Section 01

상속

1. 상속

상속의 필요성

- 상속: 기존 클래스의 기능을 사용하여 새 클래스를 만드는 기술
- 한 클래스가 다른 클래스의 특징(멤버 메서드와 변수)을 가져오도록 하는 자바 객체 지향 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

1. 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

1. 상속

상속의 개념

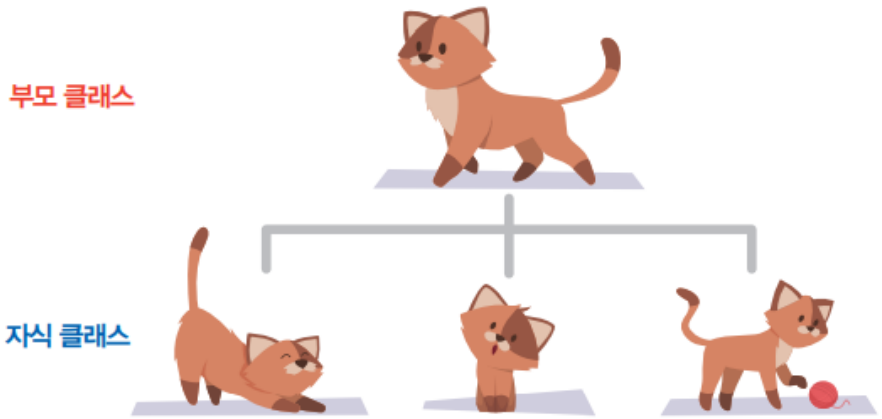
- **부모 클래스 = 슈퍼 클래스, 기본 클래스**

→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



[그림 8-1] 부모 클래스와 자식 클래스의 관계

1. 상속

상속의 개념

- 부모 클래스에서 자식 클래스로 상속하려면 extends 키워드를 사용함

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

부모 클래스와 자식 클래스 예시

ParentCat.java

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

BabyCat.java

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

1. 상속

상속의 개념

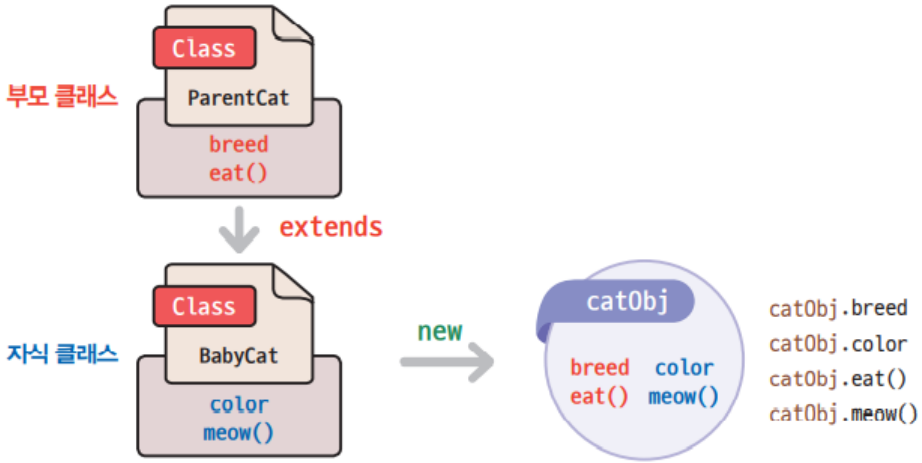
Example01.java

```
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.
```



[그림 8-2] 부모 클래스(ParentCat)와 자식 클래스(BabyCat)의 관계

1. 상속

예제 8-1 부모 클래스의 메서드 선언하고 호출하기 Calculation.java

```
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 }
```

1. 상속

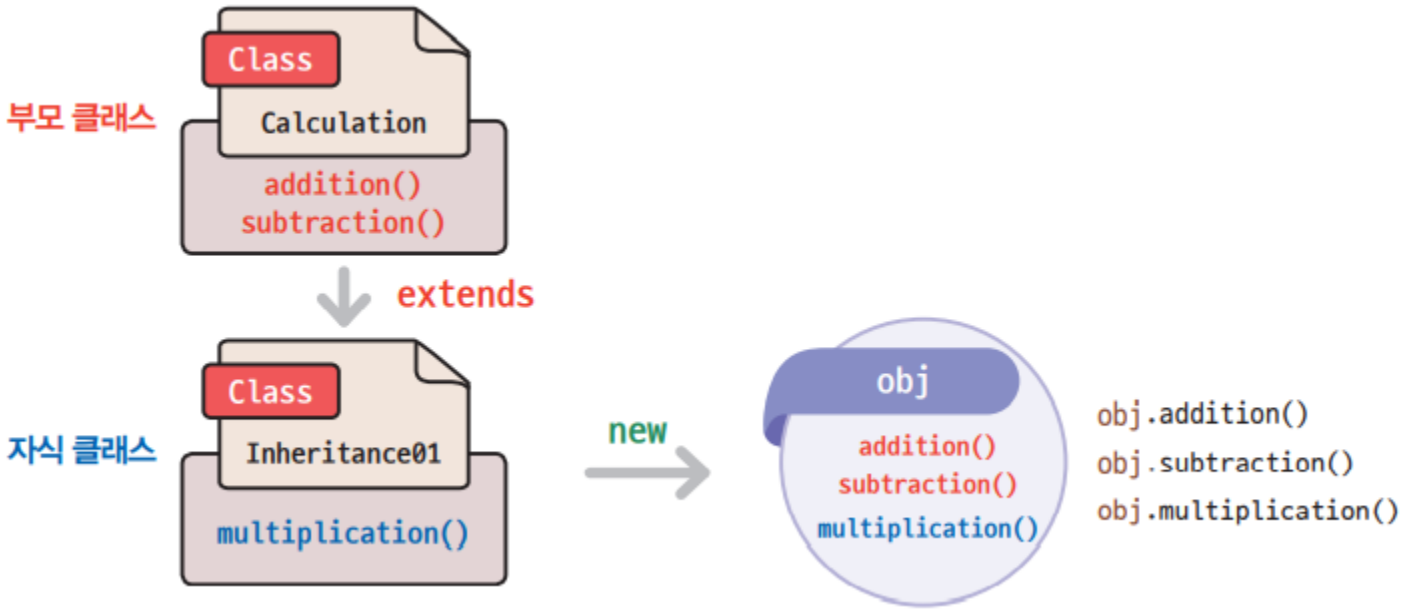
Inheritance01.java

```
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
```

