JAVA Programming

JAVA 기초

상속(Inheritance)
- 상속의 개념

학습목표

학/습/내/용

1. 상속의 개요
2. 클래스 상속(extends)

학/습/목/표

1. 상속의 개요에 대하여 학습한다.
2. 클래스 상속(extends)에 대하여 학습한다.

상속 (inheritance)의 개요

- 객체 지향의 상속
  - 부모클래스(슈퍼클래스)에 만들어진 필드, 메소드를 자식클래스(서브클래스)가 물려받음
  - 부모의 생물학적 특성을 물려받는 유전과 유사

유산 상속

유전적 상속 : 객체 지향 상속

- 상속을 통해 간결한 자식 클래스 작성
  - 동일한 특성(멤버변수와 멤버메소드)을 재정의할 필요가 없어 자식 클래스가 간결해짐

상속 (inheritance)의 개요

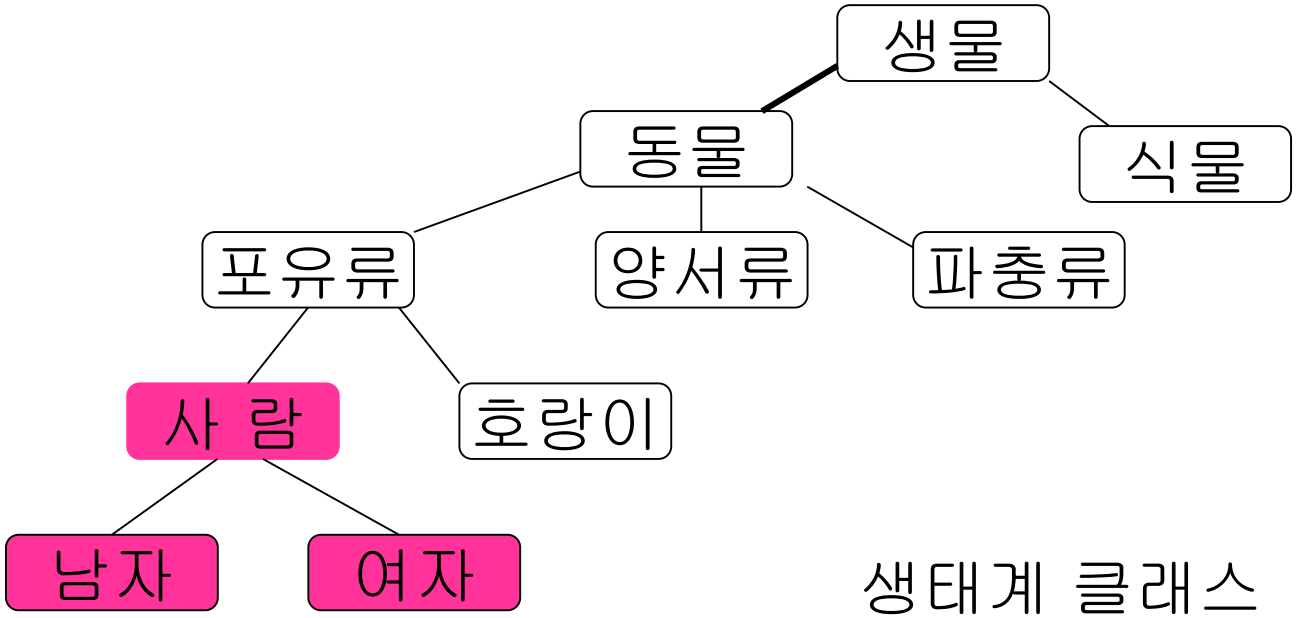
- 클래스 및 클래스 계통 개념
  - JAVA는 사용자 정의 타입(User Defined Type)을 만들 수 있는 클래스 개념 및 클래스 계통 개념을 가지고 있음.
  - 프로그래머가 구상한 개념과 응용 프로그램에서 쓰일 개념을 모형화(추상화) 하는데 클래스(class)를 사용함.
- JAVA 에서 제공하는 상속성의 개념은 클래스들 사이의 계통관계를 표현하기 위한 장치
  - 계통관계는 곧 클래스들 사이의 공통 특성임.
    - (예) 원과 삼각형 사이에는 바로 "도형이다"라는 공통점이 존재.
      - 도형이란 개념을 공유하고 있다.
      - 원과 삼각형 개념을 프로그램에 넣는다고 할 때 이 '도형' 이란 개념을 함께 포함시켜야만 제대로 표현이 가능하다.

❖ 상속성은 클래스들 사이의 공통점에 의한 관계를 표현하는 방법이다.

상속 (inheritance)의 개요

- 상속성(Inheritance)
  - 하나의 객체가 다른 객체의 특성을 이어받을 수 있게 해 주는 과정
  - 자식 클래스는 부모클래스의 모든 특성을 상속 받고, 자기 자신의 특성을 추가시켜 정의 할 수 있다.

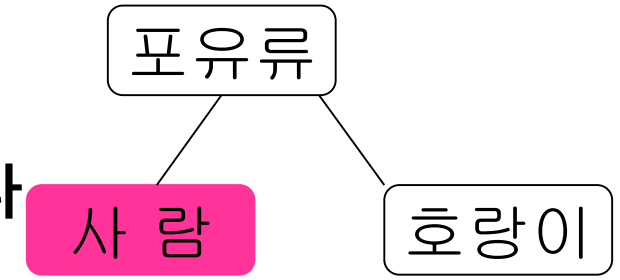
➢ 부모 클래스에 있는 멤버들은 상속 받아쓰고 부모클래스에 없는 멤버들만 자식클래스에서 새롭게 정의한다.



