프로그래밍 기초 (Java)

숭실대학교

서유화 교수

yhsuh@ssu.ac.kr

목차

학습목표

- 패키지의 개념과 특징을 이해합니다.
- 패키지를 생성하고 사용하는 방법을 알아봅니다.
- 자바의 내장 패키지에 대해 알아봅니다.
- 접근제한자에 대해 알아봅니다.

Section 01

패키지

1. 패키지

자바 API란

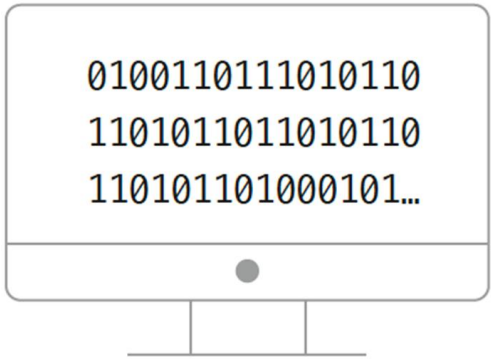
- 자바 API(Application Programming Interface)란

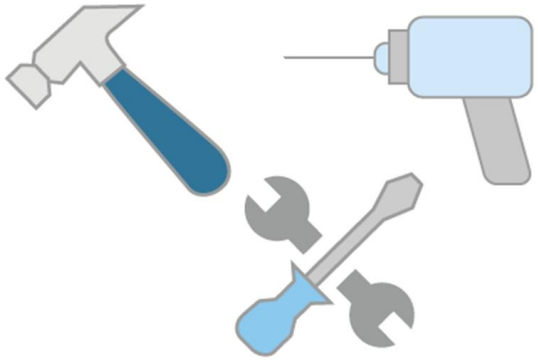
  → 프로그램 개발을 위한 도구로, 자바가 제공하는 코드

- 자바 API의 대표적인 예

  → System.out.printf(), Math.random(), Math.PI 등

- 코드가 미리 준비되어 있다면 더 편리한 프로그램의 작성을 도움





자바 API = 생활 속 여러 도구들

1. 패키지

패키지(package)의 개념

- 자바 코드의 묶음, 일종의 폴더(디렉터리)

→ 관련 코드를 한 곳에 모아 관리하거나

→ 이름만 같은 다른 코드를 구분하기 위해 사용

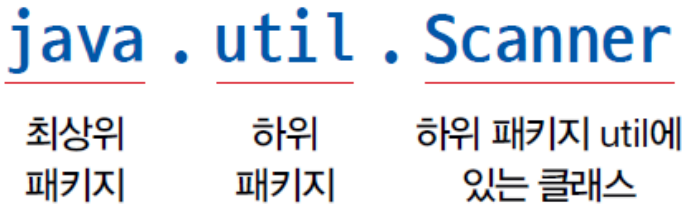
→ 클래스를 고유하게 식별할 수 있는 묶음

→ 패키지를 사용하여 클래스 또는 클래스 멤버의 접근 범위를 지정 가능

패지지 예

1. 패키지

패키지의 구조



← Scanner 클래스는 java.util.* 패키지에 포함된 클래스

패키지의 구조

- 패키지: 각각 특정 기능을 가진 클래스의 묶음
- 유사한 클래스와 기능을 캡슐화한 것
- 패키지에는 관련 클래스, 인터페이스, 하위 패키지가 포함

1. 패키지

패키지 사용의 장점

- 재사용성

→패키지를 통해 데이터 캡슐화를 쉽게 구현할 수 있으므로 프로젝트를 개발하는 동안 코드를 반복하여 작성하는 일이 거의 없음

- 더 나은 조직화

→기능에 따라 클래스를 정렬하기 때문에 클래스를 검색하기가 쉬움

- 이름 충돌 방지

→패키지는 이름의 충돌을 방지하는 데 도움이 됨

- 접근 제어

→접근제한자와 패키지를 함께 사용하여 클래스의 접근 가능성을 제어할 수 있음

1. 패키지

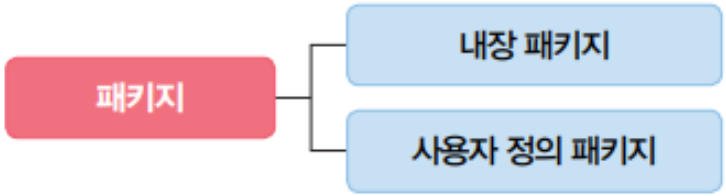
패키지와 접근제한자의 관계

패키지와 접근제한자의 관계

구분	public	protected	(default)	private
동일 클래스인 경우	○	○	○	○
동일 패키지 내부이고 자식 클래스인 경우	○	○	○	×
동일 패키지 내부이고 자식 클래스가 아닌 경우	○	○	○	×
다른 패키지이고 자식 클래스인 경우	○	○	×	×
다른 패키지이고 자식 클래스가 아닌 경우	○	×	×	×

1. 패키지

패키지의 유형



패키지의 유형

- 사용자 정의 패키지

  →사용자가 프로젝트의 클래스와 인터페이스를 분류하기 위해 생성하는 패키지

- 내장 패키지

  →java.io.*, java.lang.* 등과 같이 이미 정의된 패키지

Section 02

패키지 생성 및 사용

2. 패키지 생성 및 사용

패키지 생성

- 패키지 선언

→package 명령어 뒤에 패키지 경로명을 작성하고, 프로그램(자바 소스 파일)의 첫 행에 포함해야 함

```
package 패키지명;
```

→패키지 선언 예

```
package mypackage;

public class MyClass {
   public static void main(String[] args) {
      ...
   }
}
```

프로그램의 첫 행에 선언

MyClass 클래스가 mypackage 패키지에 포함됨

2. 패키지 생성 및 사용

패키지 생성

- 패키지명의 구성

→패키지명은 마침표(.)로 구분된 여러 단어의 조합으로, 각 단어는 파일 시스템의 한 폴더 또는 디렉터리를 나타냄

→[예] 패키지명의 구성

```
package com.javamaster.mypackage;        ● 마침표(.)로 구분된 패키지 경로명

public class MyClass {                   ● MyClass 클래스가 com.javabook.mypackage 패키지에 포함됨
   public static void main(String[] args) {
      ...
   }
}
```

2. 패키지 생성 및 사용

패키지 생성

- 패키지명 작성 규칙

→ 숫자로 시작할 수 없음

→ '_'과 '$'를 제외한 특수 문자를 사용할 수 없음.