1. 상속



[그림 8-3] 부모 클래스(Calculation)와 자식 클래스(Inheritance01)의 관계

1. 상속

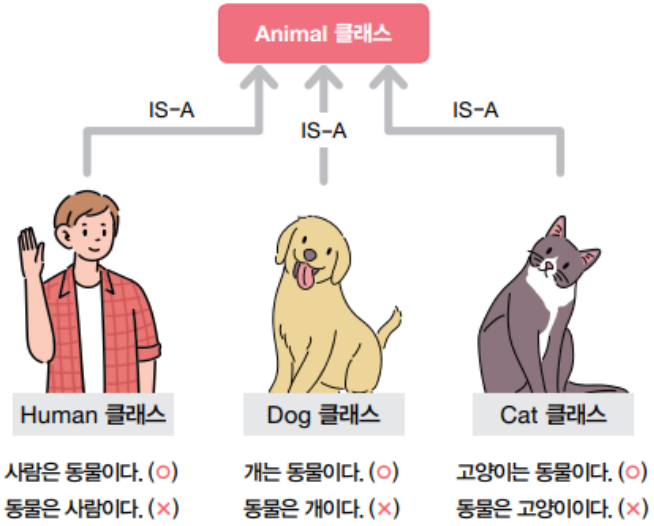
Is-A 관계

- '...는 ...이다'라는 의미, 부모-자식 관계

  → extends, implements 키워드로 구현함

  →모든 클래스는 java. lang.Object의 하위 클래스임

- Is-A 관계는 상속을 나타냄



[그림 8-4] 클래스 간의 Is-A 관계

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

Cat과 Animal은 Is-A 관계

1. 상속

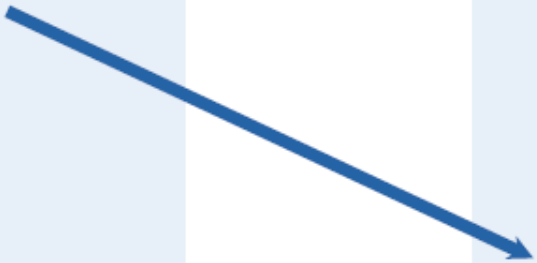
Has-A 관계

- '...는 ...소유한다'라는 의미, 소유(포함)의 관계

  → 다른 클래스로 만들어진 객체를 멤버로 갖는 경우

- 선은 점을 포함하고 있기 때문에 이러한 관계는 **"HAS~A"**관계로 표현.

```java
public class Point {
  int xPos;
  int yPos;
}
```



```java
public class Line {
  //int point1Xpos;
  //int point1Ypos;
  //int point2Xpos;
  //int point2Ypos;
  Point point1;
  Point point2;
}
```

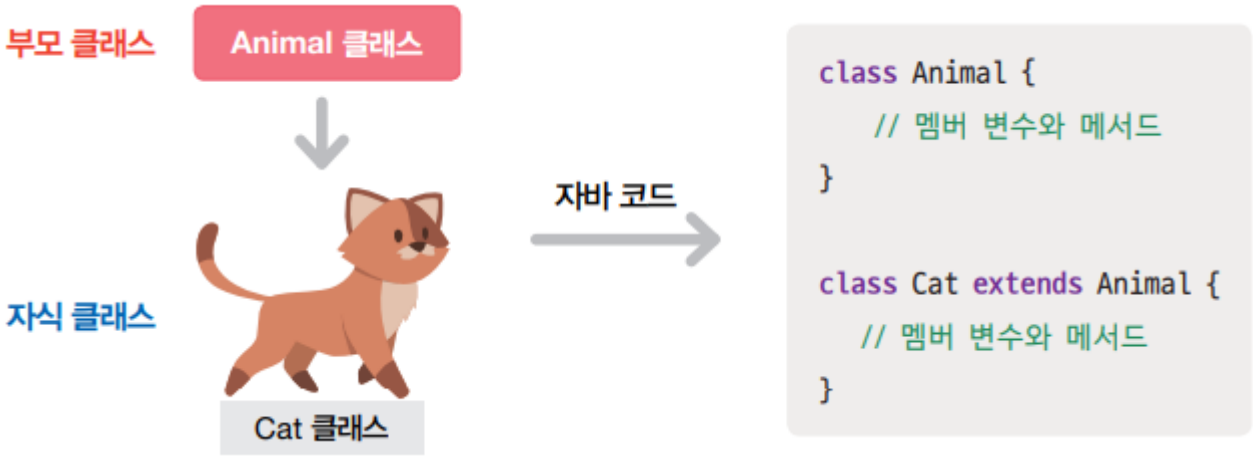
Section 02

상속의 유형

2. 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



[그림 8-5] 클래스 간의 단일 상속

2. 상속의 유형

단일 상속

단일 상속 예시

Animal.java

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

Cat.java

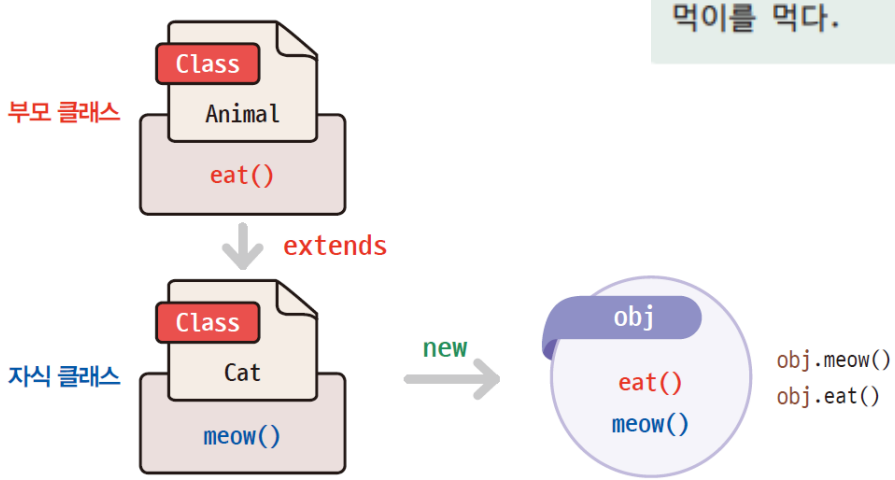
```
public class Cat extends Animal {
   void meow() {
      System.out.println("야옹하고 울다.");
   }
}
```

