

# Section 01

## 객체지향 프로그래밍

# 1. 객체지향 프로그래밍

## ■ 객체지향 프로그래밍의 개념

- 프로그래밍에서 객체를 사용하는 것을 의미함
- 데이터와 메서드를 포함하는 '객체' 개념에 기반한 프로그래밍
- 프로그램의 유연성을 향상하고 유지·관리의 가능성을 높임
- 데이터 및 데이터의 동작(메서드)을 단일 위치(객체)에 통합하여 프로그램 작동 방식을 더 쉽게 이해할 수 있음

## ■ 객체지향 프로그래밍의 장점

- 개발 속도 향상
- 소프트웨어 유지·관리 향상
- 소프트웨어 개발의 생산성 향상
- 개발 비용 절감

# 1. 객체지향 프로그래밍

## ■ 객체지향 프로그래밍의 구성 요소

[표 7-1] 객체지향 프로그래밍의 구성 요소

구성 요소	설명
클래스	같은 종류의 집단에 속한 속성과 행동을 정의한 틀이다.
객체	클래스의 인스턴스이다.
캡슐화	데이터와 행동을 하나의 단위로 묶는 기법이다.
상속	이미 존재하는 한 클래스의 멤버(변수, 메서드)를 다른 클래스에 물려주는 기법이다.
다형성	변수, 메서드 또는 객체가 여러 형태를 취하는 기법이다.
추상화	불필요한 내부 세부 사항을 숨기고 필수 사항을 표시하는 것을 의미한다.



[그림 7-1] 객체지향 프로그래밍의 구성 요소

# 1. 객체지향 프로그래밍

## ■ 캡슐화

- 데이터(변수)와 행동(메서드)을 하나의 단위(클래스)로 묶는 기법
- 데이터를 수정하지 못하도록 안전하게 하기 위해 정보의 세부 사항을 숨기고 객체의 데이터와 메서드를 보호하는 과정을 의미함
- [예] 캡슐 안에 담긴 약
  - 캡슐 속 약이 안전한 것처럼 통해 클래스의 변수와 메서드를 숨겨 안전하게 보호함



[그림 7-2] 캡슐화의 예

# 1. 객체지향 프로그래밍

## ■ 캡슐화

- 캡슐화 규칙

→ 클래스의 변수는 private 접근제한자로 선언함

→ 클래스의 변수에 접근할 수 있는 public 접근제한자로 선언한 Getter(), Setter() 메서드를 제공함

```
class Encapsulation {  
    private String name;  
  
    public String getName() { 시작  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    } 끝  
}
```

# 1. 객체지향 프로그래밍

## ■ 상속

- 이미 존재하는 한 클래스의 멤버(변수, 메서드)를 다른 클래스에 물려주는 기법
- 코드를 재사용하고 두 클래스 간에 부모-자식 관계를 설정하는 데 도움을 줌
- [예] 새끼 고양이는 부모 고양이의 생김새와 습성을 상속받음



[그림 7-3] 상속의 예

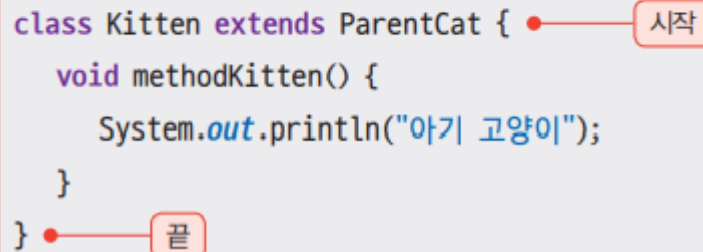
# 1. 객체지향 프로그래밍

## ■ 상속

- 부모 클래스(상위 클래스, 슈퍼 클래스, 기본 클래스): 해당 클래스가 포함하고 있는 멤버를 상속하는 클래스
- 자식 클래스(하위 클래스, 서브 클래스, 파생 클래스): 부모 클래스의 멤버를 상속받는 클래스

```
class ParentCat {  
    public String breed = "삼 고양이";  
    void methodParentCat() {  
        System.out.println("어미 고양이");  
    }  
}
```

```
class Kitten extends ParentCat {  
    void methodKitten() {  
        System.out.println("아기 고양이");  
    }  
}
```



# 1. 객체지향 프로그래밍

## ■ 다형성

- 여러 형태를 취한다는 의미를 가짐
- 객체지향 프로그래밍에서 다형성은 변수, 메서드, 객체가 여러 형태를 취하는 기법
- 상속에 의해 서로 관련된 하나 이상의 클래스 또는 객체가 있을 때 다형성이 발생
- [예] 한 사람이 고양이를 키우는 집사이면서 직장인, 자녀, 부모이기도 하여 다양한 상황에서 다른 행동을 함



[그림 7-4] 다형성의 예



# 1. 객체지향 프로그래밍

## ■ 다형성 유형

- 메서드 오버로딩

→ 메서드명이 같지만 메서드에 전달된 매개변수의 자료형 또는 개수가 다른 둘 이상의 메서드를 가짐

```
public class CatButler {  
    public void myRole(int age) {  
        System.out.println("나는 고양이 집사입니다. 나이는 " + age + "입니다");  
    }  
  
    public void myRole(String name) {  
        System.out.println(name + "은 학생입니다.");  
    }  
  
    public void myRole(String name, int age) {  
        System.out.println(name + "은 자녀입니다. 나이는 " + age + "입니다");  
    }  
}
```

메서드 오버로딩

메서드 오버로딩

# 1. 객체지향 프로그래밍

## ■ 다형성 유형

- 메서드 오버라이딩

→ 상위 클래스의 메서드를 하위 클래스에서 재정의함

```
class Animal {  
    public void Sound() {  
        System.out.println("동물의 울음소리");  
    }  
}  
  
class Cat extends Animal {  
    public void Sound() {  
        System.out.println("고양이는 야옹야옹");  
    }  
}  
  
class Dog extends Animal {  
    public void Sound() {  
        System.out.println("강아지는 멍멍멍");  
    }  
}
```

메서드 오버라이딩

메서드 오버라이딩

# 1. 객체지향 프로그래밍

## ■ 추상화

- 불필요한 세부 정보를 포함하지 않고 객체의 공통적인 속성과 행동만을 나타내는 것을 의미함
- 불필요한 세부 정보를 숨겨서 프로그래밍의 복잡성과 노력을 줄임

# 1. 객체지향 프로그래밍

## ■ 추상화를 하는 방법

- 추상 클래스(abstract class)를 이용하는 방법
  - abstract 키워드로 선언하는 추상 클래스
    - ✓ 추상 메서드와 구체적인 메서드를 포함할 수 있음
  - 추상 클래스는 인스턴스화할 수 없으므로 추상 클래스의 객체를 만들 수 없음
  - 추상 클래스를 사용하려면 다른 클래스에 상속하여 추상 메서드 자체를 구현해야 함

```
abstract class Animal {  
    abstract void Sound();  
  
    void Sleep() {  
        System.out.println("zzz");  
    }  
}
```

# 1. 객체지향 프로그래밍

## ■ 추상화를 하는 방법

- 인터페이스(interface)를 이용하는 방법

→ interface 키워드를 사용하여 선언하는 인터페이스

✓ interface 키워드는 클래스가 구현해야 하는 동작을 지정하는 데 사용되는 추상 자료형

→ 인터페이스를 사용하려면 다른 클래스에서 인터페이스를 구현해야 함

```
public interface Animal {  
    public void Sound();  
    public void Sleep();  
}  
  
public class Cat implements Animal {  
    public void Sound() {  
        System.out.println("고양이 울음소리는 야옹야옹");  
    }  
  
    public void Sleep() {  
        System.out.println("zzz");  
    }  
}
```

# Section 02

## 클래스와 객체

## 2. 클래스와 객체

### ■ 클래스와 객체의 관계

- 객체지향 프로그래밍은 객체를 조립하여 프로그램을 만드는 것
- [객체의 예] 고양이 종이 장난감
  - 고양이의 머리, 몸통, 꼬리를 종이 접기한 후 이를 조립하여 완성



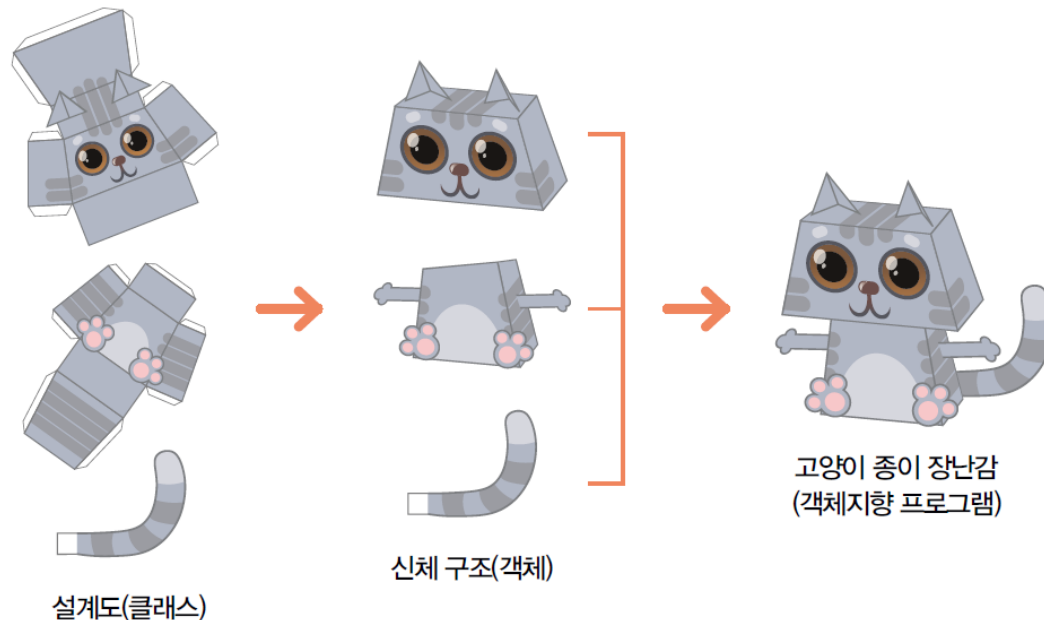
[그림 7-6] 클래스와 객체의 관계

## 2. 클래스와 객체

### ■ 클래스와 객체의 관계

- [클래스의 예] 고양이 종이 장난감 설계도

→ 고양이 종이 장난감을 만들기 위한 머리, 몸통, 꼬리는 설계도에 따라 만듦



[그림 7-7] 클래스와 객체의 구성

→ 클래스는 객체를 만들기 위한 설계도 또는 템플릿



## 2. 클래스와 객체

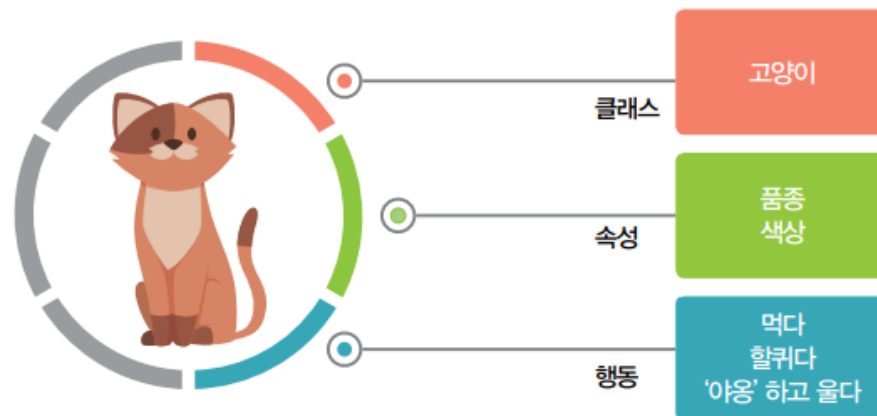
### ■ 클래스 설계 및 구현

- 클래스 설계
- [예] 고양이의 속성과 행동



[그림 7-8] 고양이의 속성과 행동 예

→ 고양이의 속성과 행동으로 클래스를 설계



[그림 7-9] 고양이에 대한 클래스 설계

## 2. 클래스와 객체

### ■ 클래스 설계 및 구현

- 클래스 구현

- class 키워드를 사용하여 클래스를 구현함
- 클래스의 본문을 중괄호({ })로 묶음
- 클래스의 구성 요소는 변수, 메서드 등

```
[접근제한자] class 클래스명 {  
    (멤버)변수  
    (멤버)메서드  
}
```