생태계 클래스

상속 (inheritance)의 개요

- 상속성
  - 하나의 클래스가 다른 클래스의 특성을 이어받는 것
- 용어
  - 수퍼(부모) 클래스(Super Class)
    - 일반(공통)적인 모든 성질을 정의한다
  - 서브(자식) 클래스(Sub Class)
    - 슈퍼 클래스의 특성을 이어 받는다
    - 서브 클래스에서 지정하고자 하는 특성을 추가한다



❖ 코드의 중복 작업 최소화

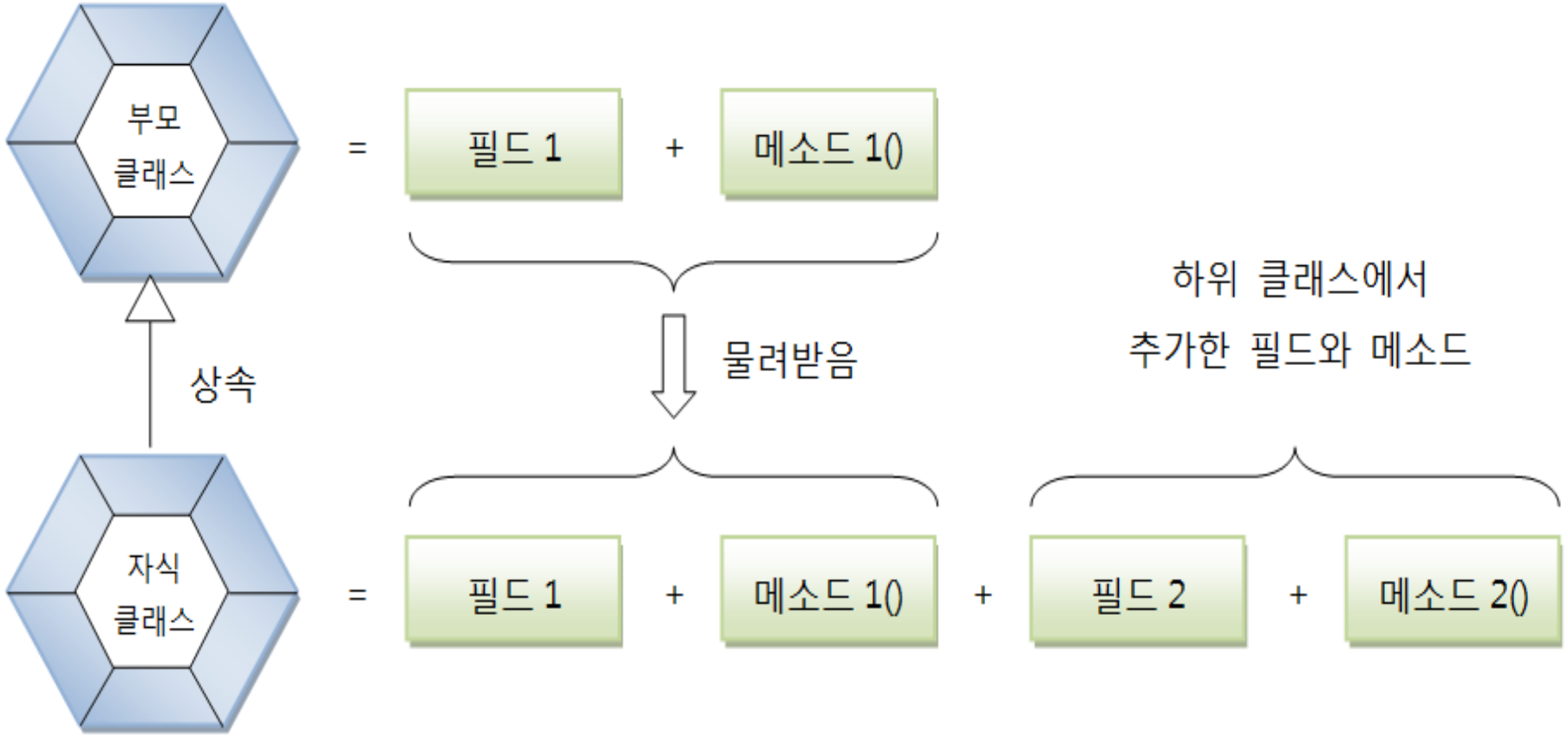
❖ 소프트웨어의 재사용성 증대

상속 (inheritance)의 개요

- 상속(Inheritance)이란?
  - 현실 세계:
    - 부모가 자식에게 물려주는 행위
    - 부모가 자식을 선택해서 물려줌
  - 객체 지향 프로그램:
    - 자식(서브, 하위) 클래스가 부모(슈퍼, 상위) 클래스의 멤버를 물려받는 것
    - 자식이 부모를 선택해 물려받음
    - 상속 대상: 부모의 필드(멤버변수와 멤버상수)와 메소드

상속 (inheritance)의 개요

- 상속(Inheritance)이란?



