6장 상속

목차

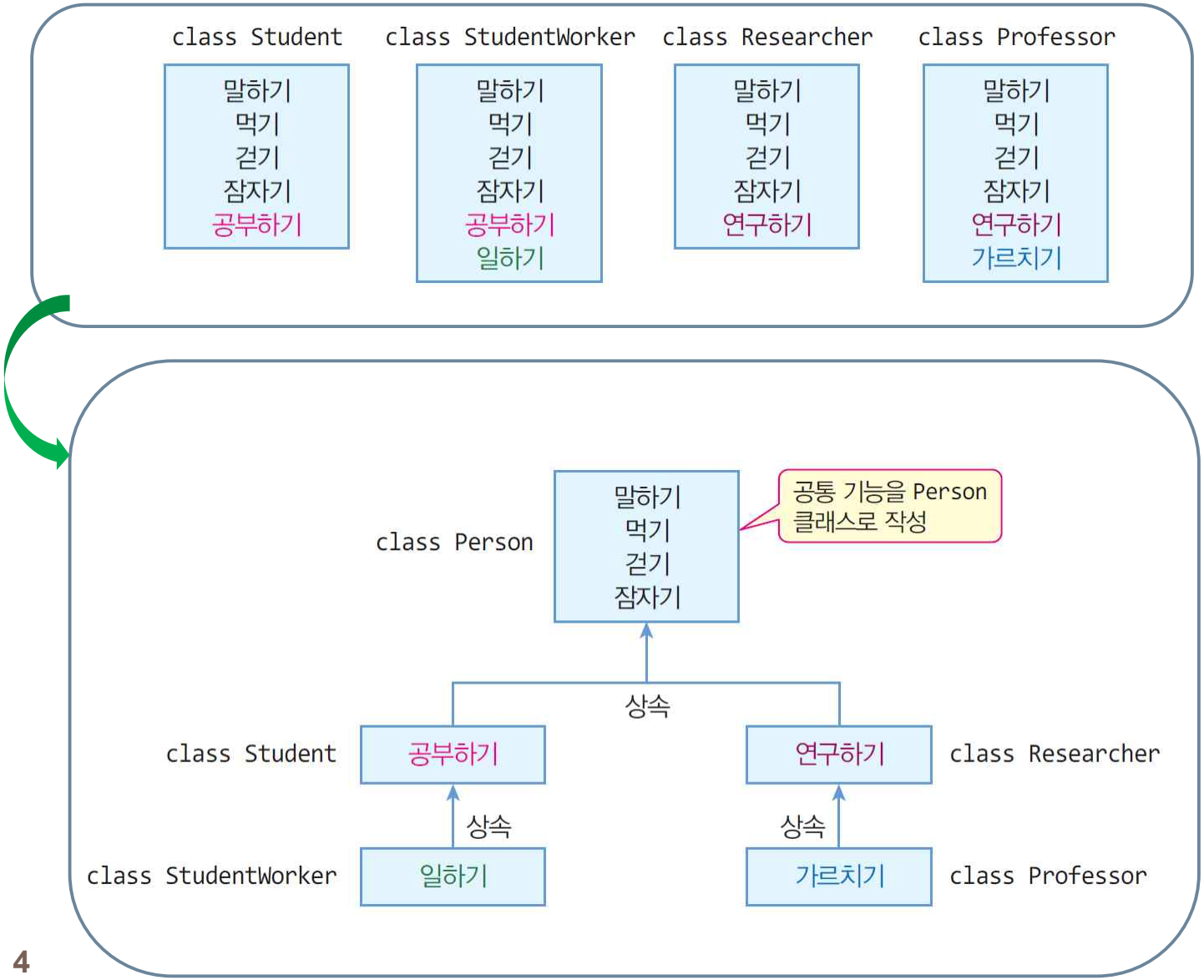
상속 (inheritance)

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

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

- **객체 지향에서 상속의 장점**
  - 클래스의 간결화
    - 멤버의 중복 작성 불필요
  - 클래스 관리 용이
    - 클래스들의 계층적 분류
  - 소프트웨어의 생산성 향상
    - 클래스 재사용과 확장 용이
    - 새로운 클래스의 작성 속도 빠름

상속의 편리한 사례



상속이 없는 경우
중복된 멤버를 가진
4 개의 클래스

상속을 이용한
경우 중복이 제거되고
간결해진 클래스 구조

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

예제 6-1 : 클래스 상속 만들기 - Point와 ColorPoint 클래스

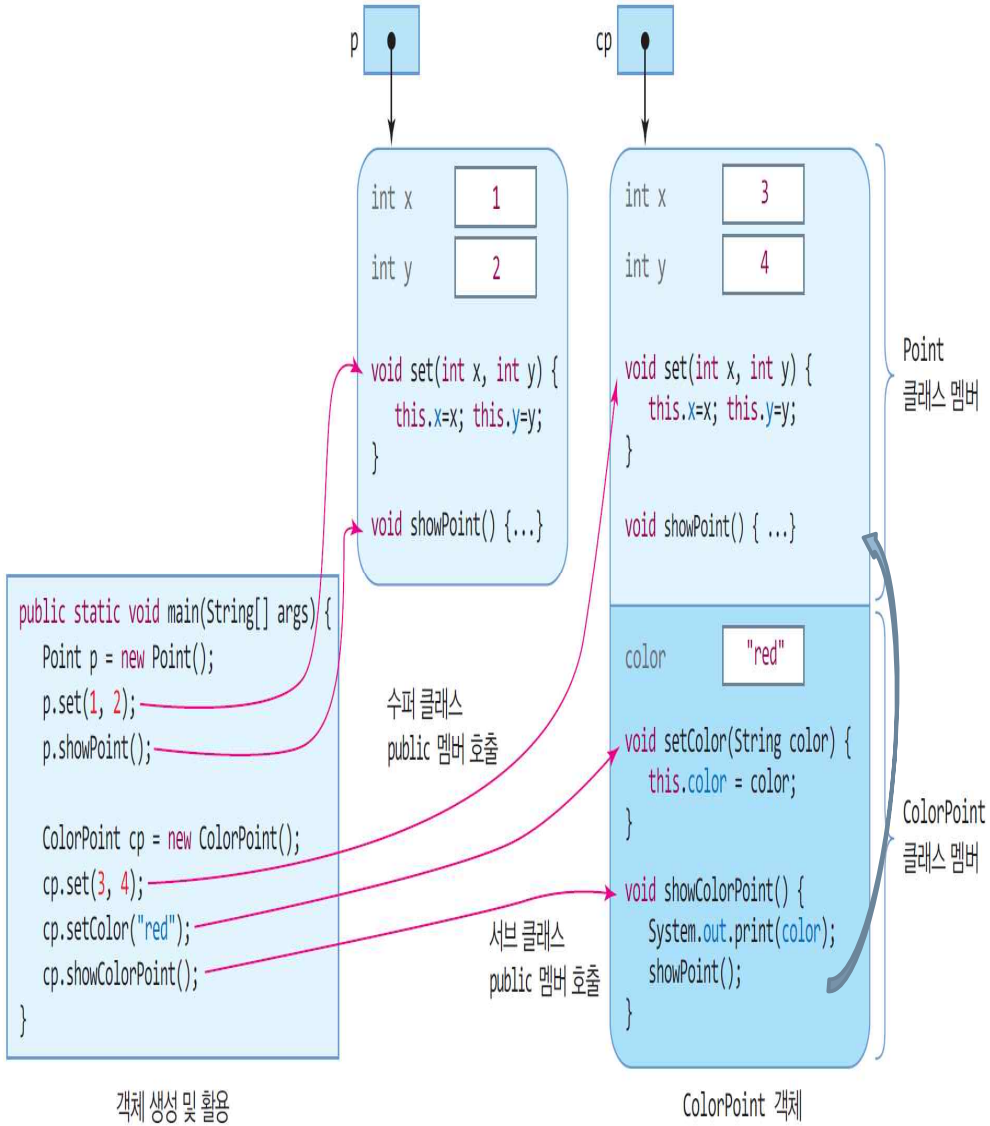
(x, y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 색을 가진 점을 표현하는 ColorPoint 클래스를 만들고 활용해보자.