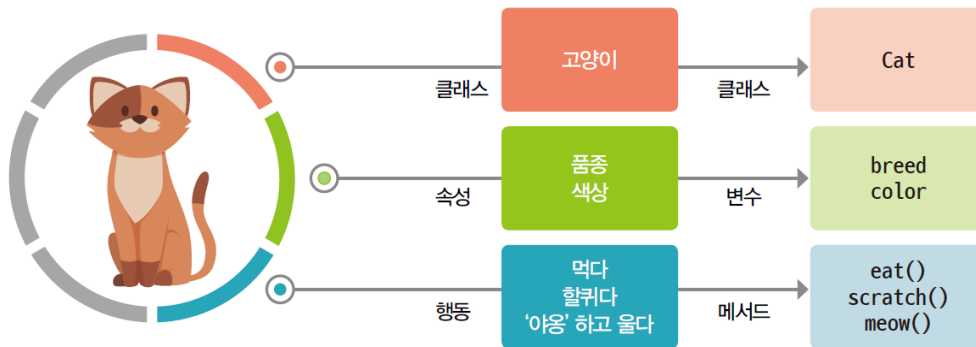
- ✓ 모든 클래스에 변수(데이터)와 메서드가 있는 것은 아님
- ✓ 변수만 포함된 클래스도 있고 메서드만 포함된 클래스도 있음
- ✓ 클래스가 수행해야 하는 작업에 따라 멤버 구성 요소를 다르게 선언함

## 2. 클래스와 객체

### ■ 클래스 설계 및 구현

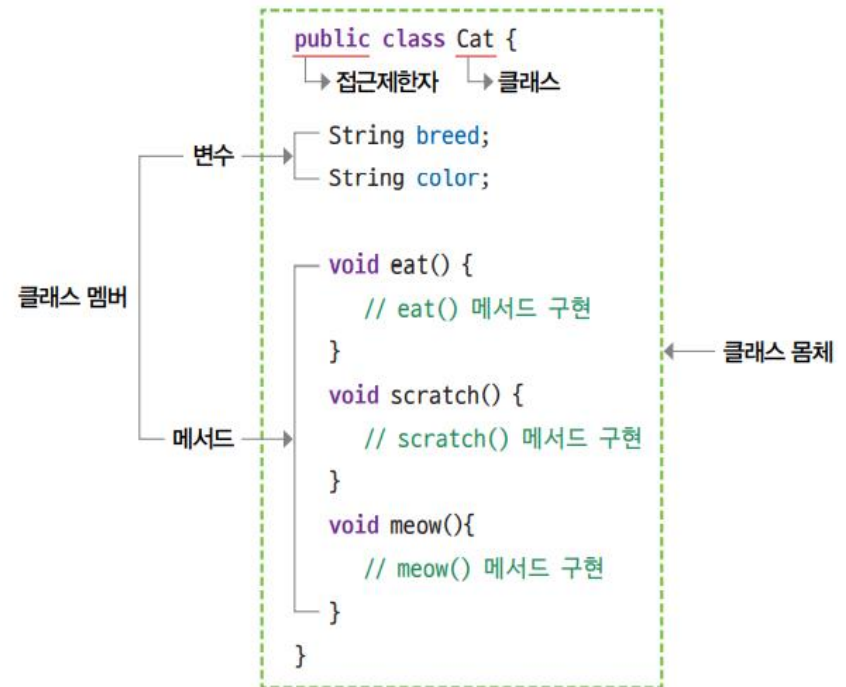
- [클래스 구현 예] 고양이에 대한 클래스 설계

→ Cat 클래스로 구현



[그림 7-10] Cat 클래스의 구현 예

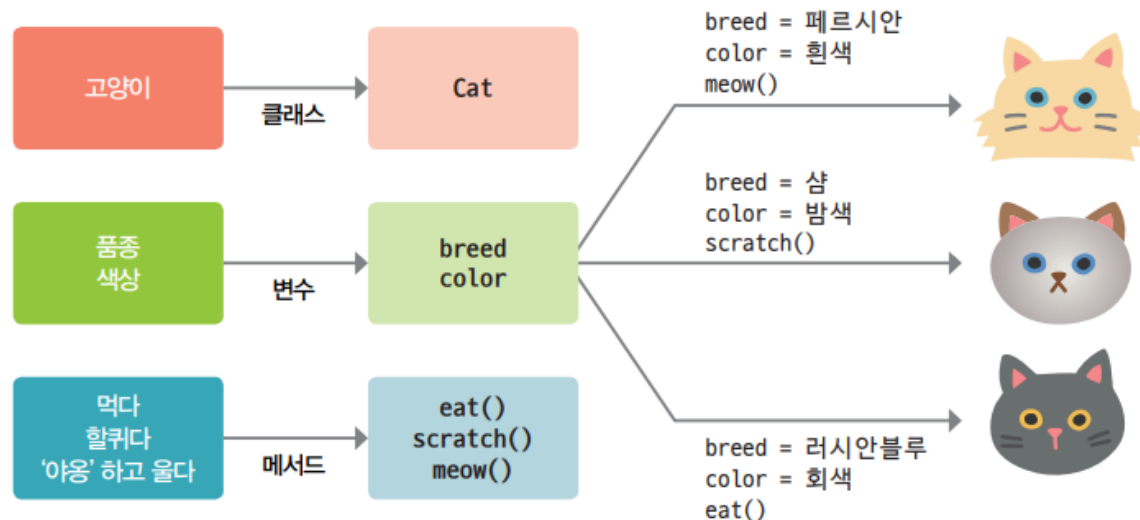
### Cat 클래스 구현 예시



## 2. 클래스와 객체

### ■ 객체

- 클래스의 인스턴스로 클래스로 생성한 실체
- 특정 유형의 데이터를 유용하게 만드는 메서드와 변수(속성)로 구성됨
- [예] 고양이 클래스
  - 속성(품종, 색깔)의 다른 값에 대해 다른 고양이 객체를 얻을 수 있음



[그림 7-11] Cat 클래스의 객체

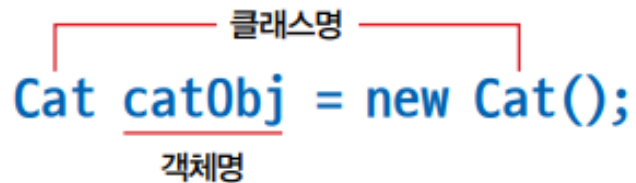
## 2. 클래스와 객체

### ■ 객체 생성

- new 키워드를 사용하여 클래스를 통해 객체를 생성함

```
클래스명 객체명 = new 클래스명();  
Cat catObj = new Cat();
```

- [예] Cat 클래스의 객체 생성



The diagram shows the code `Cat catObj = new Cat();` with annotations. A red bracket above the code spans from `Cat` to `Cat()` and is labeled "클래스명" (class name). A red underline is placed under `catObj` and is labeled "객체명" (object name).

[그림 7-12] Cat 클래스의 객체 생성 예

## 2. 클래스와 객체

### 예제 7-1

### 사람에 대한 클래스 설계 및 구현하기

Person.java

```
01 public class Person {  
02     String name;  
03     int reg_num;  
04  
05     void walk() {  
06         // walk() 메서드 구현  
07     }  
08     void run() {  
09         // run() 메서드 구현  
10     }  
11 }
```

# Section 03

## 클래스의 구성 요소

### 3. 클래스의 구성 요소

#### ■ 클래스의 구성요소

- 멤버 변수(member variable)
  - 객체의 데이터가 저장되는 곳
- 멤버 메서드(member method)
  - 객체의 동작, 데이터의 조작이 이루어지는 곳
- 생성자(constructor)
  - 객체를 생성할 때 초기화되는 항목 관리(특별한 메서드)



### 3. 클래스의 구성 요소

#### ■ 멤버 변수

- 객체의 데이터가 저장되는 곳
- 클래스의 속성으로 클래스 내의 멤버 변수 또는 필드(field)라고 부름
- 멤버 변수는 객체의 데이터를 저장하는 곳을 의미함

자료형 멤버변수명 = 초깃값; 생략 가능

- [예] Cat 클래스의 멤버 변수

```
public class Cat {  
    String breed;  
    String color;  
    int age = 10;  
}
```

멤버 변수 breed, color, age

### 3. 클래스의 구성 요소

#### ■ 멤버 변수

- 멤버 변수 접근

#### 멤버 메서드 작성 예시

Cat.java

```
public class Cat {  
    String breed;  
    String color;  
    void eat() {  
        System.out.println("먹이를 먹다");  
    }  
    void scratch() {  
        System.out.println("발톱으로 할퀴다");  
    }  
    void meow() {  
        System.out.println("야옹 하고 울다");  
    }  
}
```

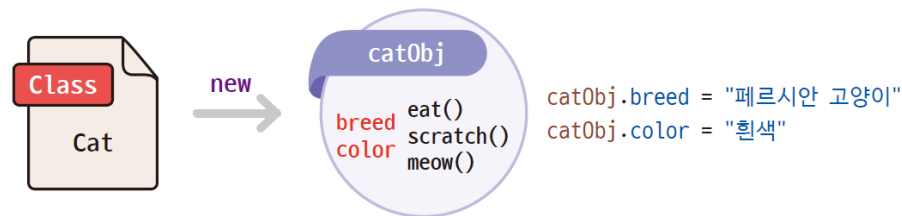
### 3. 클래스의 구성 요소

#### ■ 멤버 변수

- 멤버 변수 접근

→ 클래스 내의 멤버 변수는 객체를 만들고 마침표(.) 구문을 사용하여 접근 가능

객체명.멤버변수명;



[그림 7-13] catObj 객체에서 접근 가능한 멤버 변수

#### catObj 객체 생성 예시

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

#### 실행 결과

품종 : 페르시안 고양이  
색상 : 흰색

### 3. 클래스의 구성 요소

#### ■ 멤버 메서드

- 메서드의 유형

→ 메서드는 클래스 내에서 선언되고 특정 작업을 수행하는 데 사용됨

→ 메서드는 선언부와 중괄호 안의 구현부로 구성되며, 호출되면 중괄호 안의 모든 내용이 순차적으로 실행됨

[표 7-2] 메서드의 유형

유형	설명
정적 메서드	<ul style="list-style-type: none"><li>• static 키워드를 선언하는 메서드</li><li>• 객체를 생성하지 않고 바로 사용한다.</li></ul>
인스턴스 메서드	<ul style="list-style-type: none"><li>• static 키워드를 선언하지 않는 메서드</li><li>• 객체를 생성해야만 사용할 수 있다.</li></ul>

