프로그래밍기초

Chapter 08. 상속과 다형성

Section 01

상속

상속

■ 상속의 필요성

- 상속: 기존 클래스의 기능을 사용하여 새 클래스를 만드는 기술
- 한 클래스가 다른 클래스의 특징(멤버 메서드와 변수)을 가져오도록 하는 자바 객체지향 프로그래밍의 필수적인 부분임
- 클래스 간의 관계를 더 잘 이해할 수 있고 프로그램 구조를 더욱 조직화할 수 있기 때문에 코드의 가독성과 해석 가능성이 향상됨
- 응용 프로그램의 유지·관리에 유용함

상속

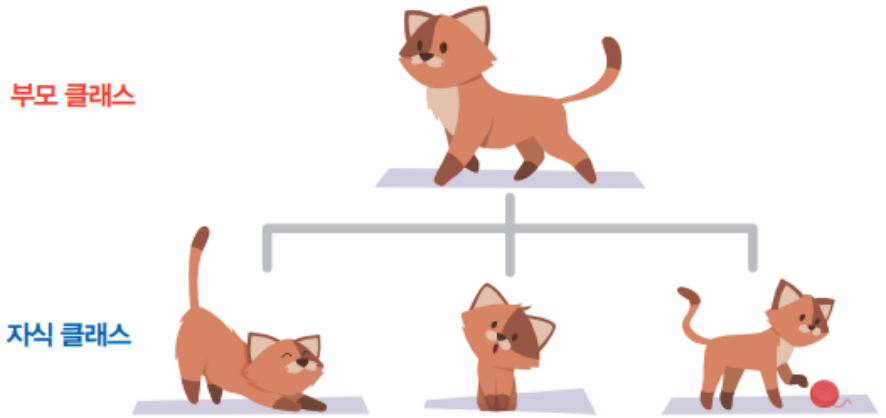
■ 상속의 개념

- 상속은 다른 클래스 간의 관계를 설정하고 계층적 순서로 정보를 관리하며 코드를 재사용하는 데 도움됨
- 새로운 클래스를 만들 때, 원하는 코드 중 일부가 포함된 클래스가 이미 있는 경우 기존 클래스에서 새 클래스를 파생(상속)시킬 수 있음
- 이렇게 함으로써 기존 클래스의 멤버 변수와 메서드를 재사용할 수 있음

상속

■ 상속의 개념

- **부모 클래스 = 슈퍼 클래스, 기본 클래스**
  → 다른 클래스에 멤버 요소(메서드와 변수)를 상속하는 클래스로 상위 클래스
- **자식 클래스 = 서브 클래스, 파생 클래스**
  → 다른 클래스의 멤버 요소를 상속받은 클래스로 하위 클래스
  → 자식 클래스는 부모 클래스의 모든 멤버 요소를 소유할 뿐만 아니라 그 밖에도 고유한 멤버 메서드와 변수를 추가할 수 있음



상속

■ 상속의 개념

- 부모 클래스에서 자식 클래스로 상속하려면 extends 키워드를 사용함

```
class 자식클래스 extends 부모클래스 {
   // 멤버 요소
}
```

상속

■ 부모 클래스와 자식 클래스 예시

```
public class ParentCat {
    public String breed = "샴고양이";
      void eat() {
        System.out.println("먹이를 먹다.");
   }
}
```

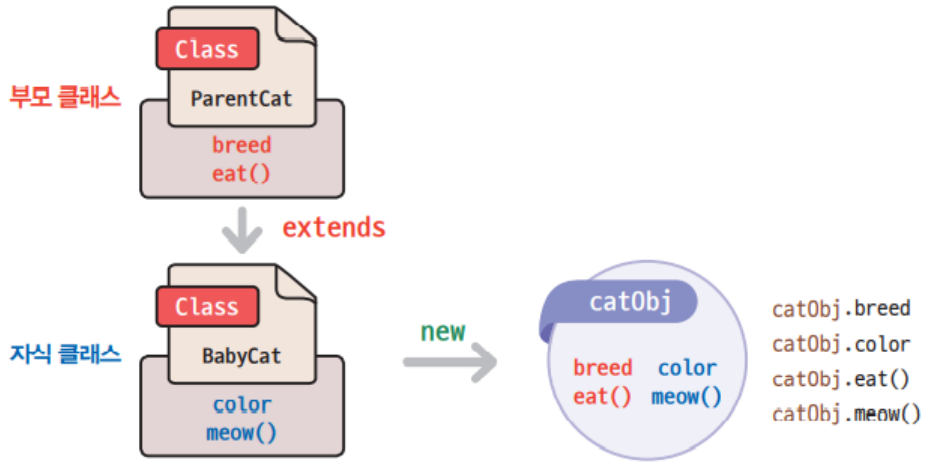
```
public class BabyCat extends ParentCat {
   public String color = "초콜릿색";
      void meow() {
        System.out.println("야옹하고 울다.");
   }
}
```

```
public class Example01 {
    public static void main(String[] args) {
       BabyCat catObj = new BabyCat();
       System.out.println("품종 : " + catObj.breed );
       System.out.println("색상 : " + catObj.color );

       catObj.eat();
      catObj.meow();
   }
}
```

실행 결과

```
품종 : 샴고양이
색상 : 초콜릿색
먹이를 먹다.
야옹하고 울다.
```



상속

■ 예제 8-1. 부모 클래스의 메서드 선언하고 호출하기

```
01 class Calculation {
02    int z;
03
04     public void addition(int x, int y) {
05        z = x + y;
06       System.out.println("두 수의 덧셈 : " + z);
07     }
08
09      public void subtraction(int x, int y) {
10         z = x - y;
11         System.out.println("두 수의 뺄셈 : " + z);
12     }
13 }
```

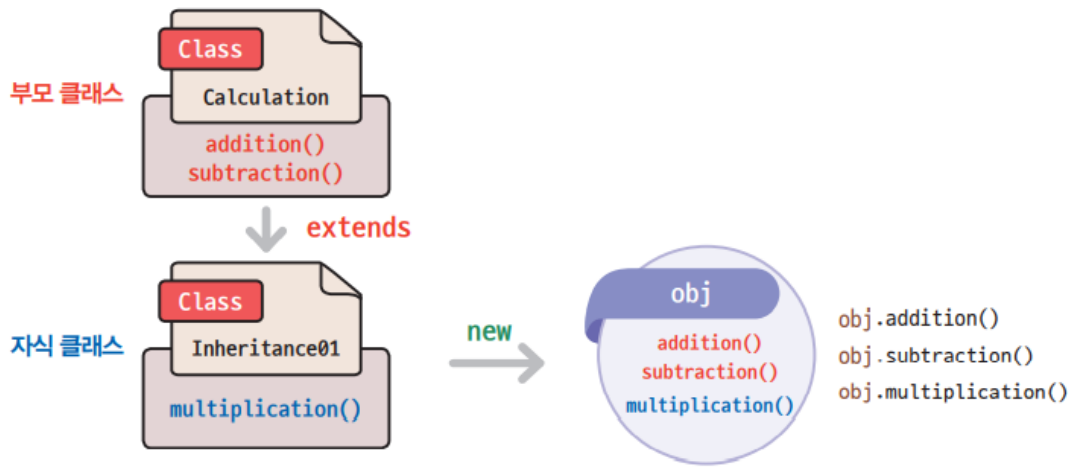
상속

■ 예제 8-1. 부모 클래스의 메서드 선언하고 호출하기

```
01 public class Inheritance01 extends Calculation {
02     public void multiplication(int x, int y) {
03         z = x * y;
04       System.out.println("두 수의 곱셈 : " + z);
05     }
06
07      public static void main(String[] args) {
08         int a = 20, b = 10;
09         Inheritance01 obj = new Inheritance01();
10         obj.addition(a, b);
11         obj.subtraction(a, b);
12        obj.multiplication(a, b);
13    }
14 }
```

