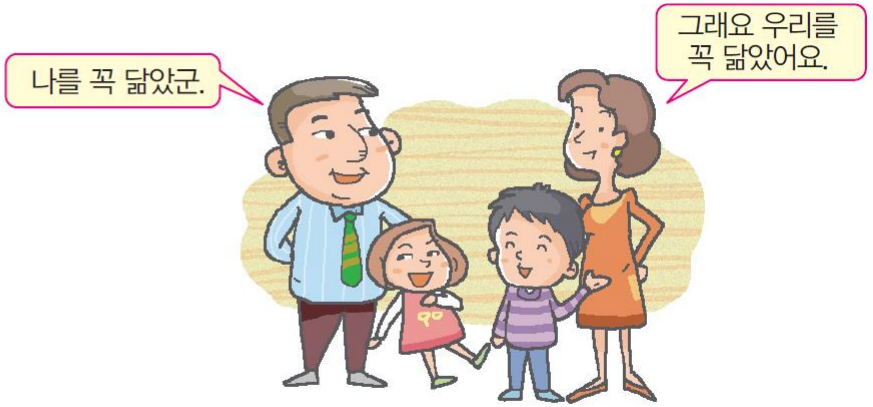
# 상속 (inheritance)



- 객체 지향의 상속
  - 부모클래스에 만들어진 필드, 메소드를 자식클래스가 물려받음
  - 부모의 생물학적 특성을 물려받는 유전과 유사


유산 상속




유전적 상속 : 객체 지향 상속

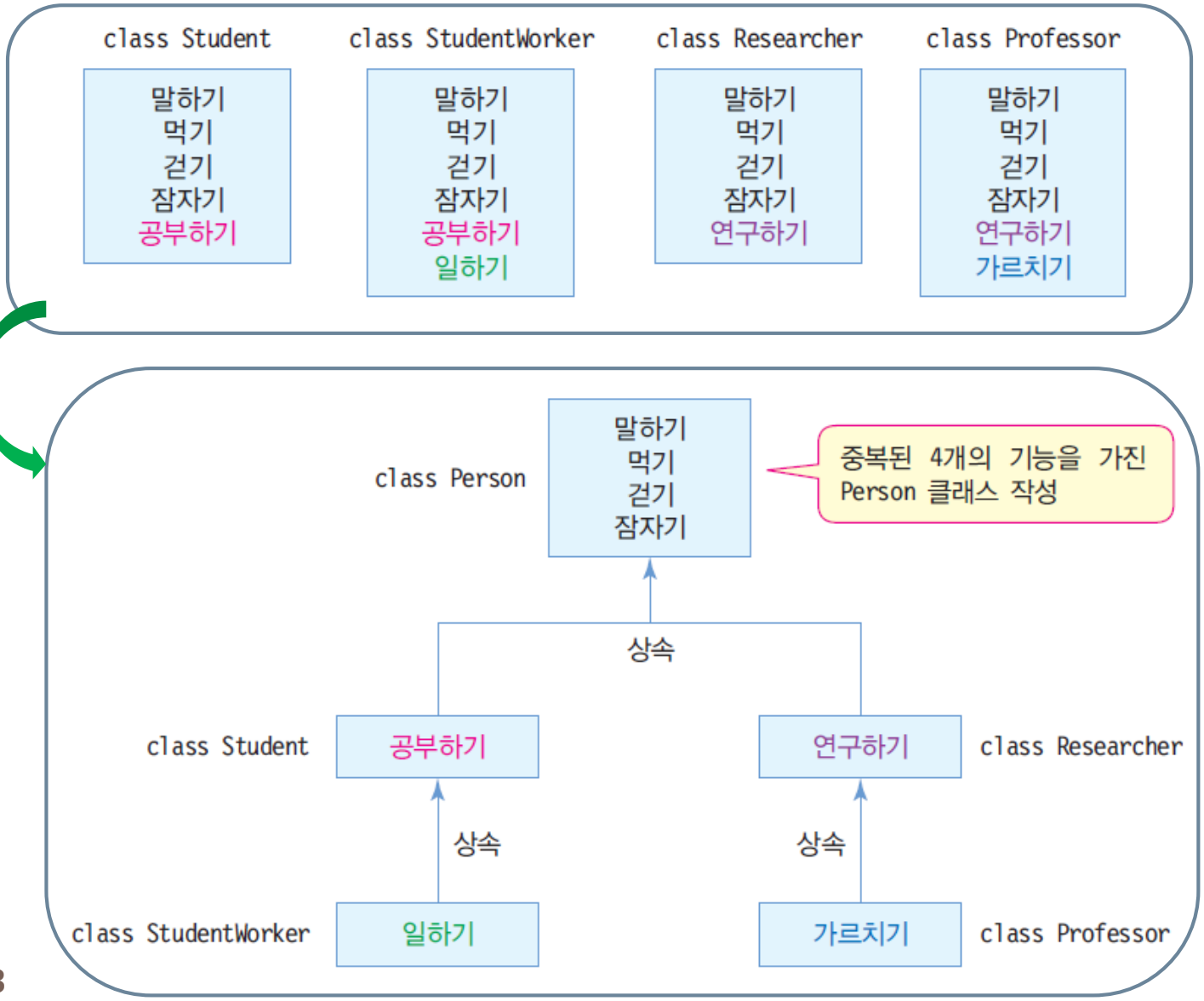
- 상속을 통해 간결한 자식 클래스 작성
  - 동일한 특성을 재정의할 필요가 없어 자식 클래스가 간결해짐