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

(1,2)
red(3,4)

```
// Point를 상속받은 ColorPoint 선언
class ColorPoint extends Point {
    private String color; // 점의 색
    public void setColor(String color) {
        this.color = color;  }
    public void showColorPoint() { // 컬러 점의 좌표 출력
        System.out.print(color);
        showPoint(); // Point 클래스의 showPoint() 호출
    }  }//class end
```

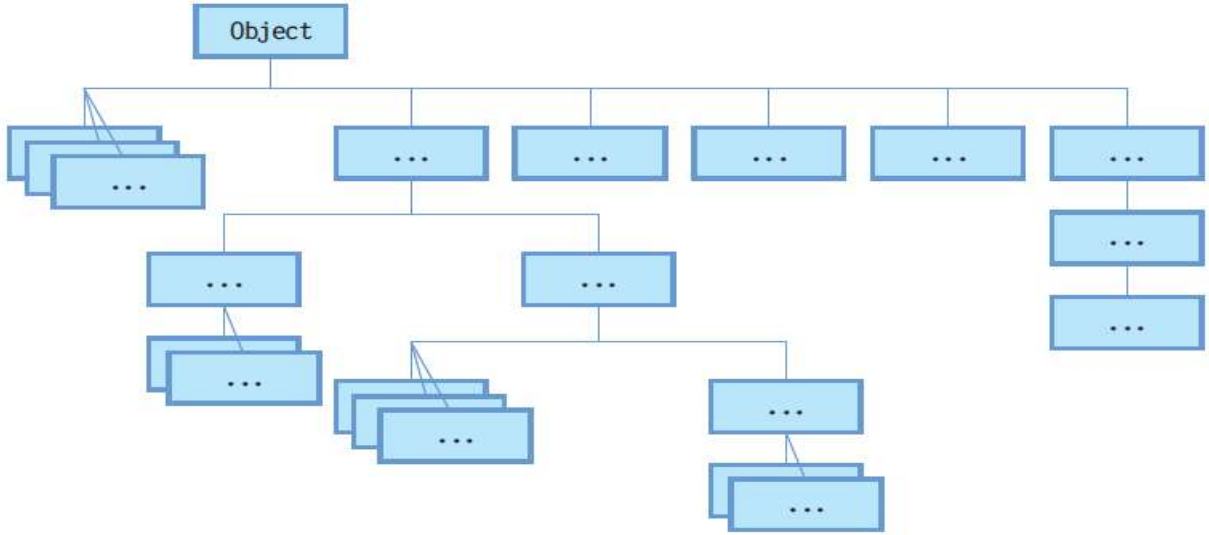
```
public class ColorPointEx {
    public static void main(String [] args) {
        Point p = new Point(); // Point 객체 생성
        p.set(1, 2); // Point 클래스의 set() 호출
        p.showPoint();
        ColorPoint cp = new ColorPoint(); // ColorPoint 객체
        cp.set(3, 4); // Point의 set() 호출
        cp.setColor("red"); // ColorPoint의 setColor() 호출
        cp.showColorPoint(); // 컬러와 좌표 출력
    } }//class end
```



* new ColorPoint()에 의해 생긴 서브 클래스 객체에 주목

자바 상속의 특징

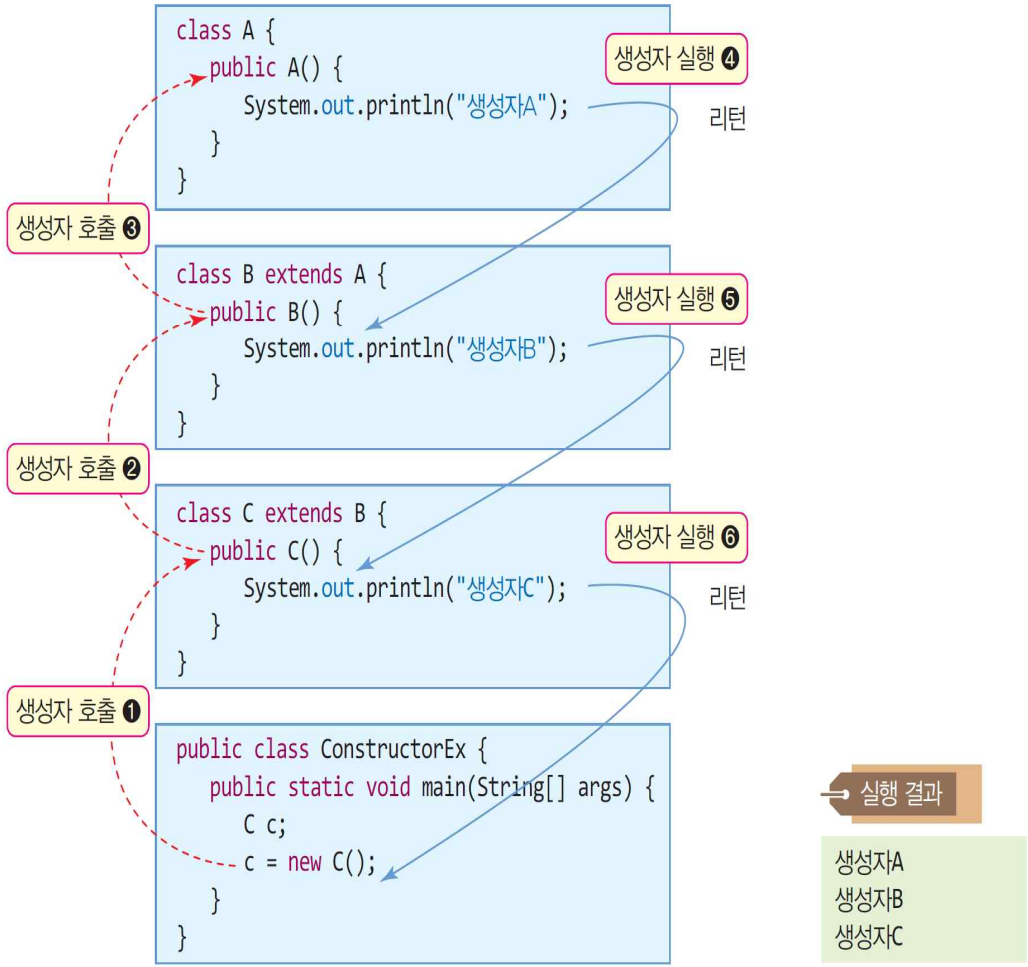
- 클래스의 다중 상속 지원하지 않음
  - 다중 상속이 안되는 이유: 다중상속을 허용하면 클래스의 성질이 복합적으로 섞여 부모와 IS-A 관계가 모호해져 정체성이 불분명해져 여러가지 객체지향 원칙에 위배될 수 있다.
- 상속 횟수 무제한
- 상속의 최상위 조상 클래스는 java.lang.Object 클래스
  - 모든 클래스는 자동으로 java.lang.Object를 상속받음
  - 자바 컴파일러에 의해 자동으로 이루어짐