실행 결과

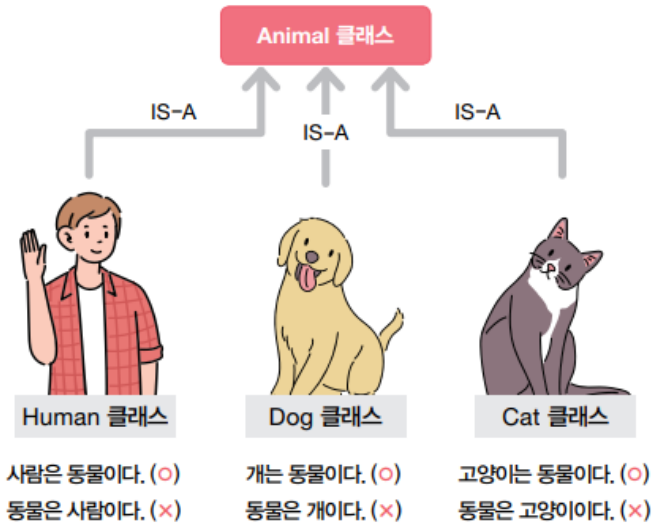
```
두 수의 덧셈 : 30
두 수의 뺄셈 : 10
두 수의 곱셈 : 200
```



상속

■ Is-A 관계(is a relationship)

- '…는 …이다'라는 의미, 부모-자식 관계
  → extends, implements 키워드로 구현함
  →모든 클래스는 java. lang.Object의 하위 클래스임
- Is-A 관계는 상속을 나타냄




```
class Animal {
   ...
}
class Cat extends Animal {
   ...
}
```

Cat과 Animal은 Is-A 관계

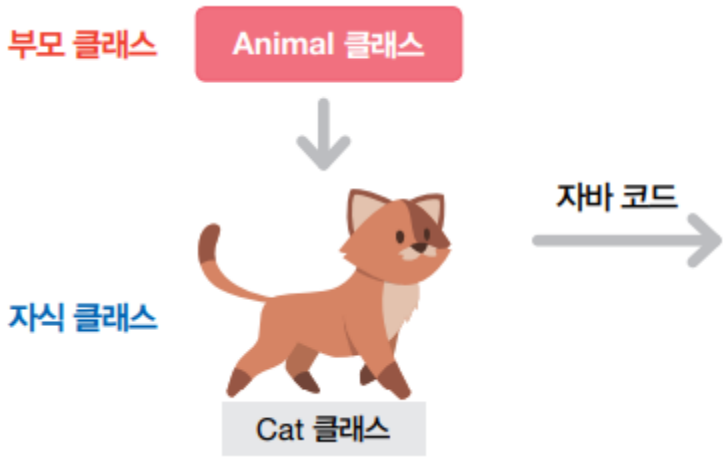
Section 02

상속의 유형

상속의 유형

■ 단일 상속

- 클래스가 하나의 클래스에 의해서만 확장되는 것으로, 단일 수준 상속이라고도 함
- 단일 부모 클래스에서 자식 클래스를 만들기 때문에 기본 클래스(부모 클래스)와 파생 클래 스(자식 클래스)가 각각 하나뿐임
- 자식 클래스는 단일 기본 클래스로부터만 속성과 행동을 상속받고 부모 클래스의 모든 메서드와 변수에 접근할 수 있음




```
class Animal {
    // 멤버 변수와 메서드
}

class Cat extends Animal {
   // 멤버 변수와 메서드
}
```

상속의 유형

■ 단일 상속 예시

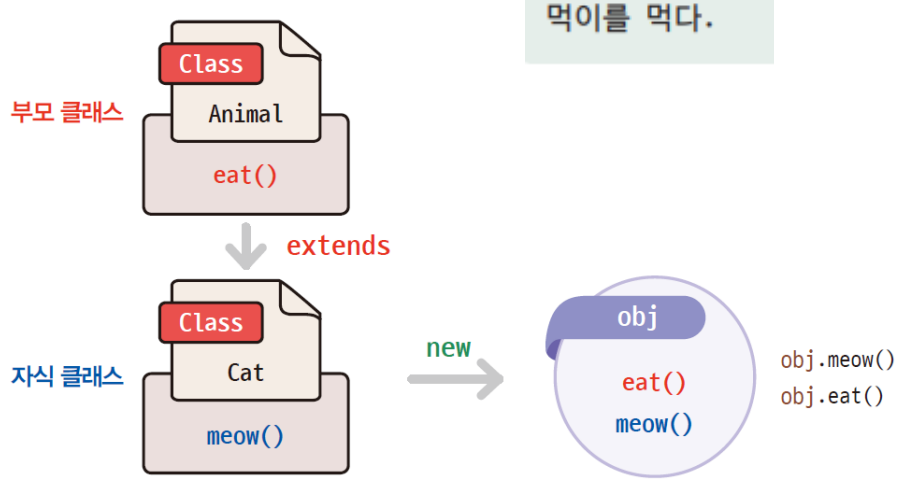
```
public class Animal {
    void eat() {
      System.out.println("먹이를 먹다.");
    }
}
```

```
public class Cat extends Animal {
   void meow() {
      System.out.println("야옹하고 울다.");
   }
}
```

```
public class Example02 {
   public static void main(String[] args) {
      Cat obj = new Cat();
      obj.meow();
      obj.eat();
   }
}
```

실행 결과

야옹 하고 울다.
먹이를 먹다.



상속의 유형

■ 예제 8-2. 단일 상속 메서드 호출하기

```
01 public class Father {
02    String familyName = "프로그래머";
03    String houseAddress = "인천";
04 }
```

```
01 public class Son extends Father {
02    String name = "홍길동";
03
04    void printDetails() {
05       System.out.println("나의 이름은 " + this.name );
06       System.out.println("나의 아버지는 " + this.familyName);
07       System.out.println("나의 집은 " + this.houseAddress);
08    }
09 }
```