# 상속 (inheritance)



- 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) 추가
    - 슈퍼 클래스의 특성(메소드)을 수정 //오버라이딩
- 슈퍼 클래스에서 하위 클래스로 갈수록 구체적
  - 예) 폰 -> 모바일폰 -> 뮤직폰

# 상속의 필요성




상속이 없는 경우
중복된 멤버를 가진
4 개의 클래스

상속을 이용한
경우 중복이 제거되고
간결해진 클래스 구조

# 클래스 상속과 객체



- 상속 선언

```
public class Person {
        …
}
public class Student extends Person { // Person을 상속받는 클래스 Student 선언
        …
}
public class StudentWorker extends Student { // Student를 상속받는 StudentWorker 선언
        …
}
```

- 자바 상속의 특징
  - 클래스 다중 상속 지원하지 않음
    - 다수 개의 클래스를 상속받지 못함
  - 상속 횟수 무제한
  - 상속의 최상위 조상 클래스는 java.lang.Object 클래스
    - 모든 클래스는 자동으로 java.lang.Object를 상속받음

# 예제: 클래스 상속 만들어 보기



(x,y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 컬러 점을 표현하는 ColorPoint 클래스를 만들어보자.

```
class Point {
   int x, y; // 한 점을 구성하는 x, y 좌표
   void set(int x, int y) {
      this.x = x; this.y = y;
   }
   void showPoint() { // 점의 좌표 출력
      System.out.println("(" + x + "," + y + ")");
   }
}
```

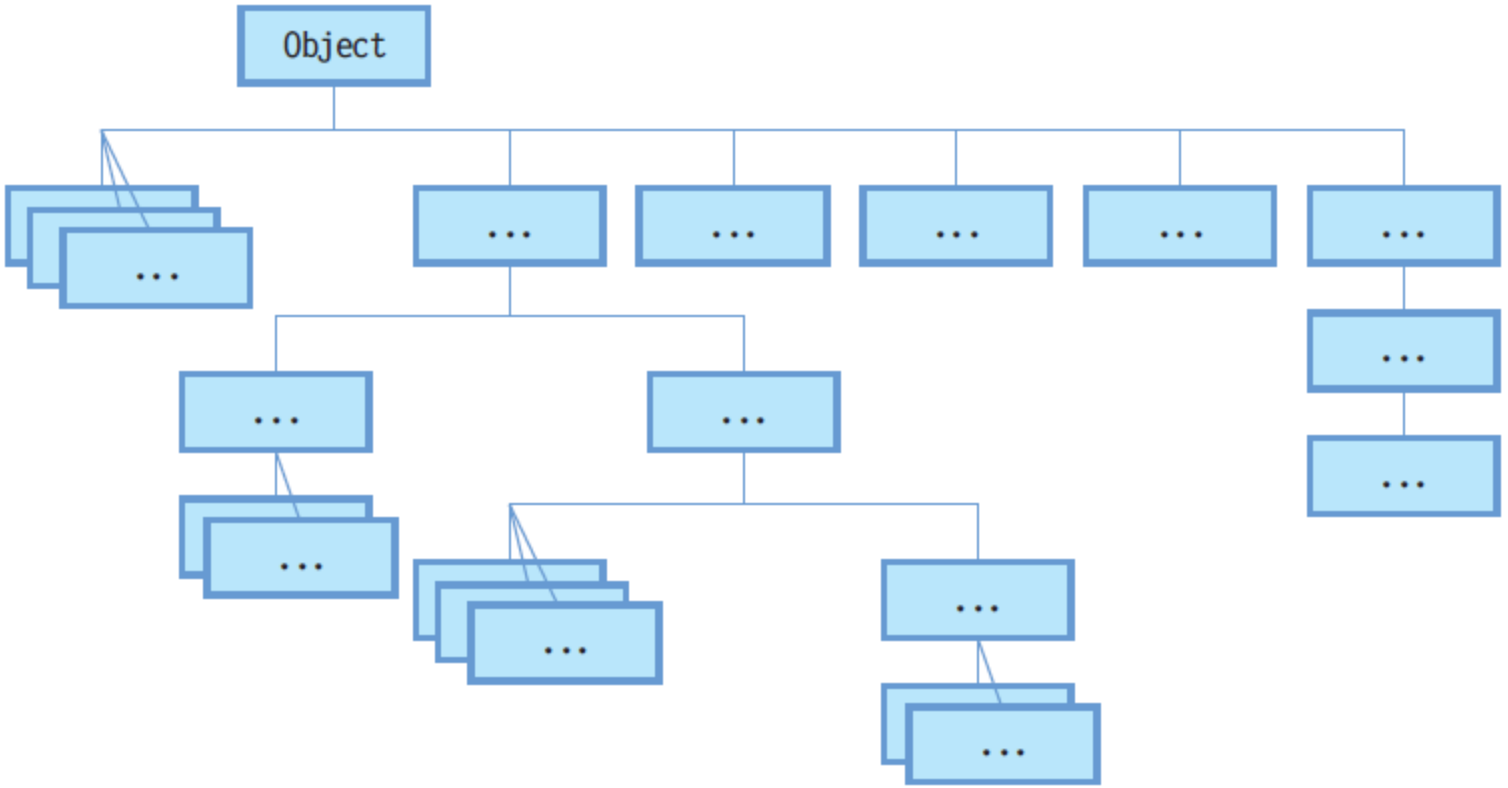
```
public class ColorPoint extends Point {
   // Point를 상속받은 ColorPoint 선언
   String color; // 점의 색
   void setColor(String color) {
      this.color = color;
   }
   void showColorPoint() { // 컬러 점의 좌표 출력
      System.out.print(color);
      showPoint(); // Point 클래스의 showPoint() 호출
   }
   public static void main(String [] args) {
      ColorPoint cp = new ColorPoint();
      cp.set(3,4); // Point 클래스의 set() 메소드 호출
      cp.setColor("red"); // 색 지정
      cp.showColorPoint(); // 컬러 점의 좌표 출력
   }
}
```

```
red(3,4)
```