실행 결과

```
야옹 하고 울다.
먹이를 먹다.
```

Example02.java

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

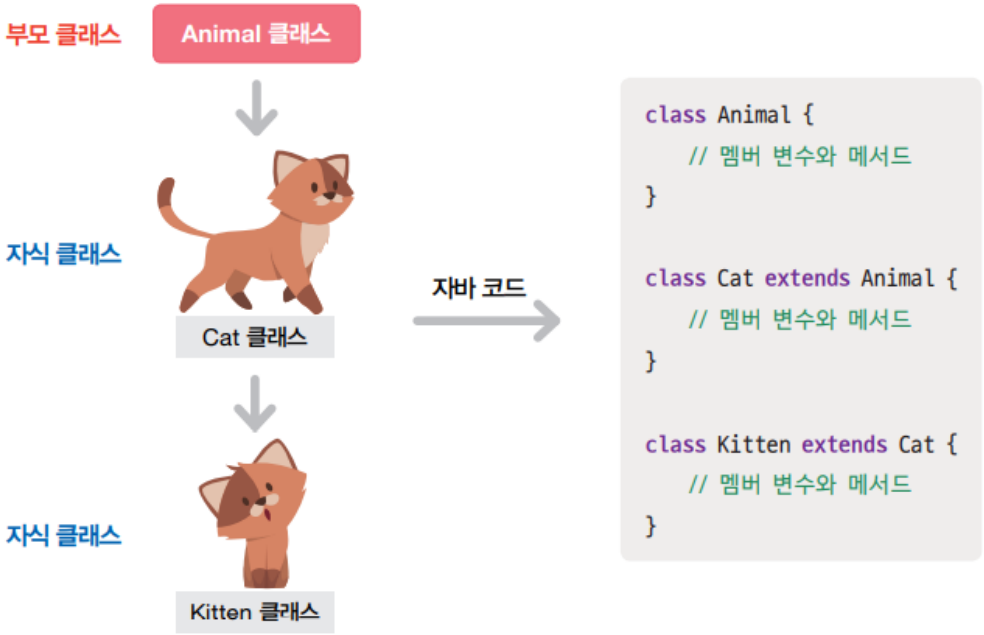


[그림 8-6] Animal 클래스에서 확장된 Cat 클래스

2. 상속의 유형

다단계 상속

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



[그림 8-8] 클래스 간의 다단계 상속

2. 상속의 유형

다단계 상속

다단계 상속 예시

Animal.java

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

Cat.java

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

Kitten.java

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

2. 상속의 유형

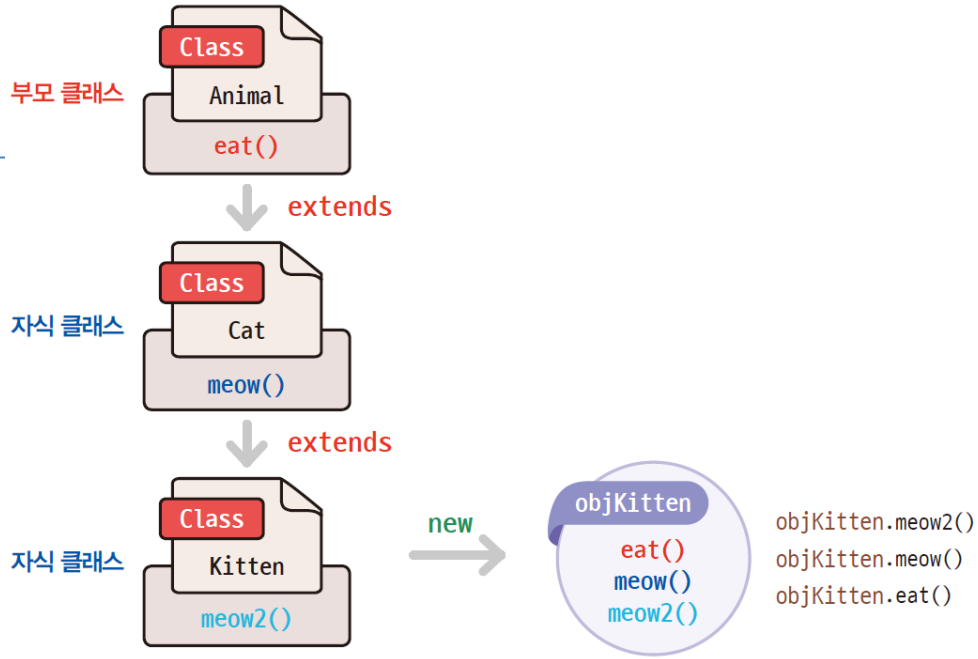
다단계 상속

Example03.java

```
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.
```

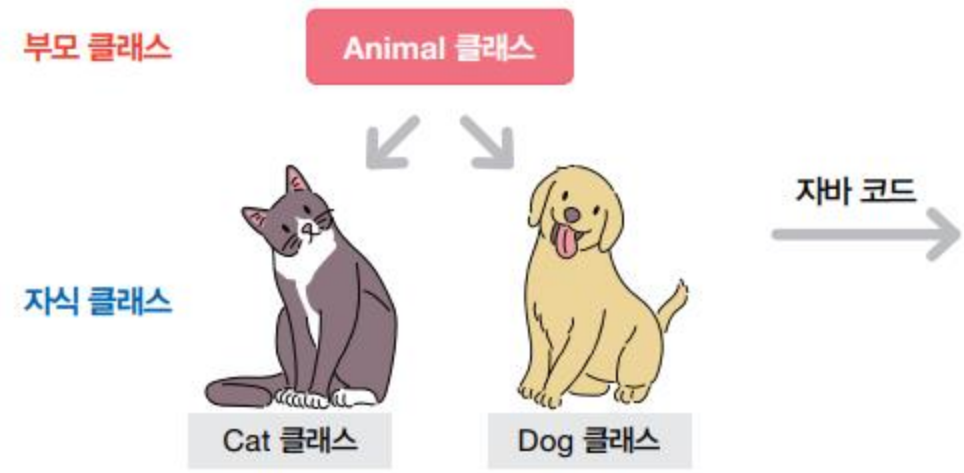


[그림 8-9] 클래스 간의 다단계 상속

2. 상속의 유형

■ 계층적 상속

- 단일 부모 클래스의 값과 메서드를 여러 자식 클래스에 전달하는 것
- 한 클래스가 부모 클래스로 사용되고 나머지 클래스는 자식 클래스가 됨




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

class Cat extends Animal {
    // 멤버 변수와 메서드
}

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