### 3. 클래스의 구성 요소

#### ■ 멤버 메서드

- 멤버 메서드 접근

→ 클래스 내의 멤버 메서드는 멤버 변수와 마찬가지로 객체를 만들고 마침표(.) 구문을 사용하여 접근 가능함

```
객체명.멤버메서드명();
```

### 3. 클래스의 구성 요소

#### 멤버 메서드 선언 예시

```
public class Example02 {  
    public static void myMethod1() {  
        System.out.println("정적 메서드 호출");  
    }  
    public void myMethod2() {  
        System.out.println("인스턴스 메서드 호출");  
    }  
}
```

### 3. 클래스의 구성 요소

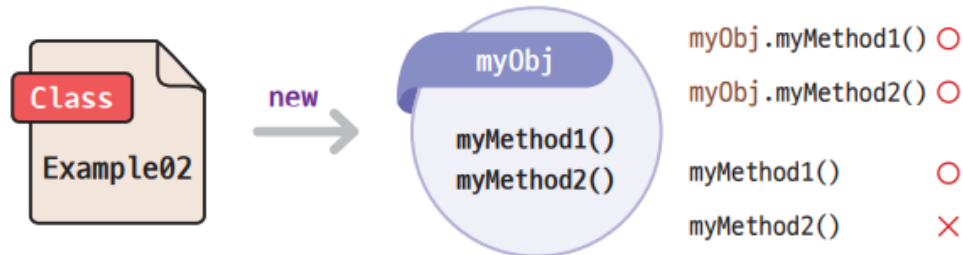
#### 멤버 메서드 선언 예시

```
public class Example01 {  
    public static void main(String[] args) {  
        Example02.myMethod1();  
        // Example02.myMethod2();  
        Example02 myObj = new Example02();  
        myObj.myMethod2();  
    }  
}
```

#### 실행 결과

정적 메서드 호출

인스턴스 메서드 호출



[그림 7-14] myObj 객체에서 접근 가능한 메서드

### 3. 클래스의 구성 요소

#### 예제 7-2 클래스의 멤버 변수와 멤버 메서드 호출하기

Student.java

```
01 public class Student {  
02     int id;  
03     String name;  
04  
05     void insertRecord(int parm1, String parm2) {  
06         id = parm1;  
07         name = parm2;  
08     }  
09  
10     void printInfo() {  
11         System.out.println("아이디: " + id);  
12         System.out.println("이름: " + name);  
13     }  
14 }
```



### 3. 클래스의 구성 요소

Object01.java

```
01 public class Object01 {  
02     public static void main(String[] args) {  
03         Student stObj1 = new Student();  
04  
05         stObj1.id = 20221004;  
06         stObj1.name = "홍길순";  
07         stObj1.printInfo();  
08  
09         Student stObj2 = new Student();  
10         stObj2.insertRecord(20021005, "홍길동");  
11         stObj2.printInfo();  
12     }  
13 }
```

#### 실행 결과

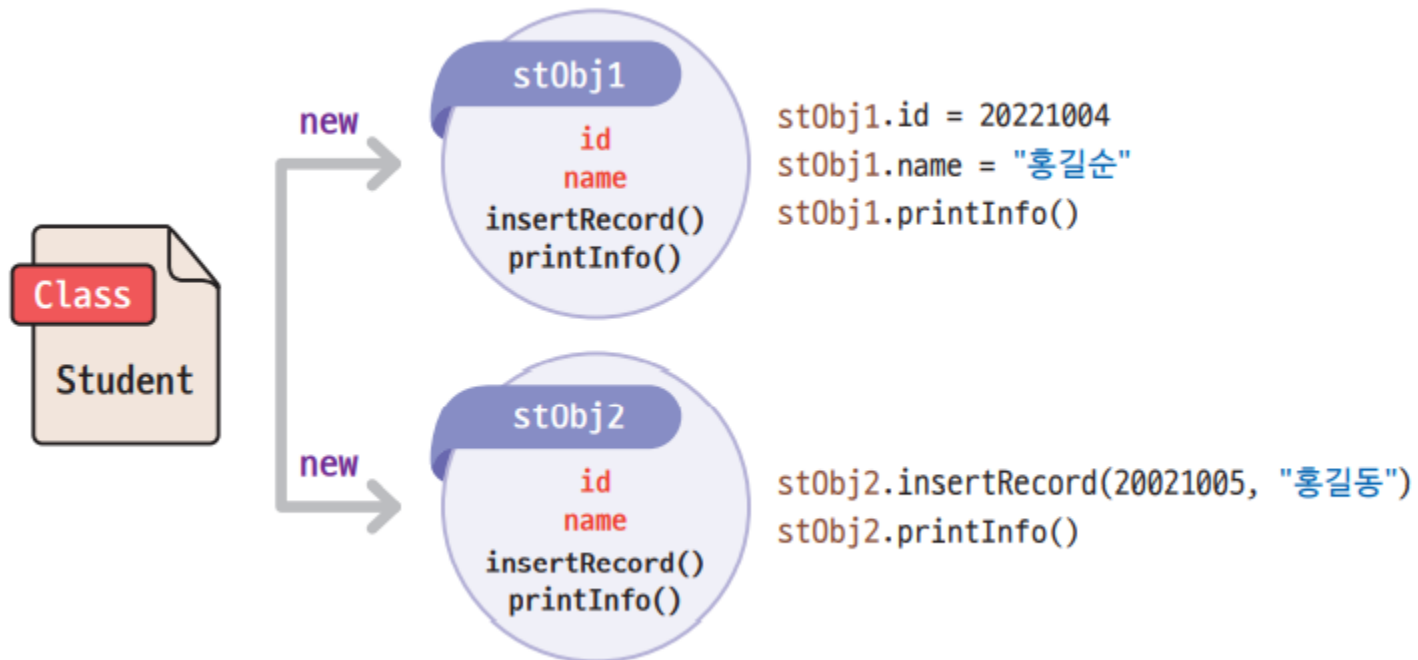
아이디: 20221004

이름: 홍길순

아이디: 20021005

이름: 홍길동

### 3. 클래스의 구성 요소



[그림 7-16] 멤버 변수값을 추가하는 두 가지 방법

# Section 04

## 생성자

## 4. 생성자

### ■ 생성자

- 객체를 생성할 때 new 연산자를 사용하여 호출되는 부분
- 클래스의 객체가 생성될 때 마다 자동으로 호출되고 객체를 초기화하는 데 이용되는 특수한 유형의 메서드
- 생성자명은 클래스명과 같고 반환 유형이 없음

### ■ 생성자를 작성하는 규칙

- 생성자명은 해당 클래스명과 동일해야 함
- 생성자는 abstract, final, static을 선언할 수 없음
- 접근제한자는 생성자의 접근을 제어하는 데 사용할 수 있음. 즉 다른 클래스가 생성자를 호출할 수 있음

## 4. 생성자

### ■ 기본 생성자(default constructor)

- 매개변수가 없는 생성자
- 클래스 내에 기본 생성자를 정의 하지 않아도 됨
  - 클래스 내에 기본 생성자를 정의하지 않으면 컴파일러는 해당 클래스에 대한 기본 생성자를 자동으로 생성함(어떤 생성자도 정의하지 않았을 때)

```
클래스명();
```

## 4. 생성자

### ■ 기본 생성자(default constructor)

#### 기본 생성자 선언 예시

Cat.java

```
public class Cat {  
    String breed;  
    String color;  
    Cat() {  
        System.out.println("Cat() 생성자 호출합니다.");  
    }  
    void eat() {  
        System.out.println("먹이를 먹다.");  
    }  
    void scratch() {  
        System.out.println("발톱으로 할퀴다.");  
    }  
    void meow() {  
        System.out.println("야옹하고 울다.");  
    }  
}
```

## 4. 생성자

### ■ 기본 생성자(default constructor)

#### 기본 생성자 호출 예시

Example04.java

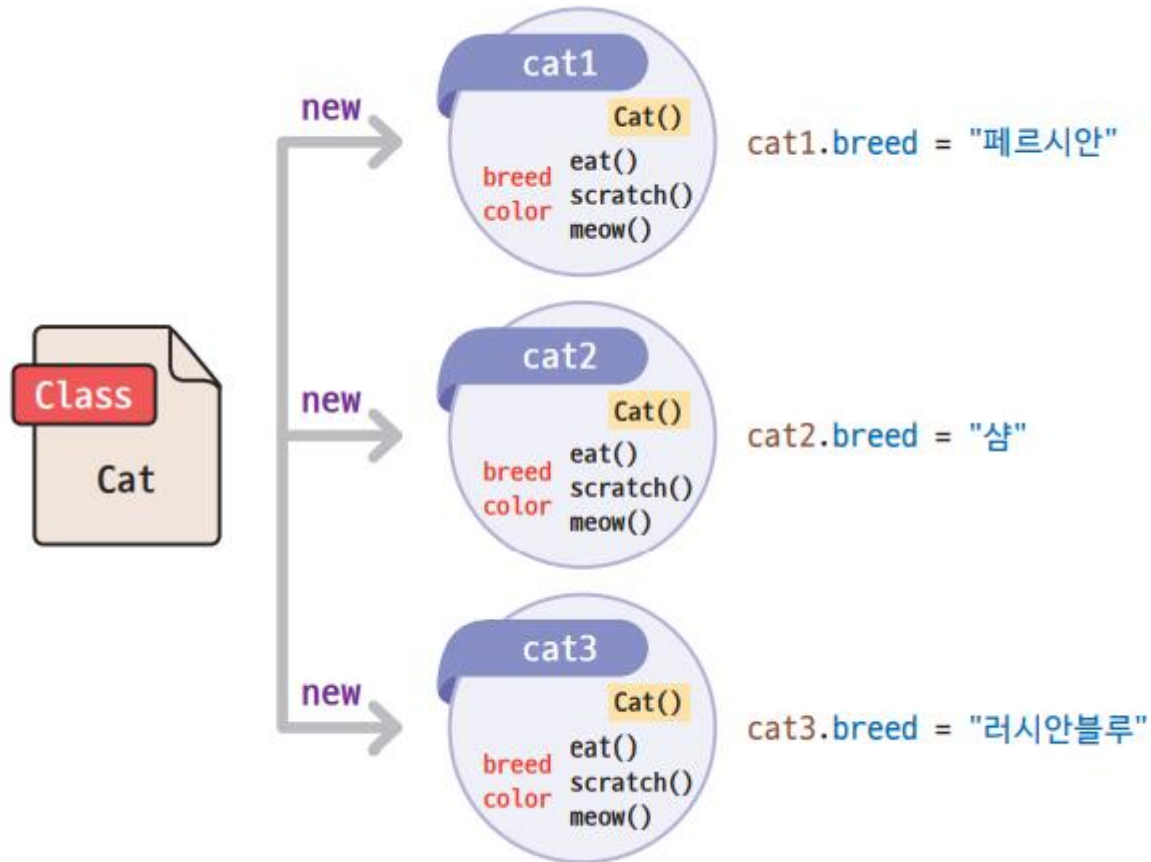
```
public class Example04 {  
    public static void main(String[] args) {  
        Cat cat1 = new Cat() ;  
        cat1.breed = "페르시안";  
        Cat cat2 = new Cat() ;  
        cat2.breed = "삼";  
        Cat cat3 = new Cat() ;  
        cat3.breed = "러시안블루";  
        System.out.println("첫 번째 고양이 품종 : " + cat1.breed);  
        System.out.println("두 번째 고양이 품종 : " + cat2.breed);  
        System.out.println("세 번째 고양이 품종 : " + cat3.breed);  
    }  
}
```

