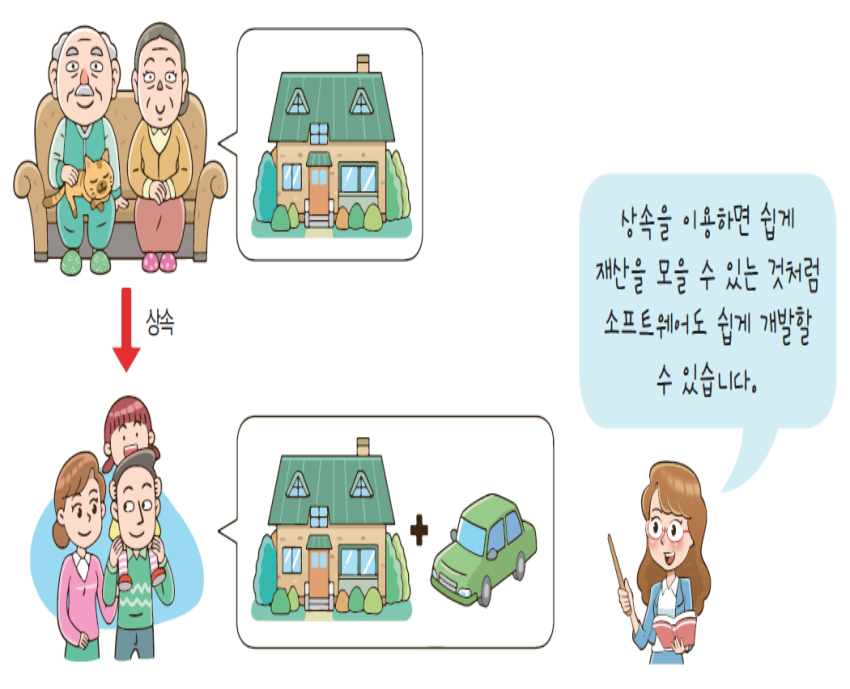
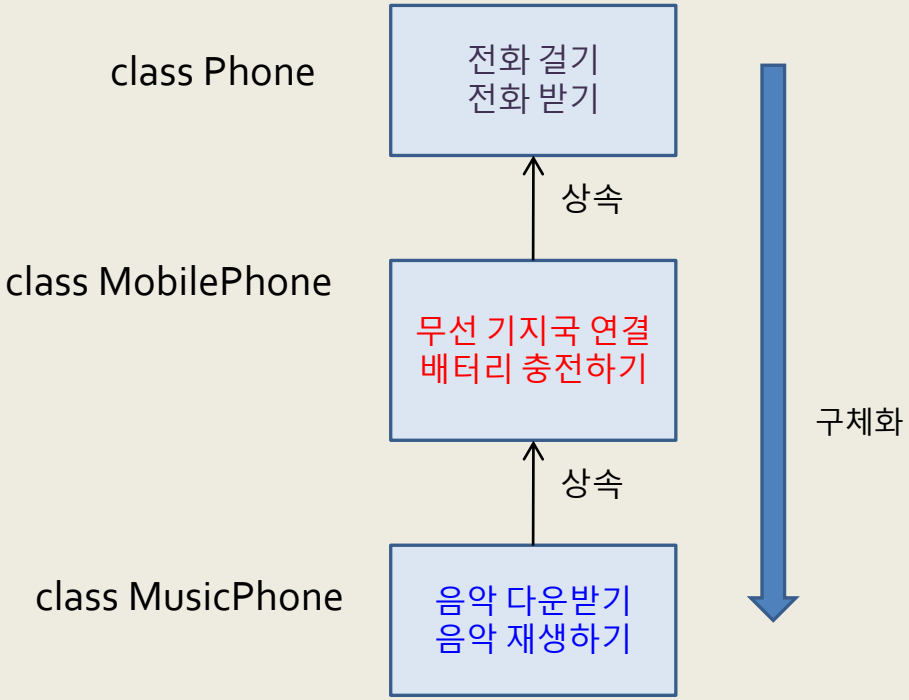
제 8 장 상속 (Inheritance) Part-1