# 자바의 클래스 계층 구조



자바에서는 모든 클래스는 반드시 java.lang.Object 클래스를 자동으로 상속받는다.



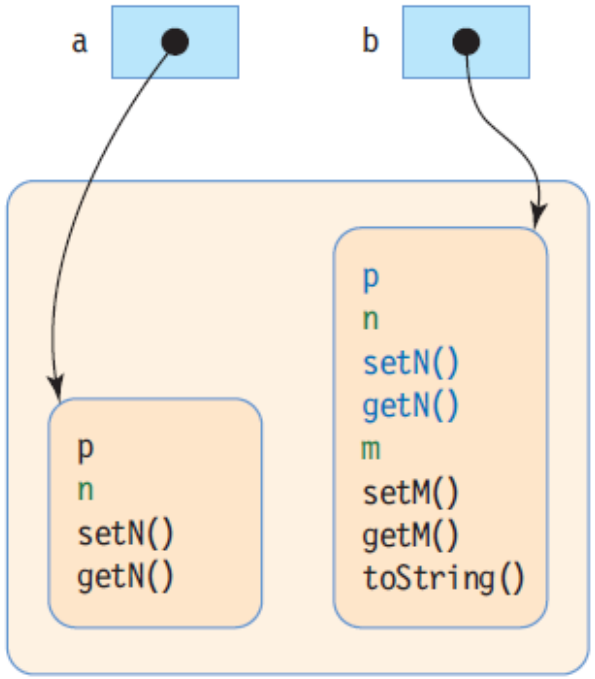
# 슈퍼 클래스와 서브 클래스의 객체 관계



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

```
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;
    }
}
```

