체는 생성할 수 없다.
- 추상 클래스의 필요성
  - ▣ 계층적 상속 관계를 갖는 클래스 구조를 만들 때
  - ▣ 설계와 구현 분리
    - 슈퍼 클래스에서는 개념적 특징을 정의하고, 서브 클래스에서 구체적인 행위를 구현한다.
- 추상 클래스의 상속
  - ▣ 추상 클래스를 상속받은 서브 클래스가 추상 메소드를 또 구현하지 않으면 서브 클래스도 추상 클래스가 된다.
    - abstract로 정의하여야 한다.
  - ▣ 서브 클래스에서 추상 메소드를 구현하면 서브 클래스는 추상 클래스가 되지 않는다.


## 2 가지 종류의 추상 클래스 (1/2)

72

```
abstract class Line { // 추상메소드를 포함하므로 반드시 추상 클래스
    int x;
    int y;
    public abstract void setX(int position);
    public abstract void setY(int position);
    public abstract int getLength();
}
```

```
public class AbstractError {
    public static void main (String args[]) {
        Line l = new Line(); // 컴파일 오류 발생
        l.setX(0);
        l.setY(10);
    }
}
```

추상클래스는  
인스턴스를  
생성할 수 없음



```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Cannot instantiate the type Line
```

```
at chap5.AbstractError.main(AbstractError.java:11)
```



## 2 가지 종류의 추상 클래스 (2/2)

73

```
abstract class Line { // 개발자가 임의로 추상 클래스 선언
    int x;
    int y;
    public void setX(int position) {
        x = position;
    }
    public void setY(int position) {
        y = position;
    }
    public int getLength() {
        return 0;
    }
}

public class AbstractError {
    public static void main (String args[]) {
        Line l = new Line(); // 컴파일 오류 발생
        l.setX(0);
        l.setY(10);
    }
}
```

추상클래스는  
인스턴스를  
생성할 수 없음

동일한 컴파일 오류 발생

# 추상 클래스의 활용 예

74

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

추상 클래스로  
수정

```
abstract class DObject {  
    public DObject next;  
  
    public DObject() { next = null;}  
    abstract public void draw();  
}
```

```
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");  
    }  
}
```

# 추상 클래스의 구현

75

다음의 추상 클래스 Calculator를 상속받는 GoodCalc 클래스를  
임의로 작성하시오.

```
abstract class Calculator {  
    public abstract int add(int a, int b);  
    public abstract int subtract(int a, int b);  
    public abstract double average(int[] a);  
}
```

# 샘플 답안

76

```
class GoodCalc extends Calculator {  
    public int add(int a, int b) {  
        return a+b;  
    }  
    public int subtract(int a, int b) {  
        return a-b;  
    }  
    public double average(int[] a) {  
        double sum = 0;  
        for (int i = 0; i < a.length; i++)  
            sum += a[i];  
        return sum/a.length;  
    }  
}
```

5  
-1  
3.0

```
public class MM {  
    public static void main(String[] args) {  
        Calculator c = new GoodCalc();  
        System.out.println(c.add(2,3));  
        System.out.println(c.subtract(2,3));  
        System.out.println(c.average(new int [] {2,3,4}));  
    }  
}
```

# 자바의 인터페이스 (interface)

77

- 인터페이스
  - ▣ 모든 메소드가 추상 메소드인 "클래스"
  - ▣ 인터페이스는 상수와 메소드만 갖는다.
  
