상 속

컴퓨터공학전공
박요한

수업내용

- 업캐스팅 / 다운캐스팅
- 메소드 오버라이딩

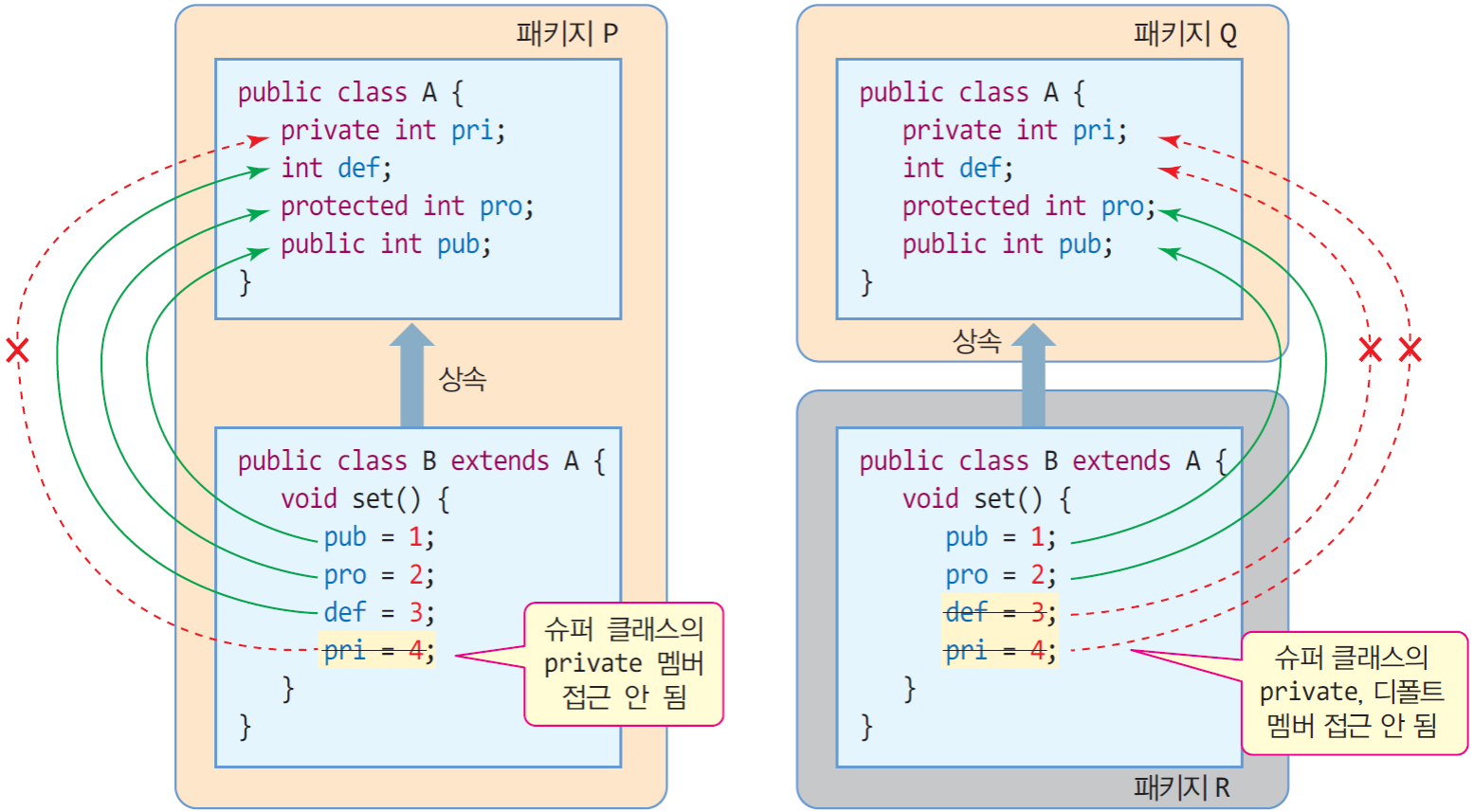
클래스 상속과 객체

- 자바의 상속 선언

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

✓부모 클래스 -> 슈퍼 클래스(super class)로 부름
✓자식 클래스 -> 서브 클래스(sub class)로 부름
✓extends 키워드 사용
  - 슈퍼 클래스를 확장한다는 개념