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

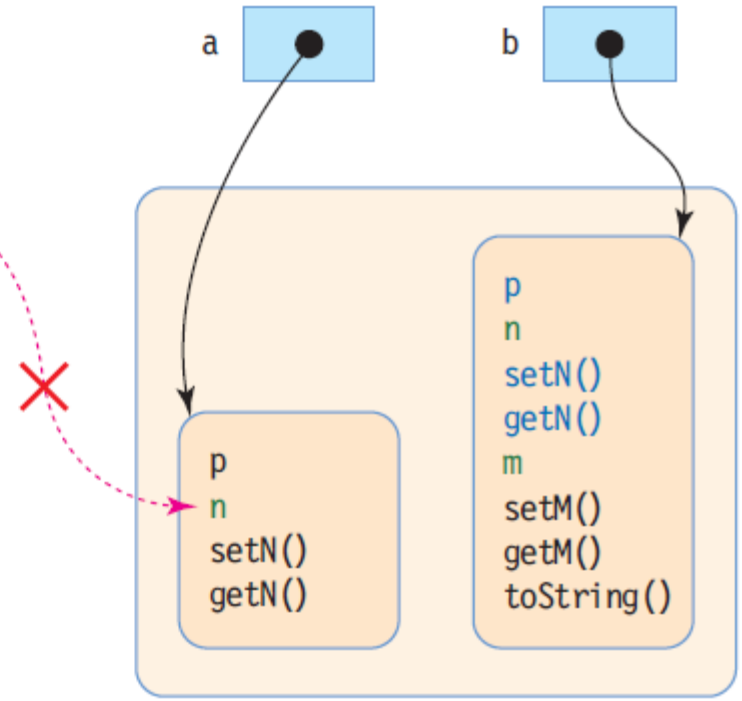



main() 실행 중 생성된 인스턴스

# 서브 클래스의 객체 멤버 접근



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

        a.p = 5;
        a.n = 5; // n은 private 멤버, 컴파일 오류 발생

        b.p = 5;
        b.n = 5; // n은 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());
        // 화면에 10 200이 출력됨
    }
}
```




10 20

# 서브 클래스의 객체와 멤버 사용



- 서브 클래스의 객체와 멤버 접근
  - 서브 클래스의 객체에는 슈퍼 클래스 멤버 포함
    - 슈퍼 클래스의 private 멤버는 상속되지만
      - 서브 클래스에서 직접 접근 불가
    - 슈퍼 클래스의 private 멤버는
      - 슈퍼 클래스의 public/protected 메소드를 통해 접근

# 상속과 접근 지정자



- 자바의 접근 지정자 4가지
  - public, protected, 디폴트, private
    - 상속 관계에서 주의할 접근 지정자는 private와 protected
- 슈퍼 클래스의 private 멤버
  - 슈퍼 클래스의 private 멤버는 다른 모든 클래스에 접근 불허
- 슈퍼 클래스의 protected 멤버
  - 같은 패키지 내의 모든 클래스 접근 허용
  - 동일 패키지 여부와 상관없이 서브 클래스에서 슈퍼 클래스의 protected 멤버 접근 가능

# 슈퍼 클래스 멤버의 접근 지정자



| 슈퍼 클래스 멤버에 접근하는 클래스 종류 | 슈퍼 클래스 멤버의 접근 지정자 | | | |
|---|---|---|---|---|
| | default | private | protected | public |
| 같은 패키지의 클래스 | ○ | × | ○ | ○ |
| 다른 패키지의 클래스 | × | × | × | ○ |
| 같은 패키지의 서브 클래스 | ○ | × | ○ | ○ |
| 다른 패키지의 서브 클래스 | × | × | ○ | ○ |

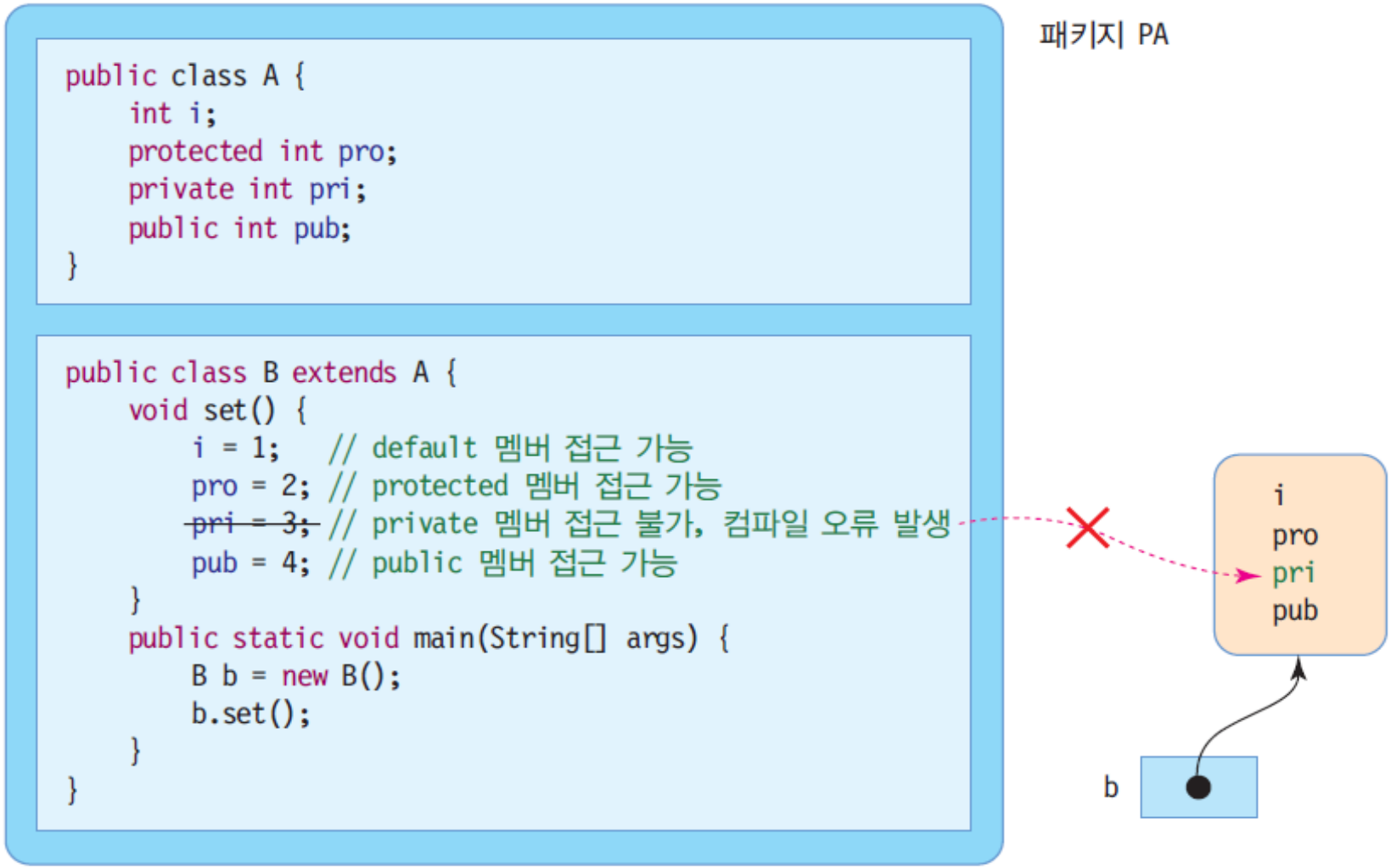
(○는 접근 가능함을, ×는 접근이 불가함을 뜻함)

# 슈퍼클래스와 서브클래스가 같은 패키지에 있는 경우



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

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



# 슈퍼클래스와 서브클래스가 서로 다른 패키지에 있는 경우

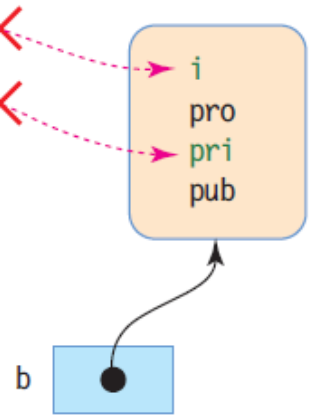


패키지 PA

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

패키지 PB

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



# 예제:상속 관계에 있는 클래스 간 멤버 접근



클래스 Person을 아래와 같은 멤버 필드를 갖도록 선언하고 클래스 Student는 클래스 Person을 상속받아 각 멤버 필드에 값을 저장하시오. 이 예제에서 Person 클래스의 private 필드인 weight는 Student 클래스에서는 접근이 불가능하여 슈퍼 클래스인 Person의 get, set 메소드를 통해서만 조작이 가능하다.

- int age;
- public String name;
- protected int height;
- private int weight;