상속(inheritance)의 개념

- 상속
  - 상위 클래스의 특성 (필드, 메소드)을 하위 클래스에 물려주는 것
  - 수퍼클래스 (superclass)
    - 특성을 물려주는 상위 클래스
  - 서브클래스 (subclass)
    - 특성을 물려 받는 하위 클래스
    - 서브 클래스에 자신만의 특성(필드, 메소드) 추가
    - 수퍼 클래스의 동적특성(메소드)을 재정의 : 오버라이딩
- 수퍼클래스는 하위 클래스들의 공통적 특성을 가지므로 고수준 추상화를 나타내게 되며, 수퍼 클래스에서 하위 클래스로 갈 수록 구체적
  - 예) 폰 -> 모바일폰 -> 뮤직폰
  - 예) 계산기 -> 공학용계산기-> 알람공학용계산기
- 상속을 통해 서브 클래스의 간결한 클래스 정의
  - 동일한 특성을 재정의할 필요가 없어 클래스 정의가 간결해짐

상속 관계 예


상속
상속을 이용하면 쉽게
재산을 모을 수 있는 것처럼
소프트웨어도 쉽게 개발할
수 있습니다.


class Phone
전화 걸기
전화 받기
상속
class MobilePhone
무선 기지국 연결
배터리 충전하기
상속
class MusicPhone
음악 다운받기
음악 재생하기
구체화

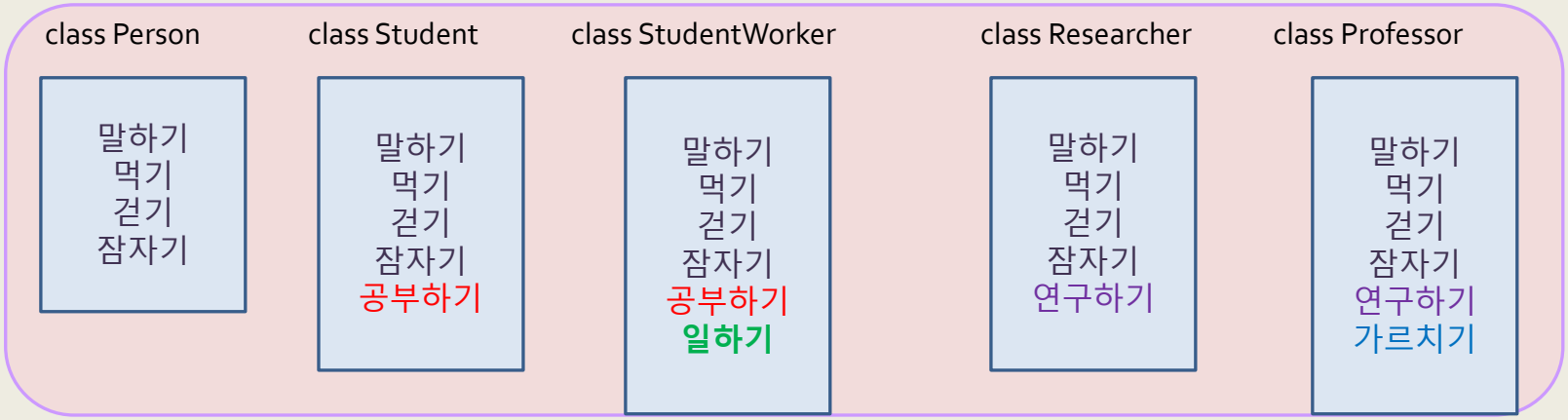
상속의 장점

- 상속의 장점
  - 기존 클래스의 필드와 메소드를 재사용(중복사용방지)
  - 기존 클래스의 동적 특성에 대한 변경도 가능(오버라이딩)
  - 상속은 이미 작성된 검증된 소프트웨어를 재사용
  - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수
  - 코드의 중복을 줄일 수 있다.
- 수퍼(부모) 클래스는 추상적이고 서브(자식) 클래스는 구체적이다