#### 실행 결과

Cat() 생성자 호출합니다.  
Cat() 생성자 호출합니다.  
Cat() 생성자 호출합니다.  
첫 번째 고양이 품종 : 페르시안  
두 번째 고양이 품종 : 삼  
세 번째 고양이 품종 : 러시안블루

## 4. 생성자

### ■ 기본 생성자(default constructor)



[그림 7-17] 기본 생성자 호출1



## 4. 생성자

### 예제 7-3

### 기본 생성자를 호출하여 객체 생성하기

Student.java

```
01 public class Student {  
02     int id;  
03     String name;  
04  
05     Student() {  
06         System.out.println("기본 생성자 Student() 호출");  
07     }  
08  
09     void insertRecord(int parm1, String parm2) {  
10         id = parm1;  
11         name = parm2;  
12     }  
13  
14     void printInfo() {  
15         System.out.println("아이디 : "+ id);  
16         System.out.println("이름 : "+ name);  
17     }  
18 }
```

## 4. 생성자

Constructor01.java

```
01 public class Constructor01 {  
02     public static void main(String[] args) {  
03         System.out.println("****학생 주소록****");  
04         Student stObj = new Student();  
05  
06         stObj.insertRecord(20221004, "홍길순");  
07         stObj.printInfo();  
08     }  
09 }
```

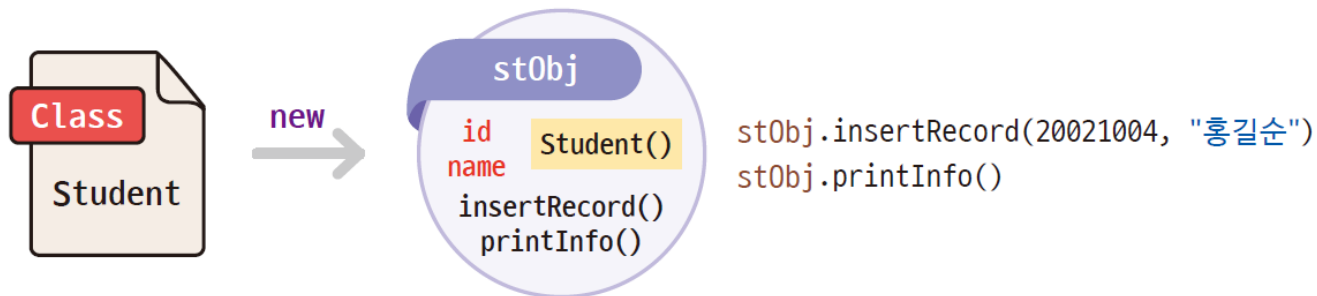
### 실행 결과

\*\*\*\*학생 주소록\*\*\*\*

기본 생성자 Student() 호출

아이디 : 20221004

이름 : 홍길순



[그림 7-18] 기본 생성자 호출2

## 4. 생성자

### ■ 일반 생성자

- 매개변수가 있는 생성자
- 기본 생성자와 달리 생략할 수 없으며 클래스의 멤버 변수를 초기화하는 데 사용됨

```
클래스명(매개변수1, 매개변수2, ...);
```

## 4. 생성자

### ■ 일반 생성자

#### 기본 생성자와 일반 생성자 선언 예시

Cat.java

```
public class Cat {  
    String breed;  
    String color;  
    Cat() {  
        System.out.println("Cat() 생성자 호출합니다.");  
    }  
    Cat(String pBreed) {  
        System.out.println("Cat(...) 생성자 호출합니다.");  
        breed = pBreed;  
    }  
    void eat() {  
        System.out.println("먹이를 먹다.");  
    }  
    void scratch() {  
        System.out.println("발톱으로 할퀴다.");  
    }  
    void meow() {  
        System.out.println("야옹 하고 울다.");  
    }  
}
```

## 4. 생성자

### ■ 일반 생성자

#### 기본 생성자와 일반 생성자 호출 예시

Example05.java

```
public class Example05 {  
    public static void main(String[] args) {  
        Cat cat1 = new Cat("페르시안") ;  
        Cat cat2 = new Cat("삼") ;  
        Cat cat3 = new Cat() ;  
        cat3.breed = "러시안블루";  
        System.out.println("첫 번째 고양이 품종 : " + cat1.breed);  
        System.out.println("두 번째 고양이 품종 : " + cat2.breed);  
        System.out.println("세 번째 고양이 품종 : " + cat3.breed);  
    }  
}
```

#### 실행 결과

Cat(...) 생성자 호출합니다.

Cat(...) 생성자 호출합니다.

Cat() 생성자 호출합니다.

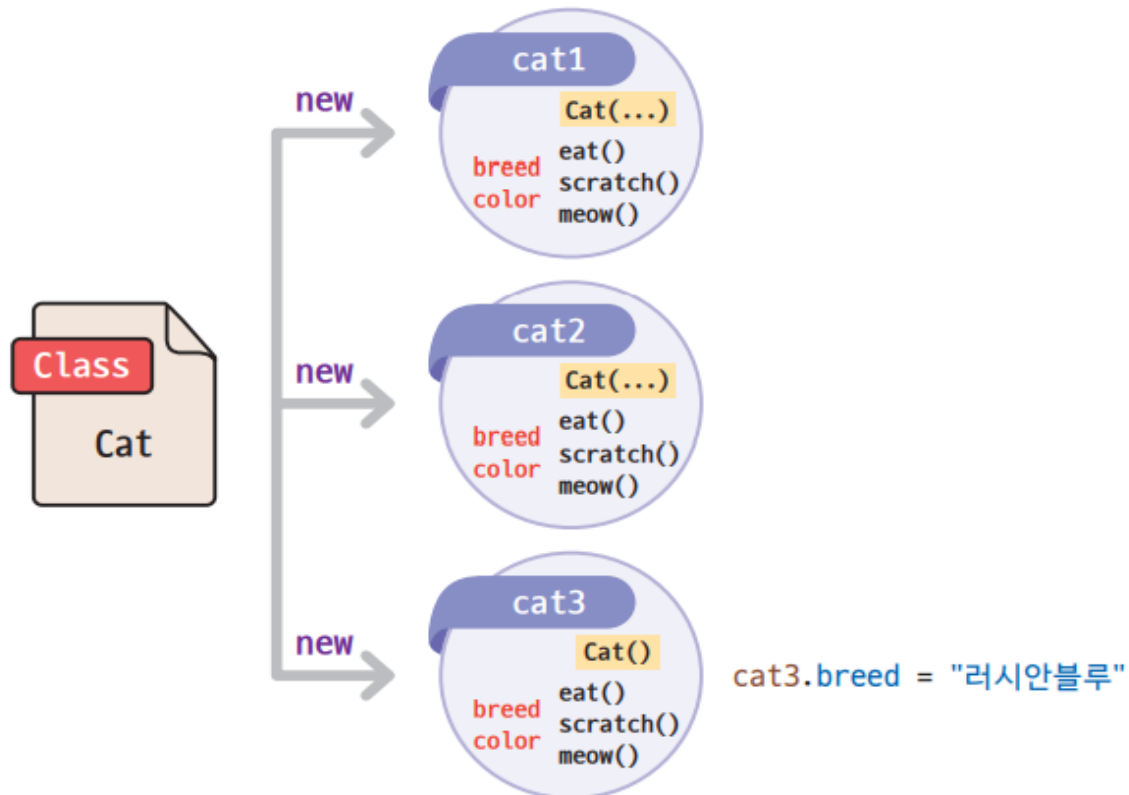
첫 번째 고양이 품종 : 페르시안

두 번째 고양이 품종 : 삼

세 번째 고양이 품종 : 러시안블루

## 4. 생성자

### ■ 일반 생성자



[그림 7-19] 기본 생성자와 일반 생성자 호출1

## 4. 생성자

### 예제 7-4

### 일반 생성자를 호출하여 객체 생성하기

Student.java

```
01 public class Student {
02     int id;
03     String name;
04
05     Student() {
06         System.out.println("기본 생성자 Student() 호출");
07     }
08
09     Student(int pram1, String pram2) {
10         System.out.println("일반 생성자 Student(...) 호출");
11         id = pram1;
12         name = pram2;
13     }
14
15     void insertRecord(int parm1, String parm2) {
16         id = parm1;
17         name = parm2;
18     }
19 }
```

## 4. 생성자

```
20 void printInfo() {  
21     System.out.println("아이디: " + id);  
22     System.out.println("이름: " + name);  
23 }  
24 }
```

Constructor02.java

```
01 public class Constructor02 {  
02     public static void main(String[] args) {  
03         System.out.println("****학생 주소록****");  
04         Student stObj1 = new Student();  
05  
06         stObj1.insertRecord(20221004, "홍길순");  
07         stObj1.printInfo();  
08  
09         Student stObj2 = new Student(20221005, "홍길동");  
10         stObj2.printInfo();  
11     }  
12 }
```