서브 클래스/슈퍼 클래스의 생성자 호출 및 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

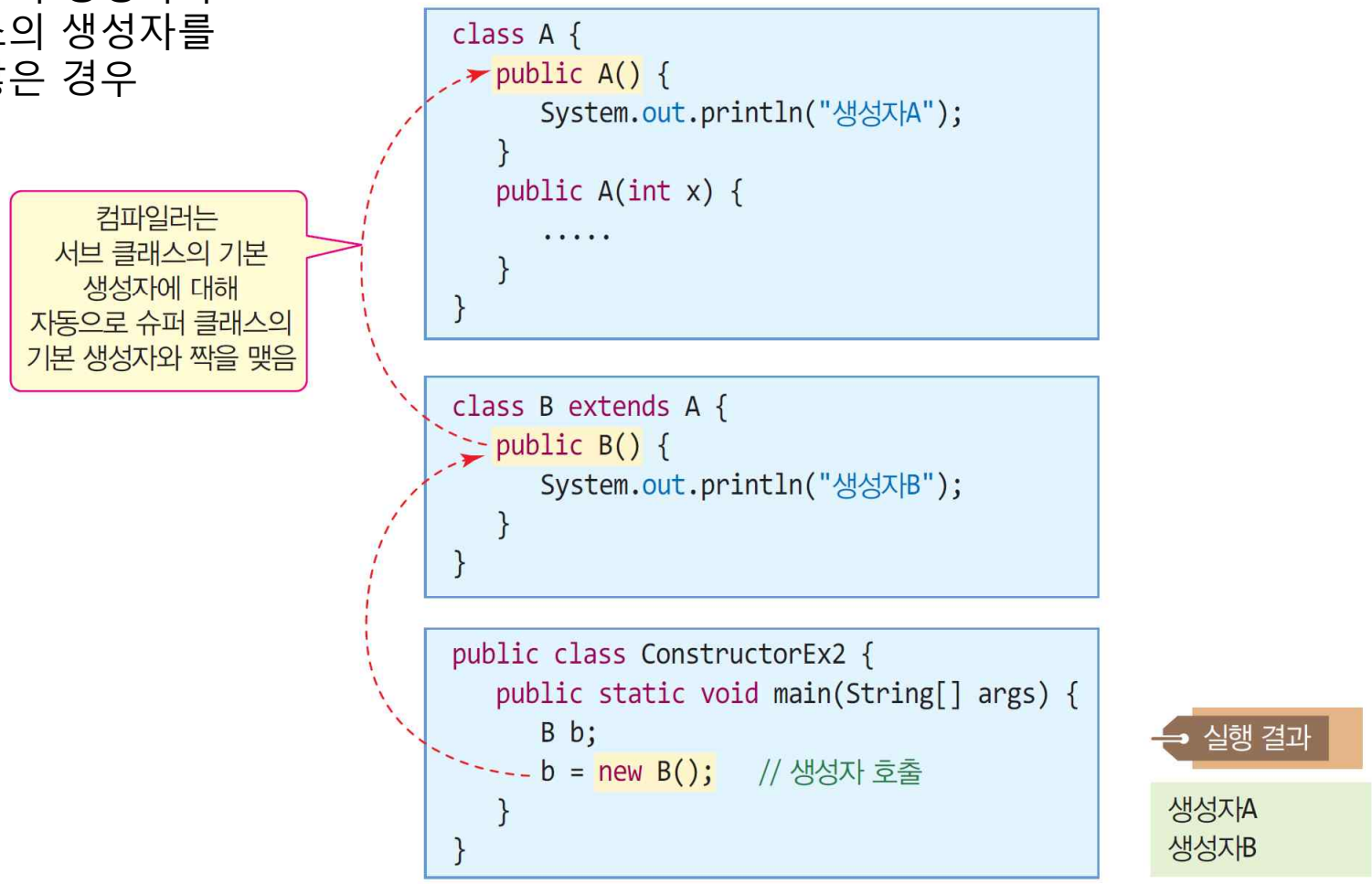


서브 클래스에서 슈퍼 클래스의 생성자 선택

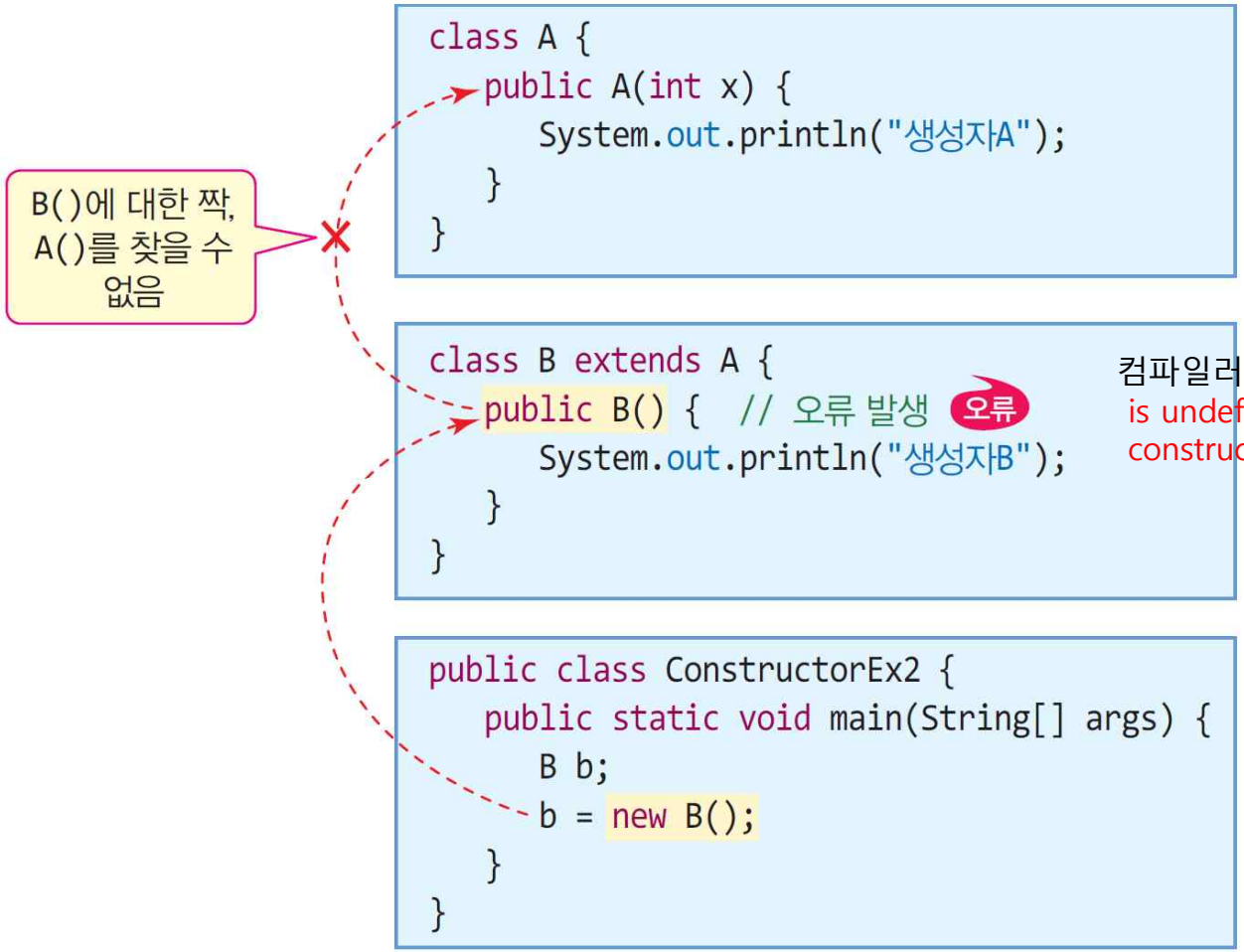
- 상속 관계에서의 생성자
  - 슈퍼 클래스와 서브 클래스 각각 각각 여러 생성자 작성 가능
- 서브 클래스 생성자 작성 원칙
  - 서버 클래스 생성자에서 슈퍼 클래스 생성자 하나 선택
- 서브 클래스에서 슈퍼 클래스의 생성자를 선택하지 않는 경우
  - 컴파일러가 자동으로 슈퍼 클래스의 기본 생성자 선택
- 서브 클래스에서 슈퍼 클래스의 생성자를 선택하는 방법
  - super() 이용