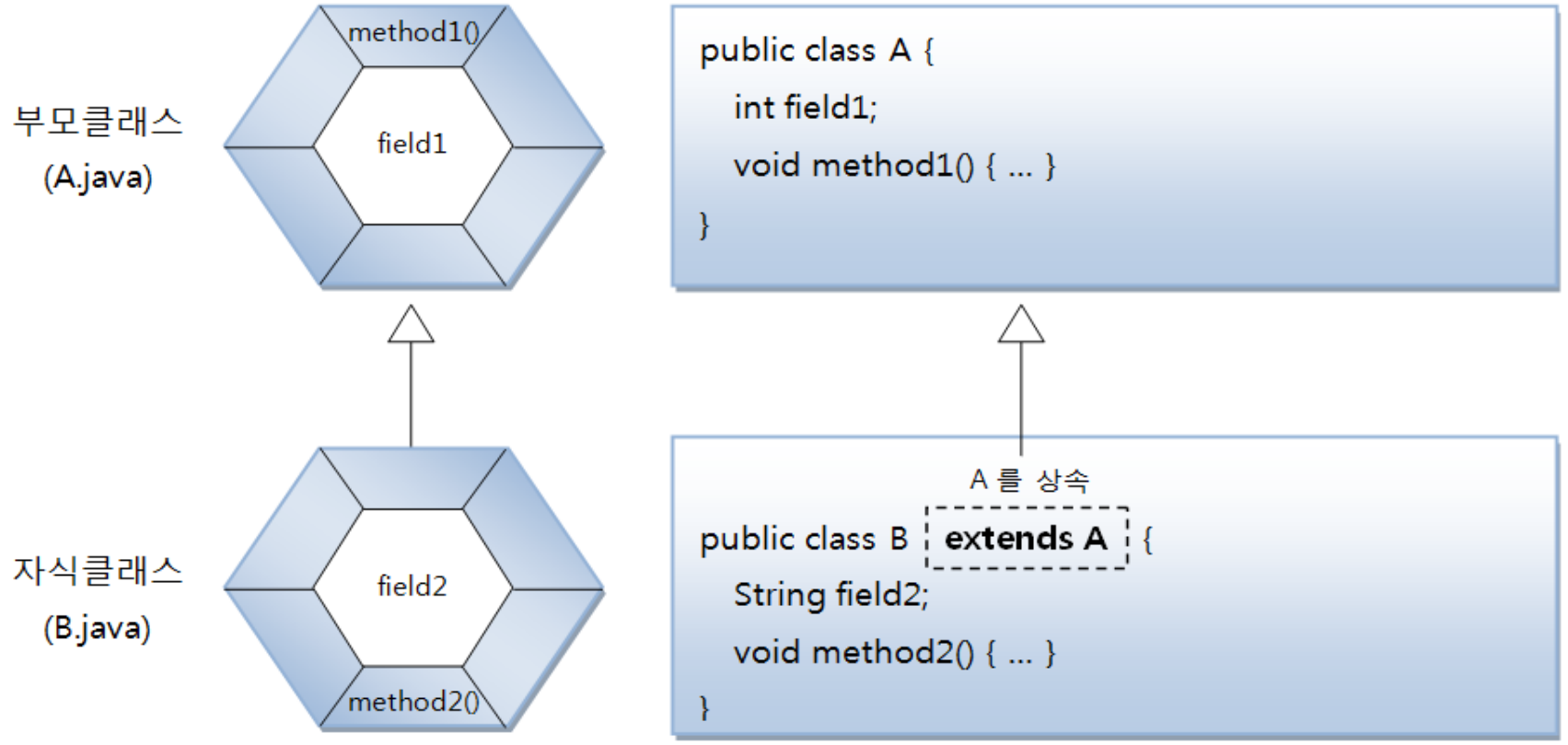
상속 (inheritance)의 개요

- 상속(Inheritance) 개념의 활용
  - 상속의 효과
    - 부모 클래스의 멤버들을 재사용함으로써 자식 클래스를 빨리 개발 가능
    - 반복된 코드의 중복을 줄임
    - 유지 보수하는데 편리성 제공
    - 객체지향에서 다형성(Polymorphism)을 구현 가능케 함.
  - 상속 대상 제한
    - 부모 클래스의 private 접근 권한을 갖는 필드와 메소드 제외
    - 부모 클래스가 다른 패키지에 있을 경우, 디폴트 접근 권한을 갖는 필드와 메소드도 제외

클래스 상속(extends)

▪ extends 키워드

– 자식 클래스가 상속받을 부모 클래스를 지정하는 키워드

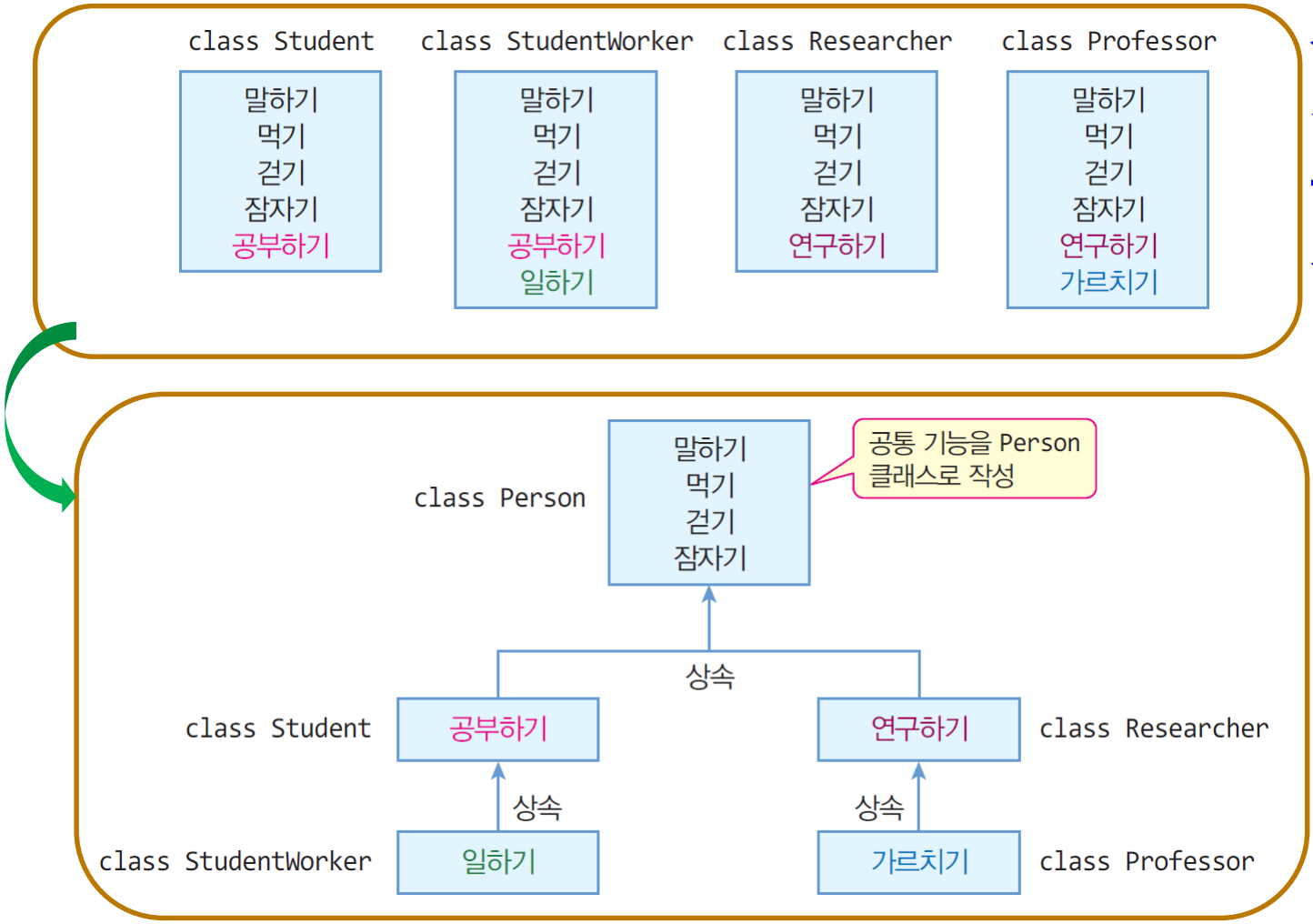


클래스 상속(extends)

✓ 자바는 단일 상속만 가능 - 부모 클래스 나열 불가

```
class 자식클래스 extends 부모클래스1, 부모클래스2 {
}
```

클래스 상속(extends)