→ java로 시작하는 패키지를 사용할 수 없음

→ java로 시작하는 패키지는 자바 표준 API에서만 사용 가능함

→ int, static 등 자바 예약어를 사용할 수 없음

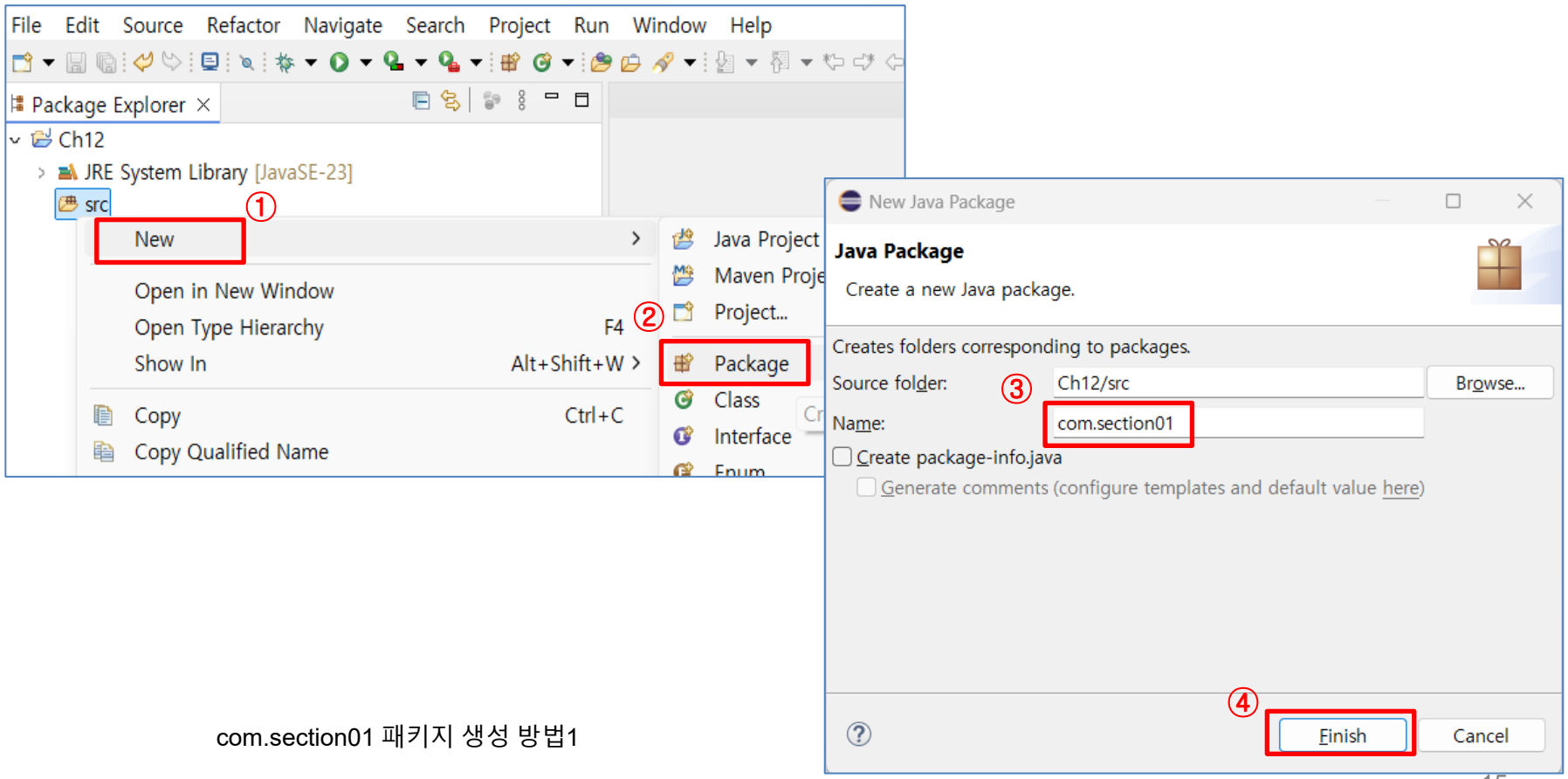
→ 클래스명 또는 인터페이스명과의 충돌 방지를 위해 모두 소문자로 작성하는 것이 관례

→ 소스 파일을 각각의 그룹으로 구분하기 위해 마침표(.)를 사용함

2. 패키지 생성 및 사용

패키지 생성

- 방법1) 이클립스에서 [New]- [Package]를 선택하면 나타나는 [New Java Package] 창에서 Name 항목에 패키지명 'com. section01'을 입력함

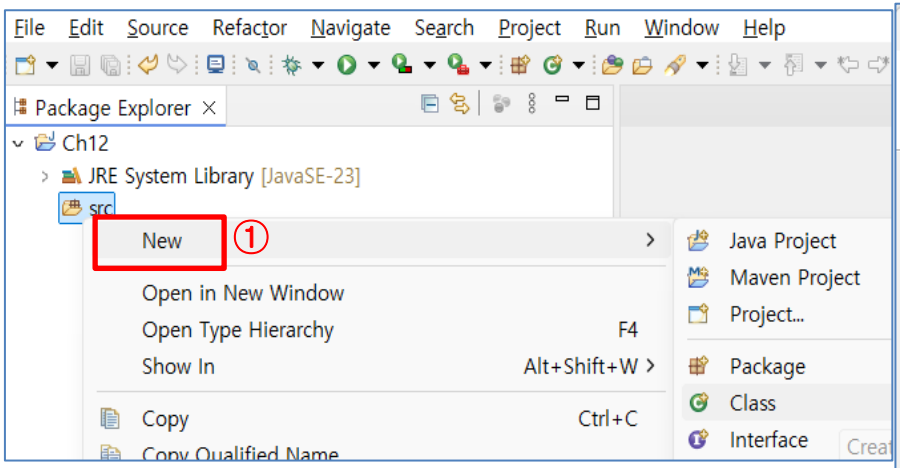


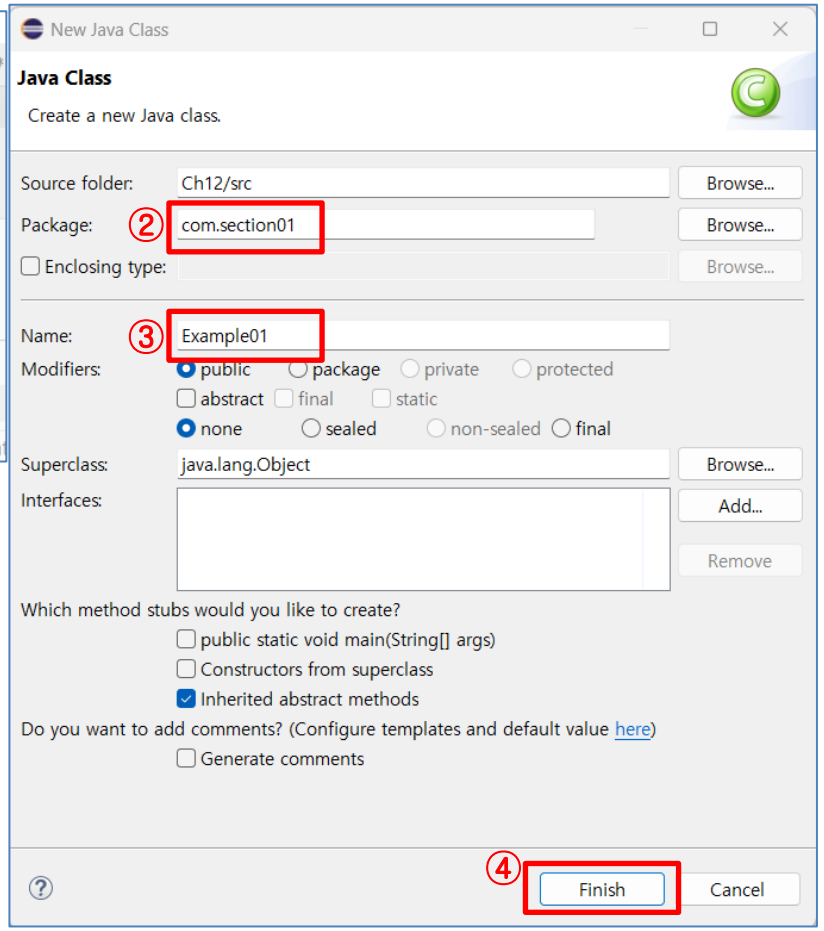
com.section01 패키지 생성 방법1

2. 패키지 생성 및 사용

패키지 생성

- 방법2) [New]-[Class]를 선택하면 나타나는 [New Java Class] 창에서 Package 항목에 패키지명 'com.section01'을 입력함





2. 패키지 생성 및 사용

패키지 생성

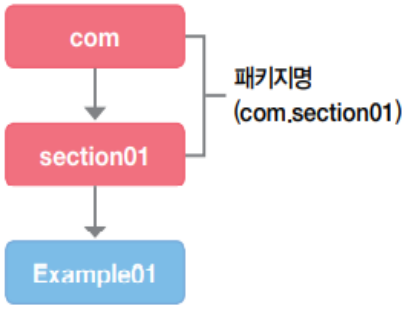
- 패키지에 소속된 클래스 파일은 첫번째 라인에서 자신이 소속된 클래스 패키지 이름을 선언해야 함

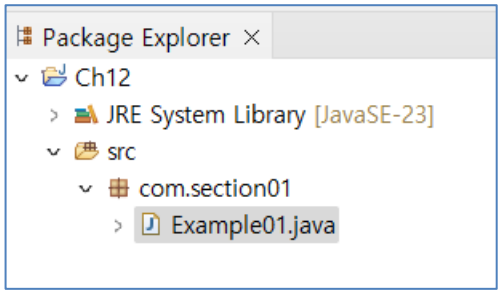
패키지 생성 예시

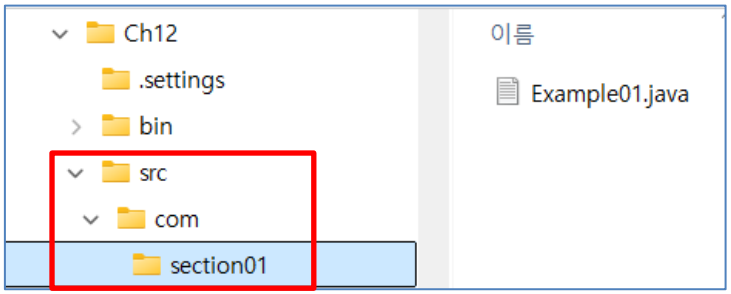
```
package com.section01;

public class Example01 {
    public int add(int a, int b) {
        return a+b;
    }
     public static void main(String[] args) {
        Example01 obj = new Example01();
        System.out.println(obj.add(10, 20));
    }
}
```