슈퍼 클래스의 기본 생성자가 자동 선택

서브 클래스의 생성자가
슈퍼 클래스의 생성자를
선택하지 않은 경우



슈퍼 클래스에 기본 생성자가 없어 오류 난 경우



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

서브 클래스에 매개변수를 가진 생성자

서브 클래스의 생성자가 슈퍼 클래스의 생성자를 선택하지 않은 경우
=>컴파일러가 자동으로 슈퍼 클래스의 기본 생성자 선택

```
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 명시적으로 슈퍼 클래스 생성자 선택

- super()
  - 서브 클래스에서 명시적으로 슈퍼 클래스의 생성자 선택 호출
    - 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);
    }
}
```

x에 5 전달

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

x는 5

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

실행 결과

```
매개변수생성자A5
매개변수생성자B5
```

상속과 접근 제어

자바의 접근 지정자 4 가지

접근 지정자	동일 클래스	동일 패키지	자식 클래스	다른 패키지
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X

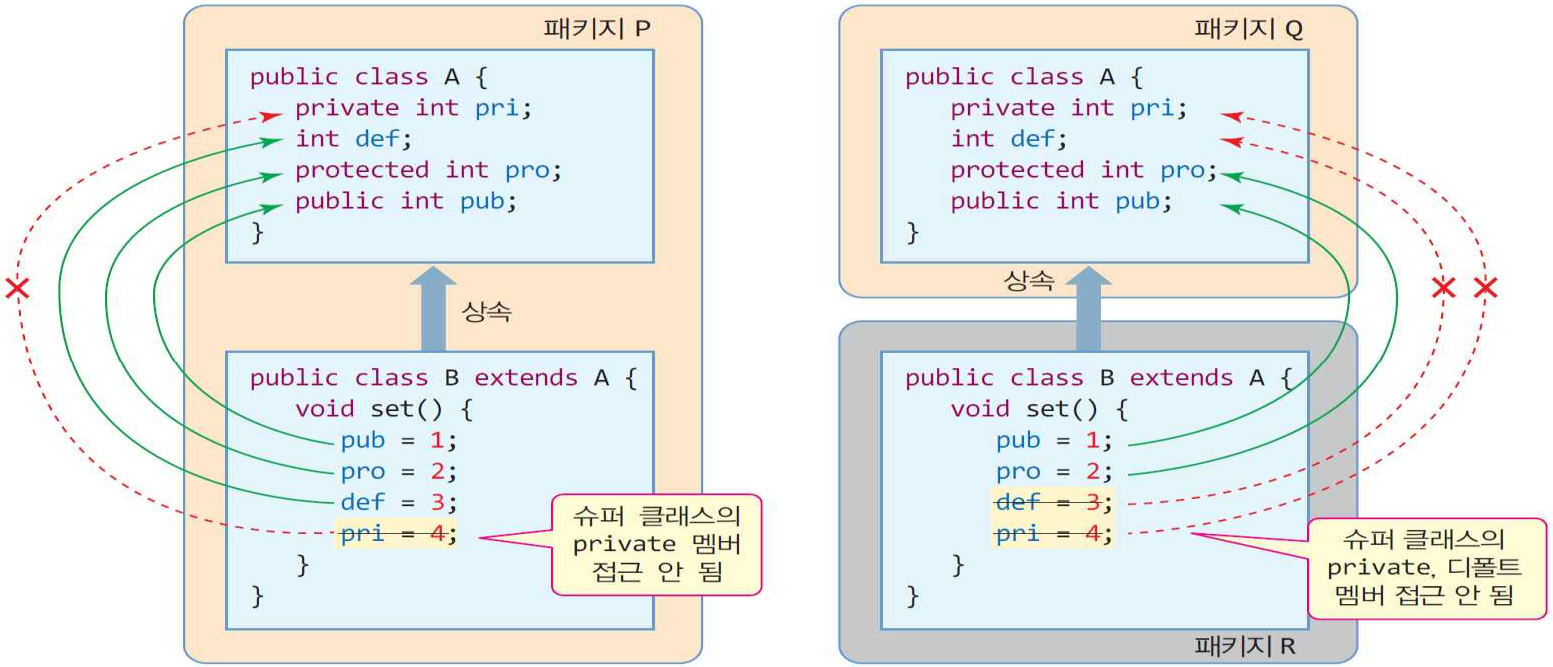
- public, protected, 디폴트, private
  - 상속 관계에서 주의할 접근 지정자는 private와 protected
- 슈퍼 클래스의 private 멤버
  - 슈퍼 클래스의 private 멤버는 다른 모든 클래스에 접근 불허
  - 클래스내의 멤버들에게만 접근 허용
- 슈퍼 클래스의 디폴트 멤버
  - 슈퍼 클래스의 디폴트 멤버는 패키지내 모든 클래스에 접근 허용
- 슈퍼 클래스의 public 멤버
  - 슈퍼 클래스의 public 멤버는 다른 모든 클래스에 접근 허용
- 슈퍼 클래스의 protected 멤버
  - 같은 패키지 내의 모든 클래스 접근 허용
  - 다른 패키지에 있어도 서브 클래스는 슈퍼 클래스의 protected 멤버 접근 가능

접근 지정자 사용 시 주의 사항

- private 멤버는 자식 클래스에 상속되지 않는다.
- 클래스 멤버는 어떤 접근 지정자로도 지정할 수 있지만, 클래스는 protected와 private으로 지정할 수 없다.
- 메서드를 오버라이딩할 때 부모 클래스의 메서드보다 가시성을 더 좁게 할 수는 없다.

슈퍼 클래스의 멤버에 대한 서브 클래스의 접근

접근 지정자	동일 클래스	동일 패키지	자식 클래스	다른 패키지
public	O	O	O	O
protected	O	O	O	×
없음	O	O	×	×
private	O	×	×	×