```
public class Student extends Person {
  void set() {
    age = 30;
    name = "홍길동";
    height = 175;
    // 몸무게 설정
  }
  public static void main(String[] args) {
    Student s = new Student();
    s.set();
  }
}
```

# 서브 클래스/슈퍼 클래스의 생성자 호출과 실행



**질문 1** 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자가 모두 실행되는가? 아니면 서브 클래스의 생성자만 실행되는가?

**답** 둘 다 실행된다. 생성자는 인스턴스를 초기화할 목적으로 사용되므로 서브 클래스의 생성자는 서브 클래스 내의 멤버를 초기화하거나 필요한 초기화 작업을 수행할 필요가 있고, 슈퍼 클래스의 생성자는 슈퍼 클래스의 멤버를 초기화하거나 필요한 초기화 작업을 수행할 필요가 있기 때문이다.

**질문 2** 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자의 실행 순서는 어떻게 되는가?

**답** 슈퍼 클래스의 생성자가 먼저 실행된 후 서브 클래스의 생성자가 실행된다.

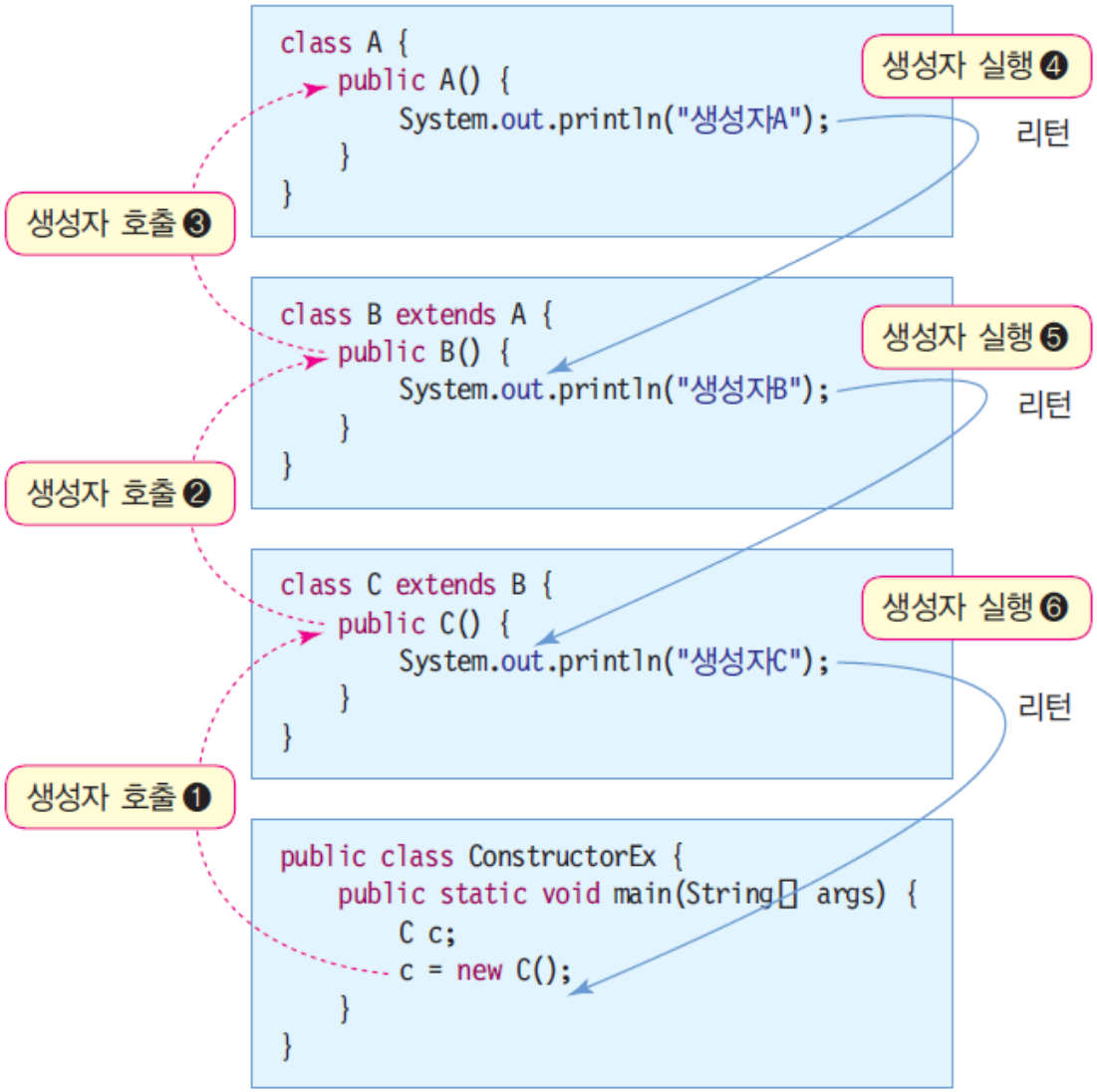
# 서브 클래스/슈퍼 클래스의 생성자 호출과 실행



- new에 의해 서브 클래스의 객체가 생성될 때
  - 슈퍼클래스 생성자와 서브 클래스 생성자 모두 실행됨
  - 호출 순서
    - 서브 클래스의 생성자가 먼저 호출, 서브 클래스의 생성자는 실행 전 슈퍼 클래스 생성자 호출
  - 실행 순서
    - 슈퍼 클래스의 생성자가 먼저 실행된 후 서브 클래스의 생성자 실행

# 슈퍼클래스와 서브 클래스의 생성자간의 호출 및 실행 관계






예상 실행 결과는 ?

생성자A
생성자B
생성자C

# 서브 클래스에서 슈퍼 클래스의 생성자 선택



- 상속 관계에서의 생성자
  - 슈퍼 클래스와 서브 클래스 각각 각각 여러 생성자 작성 가능
- 서브 클래스 생성자 작성 원칙
  - 서브 클래스 생성자에서 슈퍼 클래스 생성자 하나 선택
- 서브 클래스에서 슈퍼 클래스의 생성자를 선택하지 않는 경우
  - 컴파일러가 자동으로 슈퍼 클래스의 기본 생성자 선택
- 서브 클래스에서 슈퍼 클래스의 생성자를 선택하는 방법
  - super() 이용

# 슈퍼 클래스의 기본 생성자가 자동 선택



서브 클래스의 생성자가
슈퍼 클래스의 생성자를
선택하지 않은 경우

컴파일러는
서브 클래스의 기본
생성자에 대해
자동으로 슈퍼 클래스의
기본 생성자와 짝을 맺음

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

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

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

실행 결과