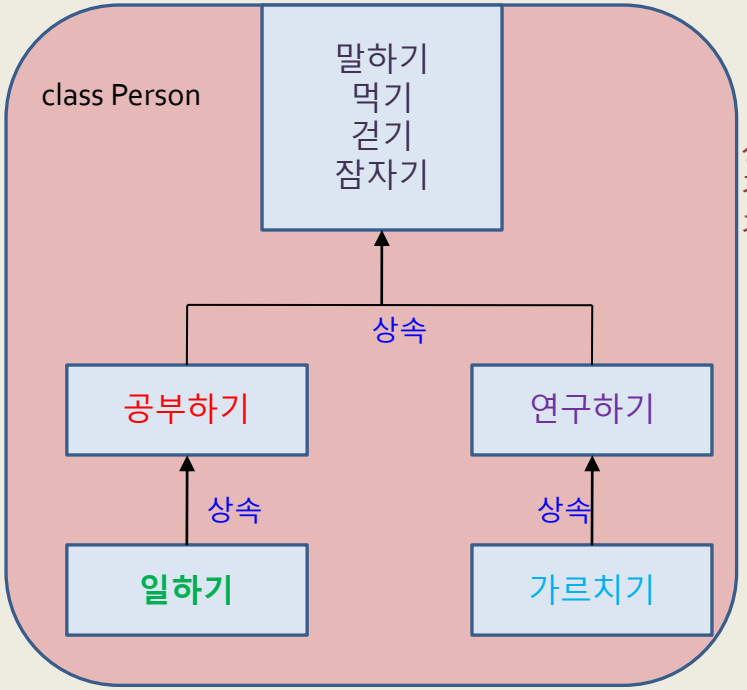
부모 클래스	자식 클래스
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악자전거), RoadBike, TandemBike
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motorcycle(오토바이), Bicycle(자전거)
Student(학생)	GraduateStudent(대학원생), UnderGraduate(학부생)
Employee(직원)	Manager(관리자)
Shape(도형)	Rectangle(사각형), Triangle(삼각형), Circle(원)

상속의 필요성

상속이 없는 경우 중복된 멤버를 가진 5개의 클래스

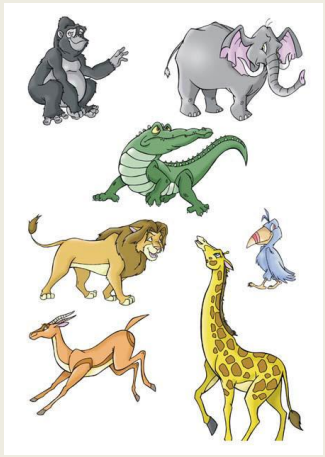


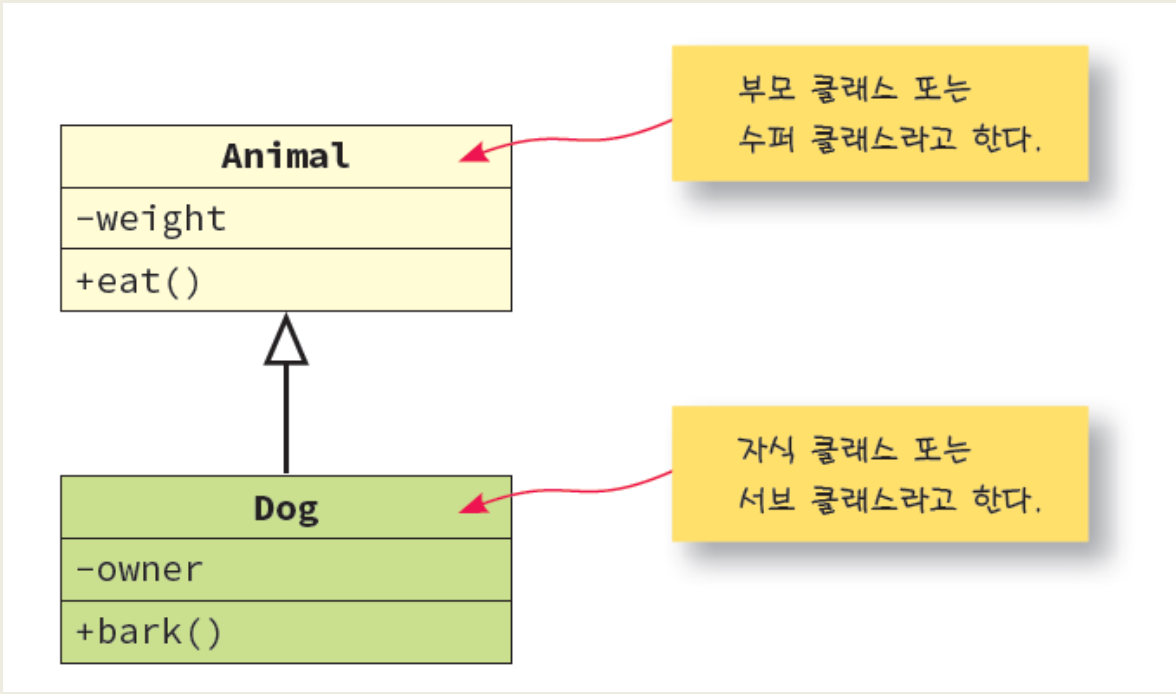
- 클래스 사이의 멤버 중복 선언 방지
- 필드와 메소드 재사용으로 클래스 간결화
- 클래스 간 계층적 분류 및 관리