### 실행 결과

\*\*\*\*학생 주소록\*\*\*\*

기본 생성자 Student() 호출

아이디: 20221004

이름: 홍길순

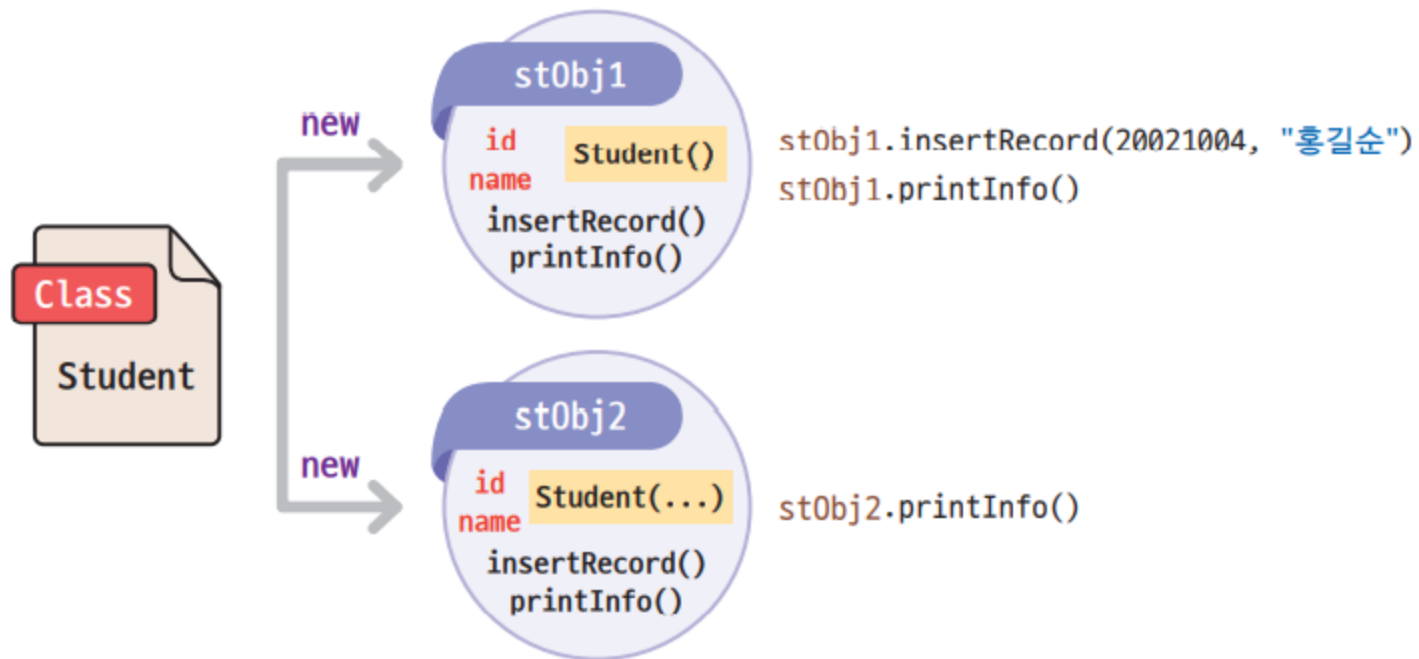
일반 생성자 Student(...) 호출

아이디: 20221005

이름: 홍길동



## 4. 생성자



[그림 7-20] 기본 생성자와 일반 생성자 호출2

## 4. 생성자

### ■ 생성자 오버로딩

- 생성자 이름이 같지만 매개변수가 다른 여러 생성자를 정의하는 것
  - 매개변수의 자료형 또는 개수에 따라 구분
- 매개변수의 자료형이나 개수가 다른 생성자가 여러 개 있으나 생성자에 반환 자료형이 없음

## 4. 생성자

### ■ 생성자 오버로딩

#### 생성자 오버로딩 예시

Cat.java

```
public class Cat {  
    String breed;  
    String color;  
    Cat() {  
        System.out.println("Cat() 생성자 호출합니다.");  
    }  
    Cat(String pBreed) {  
        System.out.println("Cat(...) 생성자 호출합니다.");  
        breed = pBreed;  
    }  
    Cat(String pBreed, String pColor) {  
        System.out.println("Cat(..., ...) 생성자 호출합니다.");  
        breed = pBreed;  
        color = pColor;  
    }  
    void eat() {  
        System.out.println("먹이를 먹다.");  
    }  
    void scratch() {  
        System.out.println("발톱으로 할퀴다.");  
    }  
    void meow() {  
        System.out.println("야옹하고 울다.");  
    }  
}
```

## 4. 생성자

### ■ 생성자 오버로딩

#### 오버로딩된 생성자 호출

Example06.java

```
public class Example06 {  
    public static void main(String[] args) {  
        Cat cat1 = new Cat();  
        Cat cat2 = new Cat("삼");  
        Cat cat3 = new Cat("러시안블루", "회색");  
        System.out.println("첫 번째 고양이 품종 : " + cat1.breed + " \t색상 : " + cat1.color);  
        System.out.println("두 번째 고양이 품종 : " + cat2.breed + " \t색상 : " + cat2.color);  
        System.out.println("세 번째 고양이 품종 : " + cat3.breed + " \t색상 : " + cat3.color);  
    }  
}
```

#### 실행 결과

Cat() 생성자 호출합니다.

Cat(...) 생성자 호출합니다.

Cat(..., ...) 생성자 호출합니다.

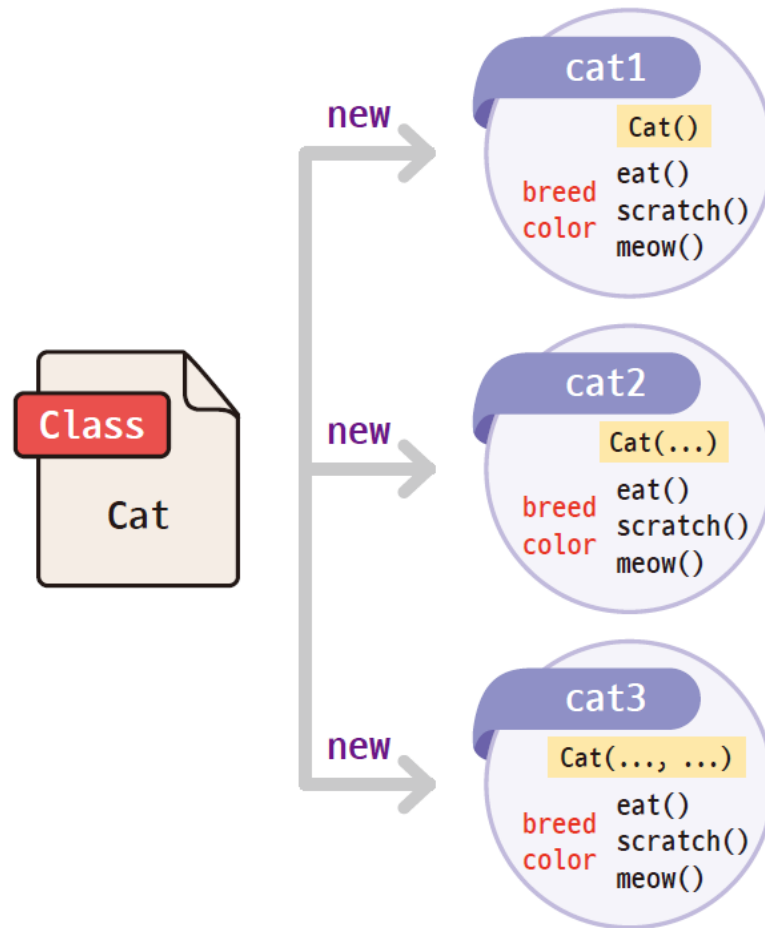
첫 번째 고양이 품종 : null    색상 : null

두 번째 고양이 품종 : 삼    색상 : null

세 번째 고양이 품종 : 러시안블루    색상 : 회색

## 4. 생성자

### ■ 생성자 오버로딩



[그림 7-21] 생성자 오버로딩1

## 4. 생성자

### 예제 7-5

덧셈 연산을 위한 생성자를 오버로딩하고 호출하기

Add.java

```
01 public class Add {
02
03     Add() {
04         System.out.println("기본 생성자 Add() 호출");
05     }
06
07     Add(int a, int b) {
08         System.out.println("일반 생성자 Add(int a, int b) 호출");
09         System.out.println(a + " + " + b + " = " + (a+b));
10     }
11
12     Add(double a, double b) {
13         System.out.println("일반 생성자 Add(double a, double b) 호출");
14         System.out.println(a + " + " + b + " = " + (a+b));
15     }
16
17     Add(int a, double b) {
18         System.out.println("일반 생성자 Add(int a, double b) 호출");
19         System.out.println(a + " + " + b + " = " + (a+b));
20     }
21 }
```

## 4. 생성자

Constructor03.java

```
01 public class Constructor03 {  
02     public static void main(String[] args) {  
03         Add numObj1 = new Add(1, 2);  
04  
05         Add numObj2 = new Add(2.4, 6.2);  
06  
07         Add numObj3 = new Add(3, 5.5);  
08     }  
09 }
```

### 실행 결과

일반 생성자 Add(int a, int b) 호출

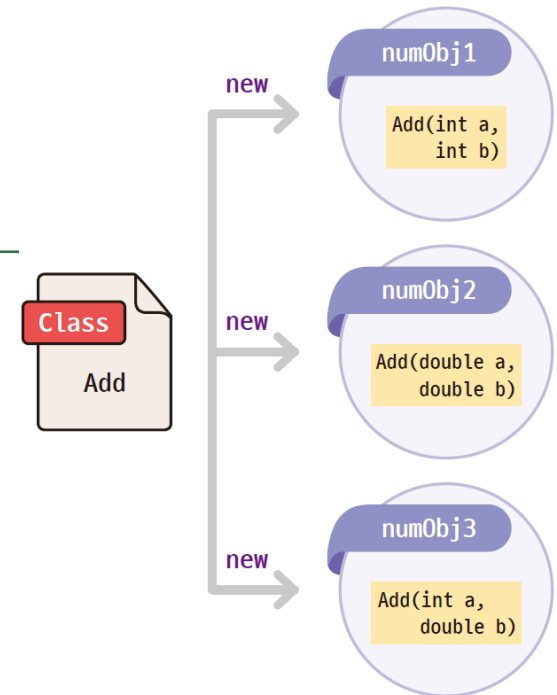
1 + 2 = 3

일반 생성자 Add(double a, double b) 호출

2.4 + 6.2 = 8.6

일반 생성자 Add(int a, double b) 호출

3 + 5.5 = 8.5



[그림 7-22] 생성자 오버로딩2

## 4. 생성자

### ■ this를 이용한 생성자 체인

- this() 생성자
  - 동일한 클래스 내의 서로 다른 생성자에서 오버로딩된 또 다른 생성자 하나를 호출하는 데 사용됨
- 생성자 체인(constructor chaining)
  - 동일한 클래스의 다른 생성자에서 하나의 생성자를 호출하는 프로세스로, 생성자는 다른 생성자에서만 호출할 수 있으므로 생성자 체인이 필요함
- 다른 생성자에서 생성자를 호출하기 위한 this()는 현재 객체를 참조하는 데 사용함



## 4. 생성자

### ■ this를 이용한 생성자 체인

#### 생성자 체인 예시

Chain.java

```
public class Chain {  
    Chain() {  
        this(10);  
        System.out.println("기본 생성자 Chain() 호출");  
    }  
    Chain(int x) {  
        this(5, 6);  
        System.out.println("일반 생성자 Chain(int x) 호출");  
        System.out.println("x의 값 : "+ x);  
    }  
    Chain(int x, int y) {  
        System.out.println("일반 생성자 Chain(int x, int y) 호출");  
        System.out.println("x와 Y 값 : "+ x + " "+ y);  
    }  
}
```