- 인터페이스 정의
  - ▣ interface 키워드로 정의된 클래스
    - ex) `public interface SerialDriver { ... }`
  
- 인터페이스의 특징
  - ▣ 모든 메소드 : `public abstract` (생략 가능)
  - ▣ 모든 상수: `public static final` (생략 가능)
  - ▣ 객체를 생성할 수 없음
  - ▣ 레퍼런스 변수 타입으로 사용 가능

# 자바 인터페이스 사례

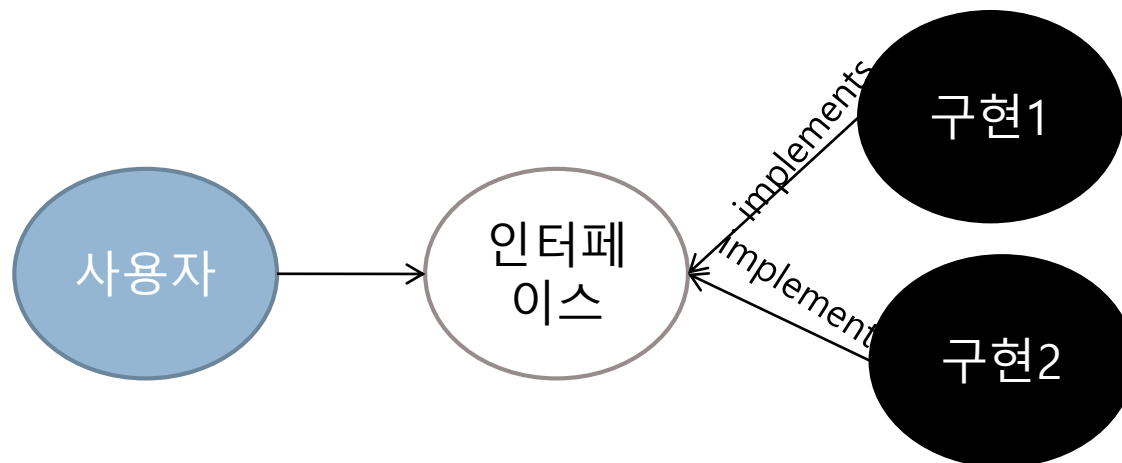
78

```
public interface Clock {  
    public static final int ONEDAY = 24; // 상수 필드 선언  
    abstract public int getMinute();  
    abstract public int getHour();  
    abstract void setMinute(int i);  
    abstract void setHour(int i);  
}  
  
public interface Car {  
    int MAXIMUM_SPEED = 260; // 상수 필드 선언  
    int moveHandle(int degree); // abstract 생략 가능  
    int changeGear(int gear); // public 생략 가능  
}
```

# 인터페이스의 필요성

79

- 인터페이스를 이용하여 다중 상속 구현
  - ▣ 클래스는 다중 상속 불가
- 인터페이스는 명세서와 같음
  - ▣ 구현은 블랙 박스와 같아 인터페이스의 사용자는 구현에 대해 알 필요가 없음
- 인터페이스만 정의하고 구현을 분리하여, 작업자가 다양한 구현을 할 수 있음



# 인터페이스 상속

80

- 인터페이스 간에도 상속이 가능하다.
  - ▣ 인터페이스를 상속하여 확장된 인터페이스를 작성할 수 있다.
- 다중 상속을 허용한다.

```
public interface MobilePhone {  
    public boolean sendCall();  
    public boolean receiveCall();  
    public boolean sendSMS();  
    public boolean receiveSMS();  
}
```

```
public interface MP3 {  
    public void play();  
    public void stop();  
}
```

```
public interface MusicPhone extends MobilePhone, MP3 {  
    public void playMP3RingTone();  
}
```



# 인터페이스 구현

81

- 인터페이스 구현
  - ▣ implements 키워드를 사용한다.
  - ▣ 여러 개의 인터페이스를 동시에 구현할 수 있다.
  - ▣ 상속과 구현이 동시에 가능하다.

```
interface USBMouseInterface {  
    void mouseMove();  
    void mouseClicked();  
}  
  
public class MouseDriver implements USBMouseInterface {  
    // 인터페이스 구현. 클래스 작성  
    void mouseMove() { .... }  
    void mouseClicked() { ... }  
  
    // 추가적으로 다른 메소드를 작성할 수 있다.  
    int getStatus() { ... }  
    int getButton() { ... }  
}
```