```
생성자A
생성자B
```

# 슈퍼 클래스에 기본 생성자가 없어 오류 난 경우



B()에 대한 짝, A()를 찾을 수 없음

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

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

오류

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

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

# 서브 클래스에 매개변수를 가진 생성자



서브 클래스의 생성자가 슈퍼 클래스의 생성자를 선택하지 않은 경우

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

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

```
public class ConstructorEx3 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

실행 결과

```
생성자A
매개변수생성자B
```

# super()를 이용하여 슈퍼 클래스 생성자 선택



- 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 함

# super()를 이용한 사례



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

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

super(); 라고 하면 A() 호출

```
public class ConstructorEx4 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

매개변수생성자A5
매개변수생성자B5

# 객체의 타입 변환

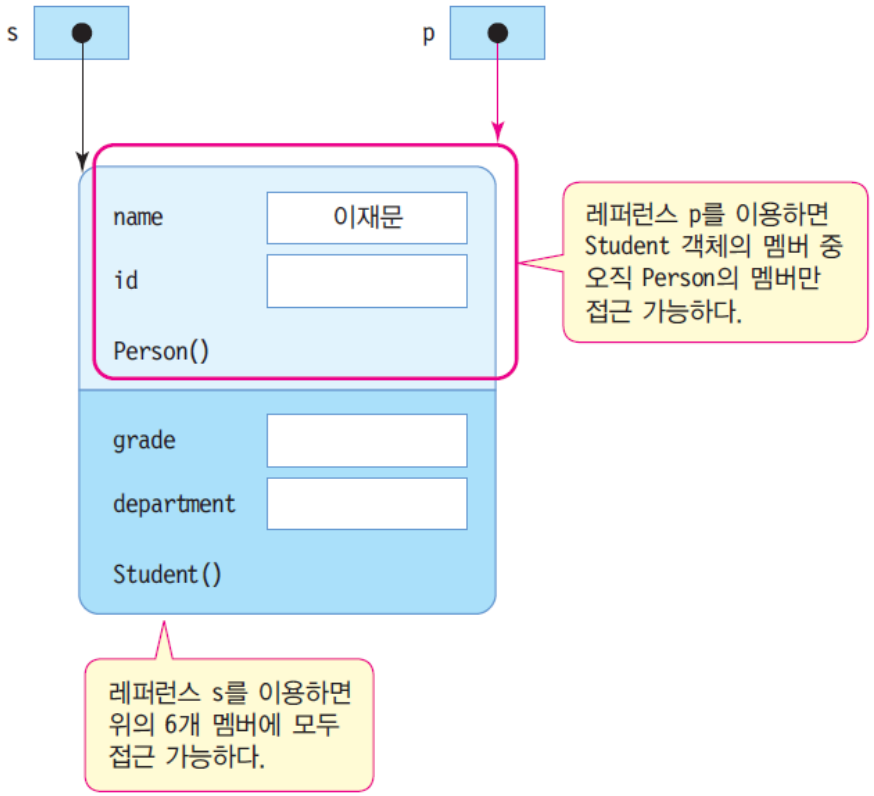


- 업캐스팅(upcasting)
  - 프로그램에서 이루어지는 자동 타입 변환
  - 서브 클래스의 레퍼런스 값을 슈퍼 클래스 레퍼런스에 대입
    - 슈퍼 클래스 레퍼런스가 서브 클래스 객체를 가리키게 되는 현상
    - 객체 내에 있는 모든 멤버를 접근할 수 없고 슈퍼 클래스의 멤버만 접근 가능