실행 결과

```
30
```







com.section01 패키지의 구조

2. 패키지 생성 및 사용

패키지 사용법

- 패키지 내부의 클래스에 접근하려면 import 명령어 뒤에 패키지명과 클래스명을 지정함

```
import 패키지명.클래스명;
import 패키지명.*;
```

✓'**패키지명.클래스명' 형식** : 이 패키지 내에 선언된 클래스에만 접근할 수 있음

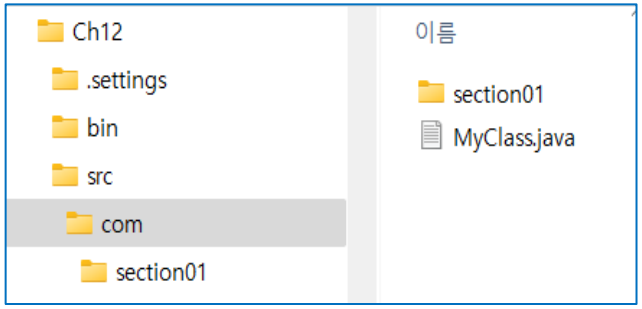
✓'**패키지명.*' 형식** : 이 패키지 내의 모든 클래스와 인터페이스에 접근할 수 있으나 하위 패키지에는 접근할 수 없음, 하위 패키지에 접근하려면 import 문을 하나 더 작성해야 함

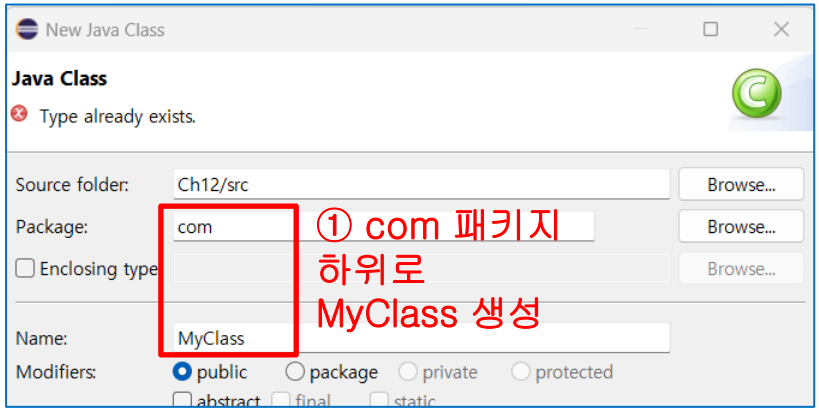
✓ import문은 패키지 선언과 클래스 사이 선언

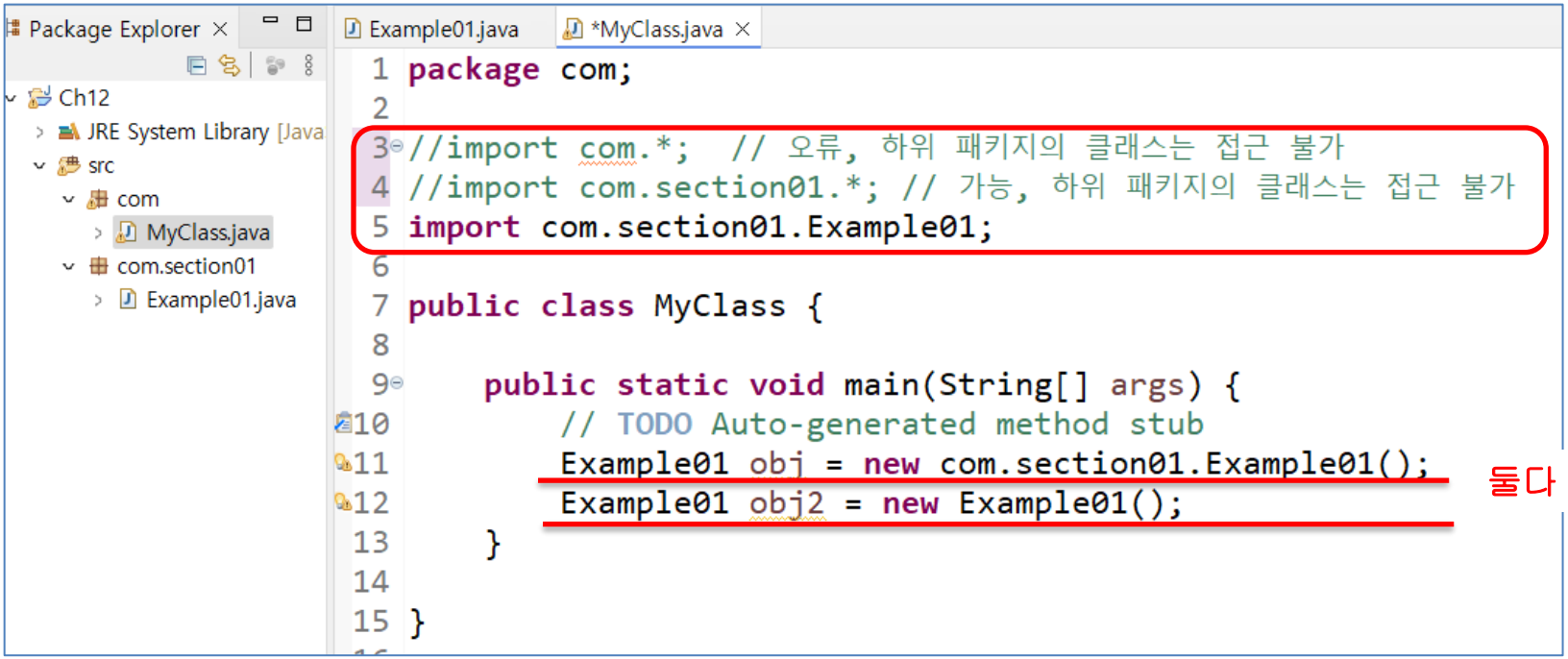
✓ import문의 개수는 제한이 없음

2. 패키지 생성 및 사용

패키지 사용법







2. 패키지 생성 및 사용

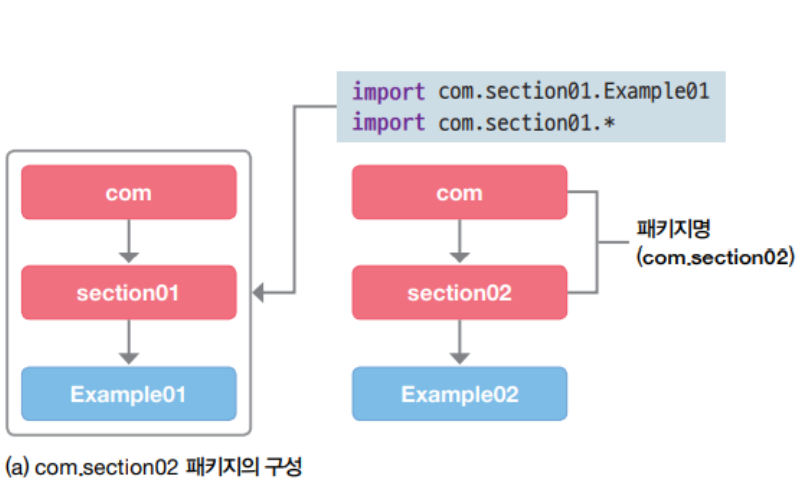
패키지 사용법

패키지 사용 예시

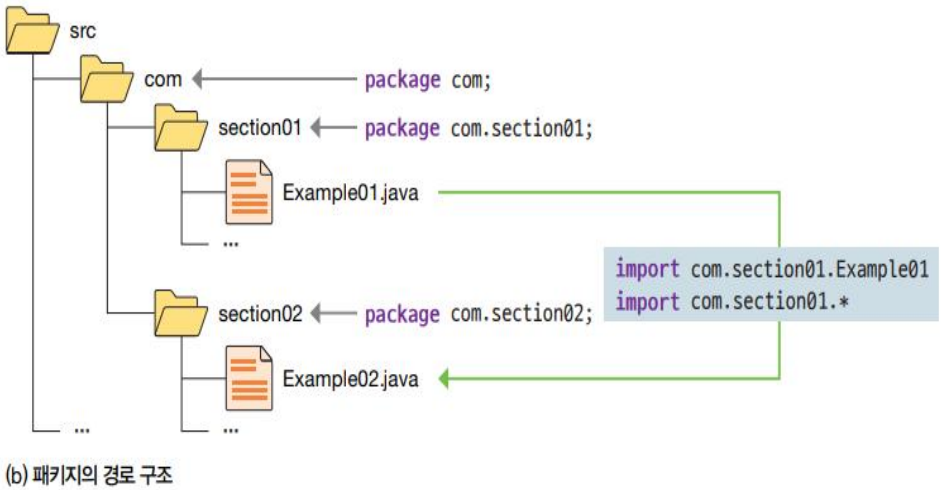
```
package com.section02;
import com.section01.Example01;

public class Example02 {
    public static void main(String[] args) {
        Example01 obj = new Example01();
        System.out.println(obj.add(10, 20));
    }
}
```