[그림 8-11] 클래스 간의 계층적 상속

2. 상속의 유형

계층적 상속

계층적 상속 예시

Animal.java

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

Cat.java

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

Dog.java

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

2. 상속의 유형

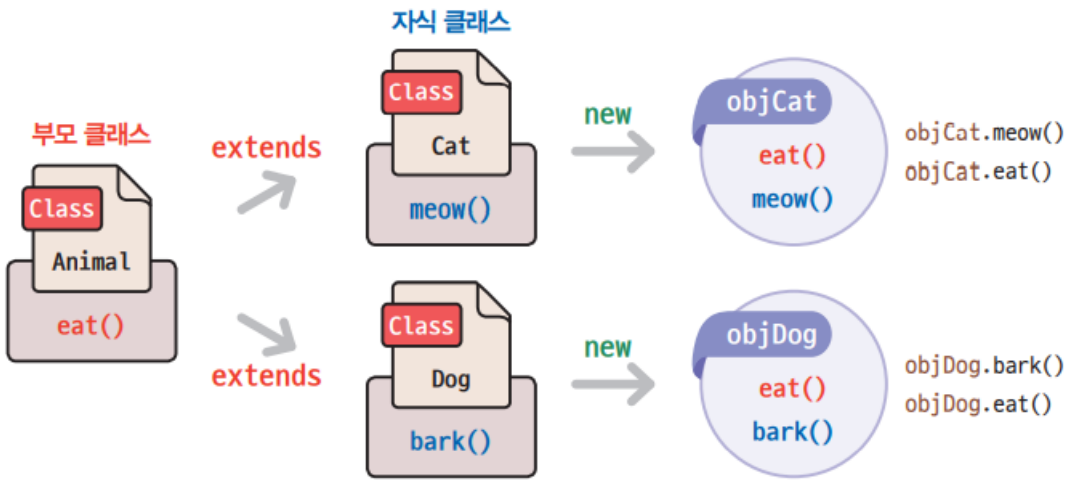
■ 계층적 상속

Example04.java

```
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.
```

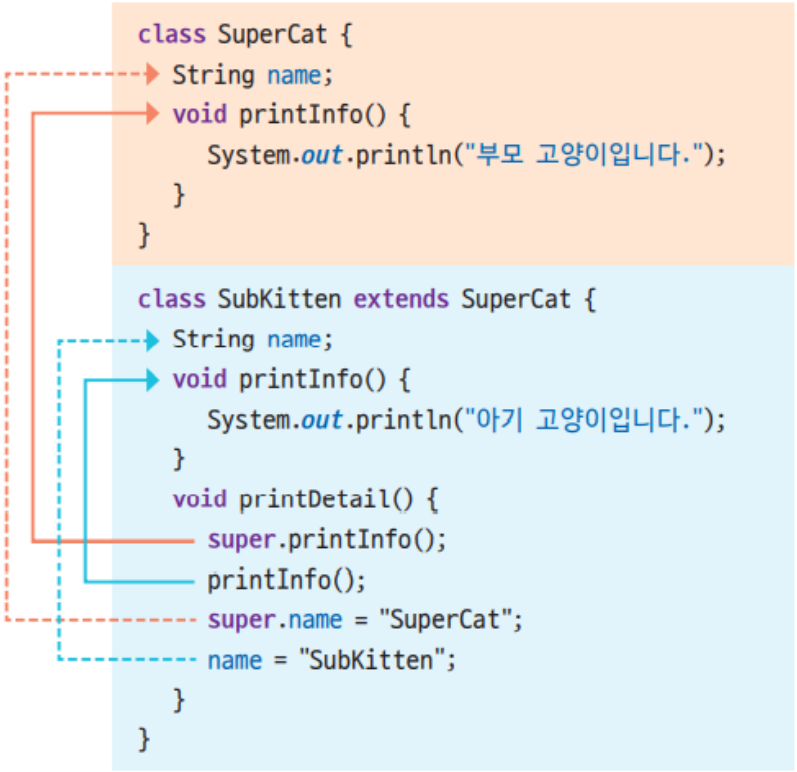


[그림 8-12] Animal, Cat, Dog 클래스 간의 계층적 상속

2. 상속의 유형

super를 이용한 부모 클래스 참조

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



[그림 8-14] supter 키워드를 이용하여 부모 클래스의 멤버 요소에 접근

2. 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

SuperCat.java

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

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

2. 상속의 유형

super를 이용한 부모 클래스 참조

- super를 이용한 변수와 메서드 접근

SubKitten.java

```
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);
    }
}
```