상속을 이용한
경우 중복이 제거되고
간결해진 클래스 구조

UML로 표현한 상속관계

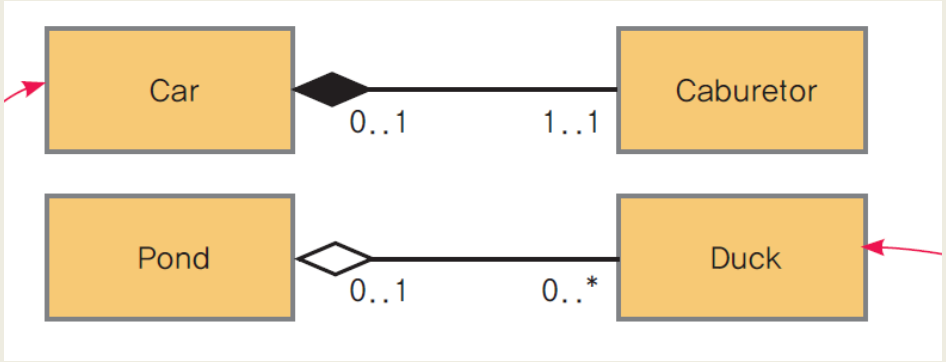




- is-a 관계: "~은 ~이다"와 같은 관계
- **상속은 is-a 관계이다**.
  - 자동차는 탈것이다(Car is a Vehicle).
  - 강아지는 동물이다(Dog is an animal).

HAS-A 관계

- has-a 관계: ~은 ~을 가지고 있다"와 같은 관계
  - 도서관은 책을 가지고 있다(Library has a book).
  - 거실은 소파를 가지고 있다(Living room has a sofa).
- 객체 지향 프로그래밍에서 has-a 관계는 구성 관계(composition) 또는 집합 관계(aggregation)를 나타낸다.




```
class Vehicle { }
class Carburetor { }
public class Car extends Vehicle{
  private Carburetor cb;
}
```

Has-a 관계와 is-a 관계를 모두 가지고 있는 경우

클래스 상속의 형식

- 상속 선언

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

- 자바 상속의 특징
  - **다중 상속 지원하지 않는다**
    - 다수 개의 클래스를 상속받지 못한다.
  - **상속의 횟수에는 제한을 두지 않는다**
  - **슈퍼 클래스의 private 멤버(혹은 final 멤버)는 상속되지 않음**
  - 계층구조의 최상위에 있는 클래스는 java.lang.Object 클래스이다.
    - 모든 클래스는 java.lang.Object의 하위 클래스이다