실행 결과

```
30
```



(a) com.section02 패키지의 구성



(b) 패키지의 경로 구조

Example02 클래스의 패키지 구조

2. 패키지 생성 및 사용

예제 1 패키지 생성하고 사용하기

com.javamaster.mypackage 패키지

```
01 package com.javamaster.mypackage;
02
03 public class Cat {
04     String breed;
05     String color;
06
07     public void eat() {
08        System.out.println("먹이를 먹다");
09     }
10
11      public void scratch() {
12        System.out.println("발톱으로 할퀴다");
13     }
14
15     public void meow() {
16       System.out.println("야옹하고 울다");
17      }
18 }
```

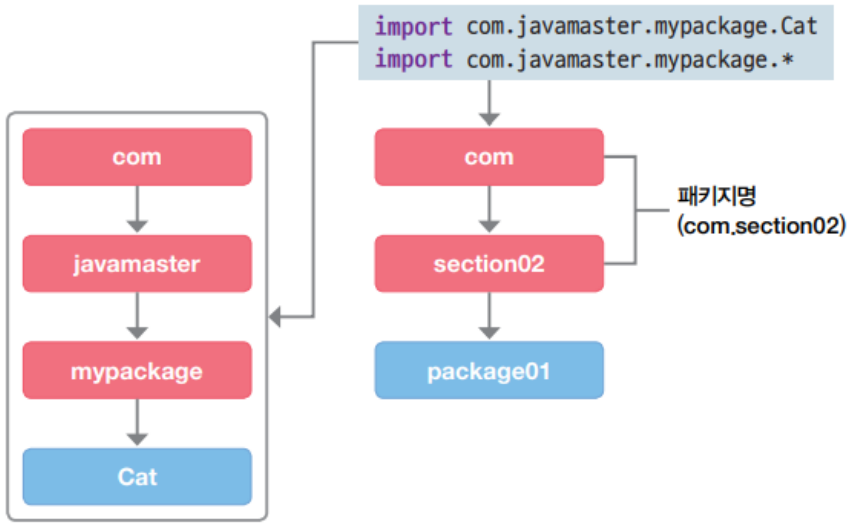
2. 패키지 생성 및 사용

com.section02 패키지

```
01 package com.section02;
02
03 import com.javamaster.mypackage.*;
04 public class Package01 {
05     public static void main(String[] args) {
06         Cat myCat = new Cat();
07         myCat.eat();
08         myCat.scratch();
09         myCat.meow();
10     }
11 }
```

실행 결과

```
먹이를 먹다
발톱으로 할퀴다
야옹하고 울다
```



com.section02 패키지의 구조

Section 03

내장 패키지

3. 내장 패키지

내장 패키지

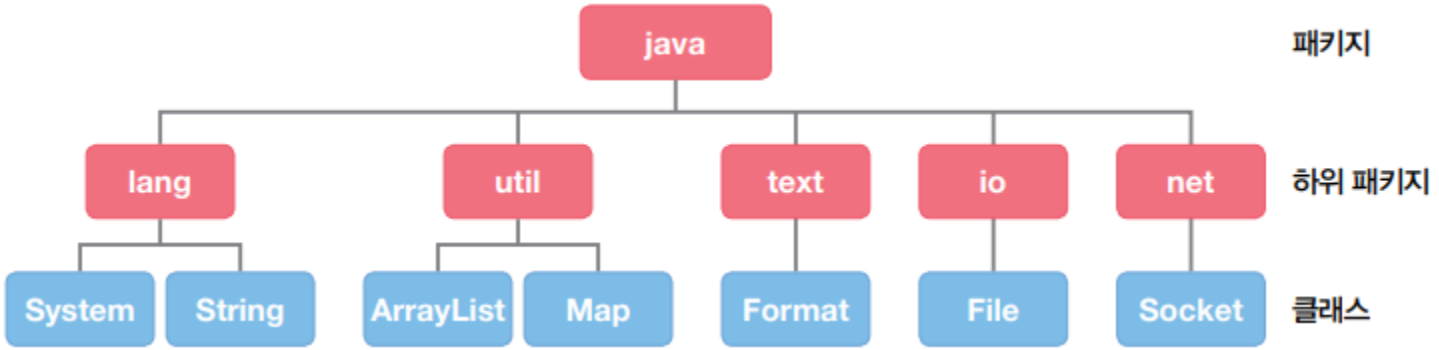
- 자바 프로그램에 사용할 수 있도록 다양한 패키지와 클래스를 제공하는 기본 패키지
- 가장 자주 사용되는 패키지: java.lang, java.util

  →java.lang 패키지는 아주 기본적인 것이라 임포트하지 않아도 사용할 수 있음
- 기본 패키지는 java로 시작하고 확장 패키지는 javax로 시작함
- 자바 API 문서

  → https://docs.oracle.com/en/java/javase/24/docs/api/

3. 내장 패키지

내장 패키지



주요 내장 패키지의 구성

주요 내장 패키지

패키지	설명
java.lang	자바 프로그램의 필수 패키지로 문자열, 수학 함수, 입출력 등 자바 프로그래밍에 필요한 기본적인 클래스와 인터페이스를 제공한다. import문 없이 사용할 수 있다.
java.util	자바 유틸리티 패키지로 날짜, 시간, 벡터, 해시맵 등의 다양한 유틸리티 클래스와 인터페이스를 제공한다.
java.text	날짜, 시간, 문자열 등 현지화를 처리하는 클래스를 제공한다.
java.io	입출력 처리 패키지로 키보드, 모니터, 프린터, 디스크 등에 입출력할 수 있는 클래스와 인터페이스를 제공한다.
java.net	네트워크 프로그램을 구현하기 위한 클래스를 제공한다.

3. java.lang 패키지

java.lang 패키지

- 자바의 가장 기본적인 클래스와 인터페이스가 담겨 있는 패키지
- import문 없이 사용할 수 있음

java.lang 패키지에 포함된 주요 클래스

클래스	설명
Object	모든 클래스의 최상위 클래스로, 배열을 포함한 모든 객체는 이 클래스의 메서드를 구현한다.
System	표준 입력 장치로부터 데이터를 입력받거나 표준 출력 장치로 출력할 때 사용한다.
Class	클래스를 메모리로 로딩할 때 사용한다.
String	문자열을 저장하고 여러 가지 정보를 얻을 때 사용한다.
StringBuffer, StringBuilder	문자열을 저장하고 내부 문자열을 조작할 때 사용한다.
Math	수학 함수를 이용할 때 사용한다.
Wrapper	기본 자료형의 객체를 만들 때, 문자열을 기본 자료형으로 변환할 때, 입력값을 검사할 때 사용한다.