상속과 접근지정자



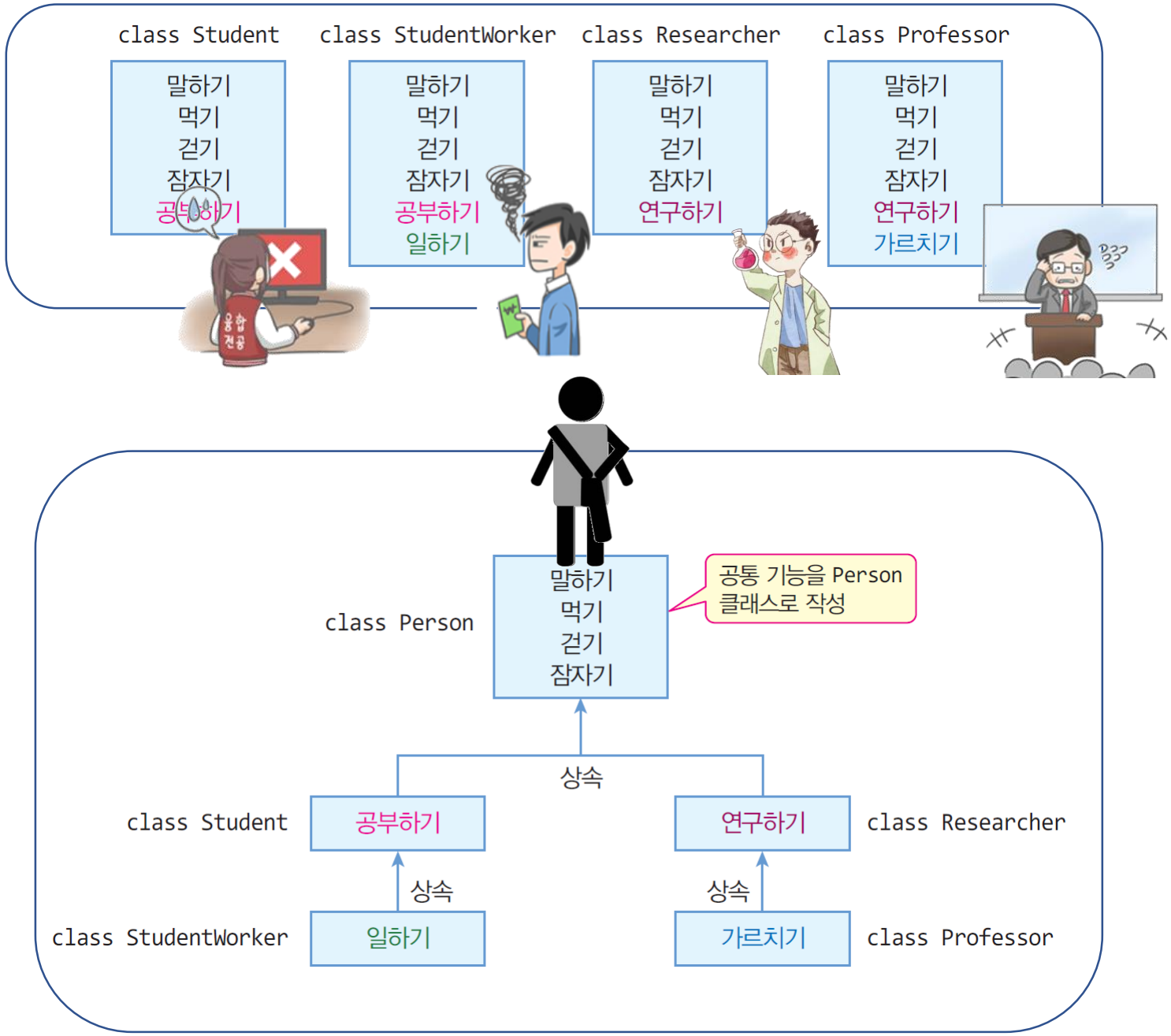
(a) 슈퍼 클래스와 서브 클래스가 동일한 패키지에 있는 경우

(b) 슈퍼 클래스와 서브 클래스가 서로 다른 패키지에 있는 경우

상속과 생성자

- 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

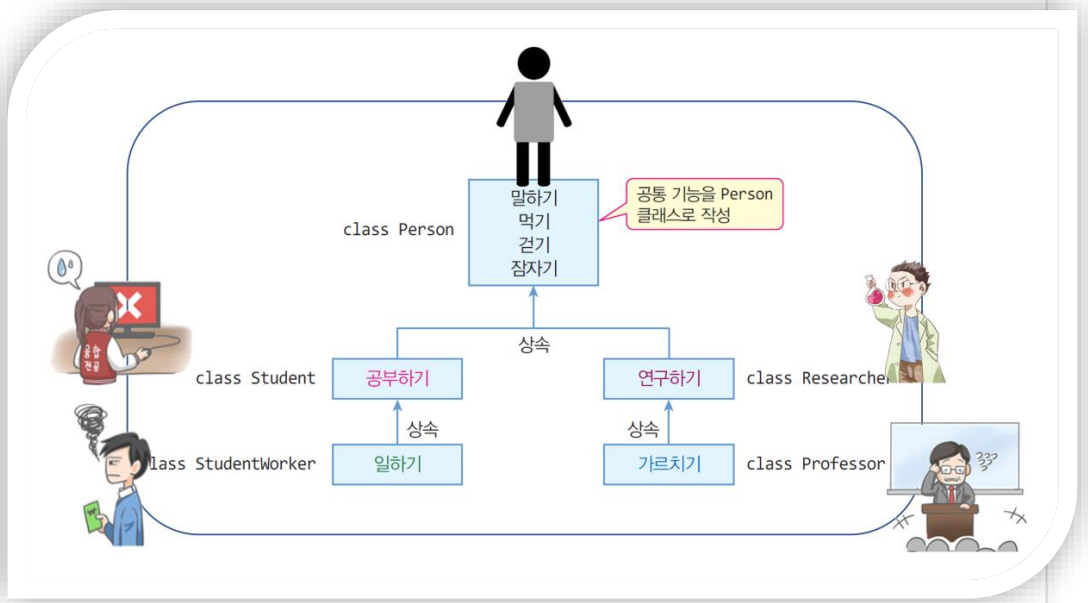
업캐스팅/다운캐스팅


class Student
말하기
먹기
걷기
잠자기
공부하기
class StudentWorker
말하기
먹기
걷기
잠자기
공부하기
일하기
class Researcher
말하기
먹기
걷기
잠자기
연구하기
class Professor
말하기
먹기
걷기
잠자기
연구하기
가르치기
class Person
말하기
먹기
걷기
잠자기
공통 기능을 Person 클래스로 작성
상속
class Student
공부하기
연구하기
class Researcher
상속
상속
class StudentWorker
일하기
가르치기
class Professor

업캐스팅(upcasting)

- 서브 클래스의 객체는…
  - ✓슈퍼 클래스의 멤버를 모두 가지고 있음
  - ✓슈퍼 클래스의 객체로 취급할 수 있음
    - ‘학생은 사람이다’의 논리와 같음
- 업캐스팅이란?
  - ✓서브 클래스 객체를 슈퍼 클래스 타입으로 타입 변환