예제 : 클래스 상속 만들어 보기

(x,y)의 한 점을 표현하는 Point 클래스와 이를 상속받아 컬러 점을 표현하는 ColorPoint 클래스

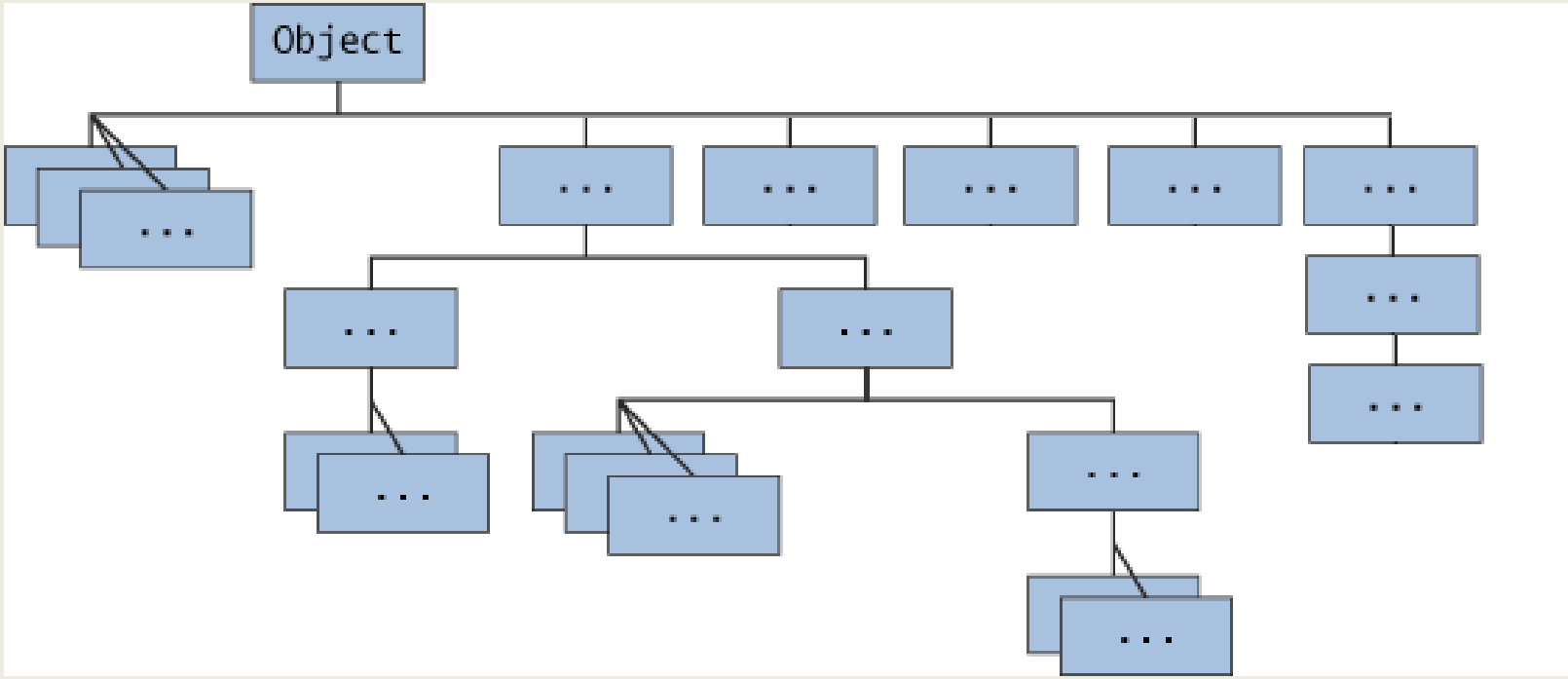
```
class Point {
   int x, y;          // 한 점을 구성하는 x, y 좌표
   void set(int x, int y) {
      this.x = x; this.y = y;
   }
   void showPoint() {   // 점의 좌표 출력
      System.out.println("(" + x + "," + y + ")");
   }
}
```

```
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 java.lang.Object 클래스의 하위 클래스이다



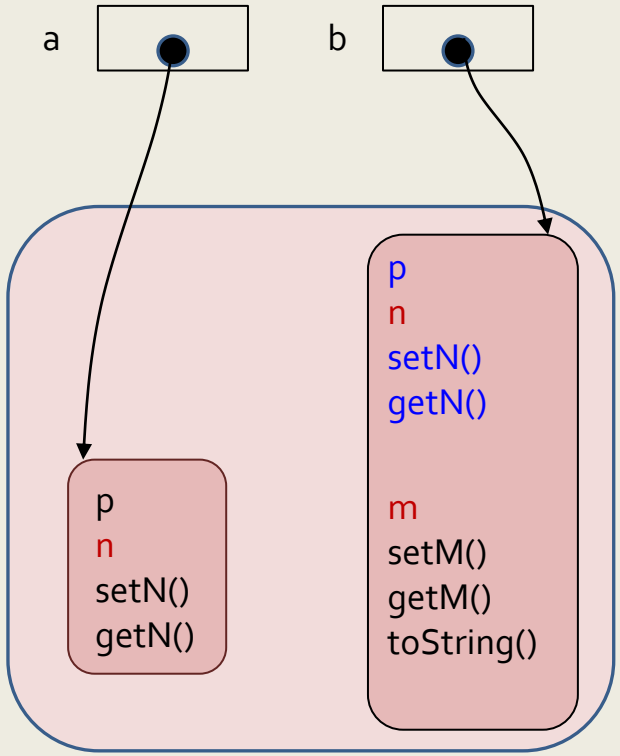
서브 클래스의 객체와 멤버 사용

- 서브 클래스의 객체와 멤버 접근
  - 서브 클래스의 객체는 슈퍼 클래스의 멤버도 포함
  - **슈퍼 클래스의 private 멤버는 상속되지 않음**
    - 서브 클래스에서 직접 접근 불가
    - 슈퍼 클래스의 prviate 멤버는 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 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

슈퍼 클래스 A와 서브 클래스 B 그리고 인스턴스 관계

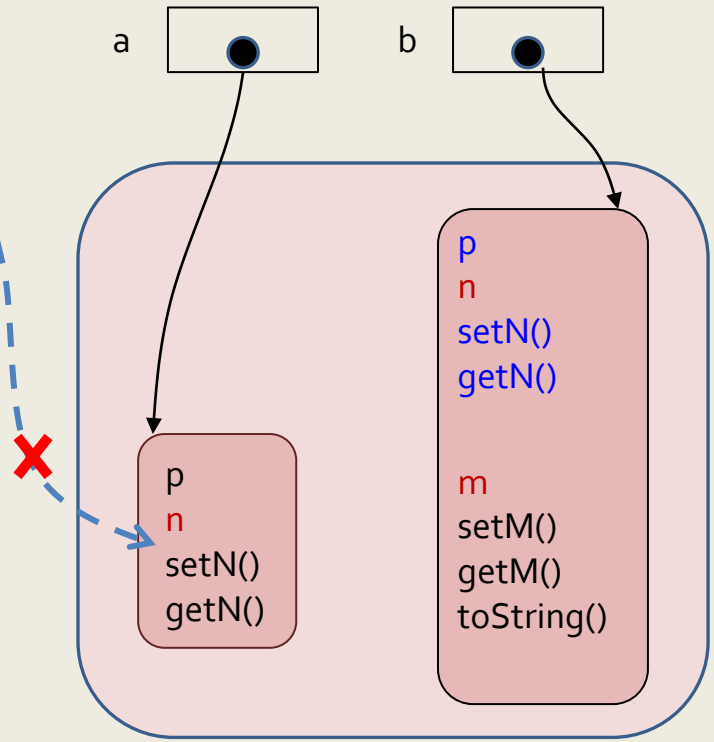
```
public class MemberAccessExample {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();