3. java.lang 패키지

Object 클래스

- 모든 클래스의 상위
- 자바의 최상위 클래스이므로 자바의 모든 클래스는 Object 클래스의 모든 메서드를 바로 사용할 수 있음
- Object 클래스의 메소드 예

→ boolean equals(Object obj) : 주어진 객체 obj와 동일한지 여부를 나타냄

→ String toString() : 객체가 가지고 있는 정보나 값들을 문자열로 만들어 리턴하는 메소드로 일반적으로 재정의하여 사용

3. java.lang 패키지

Object 클래스

Object 클래스의 메서드 사용 예시

```
package com.section03;

import com.javamaster.mypackage.Cat;

public class Example03 {
    public static void main(String[] args) {
        Cat cat01 = new Cat();
        Cat cat02 = new Cat();
        System.out.println(cat01.toString());
        System.out.println(cat02.toString());
        System.out.println(cat01.equals(cat02));
        cat01 = cat02;
        System.out.println(cat01.equals(cat02));
    }
}
```

toString() : 객체가 가지고 있는 정보나 값들을 문자열로 만들어 리턴하는 메소드로 일반적으로 재정의하여 사용

실행 결과

```
com.javamaster.mypackage.Cat@1c4af82c
com.javamaster.mypackage.Cat@379619aa
false
true
```

3. java.lang 패키지

예제 2 java.lang.Object 패키지 사용하기

com.javamaster.mypackage 패키지

```
01 package com.javamaster.mypackage;
02
03 public class Dog {
04     String breed;
05     String color;
06
07      public String bowwow() {
08         return "멍멍 짖다";
09      }
10
11     public void run() {
12       System.out.println("달리다");
13     }
14 }
```

3. java.lang 패키지

com.section03 패키지

```
01 package com.section03;
02
03 import com.javamaster.mypackage.Dog;
04
05 public class Package02 {
06     public static void main(String[] args) {
07     Dog dog01 = new Dog();
08     Dog dog02 = new Dog();
09     System.out.println(dog01.equals(dog02));
10
11     String str1 = dog01.bowwow();
12     String str2 = dog02.bowwow();
13     System.out.println(str1.equals(str2));
14    }
15 }
```

실행 결과

```
false
true
```

3. java.lang 패키지

Math 클래스

- 수학에서 자주 사용하는 상수와 함수를 미리 구현해 놓은 클래스
- Math 클래스의 모든 메서드는 정적 메서드이므로 객체를 생성하지 않고도 바로 사용할 수 있음

Math 클래스의 메서드

메서드	설명
`double abs(double a)`	절댓값을 반환한다.
`double max(double a, double b)` `int max(int a, int b)`	두 값 중 큰 값을 반환한다.
`double min(double a, double b)` `int min(int a, int b)`	두 값 중 작은 값을 반환한다.
`Long round(double a)` `int round(float a)`	십진수를 가장 가까운 값으로 반올림할 때 사용한다.
`double random()`	0.0보다 크거나 같고 1.0보다 작은, 양의 부호가 있는 실숫값을 반환한다.
`double sqrt(double a)`	반올림된 양의 제곱근을 반환한다.
`double pow(double a, double b)`	첫 번째 인수의 값을 두 번째 인수로 거듭제곱한 값을 반환한다.
`double sin(double a)`	삼각형의 사인값을 반환한다.
`double cos(double a)`	삼각형의 코사인값을 반환한다.
`double tan(double a)`	삼각형의 탄젠트값을 반환한다.

3. java.lang 패키지

Math 클래스

Math 클래스의 메서드 사용 예시

```
package com.section03;

public class Example04 {
    public static void main(String[] args) {
        System.out.println((int)(Math.random() * 100));
        System.out.println(Math.abs(-10));
        System.out.println(Math.round(4.5));
        System.out.println(Math.sqrt(4.0));
    }
}
```

실행 결과

```
5
10
5
2.0
```

랜덤으로 출력되는 값이므로 실제 실행 결과는 다를 수 있음

3. java.lang 패키지

예제 3 java.lang.Math 패키지 사용하기

```
01 package com.section03;
02
03 public class Package03 {
04     public static void main(String[] args) {
05         System.out.println(Math.pow(2, 2));  // 제곱
06         System.out.println(Math.ceil(9.3));  // 올림 : 전달된 실수의 소수부분을 무조껀 올림
07         System.out.println(Math.floor(9.9));  // 내림 : 전달된 실수의 소수부분을 무조껀 버림
           // 반올림 : 전달된 실수의 소수점 첫번째 자리를 반올림하여 정수로 리턴
08         System.out.println(Math.round(9.4));
09         System.out.println(Math.max(5, 100)); // 최댓값
10         System.out.println(Math.min(5, 100)); // 최솟값
11   }
12 }
```

```
10.0
9.0
10
100
5
```

Section 04

접근제한자

4. 접근제한자

■ 접근제한자의 개념

- 접근제한자

→접근 제한자 (access modifier)란 외부 접근을 제어하는 키워드

→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

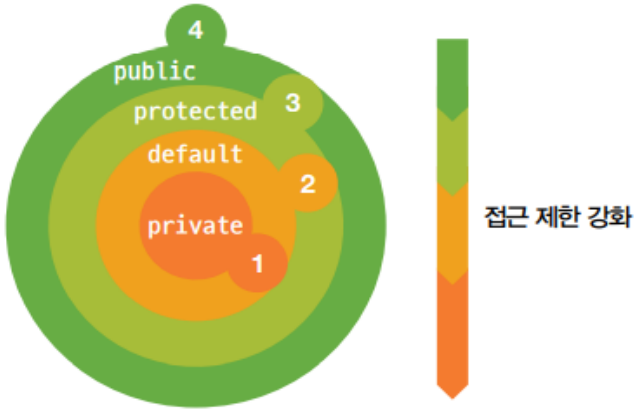
접근제한자의 접근 가능 범위

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

4. 접근제한자

■ 접근제한자의 제한 범위

- public: 모든 곳에서 접근 가능
- protected: 동일 패키지 또는 상속 관계가 있는 하위 클래스의 접근 허용
- 선언하지 않음(default): 동일 패키지(폴더)에 존재하는 클래스의 접근 허용
- private: 클래스 내에서만 접근 가능
- 클래스의 접근제한자 : 2개의 접근제한자만 **public**과 **default** 사용가능
- 생성자, 필드, 메소드의 접근 제한자 : 4가지 모두 사용 가능