상속이 없는 경우 중복된 멤버를 가진 4 개의 클래스

상속을 이용한 경우 중복이 제거되고 간결해진 클래스 구조

클래스 상속(extends)

- 객체 지향프로그래밍에서 상속의 장점
  - 클래스의 간결화
    - 멤버의 중복 작성이 불필요
  - 클래스 관리의 용이성
    - 클래스들의 계층적 분류
  - 소프트웨어의 생산성 향상
    - 클래스의 재사용과 확장 용이
    - 새로운 자식 클래스의 작성 속도가 빠름

클래스 상속(extends)

- 클래스 상속과 객체
  - 자바의 상속 선언

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

  - 부모 클래스 -> 슈퍼 클래스(super class) | 상위 | 기본 클래스로도 부름
  - 자식 클래스 -> 서브 클래스(sub class) | 하위 | 파생 클래스로도 부름
  - extends 키워드 사용
    - 슈퍼 클래스로부터 확장된다는 개념

클래스 상속(extends)

예제 1 : 클래스 상속 만들기 – Point와 ColorPoint 클래스(1)

(x, y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 색을 가진 점을 표현하는 ColorPoint 클래스를 만들고 활용해보자.(1)

```
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 + ")");
  }
}
```

클래스 상속(extends)

예제 1 : 클래스 상속 만들기 – Point와 ColorPoint 클래스(2)

(x, y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 색을 가진 점을 표현하는 ColorPoint 클래스를 만들고 활용해보자.(2)

```
// 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
  }
}
```

클래스 상속(extends)

예제 1 : 클래스 상속 만들기 – Point와 ColorPoint 클래스(3)

(x, y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 색을 가진 점을 표현하는 ColorPoint 클래스를 만들고 활용해보자.(3)

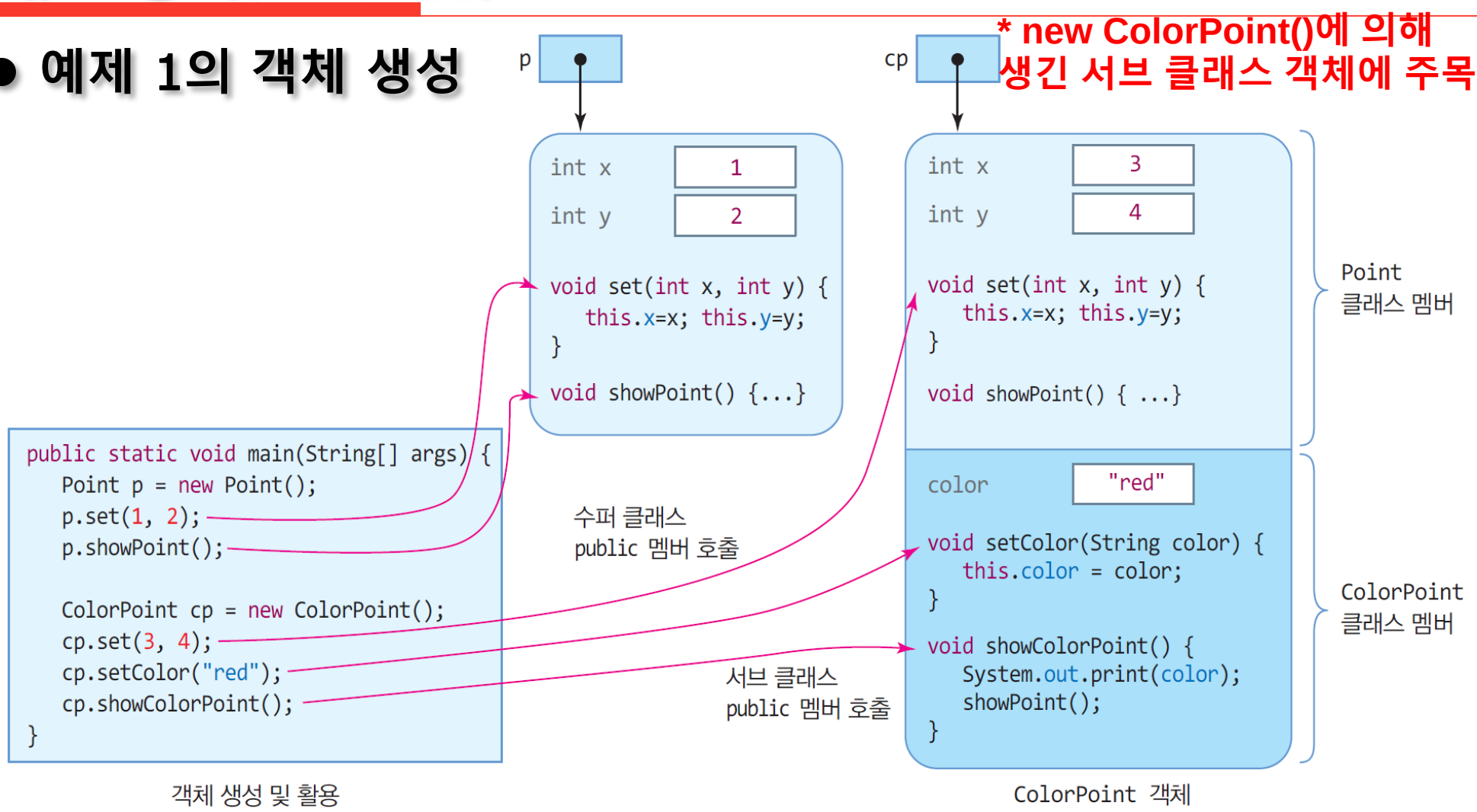
```
public class ColorPointEx {
        public static void main(String [] args) {
                Point p = new Point(); // Point 객체 생성
                p.set(1, 2); // Point 클래스의 set() 호출
                p.showPoint();

                ColorPoint  cp = new ColorPoint(); // ColorPoint 객체
                cp.set(3, 4); // Point의 set() 호출
                cp.setColor("red"); // ColorPoint의 setColor() 호출
                cp.showColorPoint(); // 컬러와 좌표 출력
        }
}
```

실행결과

```
(1,2)
red(3,4)
```

클래스 상속(extends)