        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());
        // 화면에 10 20이 출력됨
    }
}
```

10 20



상속과 멤버접근 지정자

- 접근 지정자 4 가지
  - **public, protected, default, private**
    - 상속 관계에서 주의할 접근 지정자는 private와 protected
- private 멤버
  - 슈퍼 클래스의 private 멤버는 상속되지 않으므로 서브 클래스 뿐만 아니라 기타 모든 클래스에서 접근 불허
- protected 멤버
  - 같은 패키지 내의 모든 클래스는 접근
  - 동일 패키지 여부와 상관없이 서브 클래스에서 슈퍼 클래스의 멤버 접근 가능

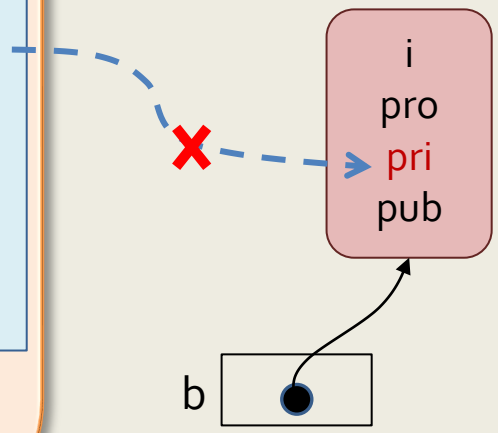
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
같은 패키지의 서브 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O
다른 패키지의 서브 클래스	X	X	O	O

같은 패키지 내 상속 관계에서 접근

패키지 PA