## 4. 생성자

### ■ this를 이용한 생성자 체인

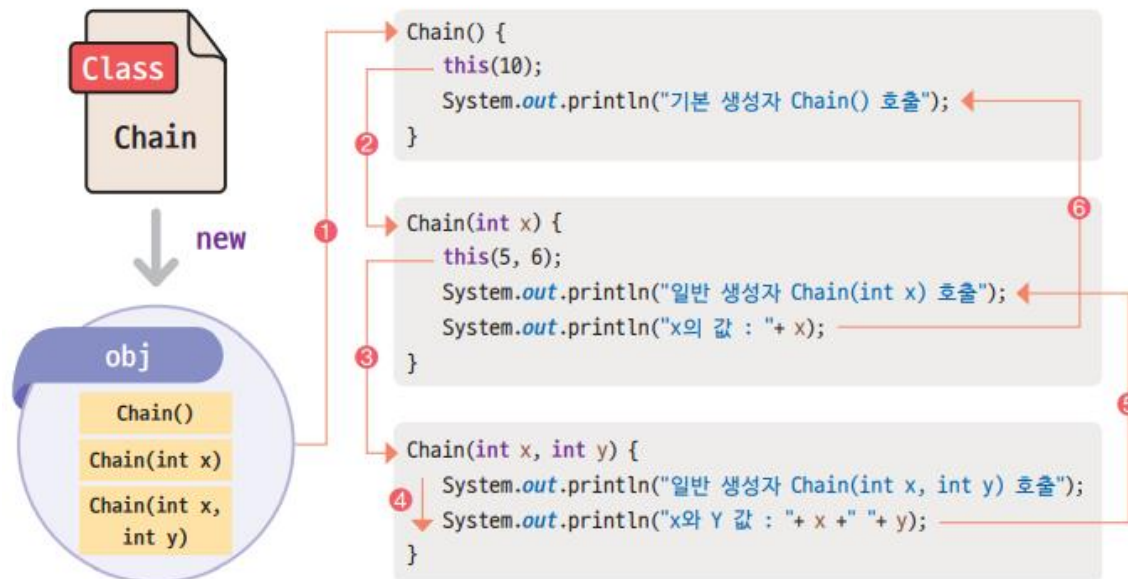
#### 생성자 체인 호출

```
public class Example07 {  
    public static void main(String[] args) {  
        Chain obj = new Chain();  
    }  
}
```

Example07.java

#### 실행 결과

일반 생성자 Chain(int x, int y) 호출  
x와 Y 값 : 5 6  
일반 생성자 Chain(int x) 호출  
x의 값 : 10  
기본 생성자 Chain() 호출



[그림 7-23] 생성자 Chain()의 호출 흐름

## 4. 생성자

### 예제 7-6

### this를 이용하여 생성자 체인 만들기

MemberChain.java

```
01 public class MemberChain {
02     String firstName;
03     String country;
04     int age;
05
06     public MemberChain() {
07         this("홍길순");
08     }
09
10     public MemberChain(String mName) {
11         this(mName, 20);
12     }
13
14     public MemberChain(String mName, int mAge) {
15         this(mName, mAge, "대한민국");
16     }
17
18     public MemberChain(String mName, int mAge, String mCountry) {
19         firstName = mName;
20         age = mAge;
21         country = mCountry;
22     }
```

## 4. 생성자

```
23
24 void printInfo() {
25     System.out.println("이름 : " + firstName );
26     System.out.println("국적 : " + country );
27     System.out.println("나이 : " + age );
28 }
29 }
```

Constructor04.java

```
01 public class Constructor04 {
02     public static void main(String[] args) {
03         MemberChain object = new MemberChain();
04         object.printInfo();
05     }
06 }
```

실행 결과

이름 : 홍길순  
국적 : 대한민국  
나이 : 20

# Section 05

## 접근제한자

## 5. 접근제한자

### ■ 접근제한자의 개념

- 접근제한자
  - 접근성을 설정하는 데 사용되는 키워드로 다른 클래스의 클래스, 생성자, 데이터 멤버, 메서드의 접근을 제한함
- 불필요한 세부 정보를 숨기고 최소한의 정보만으로 프로그램을 손쉽게 사용할 수 있도록 하는 정보 은닉의 캡슐화를 위해 접근제한자를 사용함

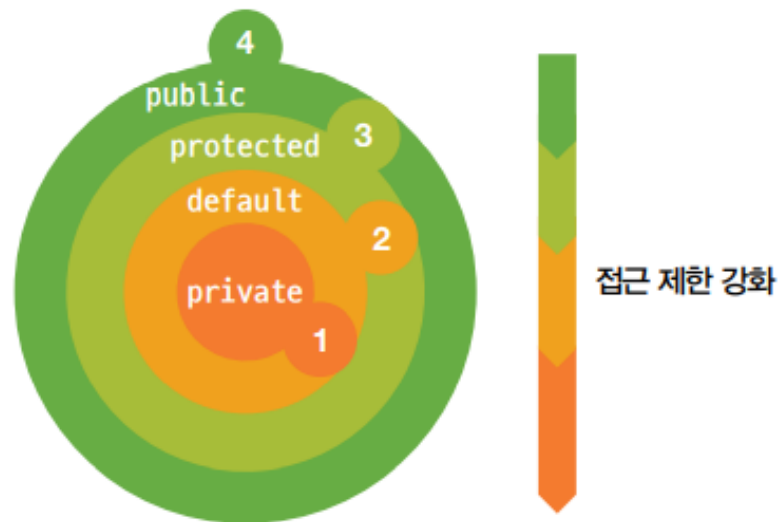
[표 7-3] 접근제한자의 접근 가능 범위

유형	같은 클래스의 멤버	같은 패키지의 멤버	자식 클래스의 멤버	기타 영역
public	○	○	○	○
protected	○	○	○	×
선언하지 않음(default)	○	○	×	×
private	○	×	×	×

## 5. 접근제한자

### ■ 접근제한자의 제한 범위

- public: 모든 곳에서 접근 가능
- protected: 패키지 및 모든 하위 클래스 내에서 접근 가능
- 선언하지 않음(default): 동일한 패키지 내에서만 접근 가능
- private: 클래스 내에서만 접근 가능

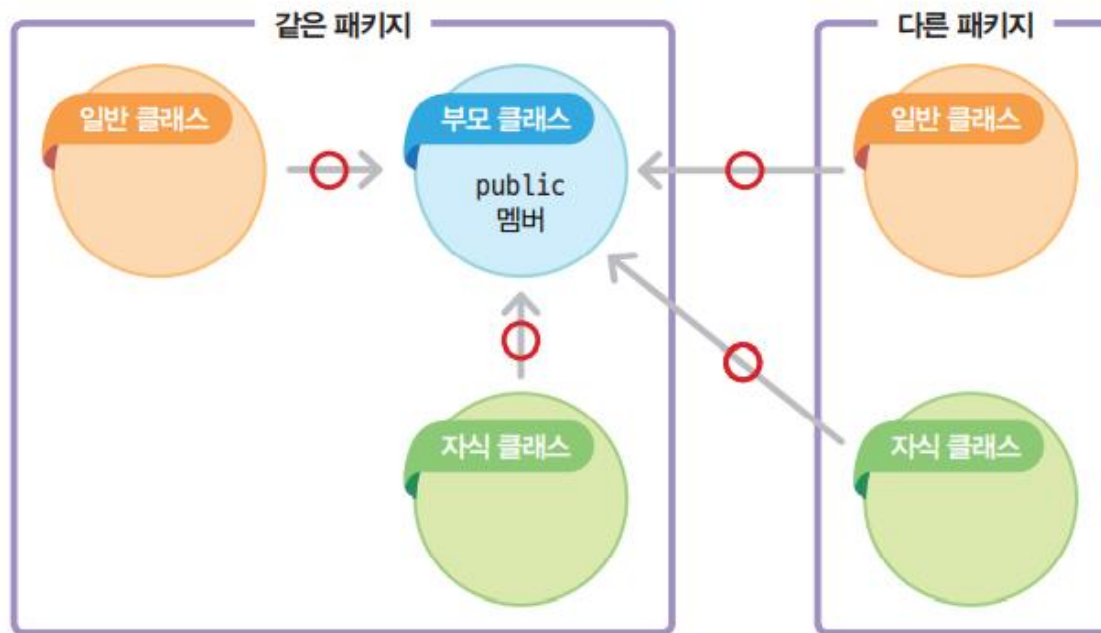


[그림 7-24] 접근제한자의 제한 범위

## 5. 접근제한자

### ■ public 접근제한자

- public 접근제한자가 선언된 클래스 멤버(변수, 메서드, 생성자)
  - 클래스가 같은 패키지에 있든 다른 패키지에 있든 상관없이 프로그램의 어디서나 직접 접근할 수 있음



[그림 7-25] public 접근제한자의 접근 예

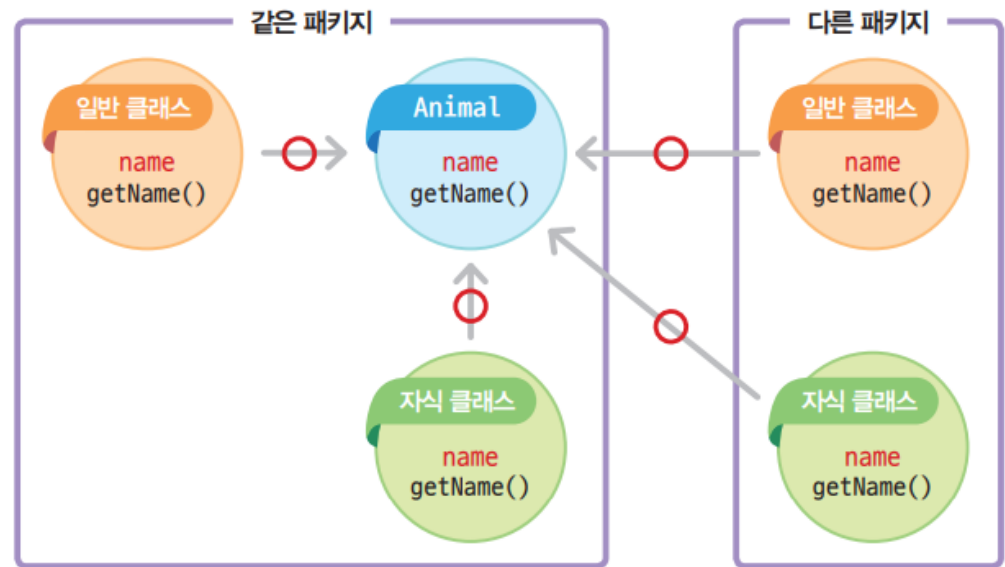


## 5. 접근제한자

### ■ public 접근제한자

- [public 접근제한자로 선언된 Animal 클래스의 예]

```
public class Animal {  
    public String name;  
  
    public String getName() {  
        return name;  
    }  
}
```

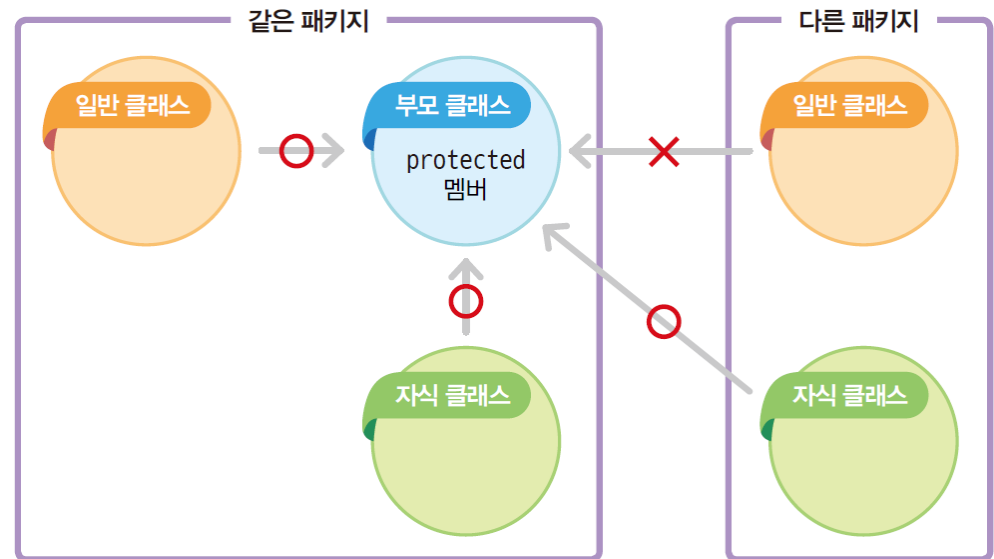


[그림 7-26] public 클래스 Animal의 접근 가능 범위

## 5. 접근제한자

### ■ protected 접근제한자

- protected는 클래스와 인터페이스에 사용할 수 없음
- 상위 클래스에서 protected가 선언된 변수, 메서드, 생성자는 다른 패키지의 하위 클래스에서만 접근할 수 있음
- 같은 패키지의 클래스는 protected가 선언된 하위 클래스가 아니더라도 클래스 멤버에 접근할 수 있음