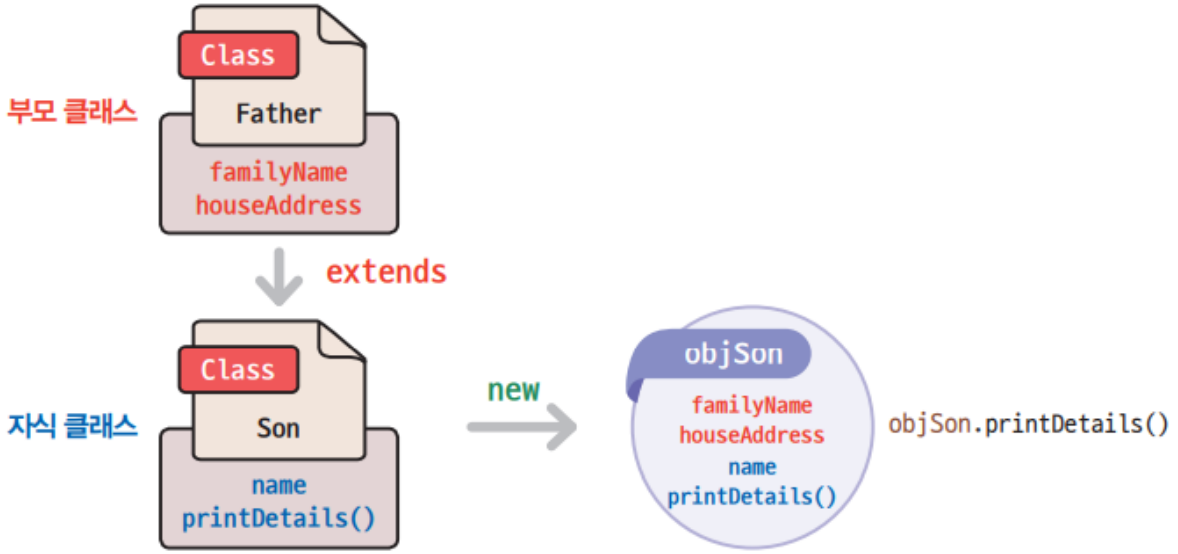
상속의 유형

■ 예제 8-2. 단일 상속 메서드 호출하기

```
01 public class Inheritance02 {
02    public static void main(String[] args) {
03      Son objSon = new Son();
04      objSon.printDetails();
05    }
06 }
```

실행 결과

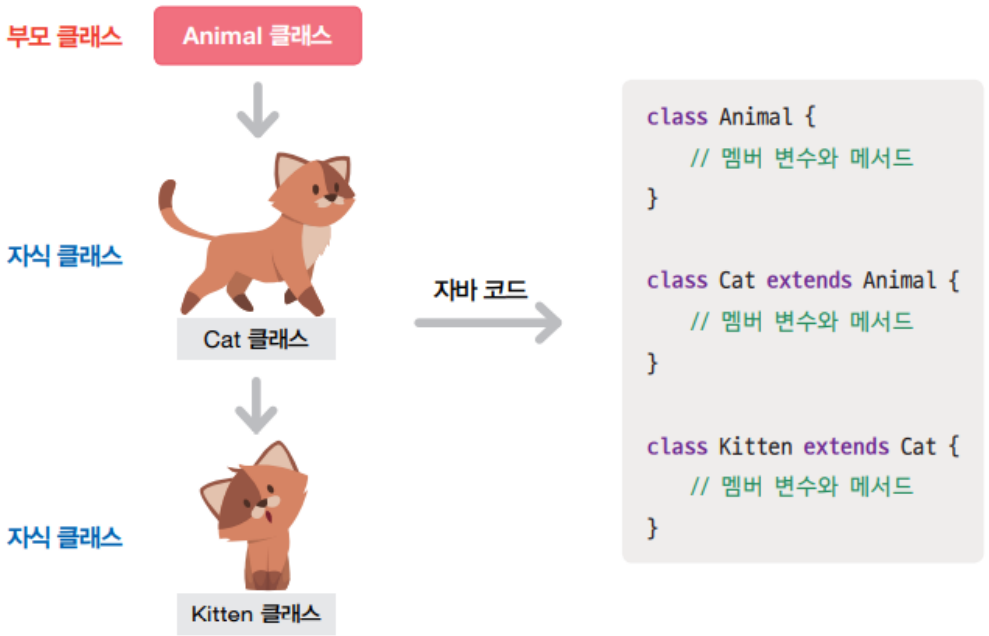
```
나의 이름은 홍길동
나의 아버지는 프로그래머
나의 집은 인천
```



상속의 유형

■ 다단계 상속

- 클래스가 하나의 클래스에 상속하고, 상속받은 자식 클래스가 또 다른 클래스에 상 속하는 것을 말함
- [예] 손주는 아버지로부터 상속받고 아버지는 할아버지로부터 상속받음



상속의 유형

■ 다단계 상속 예시

```
public class Animal {
   void eat() {
     System.out.println("먹이를 먹다.“);
   }
}
```

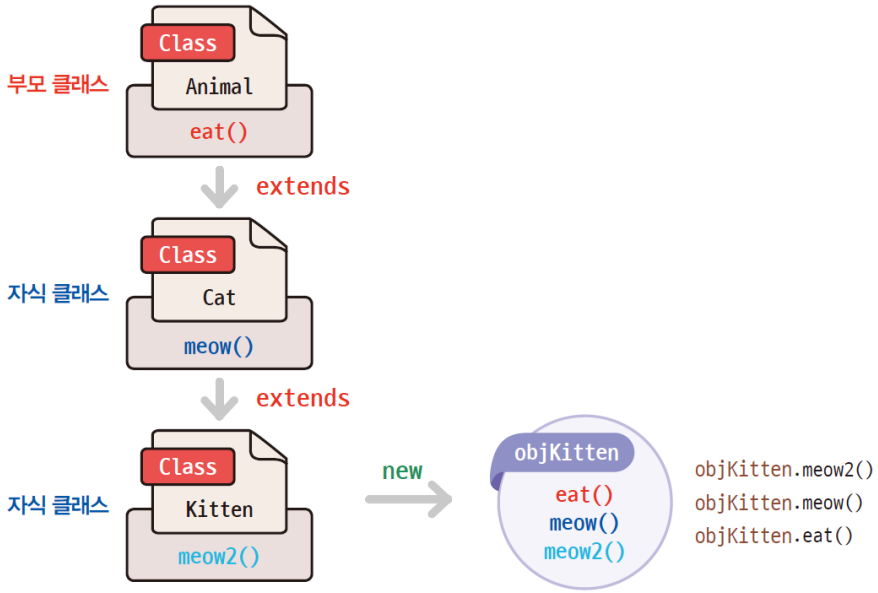
```
public class Cat extends Animal {
   void meow() {
      System.out.println("야옹하고 울다.“);
   }
}
```

```
public class Kitten extends Cat {
   void meow2() {
      System.out.println("새끼 고양이가 야옹하고 울다.");
   }
}
```

```
public class Example03 {
   public static void main(String[] args) {
      Kitten objKitten = new Kitten();
      objKitten.meow2();
      objKitten.meow();
      objKitten.eat();
   }
}
```

실행 결과

```
새끼 고양이가 야옹하고 울다.
야옹하고 울다.
먹이를 먹다.
```



상속의 유형

■ 예제 8-3. 다단계 상속 메서드 호출하기

```
01 public class GrandFather {
02   void printGrandFather() {
03     System.out.println("나는 할아버지입니다.");
04   }
05 }
```

```
01 public class SubFather extends GrandFather {
02     String familyName = "프로그래머";
03     String houseAddress = "인천";
04
05     void printFather() {
06       System.out.println("나는 아버지입니다! 나는 할아버지로부터 상속받습니다.");
07    }
08 }
```