2. 상속의 유형

super를 이용한 부모 클래스 참조

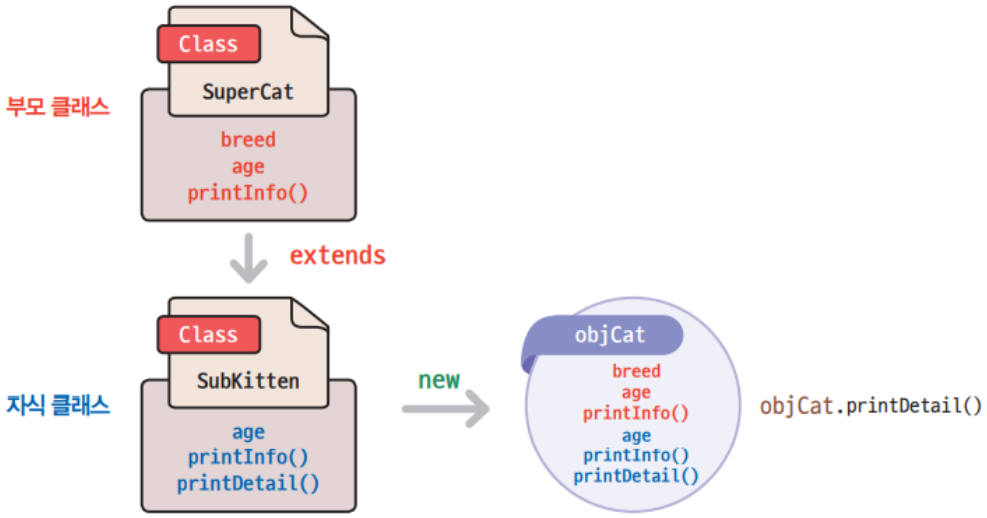
- super를 이용한 변수와 메서드 접근

Exmaple05.java

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

실행 결과

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



[그림 8-15] super 키워드를 이용하여 SubKitten에서 SuperCat의 멤버 요소에 접근

2. 상속의 유형

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

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

2. 상속의 유형

Child.java

```
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 }
```

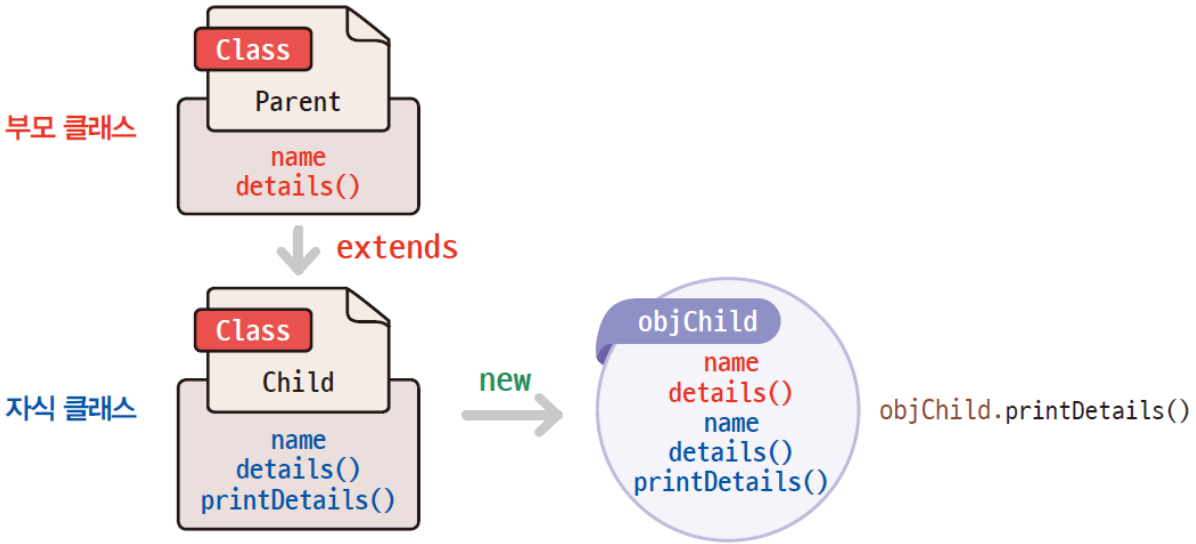
2. 상속의 유형

Inheritance05.java

```
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
```



[그림 8-16] super 키워드를 이용하여 Child에서 Parent의 멤버 요소에 접근

2. 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

2. 상속의 유형

super를 이용한 부모 클래스 참조

- super( )를 이용한 생성자 접근

super() 사용 예시

SuperCat2.java

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

2. 상속의 유형

super를 이용한 부모 클래스 참조

- super( )를 이용한 생성자 접근

SubKitten2.java

```
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);
    }
}
```