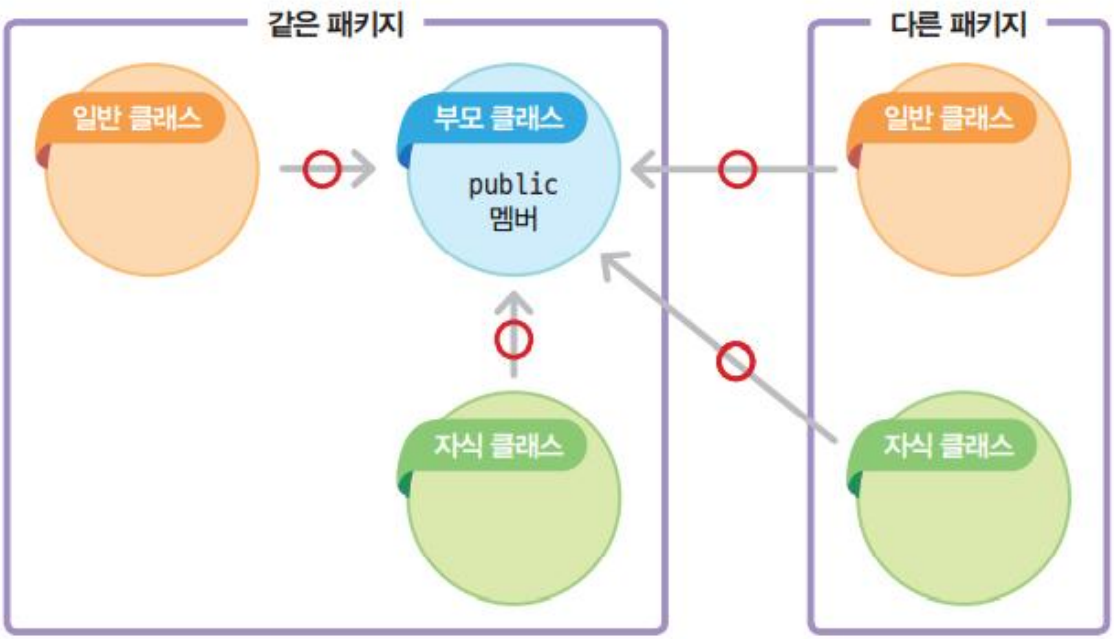
접근제한자의 제한 범위

4. 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



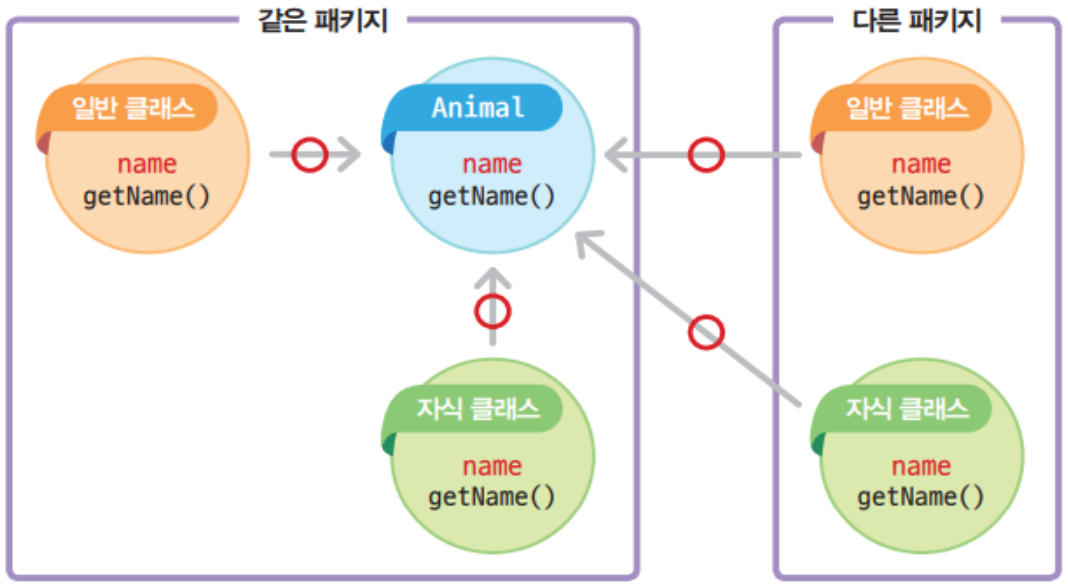
public 접근제한자의 접근 예

4. 접근제한자

public 접근제한자

- [public 접근제한자로 선언된 Animal 클래스의 예]

```
public class Animal {
   public String name;

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



public 클래스 Animal의 접근 가능 범위

4. 접근제한자

예제 1 public 접근제한자를 이용하여 클래스 멤버 호출하기 Dog.java

```
01 public class Dog {
02    public String breed;
03    public String color;
04
05    public void bowwow() {
06            System.out.println("멍멍 짖다");
07   }
08 }
```

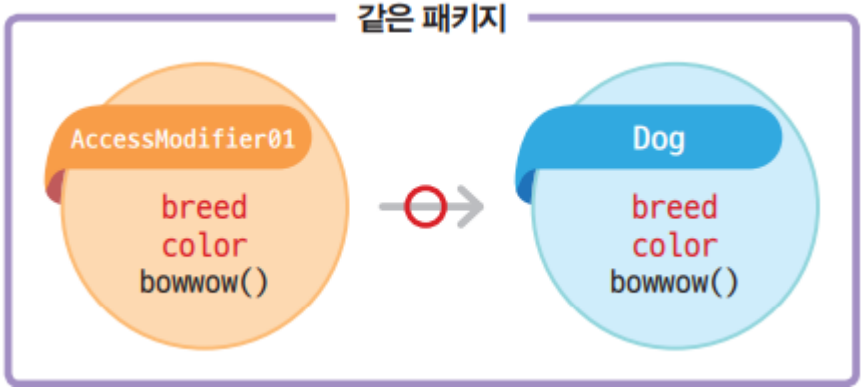
4. 접근제한자

AccessModifier01.java

```
01 public class AccessModifier01 {
02    public static void main(String[] args) {
03       Dog dogObj = new Dog();
04
05       dogObj.breed = "포메라니언";
06       dogObj.color = "갈색";
07
08       System.out.println("강아지 품종 : " + dogObj.breed);
09       System.out.println("강아지 색상 : " + dogObj.color);
10
11       dogObj.bowwow();
12    }
13 }
```

실행 결과

```
강아지 품종 : 포메라니언
강아지 색상 : 갈색
멍멍 짖다
```

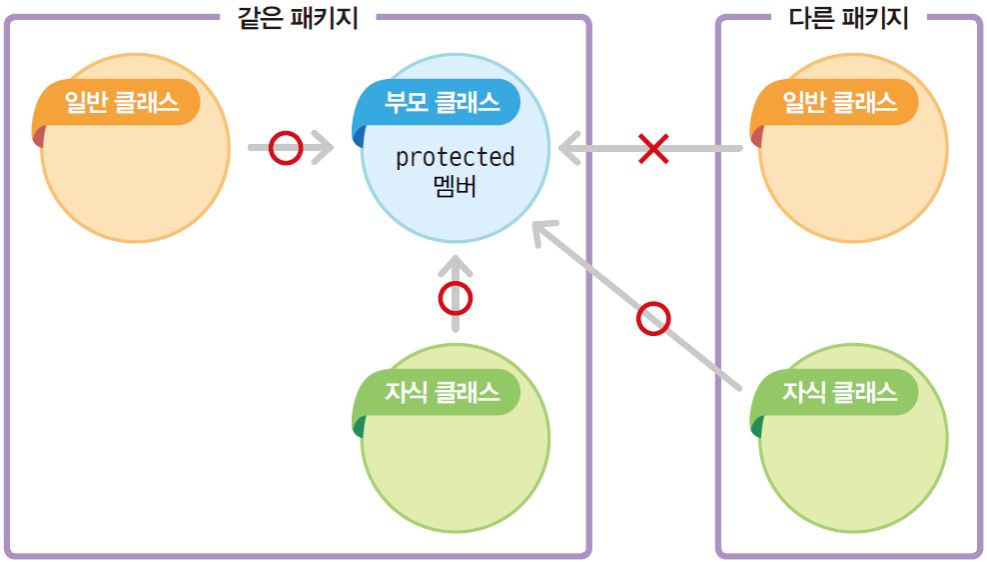


public 접근제한자 선언의 클래스 멤버 접근 예

4. 접근제한자

protected 접근제한자

- 상위(부모) 클래스에서 protected가 선언된 변수, 메서드, 생성자는 다른 패키지의 하위(자식) 클래스에서만 접근할 수 있음
- 같은 패키지의 클래스는 protected가 선언된 하위(자식) 클래스가 아니더라도 클래스 멤버에 접근할 수 있음



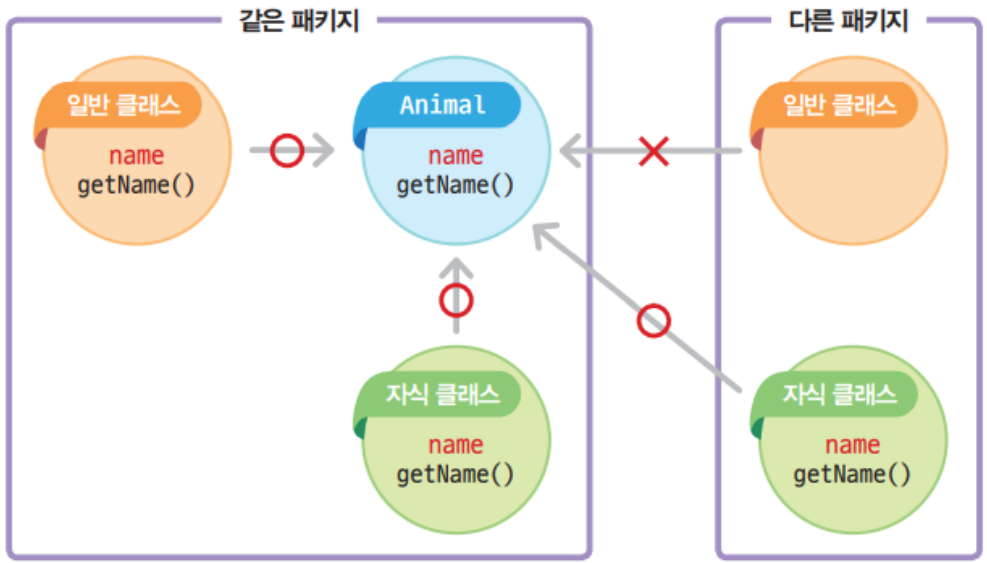
protected 접근제한자의 접근 예

4. 접근제한자

protected 접근제한자

- [protected 접근제한자로 선언된 Animal 클래스의 예]

```
public class Animal {
   protected String name;

   protected String getName() {
      return name;
   }
}
```



protected 클래스 Animal의 접근 가능 범위

4. 접근제한자

예제 2 protected 접근제한자를 이용하여 클래스 멤버 호출하기 Dog.java