상속의 유형

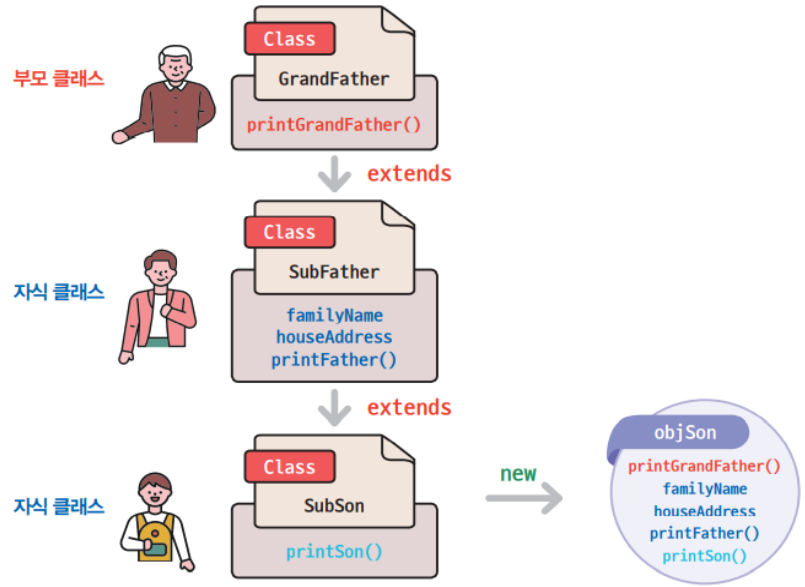
■ 예제 8-3. 다단계 상속 메서드 호출하기

```
01 public class SubSon extends SubFather {
02    void printSon() {
03       System.out.println("나는 아들입니다.");
04       System.out.println("나는 아버지로부터 상속받습니다.");
05       System.out.println("나의 아버지는 " + this.familyName);
06       System.out.println("나의 집은 " + this.houseAddress);
07    }
08 }
```

```
01 public class Inheritance03 {
02    public static void main(String[] args) {
03       SubSon objSon = new SubSon();
04       objSon.printSon();
05       objSon.printFather();
06       objSon.printGrandFather();
07    }
08 }
```

실행 결과

```
나는 아들입니다.
나는 아버지로부터 상속받습니다.
나의 아버지는 프로그래머
나의 집은 인천
나는 아버지입니다! 나는 할아버지로부터 상속받습니다.
나는 할아버지입니다.
```



상속의 유형

■ 계층적 상속 예시

```
public class Animal {
  void eat() {
    System.out.println("먹이를 먹다.");
  }
}
```

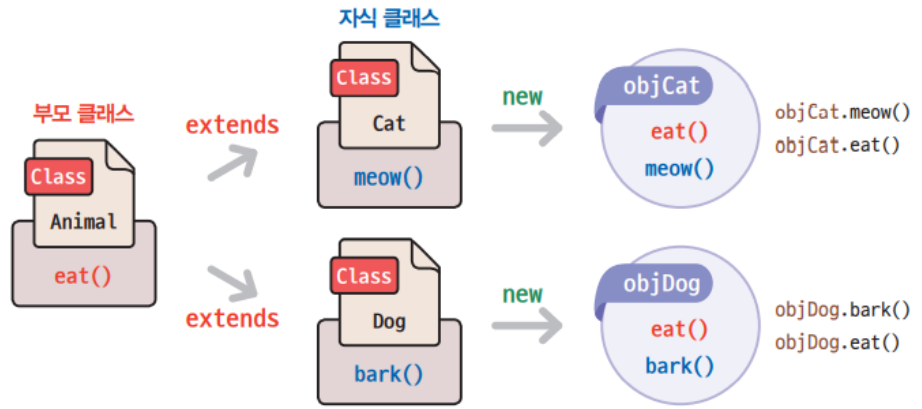
```
public class Cat extends Animal {
  void meow() {
    System.out.println("야옹하고 울다.");
  }
}
```

```
public class Dog extends Animal {
  void bark() {
    System.out.println("멍멍하고 짖다.");
  }
}
```

```
public class Example04 {
  public static void main(String[] args) {
    Cat objCat = new Cat();
    objCat.meow();
    objCat.eat();
    Dog objDog = new Dog();
    objDog.bark();
    objDog.eat();
  }
}
```

실행 결과

```
야옹하고 울다.
먹이를 먹다.
멍멍하고 짖다.
먹이를 먹다.
```



상속의 유형

■ 예제 8-4. 계층적 상속 메서드 호출하기

```
01 public class SubFather extends GrandFather {
02    String familyName = "프로그래머";
03    String houseAddress = "인천";
04
05     void printFather() {
06        System.out.println("나는 아버지입니다! 나는 할아버지로부터 상속받습니다.");
07    }
08 }
```

```
01 public class SubSon extends SubFather {
02    void printSon() {
03       System.out.print ln("나는 아들입니다.");
04       System.out.println("나는 아버지로부터 상속받습니다.");
05       System.out.println("나의 아버지는 " + this.familyName);
06       System.out.println("나의 집은 " + this.houseAddress);
07    }
08 }
```

상속의 유형

■ 예제 8-4. 계층적 상속 메서드 호출하기

```
01 public class SubDaughter extends SubFather {
02    void printDaughter() {
03       System.out.println("나는 딸입니다.");
04       System.out.println("나는 아버지로부터 상속받습니다.");
05       System.out.println("나의 아버지는 " + this.familyName);
06       System.out.println("나의 집은 " + this.houseAddress);
07    }
08 }
```

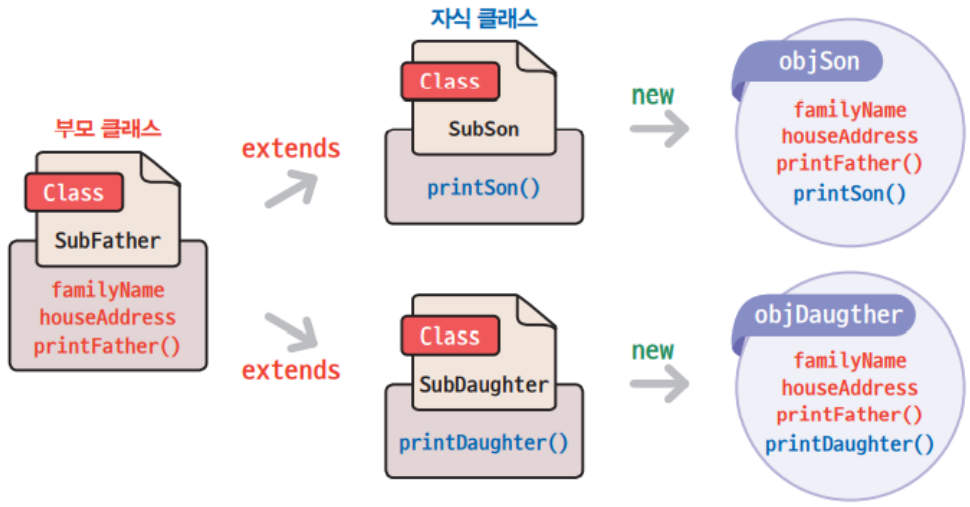
상속의 유형

■ 예제 8-4. 계층적 상속 메서드 호출하기

```
01 public class Inheritance04 {
02     public static void main(String[] args) {
03
04         SubSon objSon = new SubSon();
05         objSon.printSon();
06         objSon.printFather();
07
08         SubDaughter objDaugther = new SubDaughter();
09         objDaugther.printDaughter();
10         objDaugther.printFather();
11
12     }
13 }
```