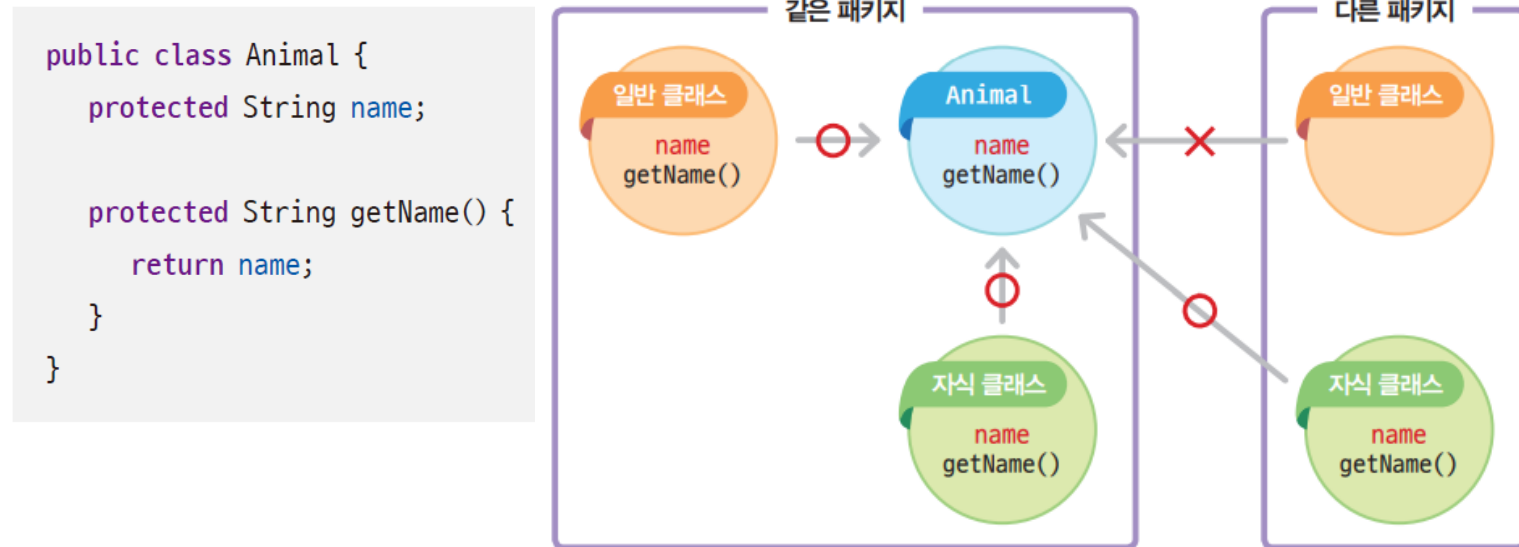


[그림 7-28] protected 접근제한자의 접근 예

## 5. 접근제한자

### ■ protected 접근제한자

- [protected 접근 제한자로 선언된 변수와 메서드를 갖는 Animal 클래스의 예]

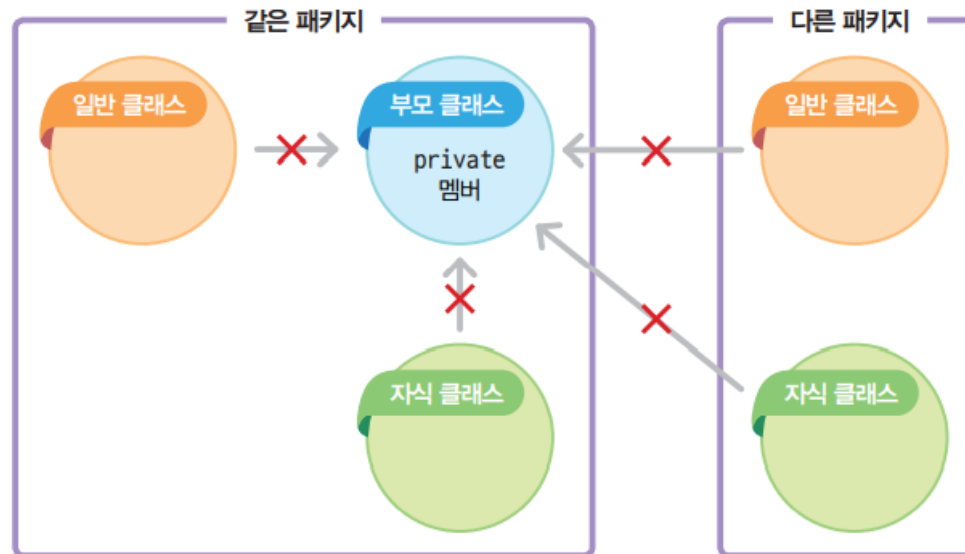


[그림 7-29] protected 클래스 Animal의 접근 가능 범위

## 5. 접근제한자

### ■ private 접근제한자

- 가장 제한적인 것으로, 외부 클래스로부터의 접근을 허용하지 않는 멤버(변수, 메서드, 생성자)에는 사용할 수 있지만 클래스와 인터페이스에는 사용할 수 없음
- private로 선언된 필드, 메서드, 생성자는 엄격하게 제어되므로 둘러싸는 클래스 외부에서 접근할 수 없음



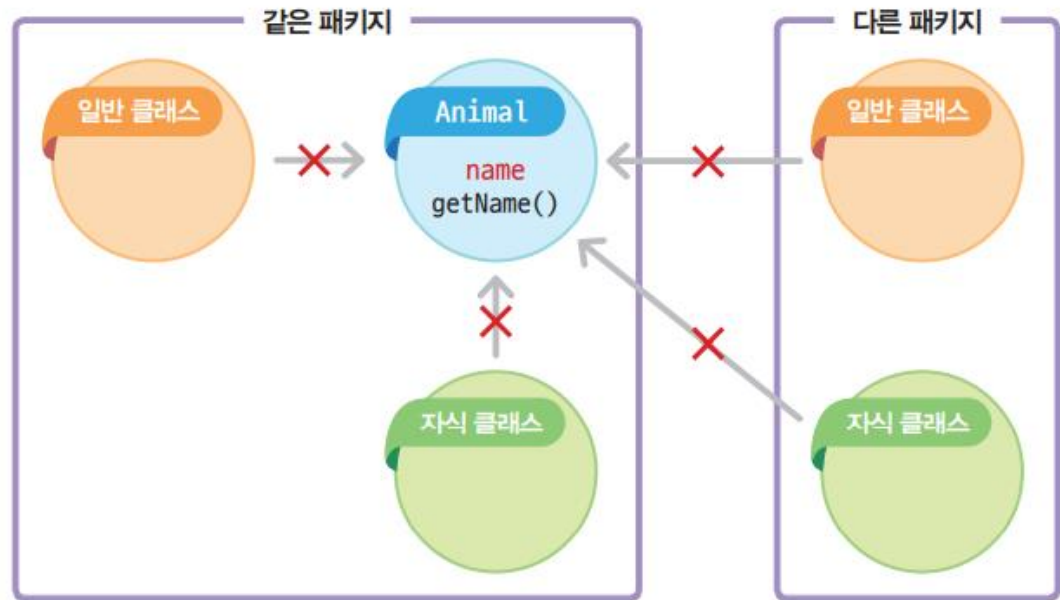
[그림 7-31] private 접근제한자의 접근 예

## 5. 접근제한자

### ■ private 접근제한자

- private 접근제한자로 선언된 Animal 클래스의 예

```
public class Animal {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
}
```



[그림 7-32] private 클래스 Animal의 접근 가능 범위

## 5. 접근제한자

### 예제 7-9

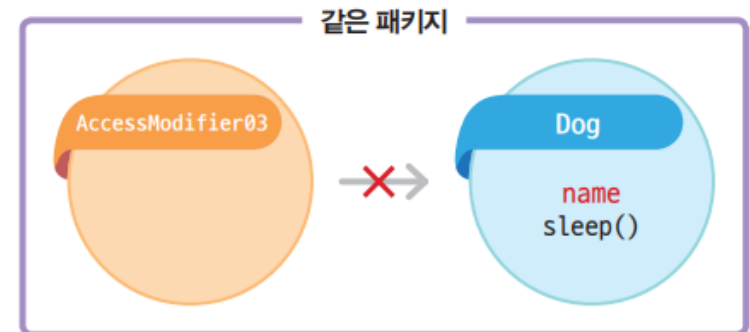
private 접근제한자를 이용하여 클래스 멤버 호출하기

Dog.java

```
01 public class Dog {
02     public String breed;
03     public String color;
04     protected int age;
05     private String name;
06
07     public void bowwow() {
08         System.out.println("멍멍 짖다");
09     }
10
11     protected void run() {
12         System.out.println("달리다");
13     }
14
15     private void sleep() {
16         System.out.println("잠을 자다");
17     }
18 }
```

#### 실행 결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    The field Dog.name is not visible
    The field Dog.name is not visible
    The method sleep() from the type Dog is not visible
    at Example08.main(Example08.java:20)
```



[그림 7-33] private 접근제한자 선언의 클래스 멤버 접근 예

## 5. 접근제한자

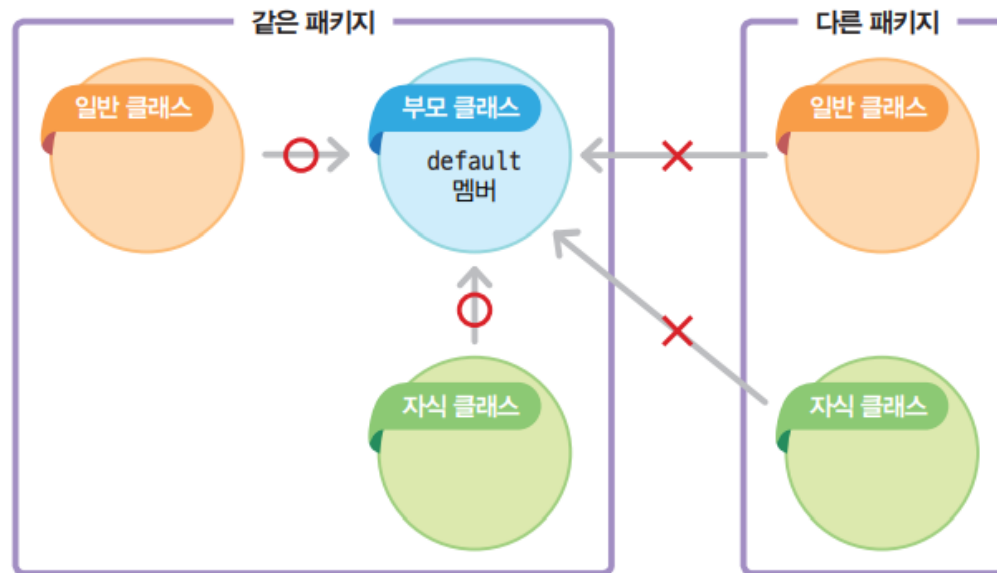
AccessModifier03.java