```
01 public class Dog {
02    public String breed;
03    public String color;
04    protected int age;
05
06    public void bowwow() {
07      System.out.println("멍멍 짖다");
08    }
09
10    protected void run() {
11     System.out.println("달리다");
12   }
13 }
```

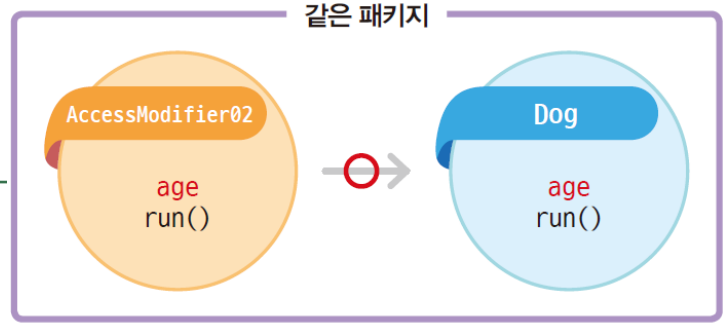
4. 접근제한자

AccessModifier02.java

```
01 public class AccessModifier02 {
02    public static void main(String[] args) {
03      Dog obj = new Dog();
04
05      obj.breed = "포메라니언";
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 }
```

실행 결과

```
강아지 품종 : 포메라니언
강아지 색상 : 갈색
멍멍 짖다
강아지 나이 : 10
달리다
```

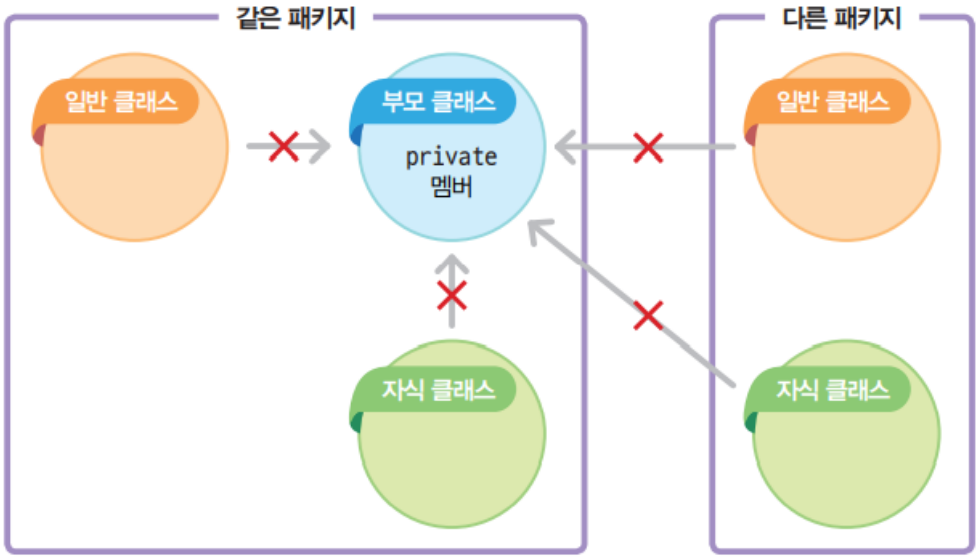


protected 접근제한자 선언의 클래스 멤버 접근 예

4. 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



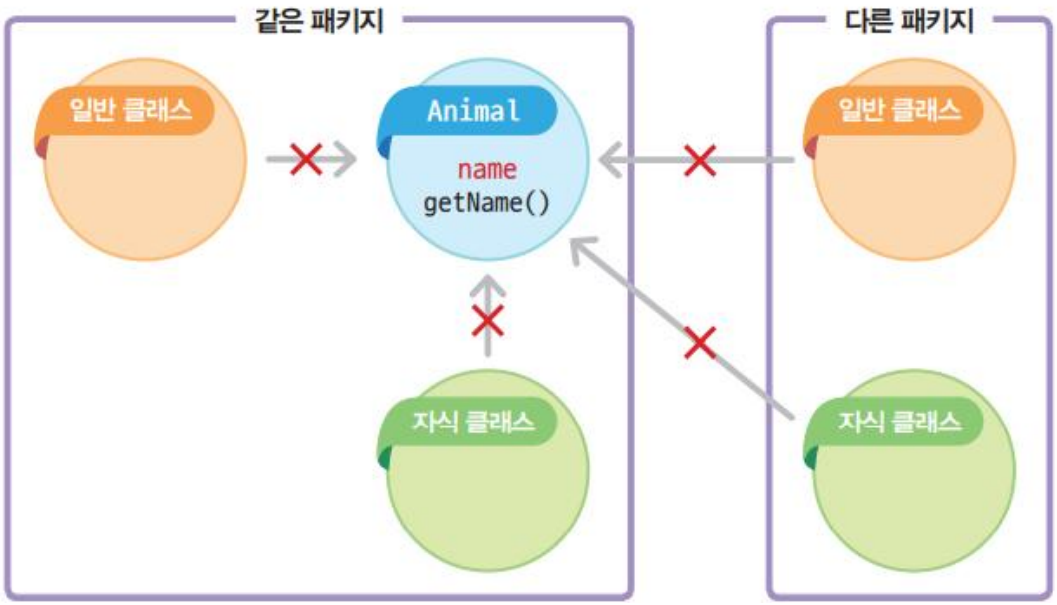
private 접근제한자의 접근 예

4. 접근제한자

■ private 접근제한자

- private 접근제한자로 선언된 Animal 클래스의 예

```
public class Animal {
   private String name;

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



private 클래스 Animal의 접근 가능 범위

4. 접근제한자

private 접근제한자

- private 접근 제한자가 적용된 경우

```
class Account {
    private int balance;     // 잔액
}
```

외부로부터 감춰져, 모든 외부 접근을 막을 수 있음

- Account 클래스 외부에서 balance 필드가 보이지 않음

```
Account acc = new Account();
acc.balance = 1000000;       // ERROR
```

private 필드는 클래스 외부에서 접근할 수 없으므로 값 변경 불가

TIP 객체의 정보를 외부로부터 감추는 것을 **정보 은닉**(information hiding)이라고 함
private을 사용한 정보 은닉은 필드뿐 아니라 메소드에도 적용 가능

4. 접근제한자

■ private 접근제한자

- 게터(getter) 메소드

→ private 선언된 필드는 클래스 내부에서만 사용할 수 있음

→ 외부에서 사용하려면, public 메소드를 통해 우회하여 접근해야 함

→ 이러한 메소드를 게터(getter)라고 함

```
class Account {
    private int balance;        // 잔액
    public int getBalance() {
        return balance;
    }
}
```

게터
- private 필드를 외부로 반환하는 **public** 메소드
- **get + 필드명**의 형태로 작성되며, 반환 타입은 대상 필드와 같음
- public 선언되었으므로 클래스 밖의 모든 코드에서 호출 가능

4. 접근제한자

■ private 접근제한자

- 게터(getter) 메소드

→ 게터 메소드를 호출하면 클래스 외부에서도 private 필드를 가져올 수 있음

```
Account acc = new Account();                    private 필드는 외부에서 접근 불가
System.out.println("balance: " + acc.balance);          // ERROR
System.out.println("balance: " + acc.getBalance());    // balance: 0
                                                  게터 메소드 호출을 통한 우회 접근
```

→ 게터의 호출 과정

4. 접근제한자

private 접근제한자

- 세터(setter) 메소드

→ 외부에서 private 필드를 변경하고 싶을 땐 세터(setter) 메소드를 정의

```
class Account {
    private int balance;

    public int getBalance() {
        return balance;
    }
    public void setBalance(int b) {
        balance = b;
    }
}
```

세터
- private 필드를 외부에서 변경하게하는 **public** 메소드
- **set + 필드명**의 형태로 작성되며, 반환 타입은 **void**
- public 선언되었으므로 클래스 밖의 모든 코드에서 호출 가능

4. 접근제한자

게터와 세터

```
01 public class AccountTest {
02     public static void main(String[] args) {
03         Account acc = new Account();
04         // acc.balance = 1000; // ERROR
05         acc.setBalance(1000);
06         System.out.printf("잔액: %d", acc.getBalance());
07     }
08 }
09
10 class Account {
11     private int balance;  // 잔액, private 적용
12
13     // 게터 메소드
14     public int getBalance() {
15         return balance;
16     }
17
18     // 세터 메소드
19     public void setBalance(int b) {
20         balance = b;
21     }
22 }
```