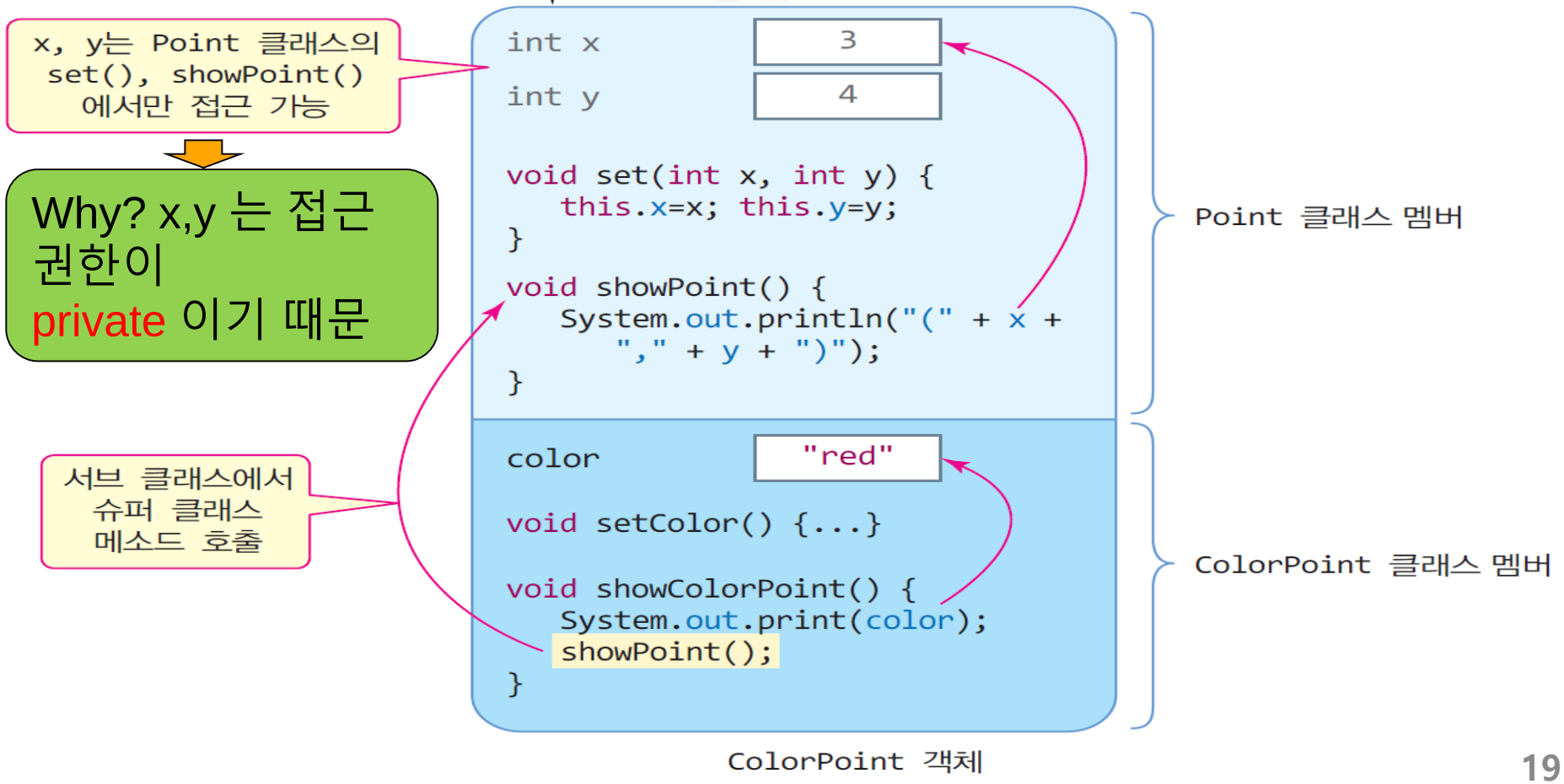
● 예제 1의 객체 생성



클래스 상속(extends)

- 예제 1의 객체 생성 cp

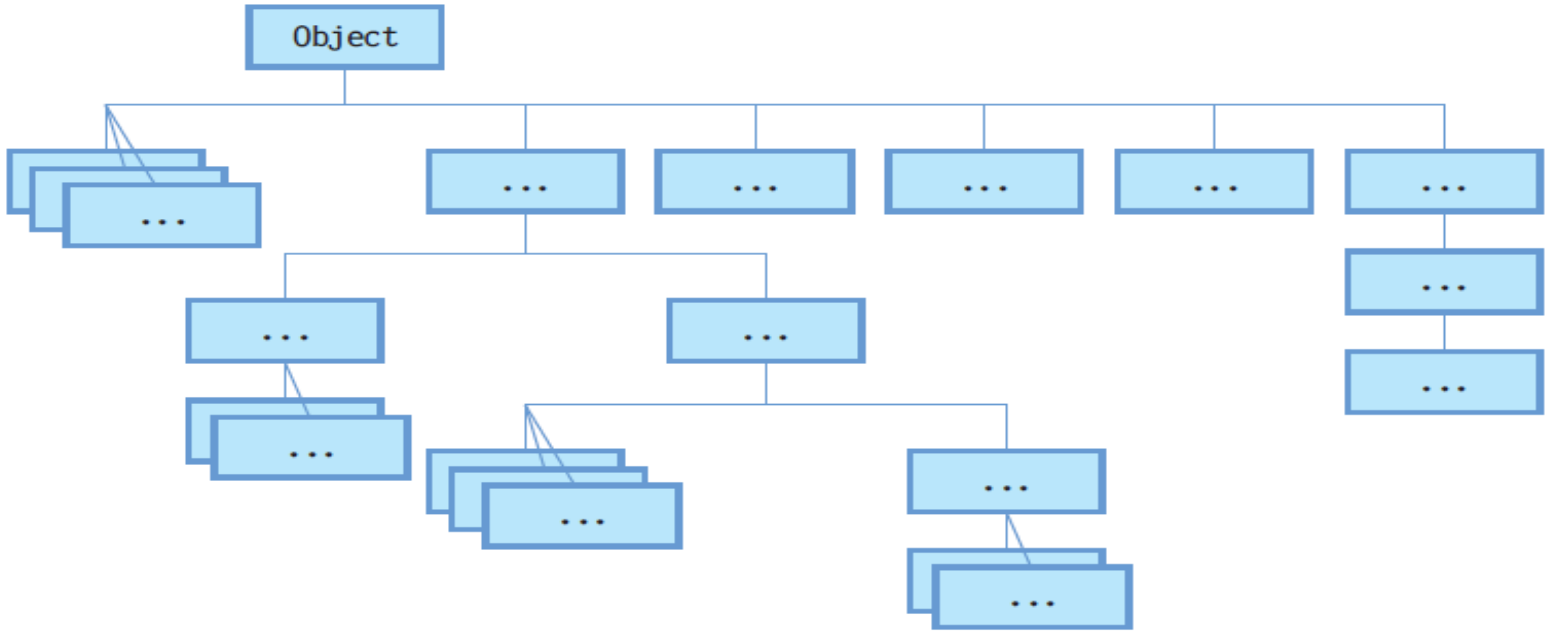
✓ 서브클래스에서 슈퍼 클래스의 멤버 접근



ColorPoint 객체

클래스 상속(extends)