```
01 public class AccessModifier03 {
02     public static void main(String[] args) {
03         Dog obj = new Dog();
04
05         obj.breed = "포메라이언";
06         obj.color = "갈색";
07
08         System.out.println("강아지 품종 : " + obj.breed);
09         System.out.println("강아지 색상 : " + obj.color);
10         obj.bowwow();
11         obj.age = 10;
12         System.out.println("강아지 나이 : " + obj.age);
13         obj.run();
14
15         obj.name = "다운";
16         System.out.println("강아지 이름 : " + obj.name);
17         obj.sleep();
18     }
19 }
```

## 5. 접근제한자

### ■ default 접근제한자

- 접근제한자로 전혀 선언되지 않은 것
- 선언된 접근제한자가 없는 모든 클래스, 변수, 메서드, 생성자는 같은 패키지의 클래스에서만 접근할 수 있음



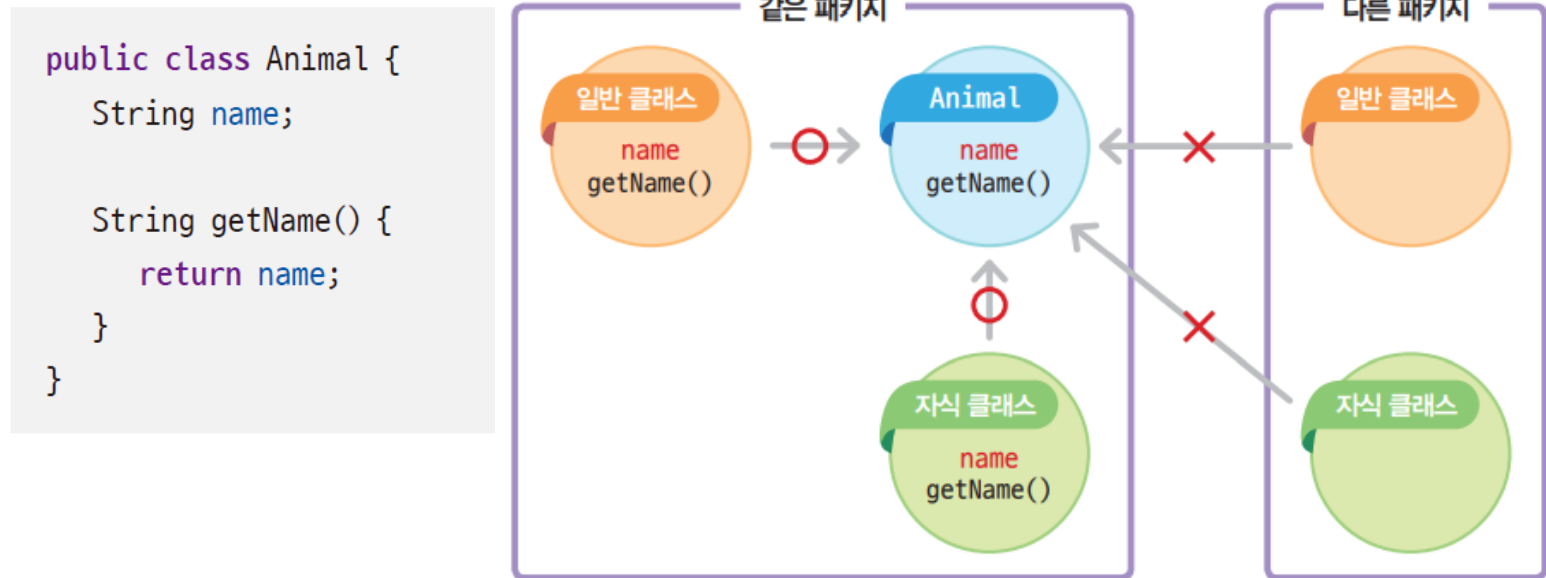
[그림 7-14] default 접근제한자의 접근 예



## 5. 접근제한자

### ■ default 접근제한자

- 접근제한자가 없는 클래스의 멤버 변수와 멤버 메서드의 예



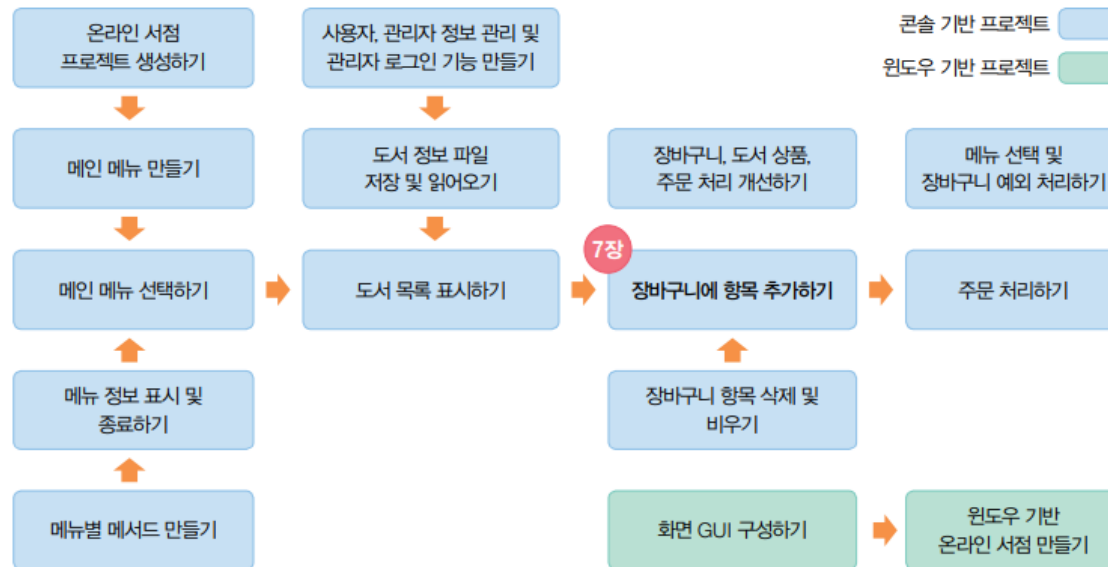
[그림 7-35] default 클래스 `Animal`의 접근 가능 범위

**[프로젝트]**

**장바구니에 항목 추가하기**

# 장바구니에 항목 추가하기

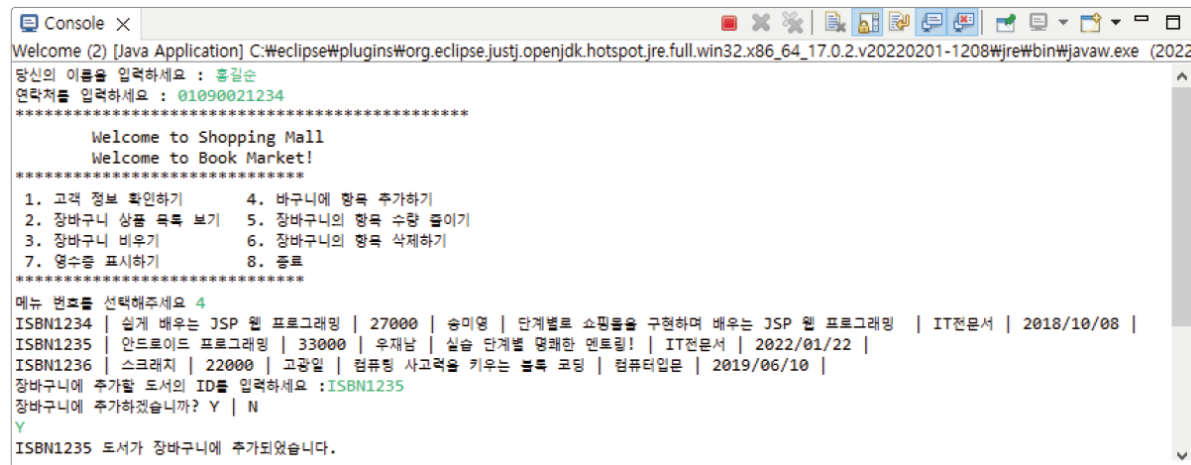
- 도서를 담은 장바구니 클래스를 만든 뒤, 메인 메뉴에서 [장바구니에 항목 추가하기] 메뉴를 선택하여 장바구니에 도서를 담고, [장바구니 상품 목록 보기] 메뉴를 선택하여 장바구니에 담긴 도서 목록을 출력하게 합니다



[그림 7-37] 장바구니에 항목 추가하기

# 장바구니에 항목 추가하기

- 도서를 담은 장바구니 클래스를 만든 뒤, 메인 메뉴에서 [장바구니에 항목 추가하기] 메뉴를 선택하여 장바구니에 도서를 담고, [장바구니 상품 목록 보기] 메뉴를 선택하여 장바구니에 담긴 도서 목록을 출력하게 합니다



```
Console X
Welcome (2) [Java Application] C:\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (2022)
당신의 이름을 입력하세요 : 홍길순
연락처를 입력하세요 : 01090021234
*****
Welcome to Shopping Mall
Welcome to Book Market!
*****
1. 고객 정보 확인하기      4. 바구니에 항목 추가하기
2. 장바구니 상품 목록 보기  5. 장바구니의 항목 수량 줄이기
3. 장바구니 비우기        6. 장바구니의 항목 삭제하기
7. 영수증 표시하기        8. 종료
*****
메뉴 번호를 선택해주세요 4
ISBN1234 | 쉽게 배우는 JSP 웹 프로그래밍 | 27000 | 송미영 | 단계별로 소프팅을 구현하며 배우는 JSP 웹 프로그래밍 | IT전문서 | 2018/10/08 |
ISBN1235 | 안드로이드 프로그래밍 | 33000 | 우재남 | 실습 단계별 명쾌한 멘토링! | IT전문서 | 2022/01/22 |
ISBN1236 | 스크래치 | 22000 | 고광일 | 컴퓨터 사고력을 키우는 블록 코딩 | 컴퓨터입문 | 2019/06/10 |
장바구니에 추가할 도서의 ID를 입력하세요 : ISBN1235
장바구니에 추가하겠습니까? Y | N
Y
ISBN1235 도서가 장바구니에 추가되었습니다.
```

[그림 7-38] 장바구니에 항목 추가하기 실행 결과

# 장바구니에 항목 추가하기

## ■ 고객 정보 클래스 만들기

- 01 고객 클래스 생성하기

- BookMarket/src 폴더에 Person.java 파일을 생성하고 Person 클래스를 작성

# 장바구니에 항목 추가하기

## ■ 장바구니에 항목 추가하기

- 01 장바구니 클래스 생성하기

- BookMarket/src 폴더에 CartItem.java 파일을 생성하고 CartItem 클래스를 작성

# 장바구니에 항목 추가하기

## ■ 장바구니에 항목 추가하기

- 02 장바구니에 도서 추가하기

→ Welcome.java 파일에서 도서 목록의 상품을 장바구니에 담는 menuCartItem() 메서드를 수정하고 isCartItemBook() 메서드를 추가