실행 결과

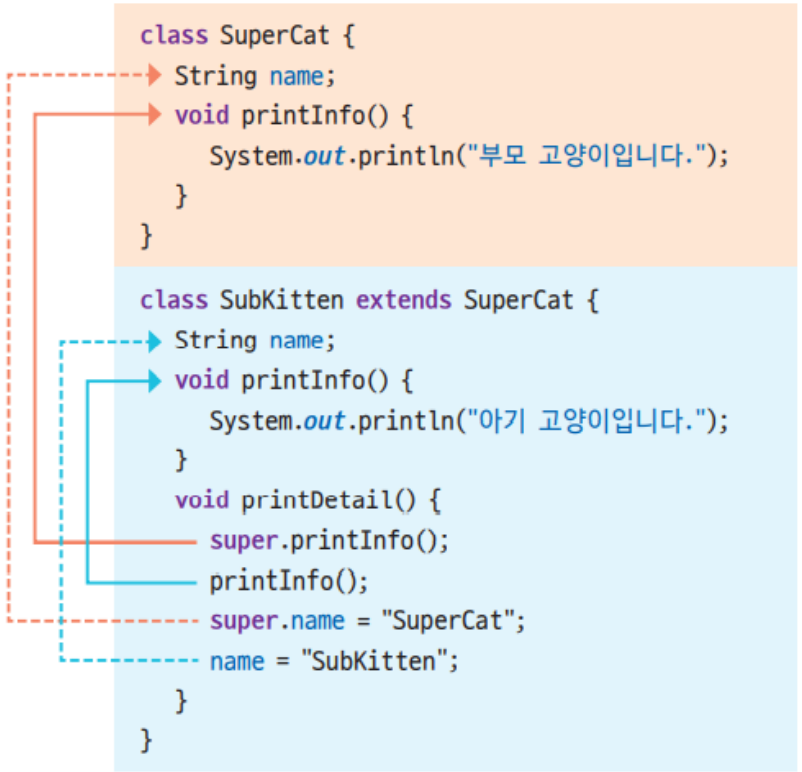
```
나는 아들입니다.
나는 아버지로부터 상속받습니다.
나의 아버지는 프로그래머
나의 집은 인천
나는 아버지입니다! 나는 할아버지로부터 상속받습니다.
나는 딸입니다.
나는 아버지로부터 상속받습니다.
나의 아버지는 프로그래머
나의 집은 인천
나는 아버지입니다! 나는 할아버지로부터 상속받습니다.
```



상속의 유형

■ super를 이용한 부모 클래스 참조

- 자바의 상속에서 자식 클래스가 부모 클래스로부터 상속을 받으면 자식 클래스는 부모 클래스를 참조하기 위해 super 키워드를 사용



상속의 유형

■ super를 이용한 부모 클래스 참조

- super를 이용한 변수와 메서드 접근

→super는 부모 클래스의 멤버 요소인 변수와 메서드에 접근할 수 있는 명령어

→자식 클래스의 메서드에서 호출하여 부모 클래스의 변수나 메서드에 접근

■ super 키워드 사용 예시

```
public class SuperCat {
   String breed = "샴고양이";
   String age = "15살";

   void printInfo() {
     System.out.println("부모 고양이입니다.");
   }
}
```

상속의 유형

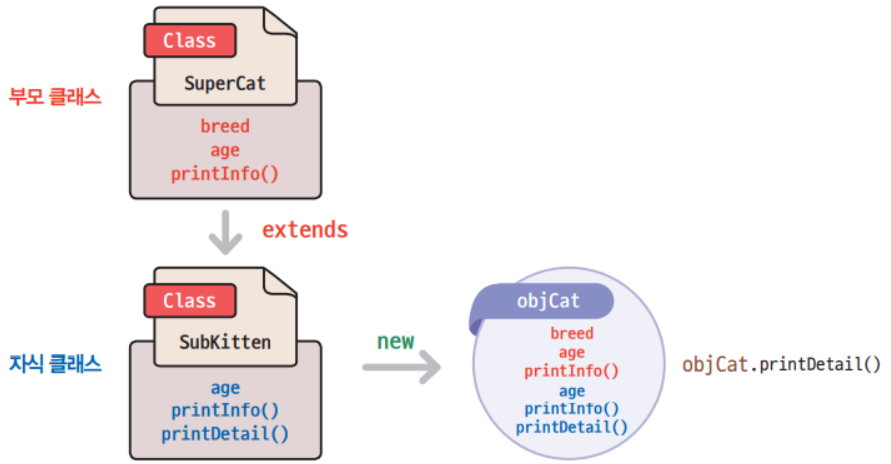
■ super 키워드 사용 예시

```
public class SubKitten extends SuperCat {
    String age = "2살";
    void printInfo() {
        System.out.println("아기 고양이입니다.");
    }
    void printDetail() {
        super.printInfo();
        System.out.println("품종은 " + super.breed + ", 나이는 " + super.age);
        printInfo();
        System.out.println("품종은 " + breed + ", 나이는 " + age);
       // System.out.println("아기 고양이는 " + this.breed + ", 나이는 " + this.age);
    }
}
```

실행 결과

```
부모 고양이입니다.
품종은 샴고양이, 나이는 15살
아기 고양이입니다.
품종은 샴고양이, 나이는 2살
```

```
public class Example05 {
    public static void main(String[] args) {
      SubKitten objCat = new SubKitten();
      objCat.printDetail();
    }
}
```



상속의 유형

■ 예제 8-5. super 키워드를 이용하여 부모 클래스의 멤버 요소에 접근하기

```
01 public class Parent {
02    String name = "홍길순";
03
04     public void details() {
05        System.out.println(name);
06    }
07 }
```

상속의 유형

■ 예제 8-5. super 키워드를 이용하여 부모 클래스의 멤버 요소에 접근하기

```
01 public class Child extends Parent {
02     String name = "홍길동";
03
04     public void details() {
05       super.details();
06       System.out.println(name);
07     }
08
09     public void printDetails() {
10        details();
11        System.out.println("부모 이름 : " + super.name);
12       System.out.println("자식 이름 : " + name);
13    }
14 }
```