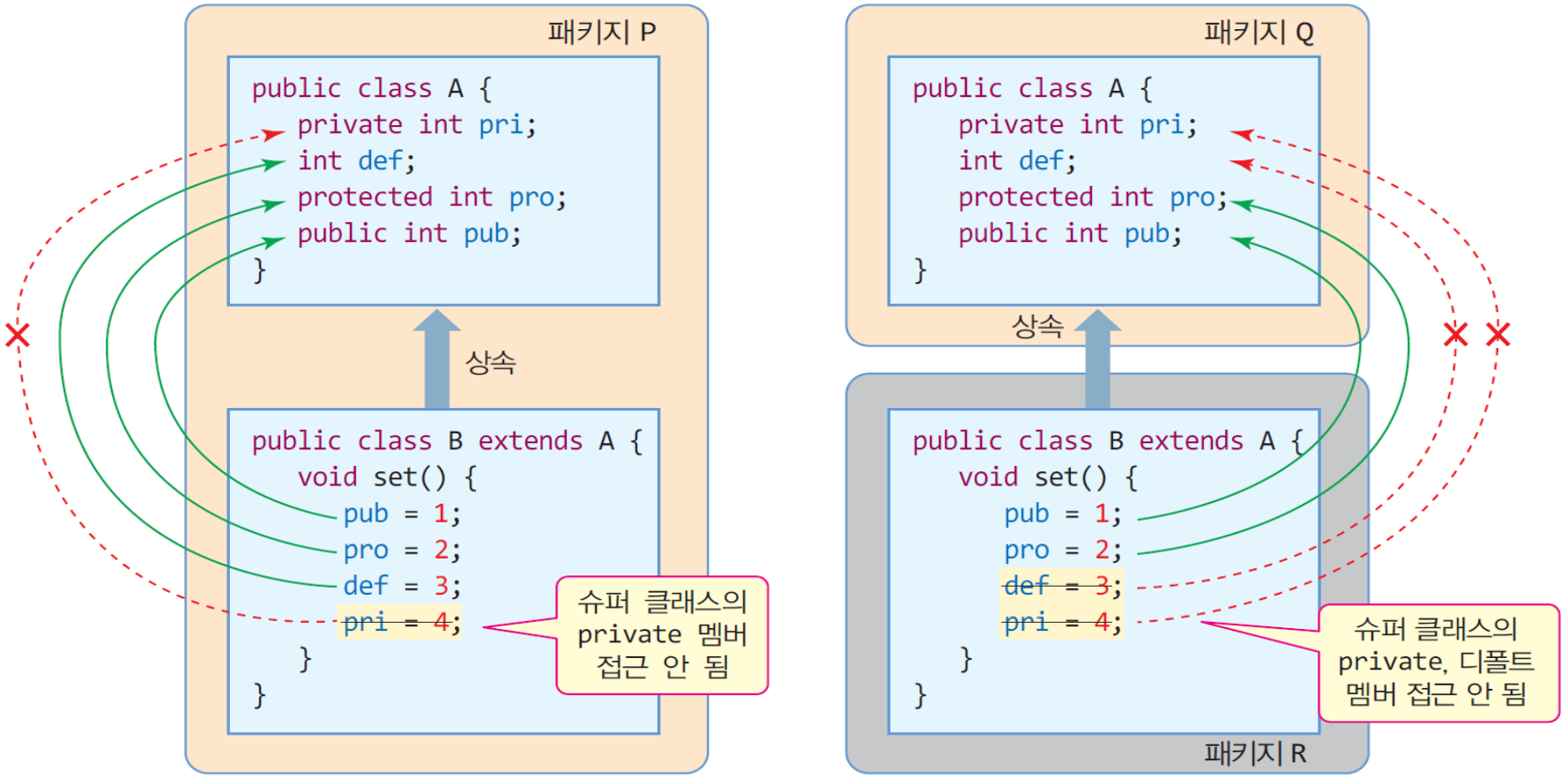
- 자바 상속의 특징
  - 클래스간의 다중 상속을 지원하지 않으며 상속 횟수는 무제한
  - 상속의 최상위 조상 클래스는 java.lang.Object 클래스
    - 모든 클래스는 자동으로 java.lang.Object를 상속받음
    - 자바 컴파일러에 의해 자동으로 이루어짐



상속과 접근권한 지정자

- 자바의 접근 지정자 4 가지
  - 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

상속과 접근권한 지정자



(a) 슈퍼 클래스와 서브 클래스가 동일한 패키지에 있는 경우

(b) 슈퍼 클래스와 서브 클래스가 서로 다른 패키지에 있는 경우

예제 2: 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 getXXX, setXXX 메소드를 통해서만 조작이 가능하다.

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

```
class Person {
    private int weight;
    int age;
    protected int height;
    public String name;

    public void setWeight(int weight) {
        this.weight = weight;
    }
    public int getWeight() {
        return weight;
    }
}
```

```
class Student extends Person {
    public void set() {
        age = 30; // 슈퍼 클래스의 디폴트 멤버 접근 가능
        name = "홍길동"; // 슈퍼 클래스의 public 멤버 접근 가능
        height = 175; // 슈퍼 클래스의 protected 멤버 접근 가능
        // weight = 99; // 오류. 슈퍼 클래스의 private 접근 불가
        setWeight(99); // private 멤버 weight은 setWeight()으로 간접 접근
    }
}
```

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

서브 클래스/슈퍼 클래스의 생성자 호출 및 실행

질문 1 서브 클래스 객체가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자가 모두 실행되는가? 아니면 서브 클래스의 생성자만 실행되는가?