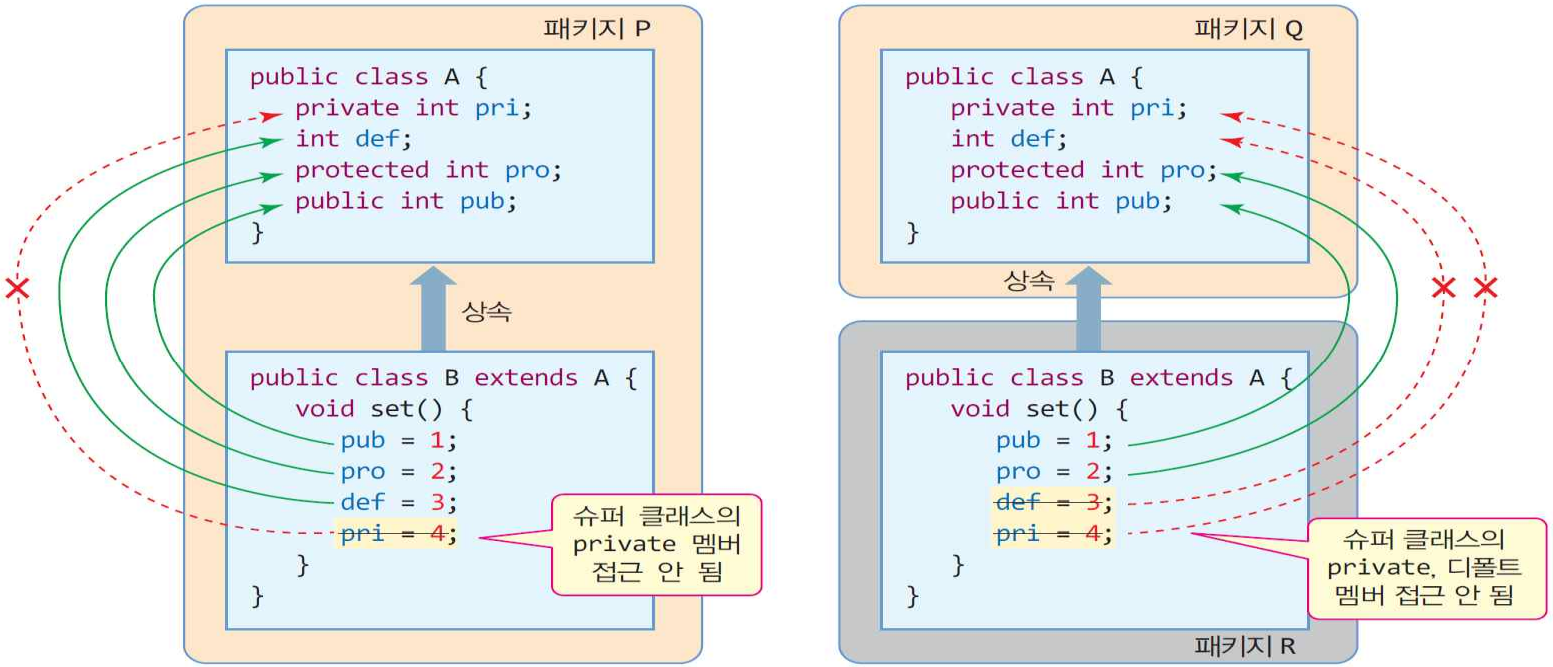


(a) 슈퍼 클래스와 서브 클래스가 동일한 패키지에 있는 경우

(b) 슈퍼 클래스와 서브 클래스가 서로 다른 패키지에 있는 경우

슈퍼 클래스의 멤버에 대한 서브 클래스의 접근

접근 지정자	동일 클래스	동일 패키지	자식 클래스	다른 패키지
public	O	O	O	O
protected	O	O	O	×
없음	O	O	×	×
private	O	×	×	×

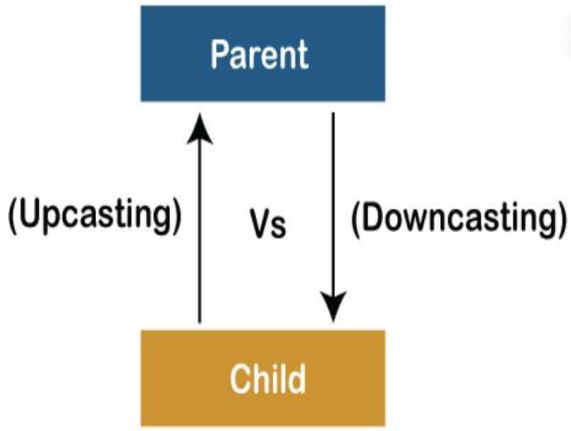


(a) 슈퍼 클래스와 서브 클래스가 동일한 패키지에 있는 경우

(b) 슈퍼 클래스와 서브 클래스가 서로 다른 패키지에 있는 경우

타입 변환과 다형성

- **타입 변환(캐스팅)**
  - 기본 타입 변환(short, int, long, float, double, ...)과 참조 타입 변환(array, enum, class, ...)이 있다.
- **객체의 타입 변환(참조형 캐스팅: 업캐스팅과 다운 캐스팅)**
  - 참조 타입 데이터도 기초 타입 데이터처럼 타입 변환 가능.
  - 그러나 **상속 관계일 경우만 타입 변환 가능**
  - 기초 타입처럼 자동 타입 변환과 강제 타입 변환이 있다.
  - **참조변수의 형변환은 사용할 수 있는 멤버의 갯수를 조절한다.**
    - 기본형 타입의 형변환은 값(3.6 → 3)으로 변환이지만
      객체 형변환 멤버 갯수만 달라지게 된다.

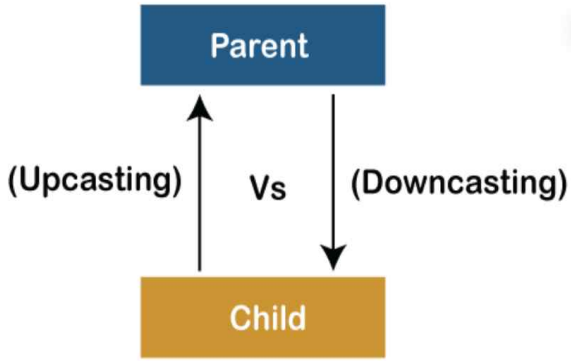


형제 클래스 끼리는 타입이 다르기 때문에 참조 형변환 불가능

- **자동 타입 변환 : 업캐스팅(upcasting)**
  - 서브 클래스 객체를 슈퍼 클래스 타입으로 타입 변환
- **강제 타입 변환 : 다운캐스팅(downcasting)**
  - 슈퍼 클래스 객체를 서브 클래스 타입으로 변환, 명시적 타입 변환 필요
- **instanceof 연산자**
  - 타입 변환된 객체의 구별
  - instanceof 연산자는 변수가 해당 타입이나 자식 타입이라면 true참을 반환하고, 그렇지 않다면 false반환 그러나 변수가 해당 타입과 관련이 없다면 오류 발생
- **타입 변환을 이용한 다형성**
  - 다형성은 하나의 참조 변수에 여러 객체를 대입해서 다양한 동작을 수행하도록 한다.

    따라서 다형성으로 다양한 객체에 동일한 명령어를 적용해 객체의 종류에 따라서 다양한 동작을 얻을 수 있다.