```
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("이재문");
        p = s; // 업캐스팅 발생

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

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




이재문

# 객체의 타입 변환



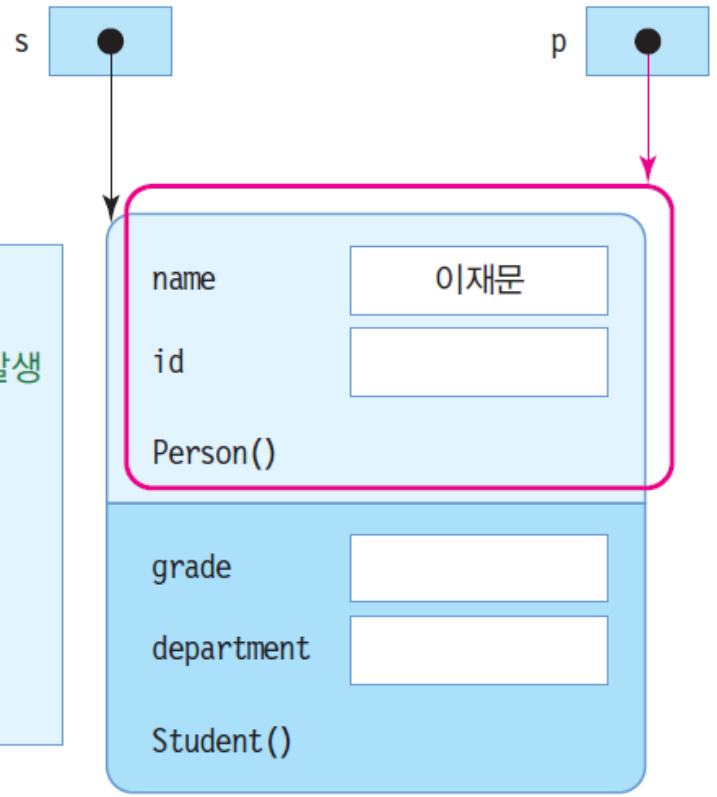
- 다운캐스팅(downcasting)
  - 슈퍼 클래스 레퍼런스를 서브 클래스 레퍼런스에 대입
  - 업캐스팅된 것을 다시 원래대로 되돌리는 것
  - 명시적으로 타입 지정