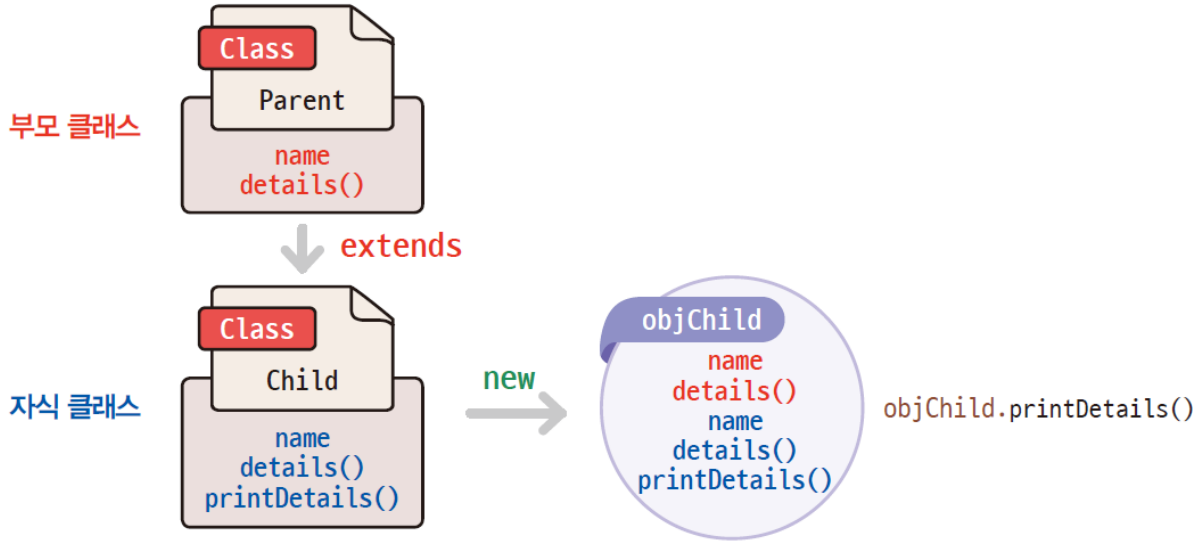
상속의 유형

■ 예제 8-5. super 키워드를 이용하여 부모 클래스의 멤버 요소에 접근하기

```
01 public class Inheritance05 {
02    public static void main(String[] args) {
03       Child objChild = new Child();
04      objChild.printDetails();
05    }
06 }
```

실행 결과

```
홍길순
홍길동
부모 이름 : 홍길순
자식 이름 : 홍길동
```



상속의 유형

■ super를 이용한 부모 클래스 참조

- super( )를 이용한 생성자 접근

→ super()는 부모 클래스의 생성자를 호출하는 명령어

→ 자식 클래스의 생성자 첫 행에 이 명령어가 있어야 함

→ 자식 클래스의 생성자가 부모 클래스의 생성자를 명시적으로 호출하지 않으면 자바 컴파일러는 부모 클래스의 매개변수가 없는 생성자를 자동으로 호출

✓ 이때 부모 클래스에 매개변수가 없는 생성자가 존재하지 않으면 컴파일타임 오류가 발생함

상속의 유형

■ super를 이용한 부모 클래스 참조
- super( )를 이용한 생성자 접근

■ super() 사용 예시

```
public class SuperCat2 {
    String name;
    String age = "15살";
    SuperCat2(String n) {
        name = n;
        System.out.println("부모 고양이입니다." + " 이름은 " + name);
    }
}
```

상속의 유형

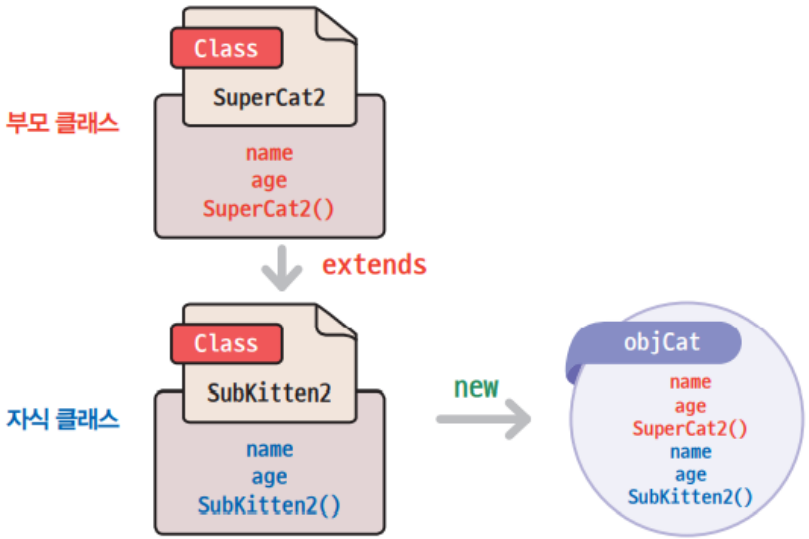
■ super() 사용 예시

```
public class SubKitten2 extends SuperCat2 {
    String name;
    String age = "2살";
    public SubKitten2(String n1, String n2) {
       super(n1);
       this.name = n2;
       System.out.println("아기 고양이입니다." + " 이름은 " + name);
   }
}
```

```
public class Example06 {
   public static void main(String[] args) {
     SubKitten2 objCat = new SubKitten2("아름이", "다운이");
   }
}
```

실행 결과

```
부모 고양이입니다. 이름은 아름이
아기 고양이입니다. 이름은 다운이
```



상속의 유형

■ 예제 8-6. super()를 이용하여 부모 클래스의 생성자에 접근하기

```
01 public class Parent2 {
02     String name = "홍길순";
03
04     Parent2() {
05       System.out.println("부모 이름 : " + name);
06     }
07 }
```

```
01 public class Child2 extends Parent2 {
02     String name = "홍길동";
03
04     Child2() {
05       super();
06       System.out.println("자식 이름 : " + name);
07     }
08 }
```

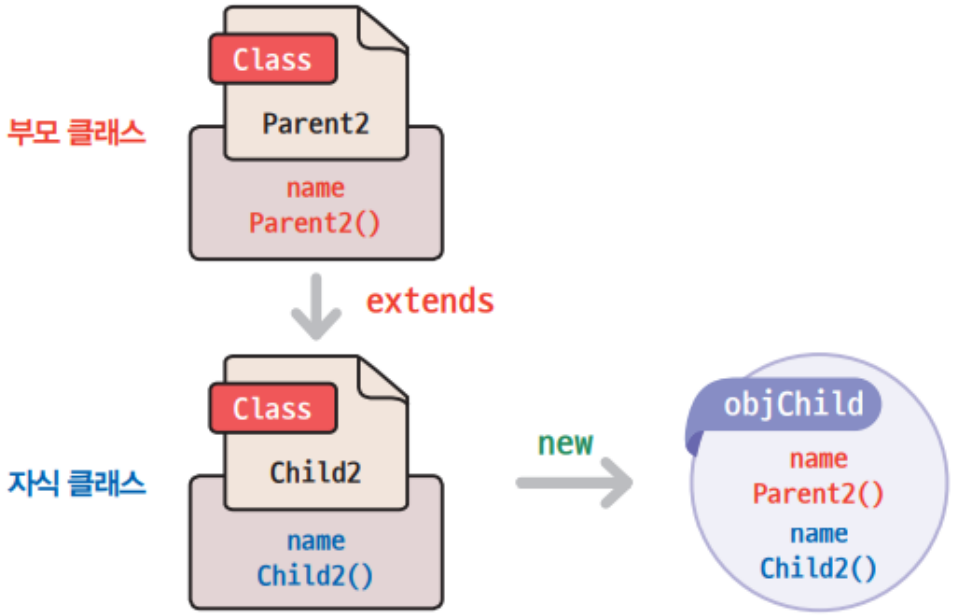
상속의 유형

■ 예제 8-6. super()를 이용하여 부모 클래스의 생성자에 접근하기

```
01 public class Inheritance06 {
02     public static void main(String[] args) {
03         Child2 objChild = new Child2();
04     }
05 }
```