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

Student s = new Student();
Person p = s; // 업캐스팅, 자동타입변환
```

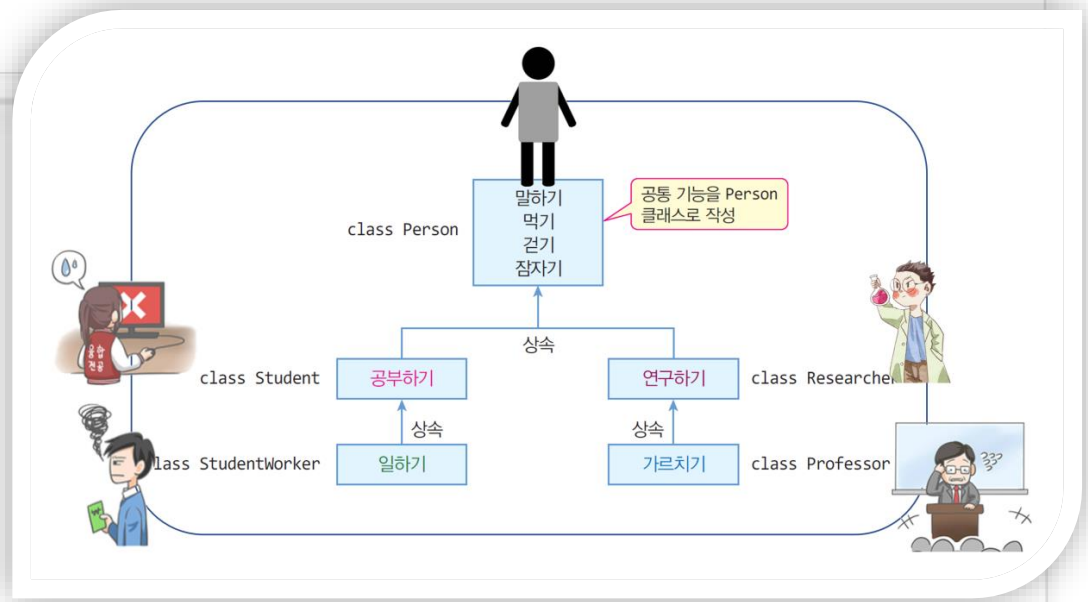


- 업캐스팅된 레퍼런스
  - ✓객체 내에 슈퍼 클래스의 멤버만 접근 가능

업캐스팅(upcasting)

- 업캐스팅된 레퍼런스
  - ✓객체 내에 슈퍼 클래스의 멤버만 접근 가능

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

Student s = new Student();
Person p = s; // 업캐스팅, 자동타입변환
```