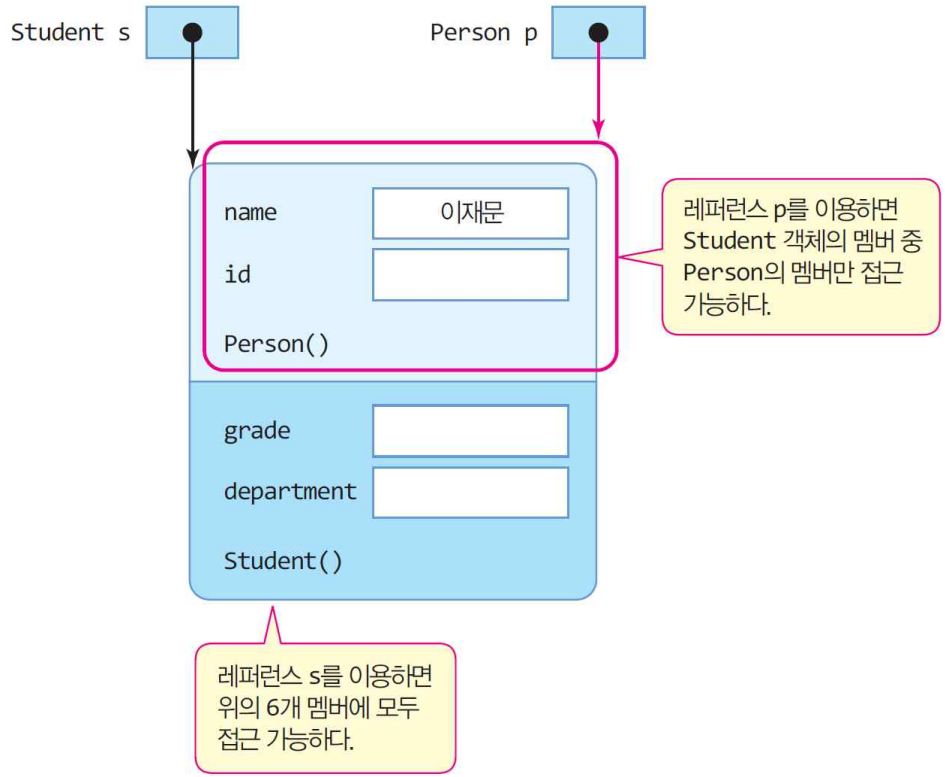
타입 변환과 다형성



- **자동 타입 변환 : 업캐스팅(upcasting)**
  - 서브 클래스 객체를 슈퍼 클래스 타입으로 타입 변환
  - 업캐스팅된 레퍼런스 : 객체 내에 슈퍼 클래스의 멤버만 접근 가능
  - 주의 사항
    - 슈퍼(부모) 클래스로 캐스팅 된다는 것은 멤버의 갯수 감소를 의미하여 업캐스팅 하면 멤버 갯수가 제한되어 자식 클래스에만 있는 멤버는 사용할 수 없게 된다
    - 업캐스팅하고 메소드를 실행 할 때 자식 클래스에서 오버라이딩한 메서드가 있으면 자식 클래스의 오버라이딩 된 메서드가 실행이 된다.
  - 업캐스팅을 사용하는 이유
    - 공통적으로 할 수 있는 부분을 만들어 간단하게 다루기 위해서이다. 상속 관계에서 상속 받은 서브 클래스가 몇 개이든 하나의 인스턴스로 묶어서 관리할 수 있기 때문이다.
- **강제 타입 변환 : 다운캐스팅(downcasting)**
  - 슈퍼 클래스 객체를 서브 클래스 타입으로 변환
  - 개발자의 명시적 타입 변환 필요
  - 다운캐스팅의 목적
    - **업캐스팅한 객체를** 다시 자식 클래스 타입의 객체로 되돌리는데 목적을 둔다. (복구)
    - 부모 클래스로 업 캐스팅된 자식 클래스를 복구하여, 본인의 필드와 기능을 회복하기 위해 있는 것이다. 즉, 원래 있던 기능을 회복하기 위해 다운캐스팅을 하는 것이다.
  - 주의 사항
    - 업캐스팅 되지 않는 생 부모 객체를 그대로 다운캐스팅 하면 오류(ClassCastException)가 발생한다.

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

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

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

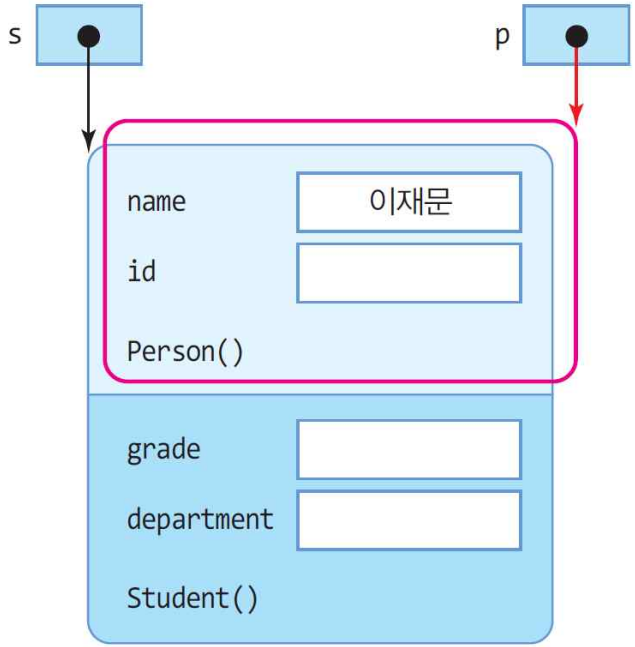


실행 결과

```
이재문
```

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
   }
}
```

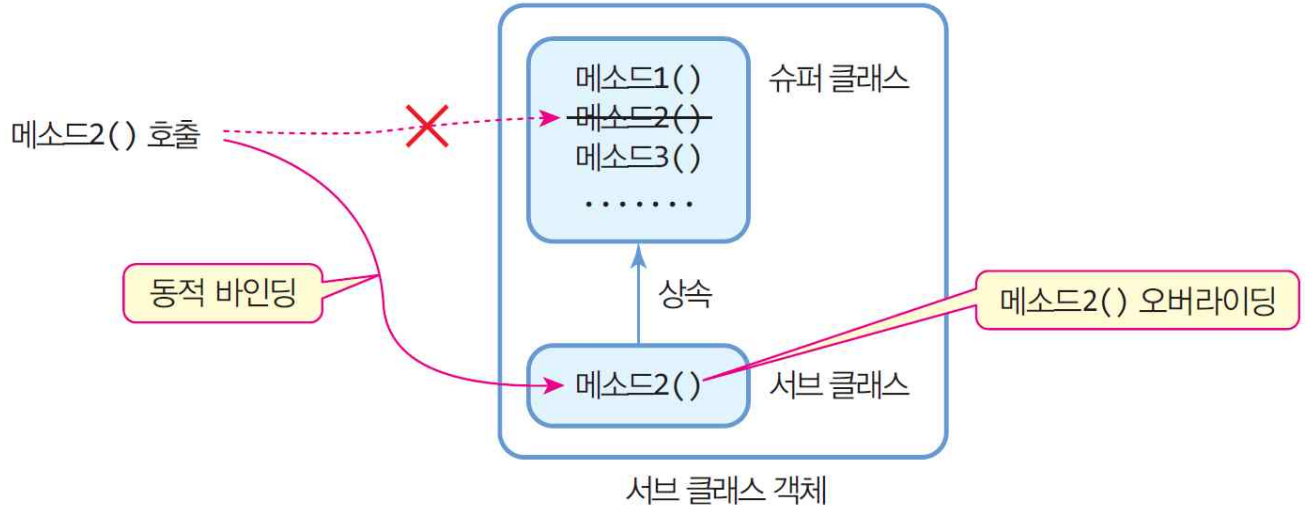


실행 결과