실행 결과

```
부모 이름 : 홍길순
자식 이름 : 홍길동
```



Section 03

다형성

다형성

■ 다형성의 필요성

- 하나의 객체가 여러 가지 유형을 띠는 것을 다형성이라고 함
  → 이해하기 쉬운 코드를 작성하는 데 도움이 됨
- 다형성을 통해 상속과 메서드 재정의를 활용함
  → 확장성 있는 프로그램을 만들 수 있음
- 부모 클래스의 참조 변수로 자식 클래스의 객체에 접근함
  → 코드의 효율성도 높일 수 있음
- 상위 클래스에서는 공통적인 부분을 제공하고 하위 클래스에서는 각 클래스에 맞는 기능을 구현할 수 있음

다형성

■ 프로그래밍에 다형성을 적용시 장점

- 단일 메서드를 다른 클래스에서 다르게 작동할 수 있음
- 동일한 구현에 대해 다른 메서드명을 선언할 필요가 없음
- 상속을 더 쉽게 구현할 수 있음

다형성

■ 다형성의 개념

- 여러 형태를 취할 수 있는 객체의 능력인 다형성은 상속에 의해 서로 관련된 하나 이상의 클래스 또는 객체가 있을 때 발생한 상속을 통해 변수와 메서드를 상속할 수 있다면 다형성은 이러한 메서드를 사용하여 다른 작업을 수행할 수 있음

Section 04

다형성의 유형

다형성의 유형

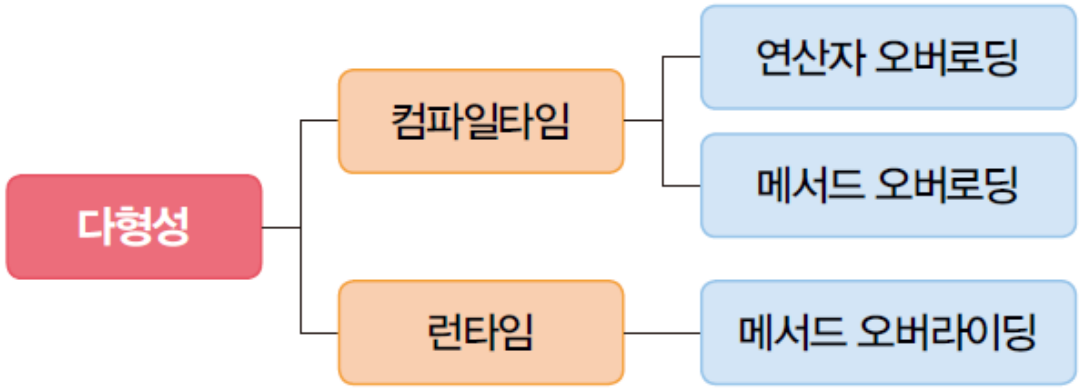
■ 다형성의 유형

- 컴파일타임 다형성

  →컴파일타임 다형성은 정적 다형성으로, 이러한 유형의 다형성은 연산자 오버로딩과 메서드 오버로딩으로 구현함

- 런타임 다형성

  →동적 메서드로 재정의된 메서드에 대한 메서드 호출이 런타임에 해결되는 프로세 스로, 이러한 유형의 다형성은 메서드 오버라이딩으로 구현함



다형성의 유형

■ 컴파일타임 다형성

- 메서드 오버로딩

→ 메서드명이 같지만 다른 매개변수를 사용하는 프로세스

→ 메서드는 인수의 개수나 유형을 변경함으로써 오버로딩될 수 있으며, 컴파일러는 프로그램을 컴파일하는 동안 호출할 메서드를 결정

→ 메서드 오버로딩은 공간 효율적이고 메서드명이 동일하여 이해하기 쉽기 때문에 프로그램 디버깅이 더 수월함

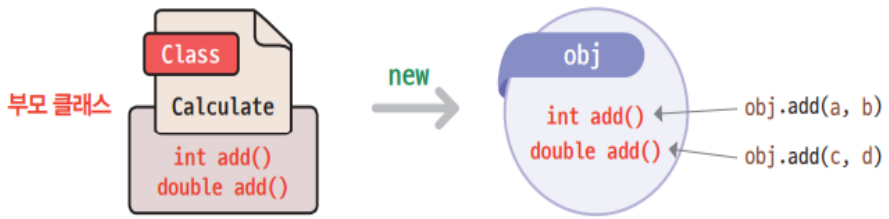
다형성의 유형

■ 컴파일타임 다형성
- 메서드 오버로딩

■ 메서드 오버로딩 예시

```
class Calculate {
   public int add(int num1, int num2) {
      return num1 + num2;
   }
   public double add(double num1, double num2) {
     return num1 + num2;
   }
}
public class Example07 {
    public static void main(String[] args) {
       int a = 4;
       int b = 5;
       double c = 11.12;
       double d = 21.34;
       Calculate obj = new Calculate();
       System.out.println(obj.add(a, b));
       System.out.println(obj.add(c, d));
  }
}
```

```
실행 결과
9
32.46
```



다형성의 유형

■ 예제 8-7. 사각형의 넓이 구하기

```
01 class CalculateSquare {
02     public void square() {
03         System.out.println("No Parameter Method Called");
04     }
05
06     public int square(int width, int height) {
07         int area = width * height;
08         return area;
09     }
10
11     public double square(double width, double height) {
12       double area = width * height;
13       return area;
14     }
15
16     public double square(int width, double height) {
17         double area = width * height;
18        return area;
19     }
20 }
```

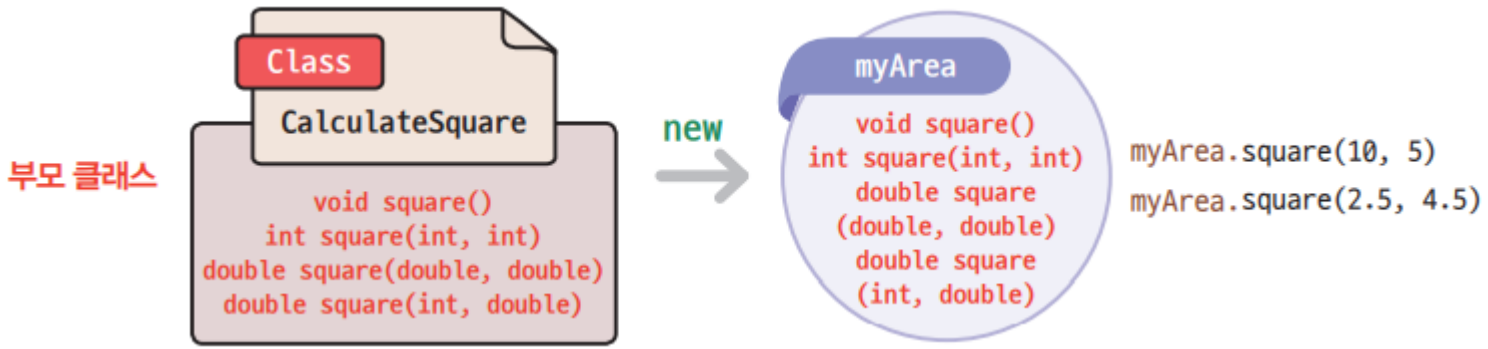
다형성의 유형

■ 예제 8-7. 사각형의 넓이 구하기

```
21
22   public class Polymorphism01 {
23      public static void main(String[] args) {
24        CalculateSquare myArea = new CalculateSquare();
25        System.out.println("가로:10, 세로:5 사각형의 넓이는 "+ myArea.square(10, 5));
26        System.out.println("가로:2.5, 세로:4.5 사각형의 넓이는 "+
                                  myArea.square(2.5, 4.5));
27 }
```