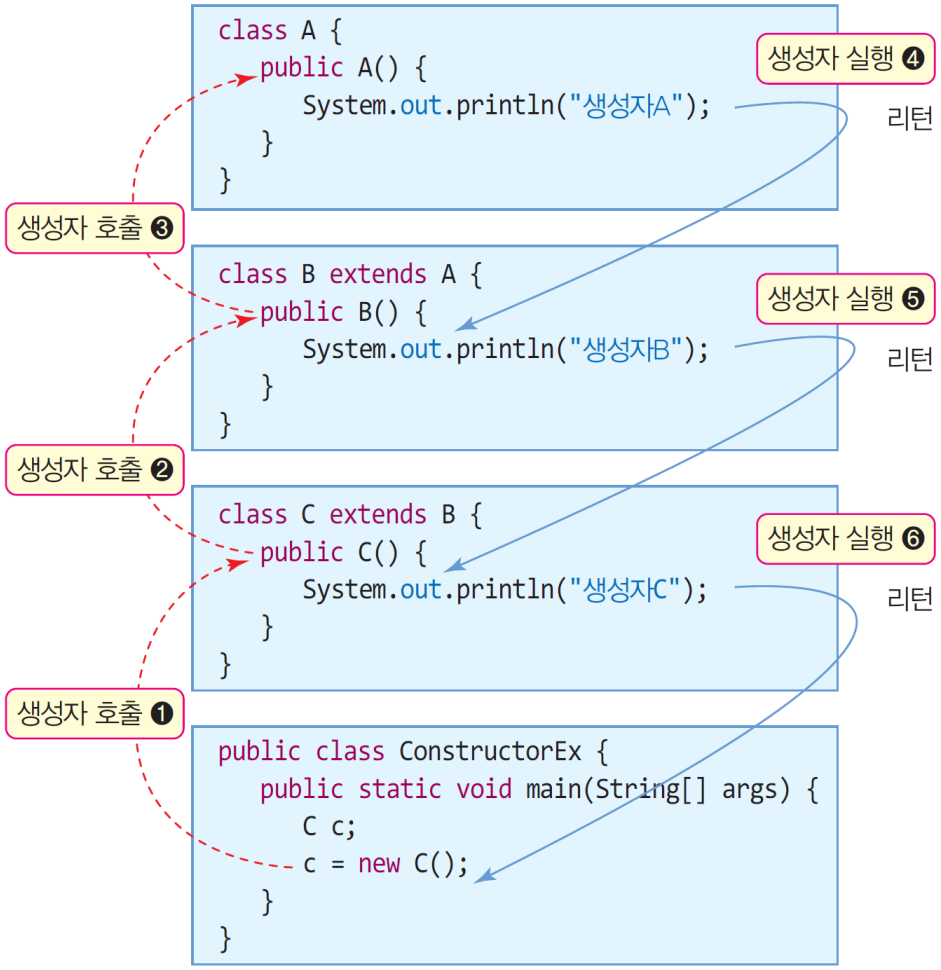
답 둘 다 실행된다. 서브 클래스의 객체가 생성되면 이 객체 속에 서브 클래스와 멤버와 슈퍼 클래스의 멤버가 모두 들어 있다. 생성자의 목적은 객체 초기화에 있으므로, 서브 클래스의 생성자는 생성된 객체 속에 들어 있는 서브 클래스의 멤버 초기화나 필요한 초기화를 수행하고, 슈퍼 클래스의 생성자는 생성된 객체 속에 있는 슈퍼 클래스의 멤버 초기화나 필요한 초기화를 각각 수행한다.

질문 1 서브 클래스의 생성자와 슈퍼 클래스의 생성자 중 누가 먼저 실행되는가?

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

- 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



실행 결과

```
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
  - 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

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) {
        .....
    }
}
```

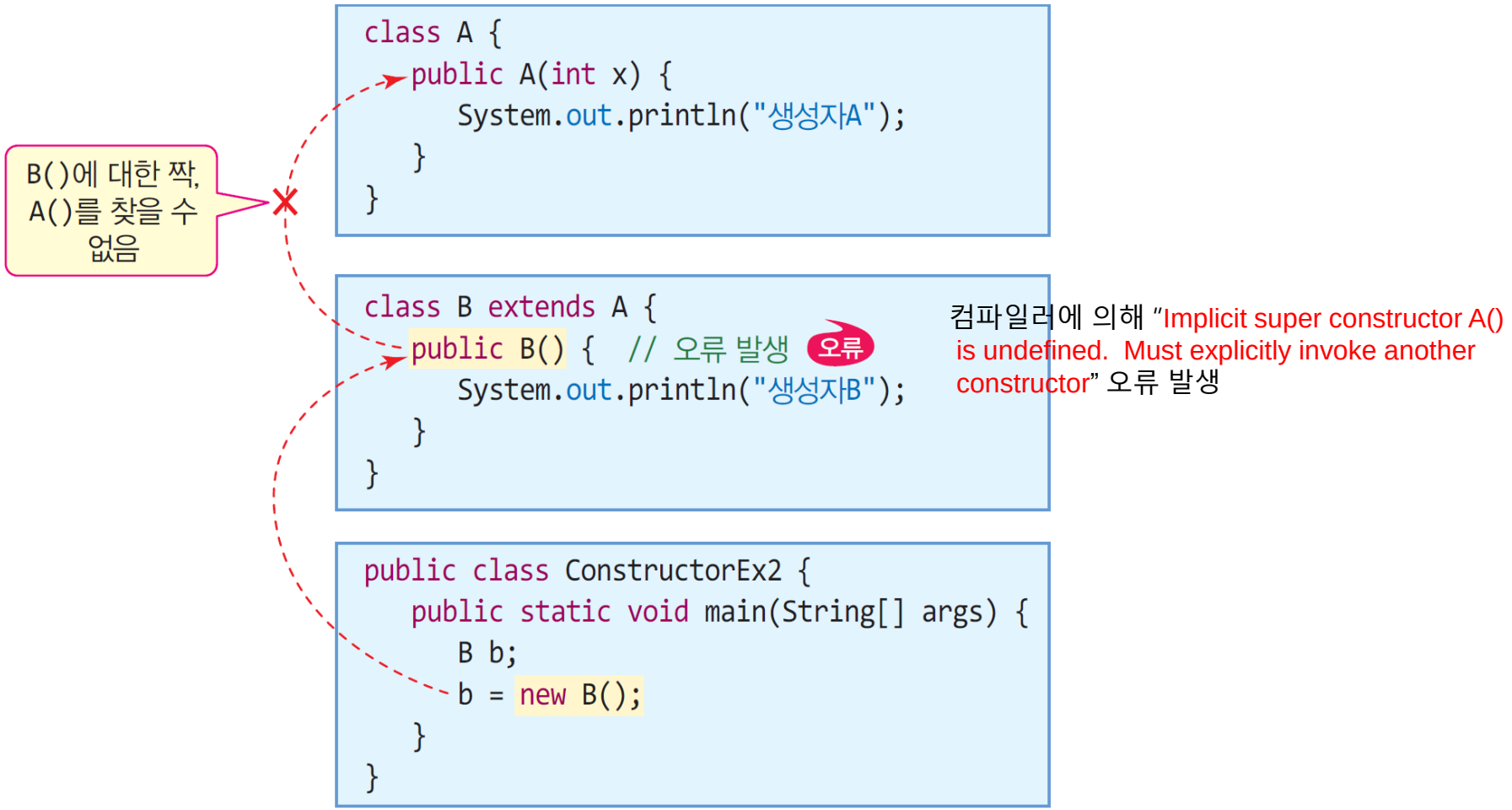
```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
}
```

```
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



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