2. 상속의 유형

super를 이용한 부모 클래스 참조

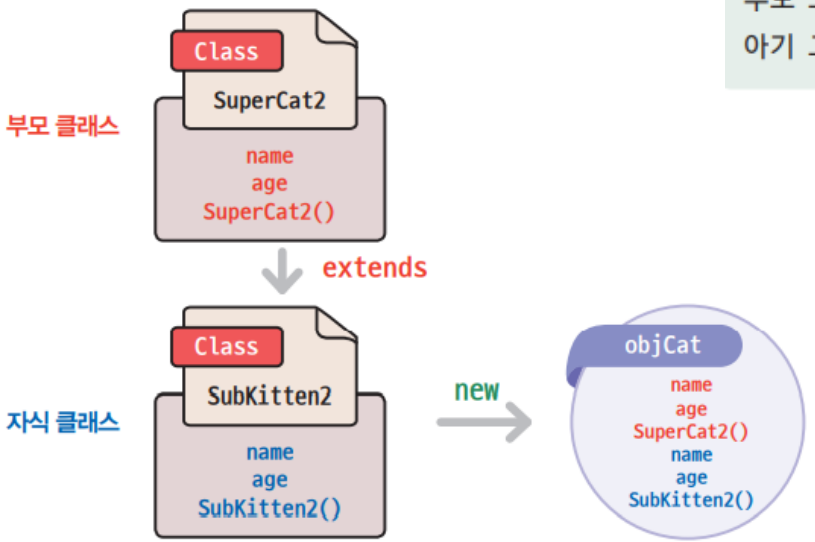
- super( )를 이용한 생성자 접근

Example06.java

```
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
```



[그림 8-17] super()를 이용하여 SubKitten2에서 SuperCat2의 생성자에 접근

2. 상속의 유형

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

Parent2.java

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

Child2.java

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

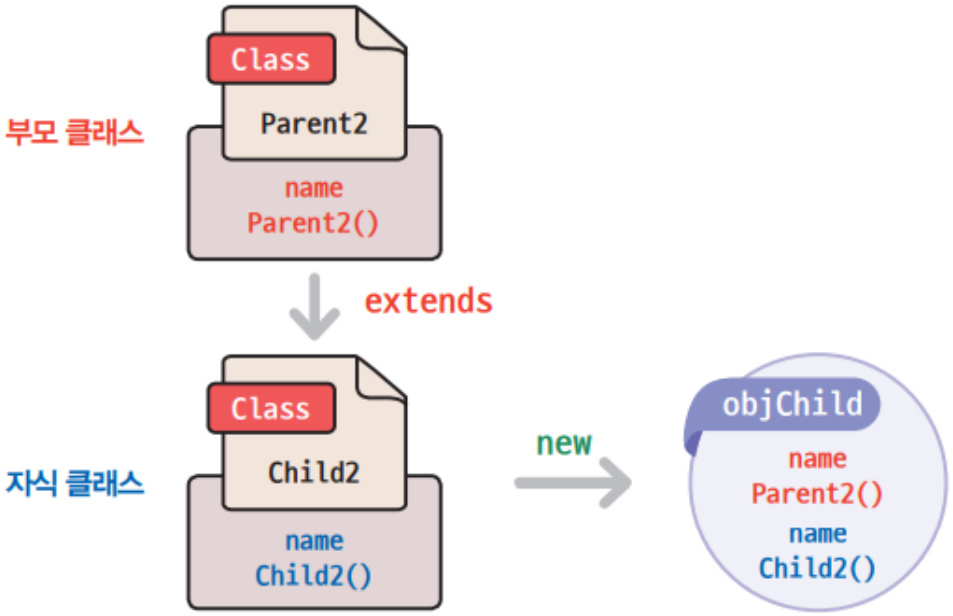
2. 상속의 유형

Inheritance06.java

```
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
```



[그림 8-18] super()를 이용하여 Child2에서 Parent2의 생성자에 접근

Section 03

다형성

3. 다형성

프로그래밍에 다형성을 적용시 장점

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

Section 04
다형성의 유형

4. 다형성의 유형

■ 컴파일타임 다형성

- 메서드 오버로딩 (method overloading)

→메서드명이 같지만 다른 매개변수를 사용하는 메서드

→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

4. 다형성의 유형

■ 컴파일타임 다형성

- 메서드 오버로딩

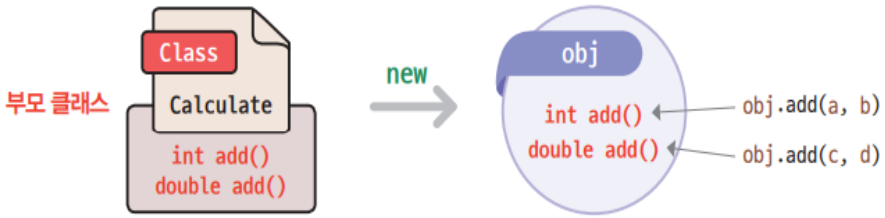
메서드 오버로딩 예시

Exampole07.java

```
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
```



[그림 8-21] add() 메서드 오버로딩

4. 다형성의 유형

예제 8-7 사각형의 넓이 구하기 Polymorphism01.java

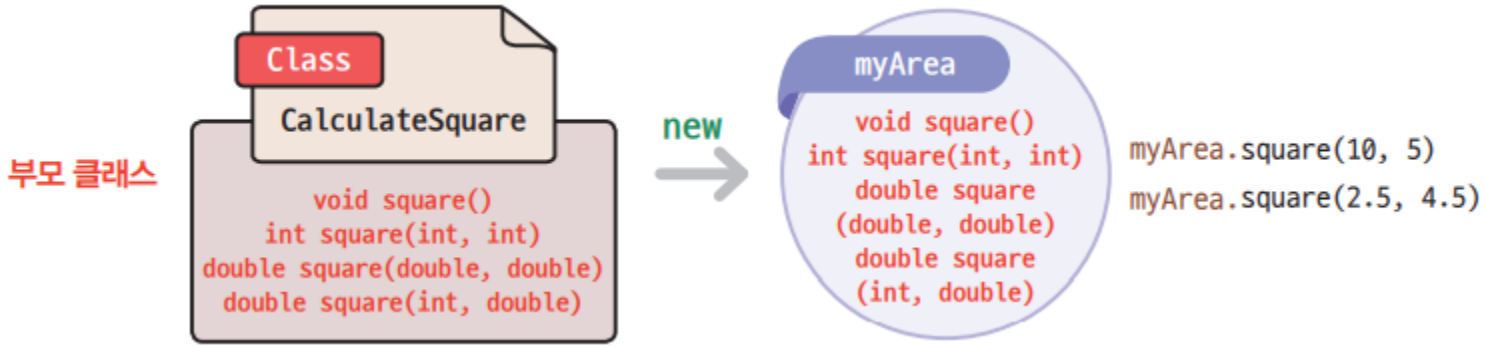
```
01 class CalculateSquare {
02     public void square() {
03        System.out.println("No Parameter Method Called");
04    }
05
06     public int square(int width, int height) {
07         int area = width * height;
08         return area;
09     }
10
11     public double square(double width, double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

4. 다형성의 유형

```
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
```



[그림 8-22] square() 메서드 오버로딩

4. 다형성의 유형

런타임 다형성

- 메서드 오버라이딩 (method overriding)

→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

4. 다형성의 유형

런타임 다형성

- 메서드 오버라이딩

메서드 오버라이딩 예시

Animal.java

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

Kitten.java

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

   }
}
```