실행 결과

```
가로:10, 세로:5 사각형의 넓이는 50
가로:2.5, 세로:4.5 사각형의 넓이는 11.25
```



다형성의 유형

■ 런타임 다형성

- 메서드 오버라이딩

→부모 클래스로부터 상속받은 메서드를 자식 클래스에서 특정한 형태로 재정의

✓ 자식 클래스가 부모 클래스로부터 같은 이름, 개수와 유형이 같은 인수, 같은 메서드 반환형을 가진 메서드를 상속받아서 구현하기 때문에 메서드 재정의라고도 함

→프로그램 실행 중에 호출할 메서드를 결정하는 메서드 오버라이딩은 동일한 기능을 구현한 자식 클래스에서 동일한 메서드를 사용함으로써 코드의 복잡성을 줄이고 일관성을 향상

다형성의 유형

■ 메서드 오버라이딩 예시

```
class Cat extends Animal {
...
   public void sound() {
      System.out.println("고양이는 야옹하고 울다.");
   }
}
```

```
class Kitten extends Cat {
...
   @Override
   public void sound() {
      System.out.println("새끼 고양이는 야옹하고 울다.");

   }
}
```

다형성의 유형

■ 메서드 오버라이딩 예시

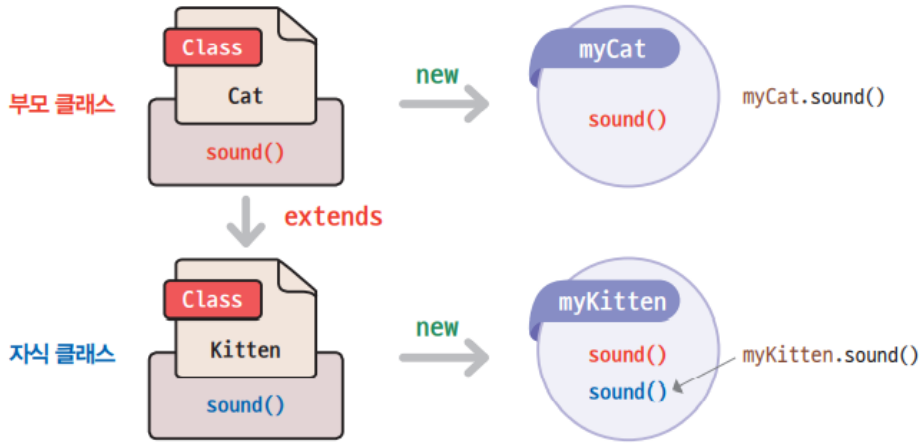
```
class Cat extends Animal {
...
   public void sound() {
      System.out.println("고양이는 야옹하고 울다.");
   }
}
```

```
class Kitten extends Cat {
...
   @Override
   public void sound() {
      System.out.println("새끼 고양이는 야옹하고 울다.");
   }
}
```

```
public class Example08 {
     public static void main(String[] args) {
        Cat myCat = new Cat();
        Kitten myKitten = new Kitten();
        myCat.sound();
       myKitten.sound();
     }
}
```

실행 결과

```
고양이는 야옹하고 울다.
새끼 고양이는 야옹하고 울다.
```



다형성의 유형

■ 예제 8-8. 동물의 울음소리 출력하기

```
01 public class Animal {
02    void eat() {
03       System.out.println("먹이를 먹다");
04    }
05
06    public void animalSound() {
07       System.out.println("동물이 소리를 낸다.");
08    }
09 }
```

```
01 public class Pig extends Animal {
02    public void animalSound() {
03      System.out.println("돼지는 꿀꿀꿀");
04    }
05 }
```

다형성의 유형

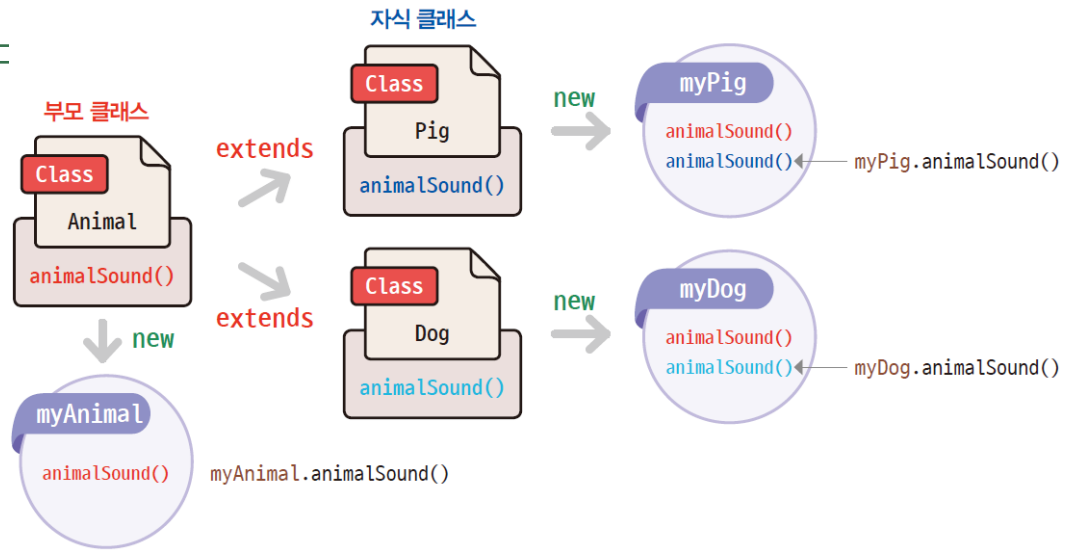
■ 예제 8-8. 동물의 울음소리 출력하기

```
01 public class Dog extends Animal {
02    void bark() {
03        System.out.println("멍멍하고 짖다.");
04    }
05
06     public void animalSound() {
07        System.out.println("개는 멍멍멍");
08    }
09 }
```

```
01 public class Polymorphism02 {
02    public static void main(String[] args) {
03       Animal myAnimal = new Animal();
04       Animal myPig = new Pig();
05       Animal myDog = new Dog();
06       myAnimal.animalSound();
07      myPig.animalSound();
08      myDog.animalSound();
09    }
10 }
```

실행 결과

```
동물이 소리를 낸다.
돼지는 꿀꿀꿀
개는 멍멍멍
```



프로그래밍기초

<Q&A : lsw@kopo.ac.kr>