```
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 발생
        pub = 4;
    }
    public static void main(String[] args) {
        B b = new B();
        b.set();
    }
}
```



다른 패키지의 상속 관계에서 접근

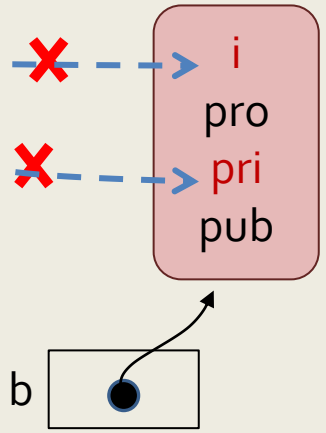
패키지 PA

```
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;
        pri = 3;   // private 멤버 접근 불가, 컴파일 오류 발생
        pub = 4;
    }

    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 getter와 setter를 통해서만 조작이 가능하다.

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

```
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 Student extends Person {
   void set() {
      age = 30;
      name = "홍길동";
      height = 175;
      setWeight(99);
   }
   public static void main(String[] args) {
      Student s = new Student();
      s.set();
   }
}
```

서브 클래스와 슈퍼 클래스의 생성자 호출 및 실행 관계

질문 1> 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자가 모두 실행되는가? 아니면 서브 클래스의 생성자만 실행되는가? 둘 다 실행된다.

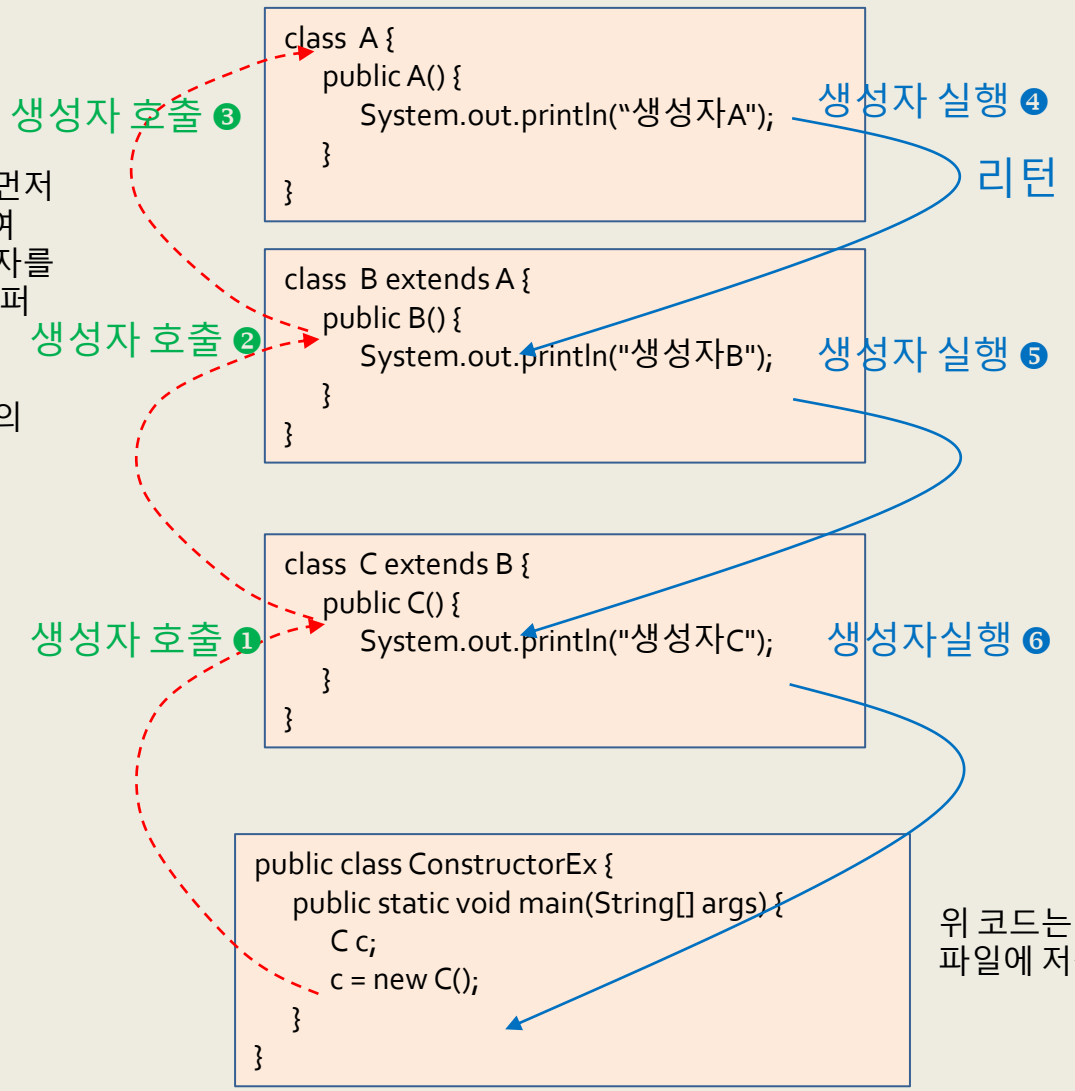
질문 2> 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자의 실행 순서는 어떻게 되는가?