```
이재문
```

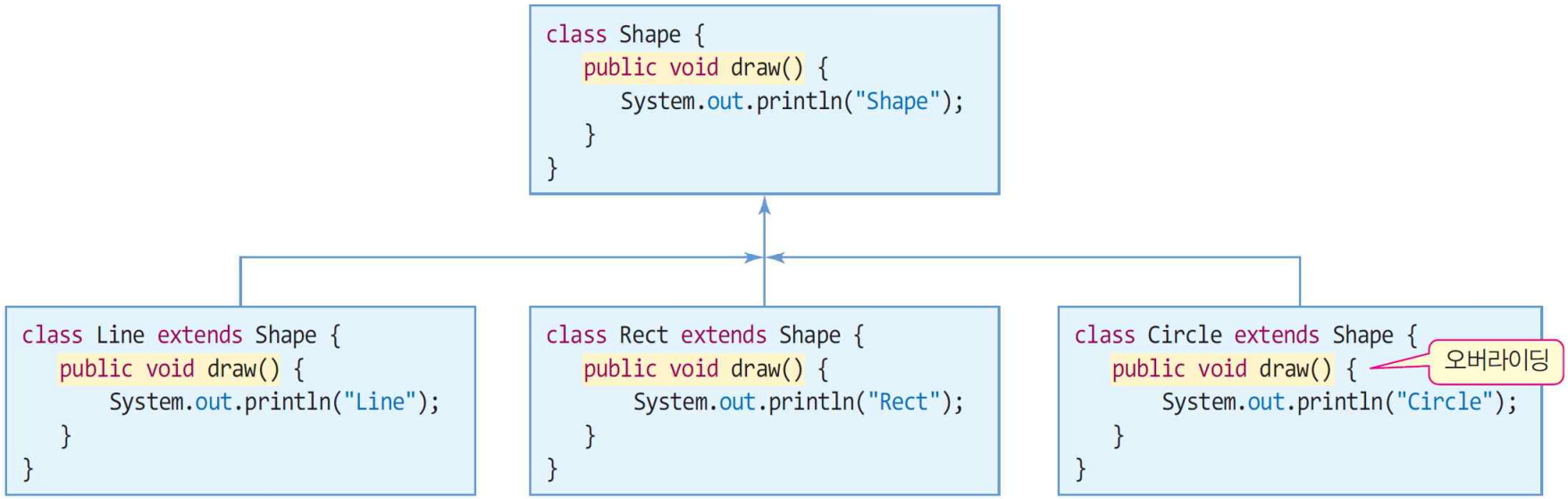
메소드 오버라이딩

- 메소드 오버라이딩(Method Overriding)
  - 슈퍼 클래스의 메소드를 서브 클래스에서 재정의
    - 슈퍼 클래스 메소드의 이름, 매개변수 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
  - 메소드 무시하기, 덮어쓰기로 번역되기도 함
  - 동적 바인딩 발생
    - 서브 클래스에 오버라이딩된 메소드가 무조건 실행되는 동적 바인딩



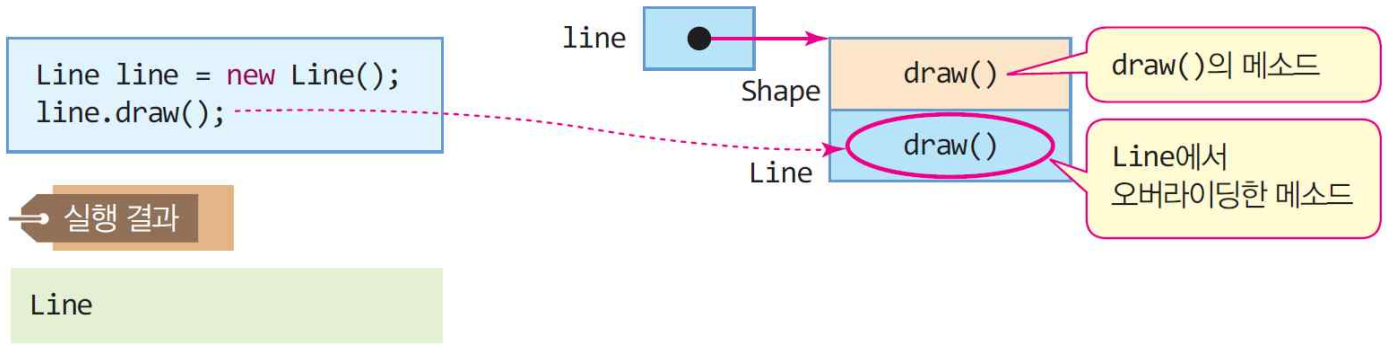
메소드 오버라이딩 사례

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

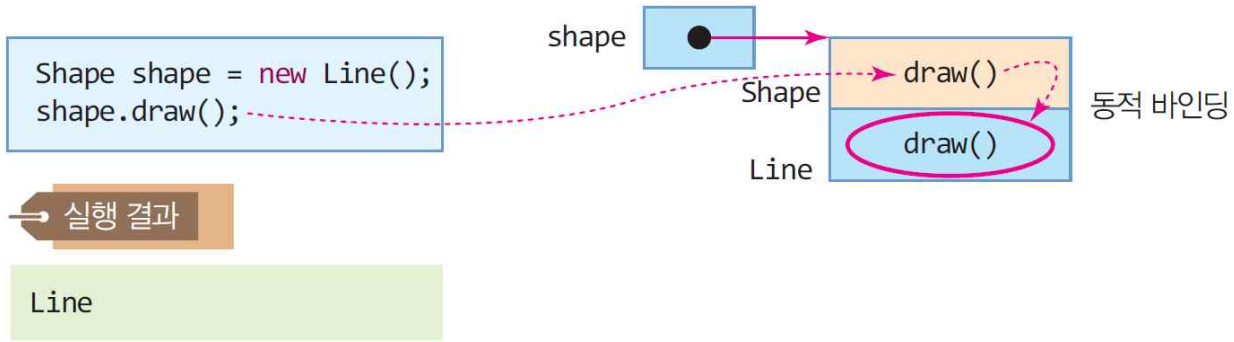


오버라이딩에 의해 서브 클래스의 메소드 호출

(1) 서브 클래스 레퍼런스로 오버라이딩된 메소드 호출



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



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

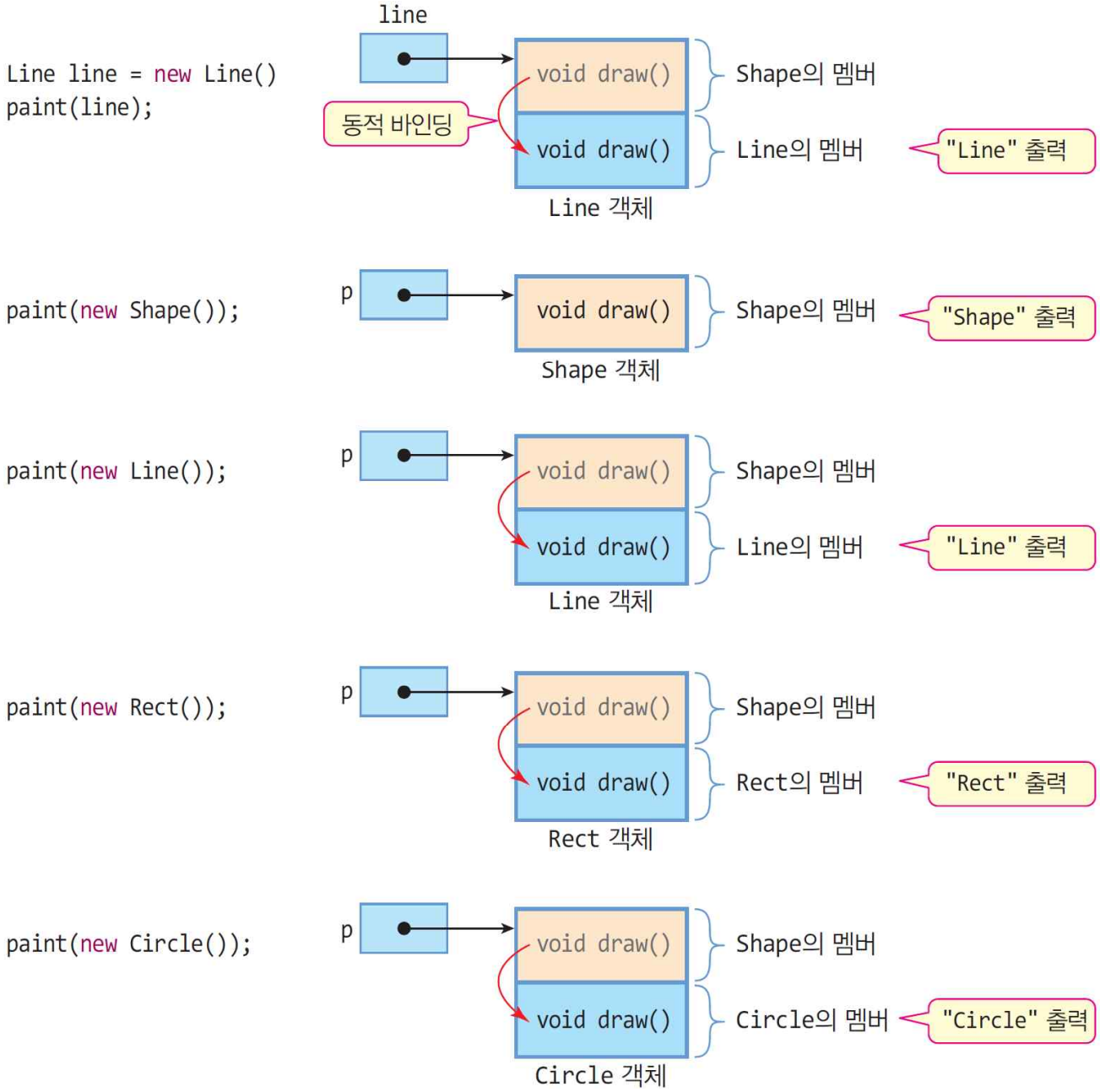
예제 6-2 : 메소드 오버라이딩으로 다형성 실현