# 인터페이스의 다중 구현

82

```
interface USBMouseInterface {  
    void mouseMove();  
    void mouseClicked();  
}
```

```
interface RollMouseInterface {  
    void roll();  
}
```

```
public class MouseDriver implements RollMouseInterface , USBMouseInterface {  
    void mouseMove() { .... }  
    void mouseClicked() { ... }  
    void roll() { ... }
```

// 추가적으로 다른 메소드를 작성할 수 있다.

```
    int getStatus() { ... }  
    int getButton() { ... }  
}
```

# 수열 관리 인터페이스 (1/6)

83

```
public interface Series {  
    int getNext(); // return next number in series  
    void reset(); // restart  
    void setStart(int x); // set starting value  
}
```

- 2, 4, 6, 8, 10, 12, 14, . . . . .
- 100, 102, 104, 106, 108, . . . . .
- 3, 6, 9, 12, 15, 18, . . . . .
- 60, 63, 66, 69, 72, . . . . .

# 수열 관리 인터페이스 (2/6)

84

```
public class ByTwos implements Series{
    int start;
    int val;

    public ByTwos() {
        start = 0;
        val = 0;
    }
    public int getNext() {
        val += 2;
        return val;
    }
    public void reset() {
        start = 0;
        val = 0;
    }
    public void setStart(int x) {
        start = x;
        val = x;
    }
}
```

# 수열 관리 인터페이스 (3/6)

85

```
public class ByThrees implements Series {
    int start;
    int val;

    public ByThrees() {
        start = 0;
        val = 0;
    }
    public int getNext() {
        val += 3;
        return val;
    }
    public void reset() {
        start = 0;
        val = 0;
    }
    public void setStart(int x) {
        start = x;
        val = x;
    }
}
```

# 수열 관리 인터페이스 (4/6)

86

```
public class SeriesDemo {  
    public static void main(String[] args) {  
        ByTwos ob = new ByTwos();  
        for(int i=0; i < 5; i++)  
            System.out.println("Next value is " + ob.getNext());  
        System.out.println("\nResetting");  
        ob.reset();  
        for(int i=0; i < 5; i++)  
            System.out.println("Next value is " + ob.getNext());  
        System.out.println("\nStarting at 100");  
        ob.setStart(100);  
        for(int i=0; i < 5; i++)  
            System.out.println("Next value is " + ob.getNext());  
    }  
}
```

Next value is 2  
Next value is 4  
Next value is 6  
Next value is 8  
Next value is 10



Resetting  
Next value is 2  
Next value is 4  
Next value is 6  
Next value is 8  
Next value is 10