4. 다형성의 유형

■ 런타임 다형성

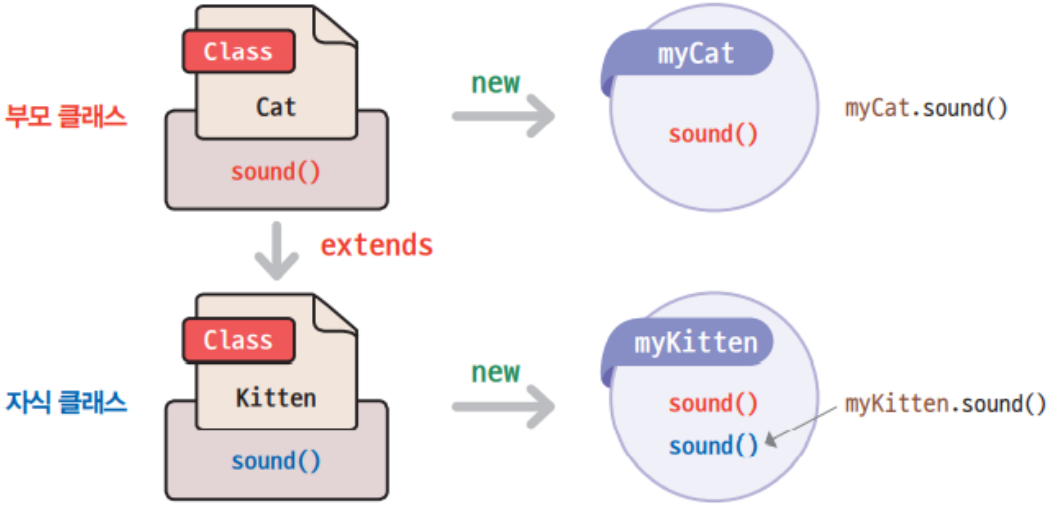
- 메서드 오버라이딩

Example08.java

```
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.
```



[그림 8-23] sound() 메서드 오버라이딩

4. 다형성의 유형

예제 8-8 동물의 울음소리 출력하기

Animal.java

```
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 }
```

Pig.java

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

4. 다형성의 유형

Dog.java

```
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 }
```

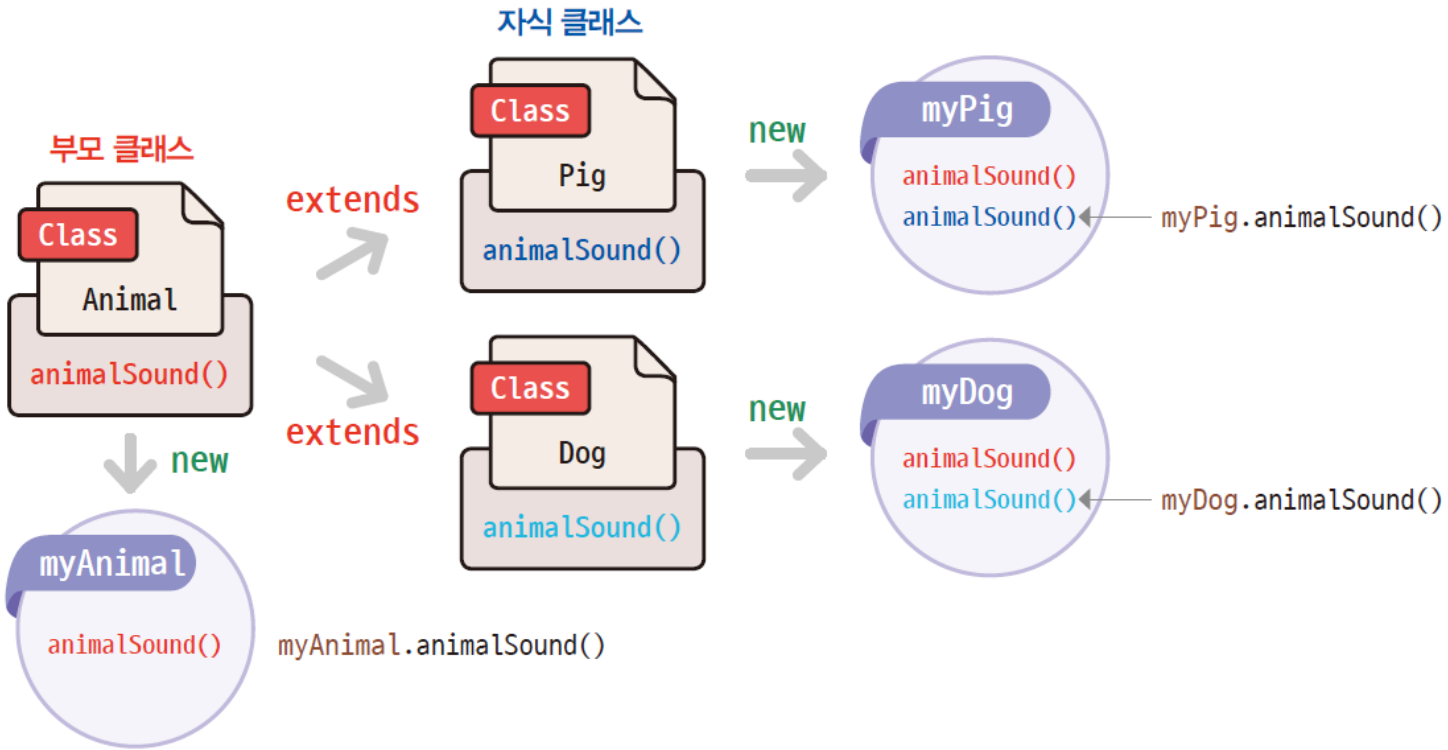
Polymorphism02.java

```
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
```