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
    - 서브클래스 생성자 먼저 호출, 실행 전 슈퍼 클래스의 생성자 호출
  - 실행 순서
    - 슈퍼 클래스 생성자 먼저 실행 후에, 서브 클래스 생성자 실행
    - 그러므로 호출 순서와 실행 순서는 반대

슈퍼클래스와 서브 클래스의 생성자간의 호출 및 실행 관계

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



예상 실행 결과는?

```
생성자A
생성자B
생성자C
```

위 코드는 모두 ConstructorEx.java 파일에 저장된다.

서브 클래스와 슈퍼 클래스의 생성자 매핑

- 슈퍼 클래스와 서브 클래스
  - 각각 여러 개의 생성자 가능(생성자 overloading)
- 슈퍼 클래스와 서브 클래스의 생성자 사이의 mapping
  - 서브클래스의 객체 생성 시, 실행 가능한 슈퍼 클래스와 서브 클래스의 생성자 조합
    - 컴파일러는 서브 클래스의 생성자를 기준으로 아래와 같이 슈퍼 클래스의 생성자를 찾음
      - 기본적으로 서브클래스의 생성자 유형에 관계없이 수퍼클래스에서는 기본 생성자가 호출됨
      - 경우 1, 3
    - 프로그래머가 서브 클래스의 생성자에서 슈퍼 클래스의 쌍을 지정하는 경우
      - 경우 2, 4
      - **super() 키워드 이용**

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)

아래 코드는 모두 ConstructorEx2.java 파일에 저장된다.

서브 클래스의 생성자가 기본생성자인 경우 컴파이일러는 자동으로 슈퍼클래스의 기본생성자와 짝 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();
    }
}
```

생성자A
생성자B

아래 코드는 모두 ConstructorEx2.java 파일에 저장된다.

컴파일러가 public B()에 대한 짝을 찾을 수 없음

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

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

```
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 생성자, 슈퍼클래스의 기본생성자 매칭

엮의 코드는 모두 ConstructorEx3.java 파일에 저장된다.

```
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");
    }
}
```

```
생성자A
매개변수생성자B
```

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

수퍼클래스의 생성자 명시적 호출: **super()**

- super()
  - 서브 클래스에서 명시적으로 슈퍼 클래스의 생성자를 선택하여 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 사례

엮의 코드는 모두 ConstructorEx4.java 파일에 저장된다.

```
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);
    }
}
```

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

```
매개변수생성자A5
매개변수생성자B5
```