```
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());
    }
}
```

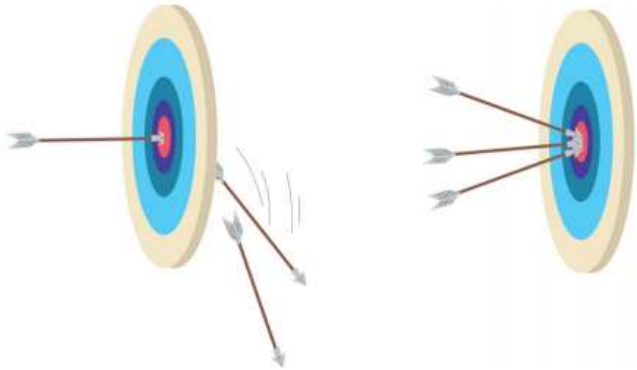
```
Line
Shape
Line
Rect
Circle
```

예제 실행 과정


Line line = new Line()
paint(line);
line
void draw()
void draw()
동적 바인딩
Shape의 멤버
Line의 멤버
"Line" 출력
Line 객체
paint(new Shape());
p
void draw()
Shape의 멤버
"Shape" 출력
Shape 객체
paint(new Line());
p
void draw()
void draw()
Shape의 멤버
Line의 멤버
"Line" 출력
Line 객체
paint(new Rect());
p
void draw()
void draw()
Shape의 멤버
Rect의 멤버
"Rect" 출력
Rect 객체
paint(new Circle());
p
void draw()
void draw()
Shape의 멤버
Circle의 멤버
"Circle" 출력
Circle 객체

메서드 오버라이딩

■ 메서드 오버라이딩과 메서드 오버로딩



(a) 메서드 오버라이딩 (b) 메서드 오버로딩

비교 요소	메서드 오버라이딩 Overriding	메서드 오버로딩 Overloading
메서드 이름	동일하다.	동일하다.
매개변수	동일하다.	다르다.
반환 타입	동일하다.	관계없다.
상속 관계	필요하다.	필요 없다.
예외와 접근 범위	제약이 있다.	제약이 없다.
바인딩	호출할 메서드를 실행 중 결정하는 동적 바인딩이다.	호출할 메서드를 컴파일할 때 결정하는 정적 바인딩이다.

상속
(소스 재사용과 확장 용이)

다형성

[quiz6_1_1_식별자]상속 (타입 변환,동적 바인딩)

다음과 같은 멤버를 가진 상속 관계의 클래스를 3개를 명시적인 생성자 없이 작성한 프로그램이다.

- Girl클래스는 String name만 멤버로 포함하고 각 클래스의 상속 관계는 Girl <- GoodGril <- BestGirl 이다.
- GoodGril클래스와 BestGirl클래스의 show() 메서드는 각각 '그녀는 자바를 잘 안다', '그녀는 자바를 무지하게 잘 안다'를 출력하는 실행문만 포함한다.

다음 main 소스에서 주석을 제거한 후 실행하면 오류가 발생하는 원인을 설명하시오.

[실행결과]
그녀는 자바를 무지하게 잘 안다.


```
package w4_111;


//3개의 클래스 작성

public class quiz6_1_1_111 {

		public static void main(String[] args) {
				Girl g1 = new Girl();
				Girl g2 = new GoodGirl();
				GoodGirl gg = new BestGirl();

				// g2.show();
				gg.show();   }

}
```

[quiz6_1_2_식별자]상속 (타입 변환,동적 바인딩)

Quiz6_1_1_식별자 문제에서 다음 조건을 추가 하시오.

- Girl클래스에만 생성자, private 필드, show()메서드을 추가 또는 수정한다.

  private String name ; void show(){ //'그녀는 자바 초보자이다' 출력 }; Gril(String name);

프로그램을 실행한 후 발생하는 오류의 원인을 찾아 수정 하시오.

[실행결과]
그녀는 자바를 잘 안다.
그녀는 자바를 무지하게 잘 안다.

```
package w4_111;


//3개의 클래스 작성



public class quiz6_1_2_111 {
		public static void main(String[] args) {
				Girl g1 = new Girl();
				Girl g2 = new GoodGirl();
				GoodGirl gg = new BestGirl();

				g2.show();
				gg.show();   }

}
```

[quiz6_1_3_식별자]상속 (타입 변환,동적 바인딩)

Quiz6_1_2_식별자 문제에서 다음 조건을 추가 하시오.

- Girl클래스에서 매개변수 한 개를 가진 생성자, protected 필드, show()메서드을 추가 또는 수정한다.

  protected String name ; void show(); Gril(String name);

- GoodGril클래스와 BestGirl클래스의 생성자를 super(name); 의 실행문을 가진 생성자를 정의한다.
- 모든 클래스의 show()메서드의 실행 결과가 '그녀는' 대신에 name 이 출력되도록 수정한다.

[실행결과]
갑순이는 자바 초보자이다.
콩쥐는 자바를 잘 안다.
황진이는 자바를 무지하게 잘 안다.

```
package w4_111;



//3개의 클래스 작성




public class quiz6_1_3_111 {
        public static void main(String[] args) {
   Girl[] girls = { new Girl("갑순이"), new GoodGirl("콩쥐"), new BestGirl("황진이") };
                for (Girl g : girls)
                        g.show();
    }
}
```

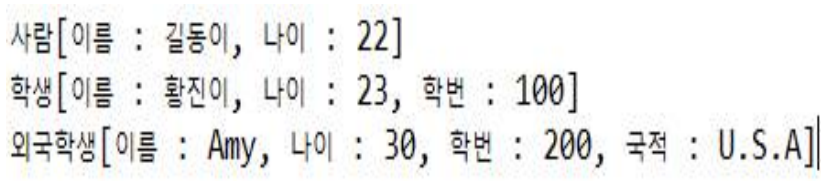
[quiz6_2_식별자]상속 프로그램을 작성하시오.

다음 표와 실행 결과를 참고해서 프로그램을 작성하시오. Show()메서드는 객체의 정보를 문자열로 반환한다.

- Person, Person의 자식 Student, Student의 자식 ForeignStudent클래스를 작성하시오.
- Person 타입 배열이 Person, Student, ForeignStudent 타입의 객체를 1개씩 포함하며, Person타입 배열 원소를 for ~ each 문을 사용해 각 원소의 정보를 실행 결과와 같이 출력하는 프로그램을 작성하시오.

	Person	Student	ForeignStudent
필드	이름, 나이	학번	국적
메서드	접근자와 생성자, show()		
생성자	모든 필드를 초기화하는 생성자		

[실행결과]
사람[이름 : 길동이, 나이 : 22]
학생[이름 : 황진이, 나이 : 23, 학번 : 100]
외국학생[이름 : Amy, 나이 : 30, 학번 : 200, 국적 : U.S.A]




```
package w4_111;

//3개의 클래스 작성

public class quiz6_2_111 {
		public static void main(String[] args) {

			// 객체 배열 이용
			//for ~ each 이용
	}
}
```