4. 다형성의 유형



[그림 8-24] animalSound() 메서드 오버라이딩

4. 다형성의 유형

Polymorphism02.java

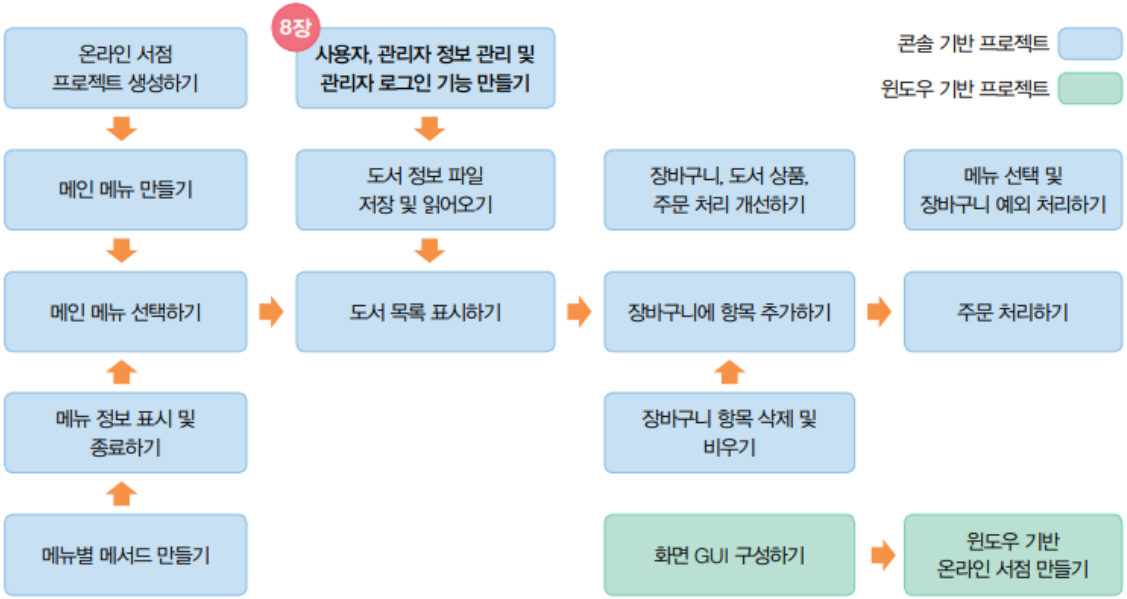
```
import java.util.Scanner;

public class Polymorphism02 {
	public static void main(String[] args) {
		Scanner input=new Scanner(System.in);
		System.out.println("동물을 선택하세요(돼지(1), 개(2):))");
		int select=input.nextInt();
		if (select==1) {
			Polymorphism02.printanimalSound(new Pig());
		}
		else if(select==2) {
			Polymorphism02.printanimalSound(new Dog());
		}
	}
	public static void printanimalSound(Animal animal) {
		animal.animalSound();
	}
}
```

[프로젝트]
사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

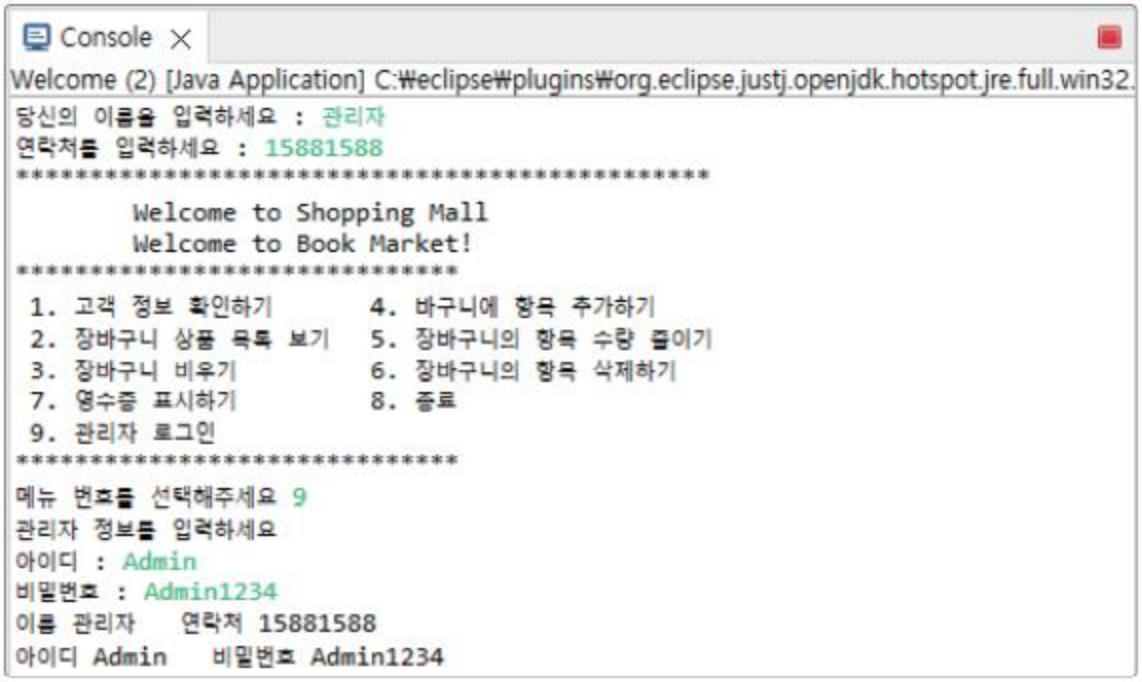
- 고객 정보를 관리하는 기존 클래스의 자식 클래스로 사용자, 관리자 클래스 를 만든 뒤, 메인 메뉴에 관리자 로그인 메뉴를 추가하여 관리자 로그인 인증을 통해 관리자 정보를 출력하게 합니다



[그림 8-25] 사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

- 고객 정보를 관리하는 기존 클래스의 자식 클래스로 사용자, 관리자 클래스 를 만든 뒤, 메인 메뉴에 관리자 로그인 메뉴를 추가하여 관리자 로그인 인증을 통해 관리자 정보를 출력하게 합니다



[그림 8-25] 관리자 로그인 인증 실행 화면

사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

■ 사용자 정보 관리하기

- **01 사용자 클래스 생성하기**

  → User.java 파일을 생성하고 Person 클래스로부터 상속받는 자식 클래스인 User를 만듦

- **02 사용자 정보 저장하기**

  →Welcome.java 파일의 main()에서 User 클래스의 객체를 생성하여 입력 받은 고객의 정보 저장

- **03 사용자 정보 출력하기**

  →Welcome.java 파일에서 고객 정보를 확인하는 menuGuestInfo() 메서드 수정

사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

관리자 정보 관리하기

- **01 관리자 클래스 생성하기**

→Admin.java 파일을 생성하고 Person 클래스로부터 상속받는 자식 클래스인 Admin을 만듦

사용자, 관리자 정보 관리 및 관리자 로그인 기능 만들기

관리자 로그인 기능 만들기

- **01 관리자 로그인 메뉴 만들기**

  →관리자 로그인 메뉴 추가를 위해 Welcome.java 파일에서 menuIntroduction() 메서드 수정

- **02 관리자 로그인 메뉴 선택하기**

  →관리자가 로그인 메뉴를 선택하여 실행할 수 있도록 Welcome. java 파일을 수정

- **03 관리자 로그인 정보 확인하기**

  →관리자의 로그인을 위한 menuAdminLogin() 메서드를 Welcome.java 파일에 추가