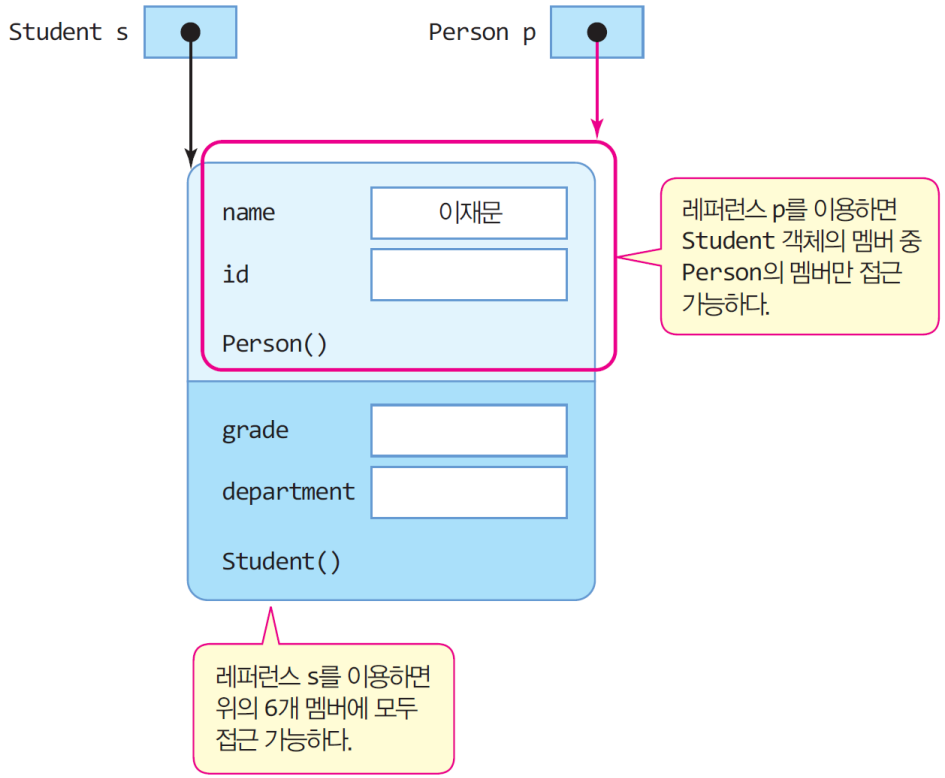
업캐스팅 사례

```java
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

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

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

오류

실행 결과

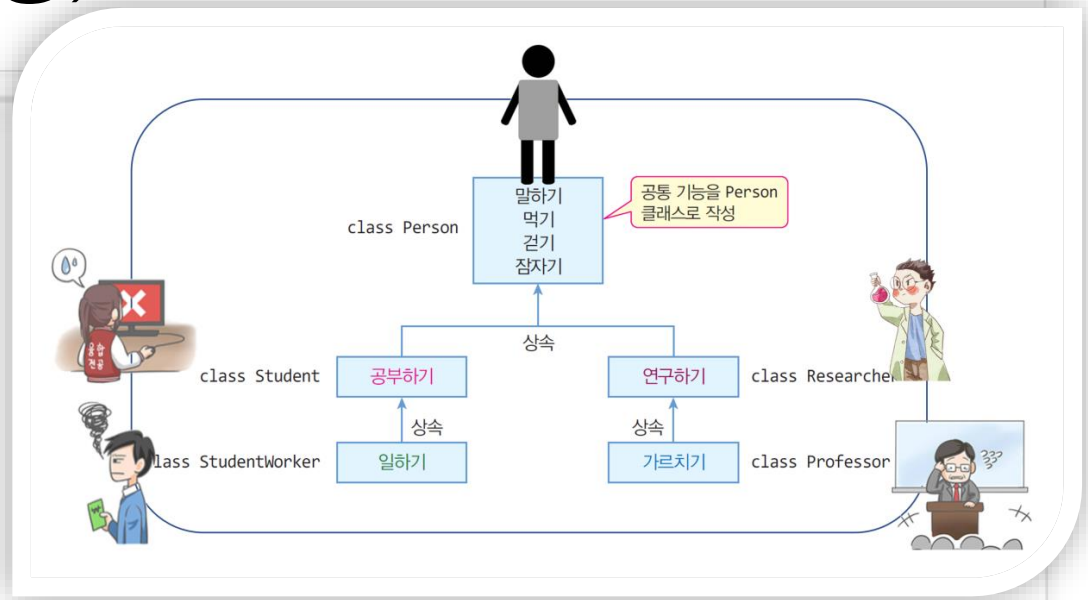
```
이재문
```



다운캐스팅(downcasting)

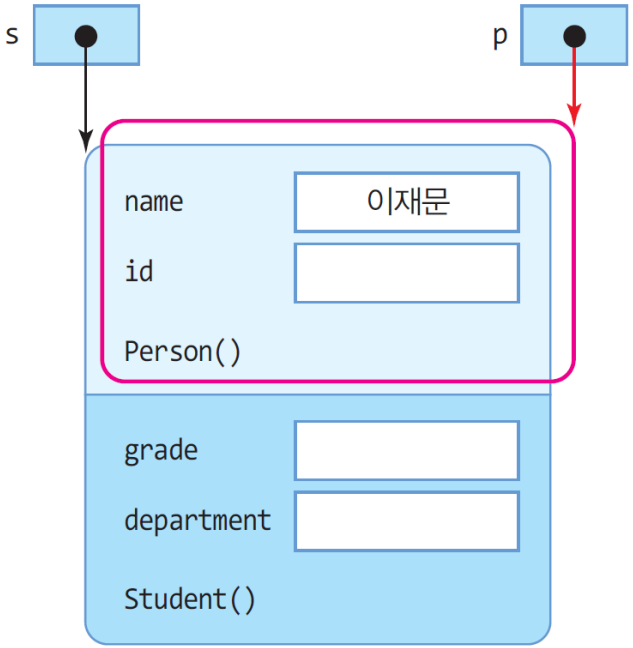
- 다운캐스팅이란?
  - ✓슈퍼 클래스 객체를 서브 클래스 타입으로 변환
  - ✓개발자의 명시적 타입 변환 필요