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

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

# 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
```

# instanceof 연산자와 객체의 타입 구별



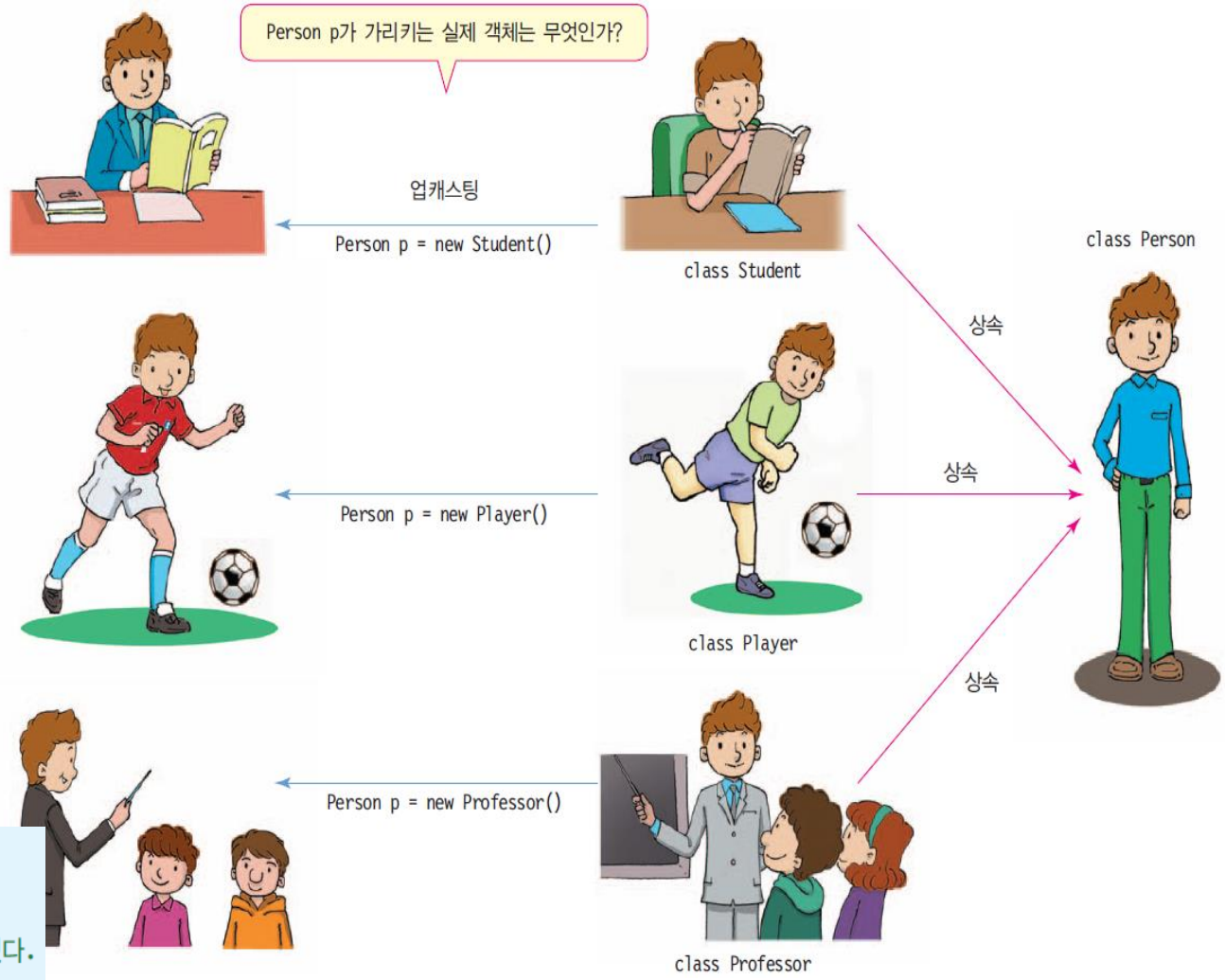
- 업캐스팅된 레퍼런스로는 객체의 진짜 타입을 구분하기 어려움
  - 슈퍼 클래스는 여러 서브 클래스에 상속되기 때문
    - 슈퍼 클래스 레퍼런스로 서브 클래스 객체를 가리킬 수 있음
- instanceof 연산자
  - instanceof 연산자
    - 레퍼런스가 가리키는 객체의 진짜 타입 식별
  - 사용법

객체레퍼런스 **instanceof** 클래스타입

연산의 결과 : true/false의 불린 값

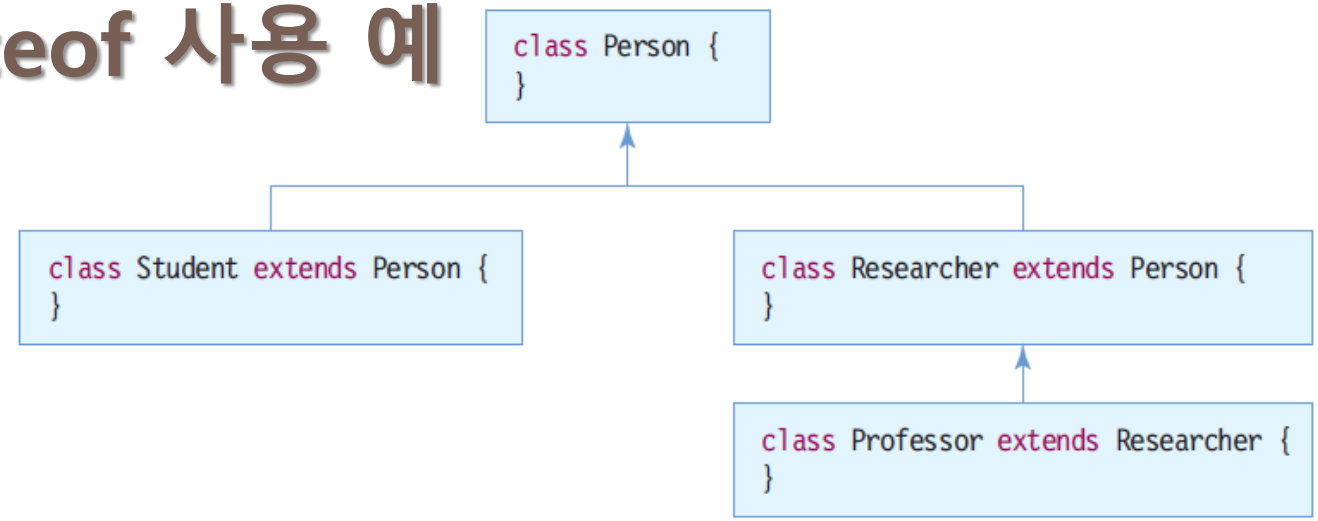
# 업캐스팅된 객체의 실제 타입은 무엇?

```
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.
.....
}
```

# 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
```

# 예제 : instanceof를 이용한 객체 구별



instanceof를 이용하여 객체의 타입을 구별하는 예를 만들어보자.

```
class Person {}
class Student extends Person {}
class Researcher extends Person {}
class Professor extends Researcher {}

public class InstanceofExample {
    public static void main(String[] args) {
        Person jee= new Student();
        Person kim = new Professor();
        Person lee = new Researcher();
        if (jee instanceof Student) // jee는 Student 타입이므로 true
            System.out.println("jee는 Student 타입");
        if (jee instanceof Researcher) // jee는 Researcher 타입이 아니므로 false
            System.out.println("jee는 Researcher 타입");
        if (kim instanceof Student) // kim은 Student 타입이 아니므로 false
            System.out.println("kim은 Student 타입");
        if (kim instanceof Professor) // kim은 Professor 타입이므로 true
            System.out.println("kim은 Professor 타입");
        if (kim instanceof Researcher) // kim은 Researcher 타입이기도 하므로 true
            System.out.println("kim은 Researcher 타입");
        if (kim instanceof Person) // kim은 Person 타입이기도 하므로 true
            System.out.println("kim은 Person 타입");
        if (lee instanceof Professor) // lee는 Professor 타입이 아니므로 false
            System.out.println("lee는 Professor 타입");
        if ("java" instanceof String) // "java"는 String 타입의 인스턴스이므로 true
            System.out.println("₩"java₩"는 String 타입");
    }
}
```

```
jee는 Student 타입
kim은 Professor 타입
kim은 Researcher 타입
kim은 Person 타입
"java"는 String 타입
```