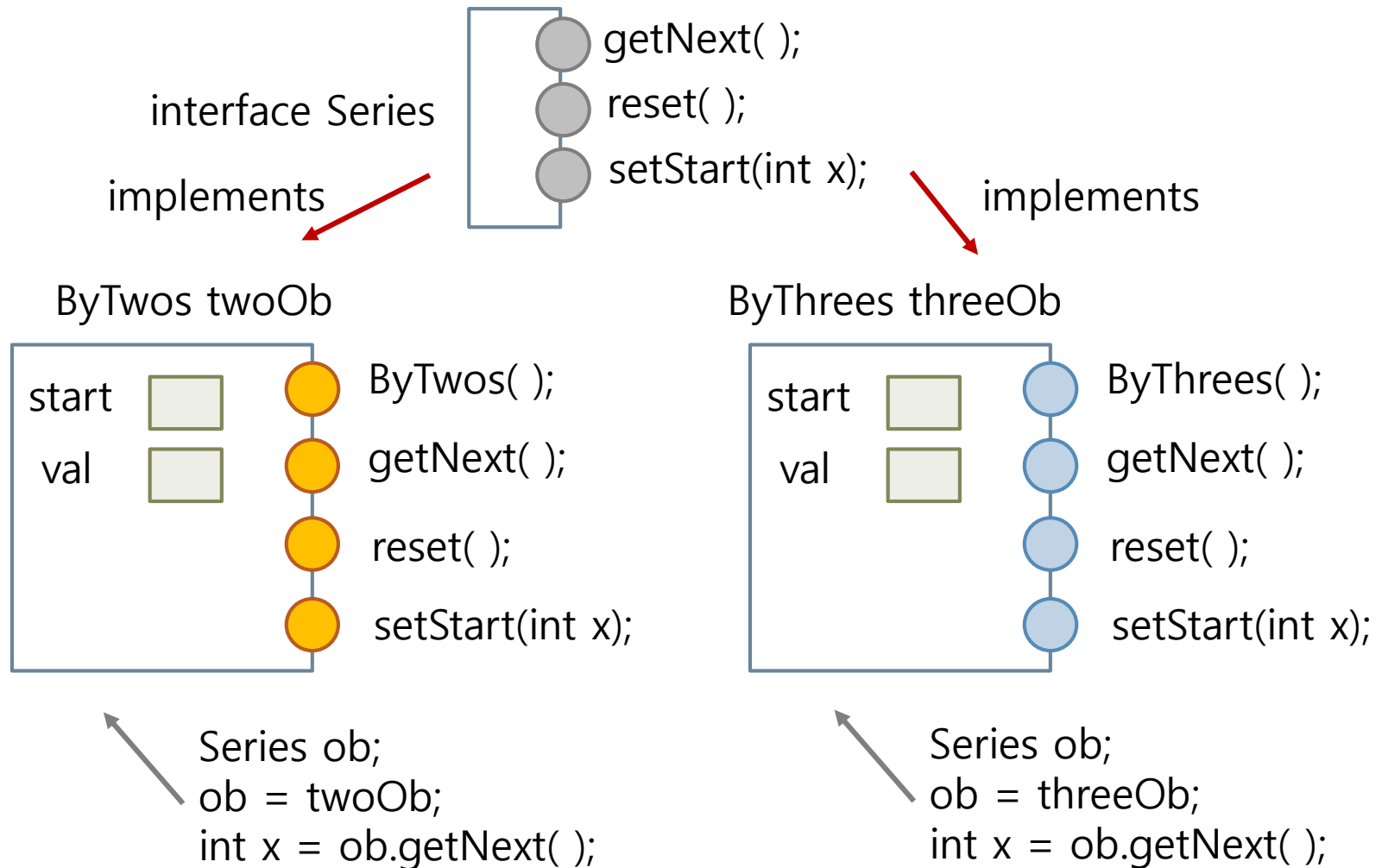


Starting at 100  
Next value is 102  
Next value is 104  
Next value is 106  
Next value is 108  
Next value is 110

# 수열 관리 인터페이스 (5/6)

## 인터페이스 레퍼런스

87



# 수열 관리 인터페이스 (6/6)

88

```
public class SeriesDemo {  
    public static void main(String[] args) {  
        ByTwos twoOb = new ByTwos();  
        ByThrees threeOb = new ByThrees();  
        Series ob;  
  
        for(int i=0; i < 5; i++) {  
            ob = twoOb;  
            System.out.println("Next ByTwos value is " + ob.getNext());  
            ob = threeOb;  
            System.out.println("Next ByThrees value is " + ob.getNext());  
        }  
    }  
}
```

```
Next ByTwos value is 2  
Next ByThrees value is 3  
Next ByTwos value is 4  
Next ByThrees value is 6  
Next ByTwos value is 6  
Next ByThrees value is 9  
Next ByTwos value is 8  
Next ByThrees value is 12  
Next ByTwos value is 10  
Next ByThrees value is 15
```



# 추상 클래스와 인터페이스 비교

89

- 추상 클래스
  - ▣ 일반 메소드 포함 가능
  - ▣ 상수, 변수 필드 포함 가능
  - ▣ 모든 서브 클래스에 공통된 메소드가 있는 경우, 추상 클래스가 적합
- 인터페이스
  - ▣ 모든 메소드가 추상 메소드
  - ▣ 상수 필드만 포함 가능
  - ▣ 다중 상속 지원