```
class Person { … }
class Student extends Person { … }
…
Person p = new Person("이영희");
Student s = (Student)p; // 다운캐스팅
```



다운캐스팅 사례

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

        s = (Student)p;  // 다운캐스팅

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

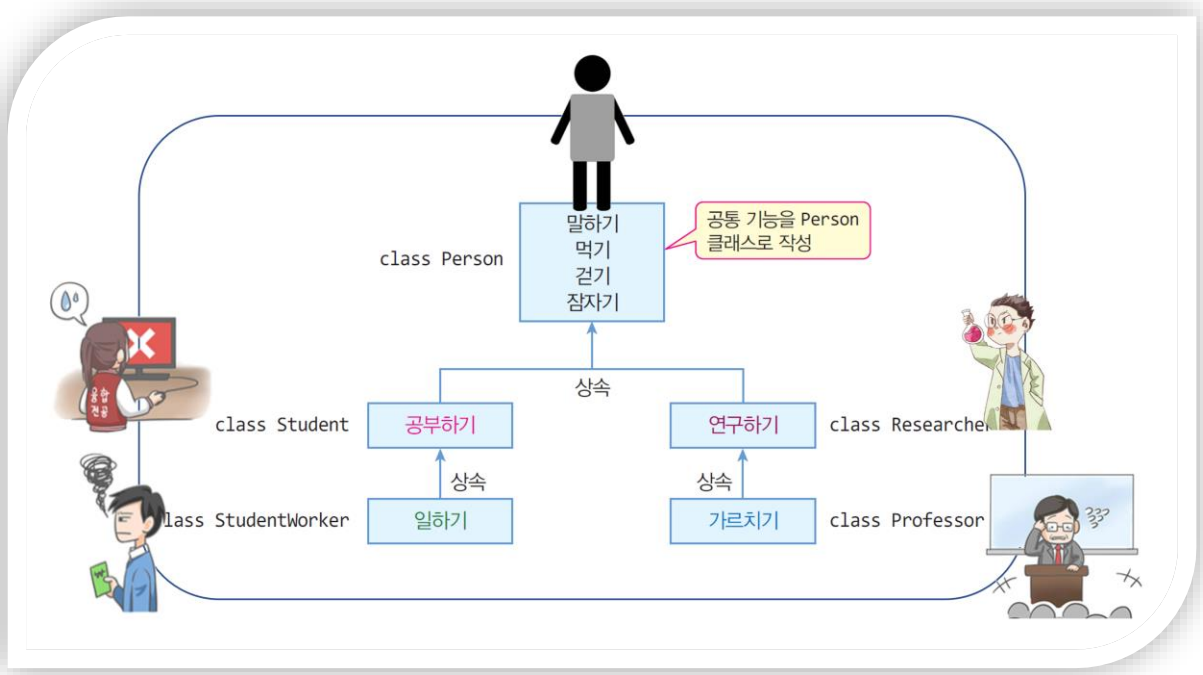


실행 결과

```
이재문
```

왜 casting 작업이 필요한가?

- 하나의 배열로 객체 관리



instanceof 연산자와 객체의 타입 판단

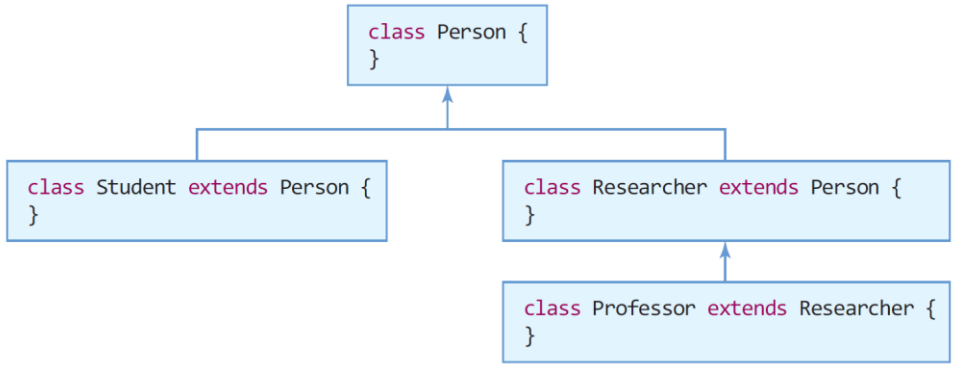
- 업캐스팅된 레퍼런스로 객체의 타입 판단 어려움
  - ✓슈퍼 클래스는 여러 서브 클래스에 상속되기 때문
  - ✓예) '사람' 레퍼런스(팻말)이 가리키는 박스에 들어 있는 객체의 타입이 학생인지, 교수인지 레퍼런스 타입(펫말)만 보고서는 알 수 없음

- instanceof 연산자
  - ✓레퍼런스가 가리키는 객체의 타입 식별을 위해 사용
  - ✓사용법

객체레퍼런스 **instanceof** 클래스타입

연산의 결과 : true/false의 불린 값

instanceof 사용 예




```
Person jee= new Student();
Person kim = new Professor();
Person lee = new Researcher();
if (jee instanceof Person)
if (jee instanceof Student)
if (kim instanceof Student)
if (kim instanceof Professor)
if (kim instanceof Researcher)
if (lee instanceof Professor)
```