```
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 선택강

- 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


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

x에 5 전달

```java
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

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

실행 결과

```
매개변수생성자A5
매개변수생성자B5
```

예제 5-3 : super()를 활용한 ColorPoint 작성

super()를 이용하여 ColorPoint 클래스의 생성자에서 슈퍼 클래스 Point의 생성자를 호출하는 예를 보인다.

```
class Point {
    private int x, y; // 한 점을 구성하는 x, y 좌표
    public Point() {
        this.x = this.y = 0;
    }
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public void showPoint() { // 점의 좌표 출력
        System.out.println("(" + x + "," + y + ")");
    }
}

class ColorPoint extends Point {
    private String color; // 점의 색
    public ColorPoint(int x, int y, String color) {
        super(x, y); // Point의 생성자 Point(x, y) 호출
        this.color = color;
    }
    public void showColorPoint() { // 컬러 점의 좌표 출력
        System.out.print(color);
        showPoint(); // Point 클래스의 showPoint() 호출
    }
}
```

x=5, y=6 전달

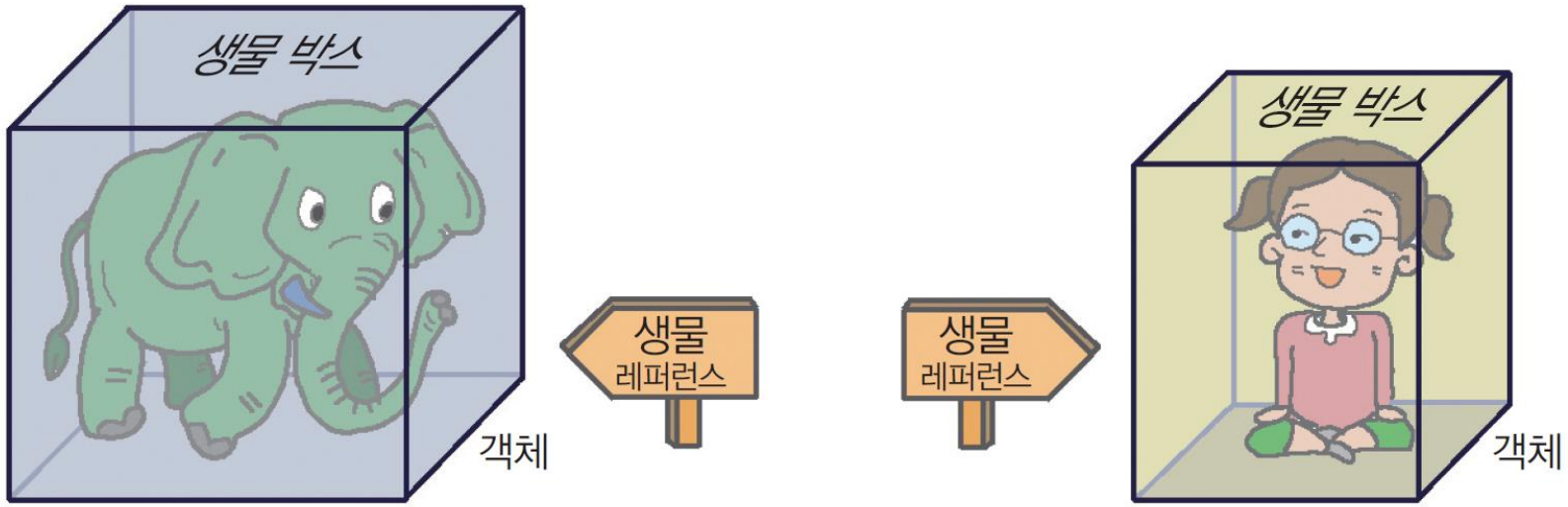
```
public class SuperEx {
    public static void main(String[] args) {
        ColorPoint cp = new ColorPoint(5, 6, "blue");
        cp.showColorPoint();
    }
}
```

x=5, y=6,
color = "blue" 전달

```
blue(5,6)
```

사람은 생물이다.

생물을 넣는 박스에 코끼리나 사람을 넣고 박스 앞에 생물을 가리키는 팻말을
사용해도 무방하다. 왜냐하면, **사람은 생물을 상속 받았기 때문이다.**



- **(Quiz) 다음 중 임베디드 시스템의 응용분야와 가장 거리가 먼 것은 무엇인가?**

① 세탁기, 오디오와 같은 정보 가전

② 공장자동화와 같은 제어분야

③ 항공/군용 분야

④ 핸드폰과 같은 정보 단말 분야

⑤ 워드 편집 프로그램

핵심정리

- 임베디드 시스템의 정의
  - 하드웨어와 소프트웨어가 조합되어 특정한 목적을 수행하는 시스템
  - 특정한 기능을 수행하도록 마이크로 프로세서와 입출력 장치를 내장
  - 이를 제어하기 위한 프로그램이 내장되어 있는 우리의 일상 생활에서 사용되는 각종 전자기기, 가전제품, 제어장치, 스마트폰, 스마트패드 등
- 임베디드 시스템의 응용분야
  - 정보 가전 / 제어 분야 / 정보 단말 / 게임기기 / 항공과 군용 / 물류와 금융 / 차량과 교통 / 사무와 의료