```
if(3 instanceof int)
```

```
if("java" instanceof String)
```

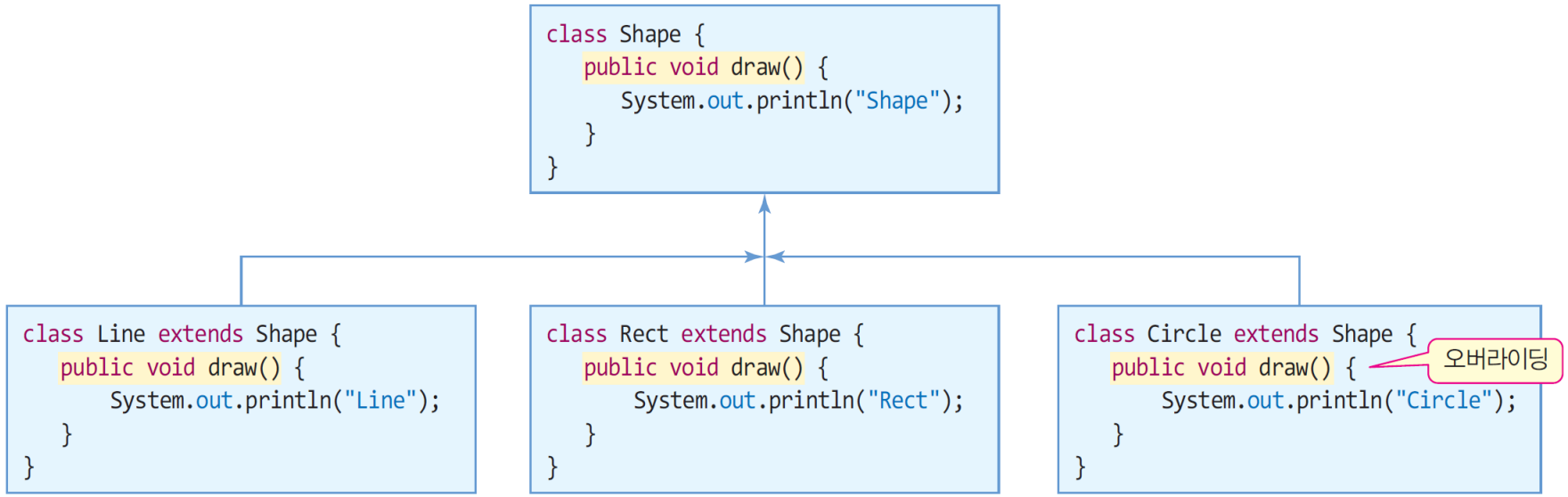
메소드 오버라이딩

메소드 오버라이딩

- 메소드 오버라이딩(Method Overriding)
  - ✓슈퍼 클래스의 메소드를 서브 클래스에서 재정의(수정)
    - 슈퍼 클래스 메소드의 이름, 매개변수 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
  - ✓메소드 무시하기, 덮어쓰기, 재정의로 번역되기도 함
  - ✓동적 바인딩 발생
    - 서브 클래스에

메소드 오버라이딩 사례

Shape 클래스의 draw() 메소드를 Line, Rect, Circle 클래스에서 각각 오버라이딩한 사례



```
class Shape {
    public void draw() {
        System.out.println("Shape");
    }
}
```

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

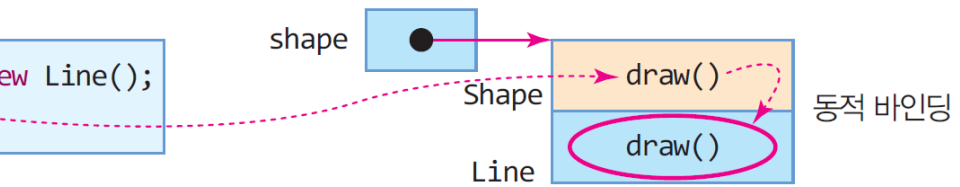
```
class Rect extends Shape {
    public void draw() {
        System.out.println("Rect");
    }
}
```

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

오버라이딩

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

```
Shape shape = new Line();
shape.draw();
```



실행 결과

```
Line
```

오버라이딩의 목적, 다형성 실현

- 오버라이딩
  - ✓수퍼 클래스에 선언된 메소드를, 각 서브 클래스들이 자신만의 내용으로 새로 구현하는 기능
  - ✓상속을 통해 '하나의 인터페이스(같은 이름)에 서로 다른 내용 구현'이라는 객체 지향의 다형성 실현
    - Line 클래스에서 draw()는 선을 그리고
    - Circle 클래스에서 draw()는 원을 그리고
    - Rect 클래스에서 draw()는 사각형 그리고
- 오버라이딩은 실행 시간 다형성 실현
  - ✓동적 바인딩을 통해 실행 중에 다형성 실현
    - 오버로딩은 컴파일 타임 다형성 실현

예제 5-5 : 메소드 오버라이딩으로 다형성 실현

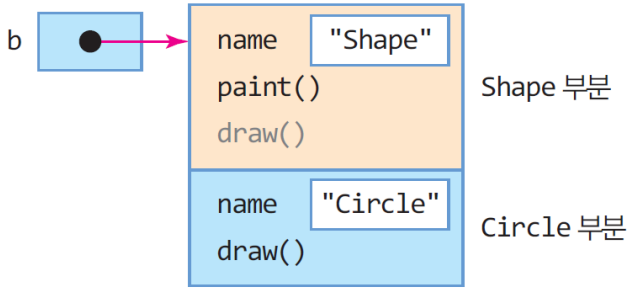
```
class Shape { // 슈퍼 클래스
    public Shape next;
    public Shape() { next = null; }

    public void draw() {
        System.out.println("Shape");
    }
}

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

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

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

```
public class MethodOverridingEx {
    static void paint(Shape p) {
        p.draw(); // p가 가리키는 객체 내에 오버라이딩된 draw() 호출.
                  // 동적 바인딩
    }

    public static void main(String[] args) {
        Line line = new Line();
        paint(line);
        paint(new Shape());
        paint(new Line());
        paint(new Rect());
        paint(new Circle());
    }
}
```

```
Line
Shape
Line
Rect
Circle
```

오버라이딩과 super 키워드

- super는 슈퍼 클래스의 멤버를 접근할 때 사용되는 레퍼런스
- 서브 클래스에서만 사용
- 슈퍼 클래스의 메소드 호출
- 컴파일러는 super의 접근을 **정적 바인딩**으로 처리




```
class Shape {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println(name);
    }
}
public class Circle extends Shape {
    protected String name;
    @Override
    public void draw() {
        name = "Circle";
        super.name = "Shape";
        super.draw();
        System.out.println(name);
    }
    public static void main(String [] args) {
        Shape b = new Circle();
        b.paint();
    }
}
```

정적 바인딩

실행 결과

```
Shape
Circle
```

오버로딩 vs 오버라이딩

비교 요소	메소드 오버로딩	메소드 오버라이딩
선언	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 작성하여 사용의 편리성 향상. 다형성 실현	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함. 다형성 실현
조건	메소드 이름은 반드시 동일하고, 매개변수 타입이나 개수가 달라야 성립	메소드의 이름, 매개변수 타입과 개수, 리턴 타입이 모두 동일하여야 성립
바인딩	정적 바인딩. 호출될 메소드는 컴파일 시에 결정	동적 바인딩. 실행 시간에 오버라이딩된 메소드 찾아 호출

```
public class HeadQuarter {

    public HeadQuarter() {
        System.out.println("HeadQuarter의 생성자");
    }

    public void orderBulgogi() {
        System.out.println("5000원 입니다.");
    }

    public void orderChicken() {
        System.out.println("3000원 입니다.");
    }

    public void orderShrimp() {
        System.out.println("6000원 입니다.");
    }
}
```

```
public class Daegu extends HeadQuarter{

    public Daegu() {
        System.out.println("Daegu의 생성자");
    }

    public void orderBulgogi() {
        System.out.println("4000원 입니다.");
    }

    public void orderBulgogi(int i) {
        System.out.println(i +"개는 4000원 입니다.");
    }

    public void orderApple() {
        System.out.println("6000원 입니다.");
    }
}
```

```
public class MainClass{

    public static void main(String[] args) {
      Daegu dgNo1 = new Daegu();
      dgNo1.orderBulgogi();
      dgNo1.orderChicken();
      dgNo1.ordershrimp();
      dgNo1.orderApple();
}
```