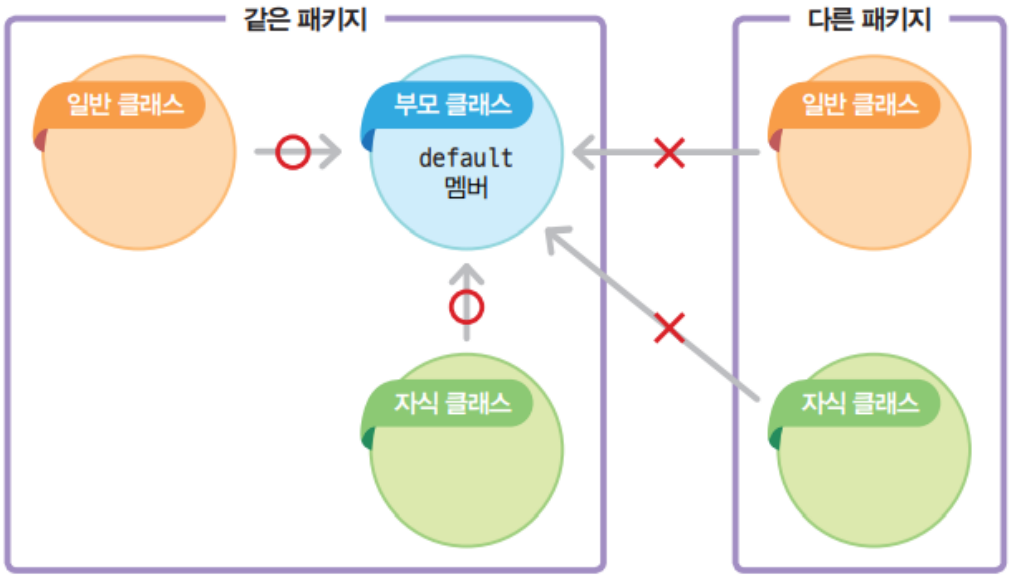
private 필드는 외부에서 사용 불가

세터 메소드 호출

4. 접근제한자

default 접근제한자

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



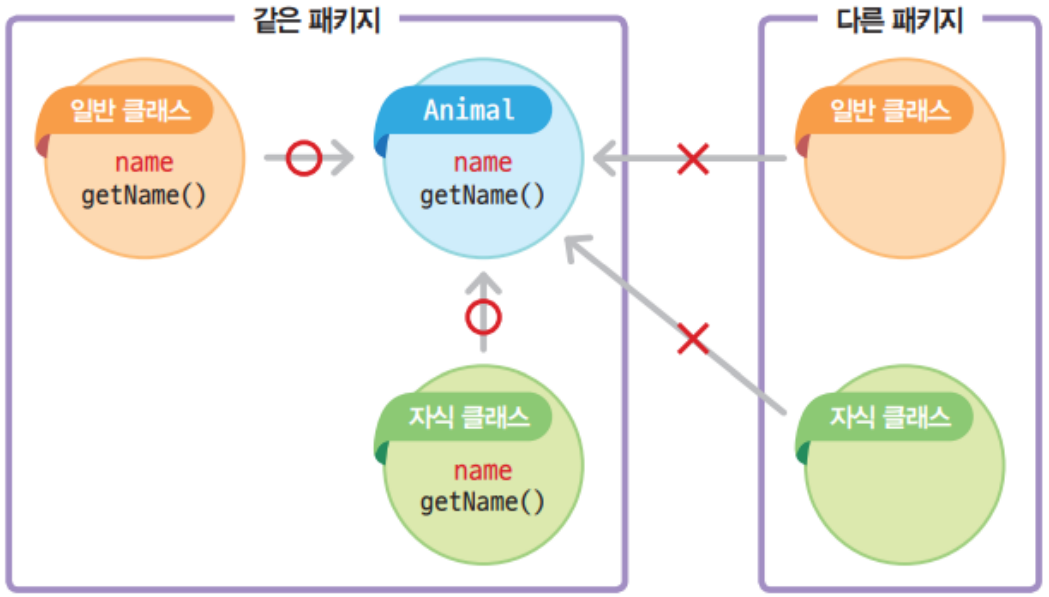
default 접근제한자의 접근 예

4. 접근제한자

default 접근제한자

- 접근제한자가 없는 클래스의 멤버 변수와 멤버 메서드의 예

```
public class Animal {
   String name;

   String getName() {
      return name;
   }
}
```



default 클래스 Animal의 접근 가능 범위

4. 접근제한자

예제 3 접근제한자가 없는 클래스 멤버 호출하기 Dog.java

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

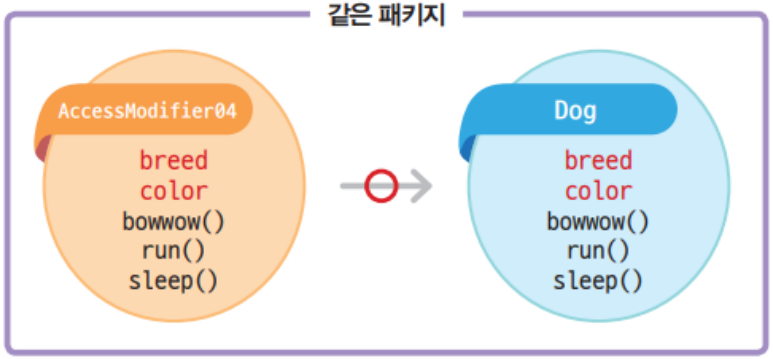
4. 접근제한자

AccessModifier04.java

```
01 public class AccessModifier04 {
02    public static void main(String[] args) {
03       Dog obj = new Dog();
04
05       obj.breed = "포메라니언";
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    }
12 }
```

실행 결과

```
강아지 품종 : 포메라니언
강아지 색상 : 갈색
멍멍 짖다
```



접근제한자가 없는 클래스 멤버 접근 예

실습해보기

■ **다음에 제시된 웹툰 목록을 객체화하여 배열로 만들고, 출력 예와 같은 결과를 얻으시오.**

- 나 혼자만 레벨업업(장르: 판타지, 작가: 추공님)
- 스위트 집(장르: 스릴러, 작가: 칸비님)
- 이태원 클래식(장르: 드라마, 작가: 광진님)

- 실행결과

```
Webtoon { title: 나 혼자만 레벨업업, genre: 판타지, author: 추공님 }
Webtoon { title: 스위트 집, genre: 스릴러, author: 칸비님 }
Webtoon { title: 이태원 클래식, genre: 드라마, author: 광진님 }
웹툰 객체의 총 수: 3
```

실습해보기

■ **다음에 제시된 웹툰 목록을 객체화하여 배열로 만들고, 출력 예와 같은 결과를 얻으시오.**

- 나 혼자만 레벨업업(장르: 판타지, 작가: 추공님)
- 스위트 집(장르: 스릴러, 작가: 칸비님)
- 이태원 클래식(장르: 드라마, 작가: 광진님)

- 실행결과

```
Webtoon { title: 나 혼자만 레벨업업, genre: 판타지, author: 추공님 }
Webtoon { title: 스위트 집, genre: 스릴러, author: 칸비님 }
Webtoon { title: 이태원 클래식, genre: 드라마, author: 광진님 }
웹툰 객체의 총 수: 3
```

실습해보기

과정 1

- WebtoonTest 클래스를 만들고 내부에는 Webtoon 클래스를 작성
- 인스턴스 변수와 클래스 변수는 정보보호를 위해 private 선언

```
public class WebtoonTest {
	public static void main(String[] args) {
	}
}

class Webtoon {
	private String title; // 제목
	private String genre; // 장르
	private String author; // 저자
	private static int count = 0; // 생성된 웹툰 객체의 수
}
```

실습해보기

과정 2

- Webtoon 생성자를 작성하고 객체 생성

```
class Webtoon {
	private String title; // 제목
	private String genre; // 장르
	private String author; // 저자
	private static int count = 0; // 생성된 웹툰 객체의 수

	public Webtoon(String t, String g, String a) {
		title = t;
		genre = g;
		author = a;
		Webtoon.count++;
	}
}
```

실습해보기

과정 2

- Webtoon 생성자를 작성하고 객체 생성

```
public class WebtoonTest {
      public static void main(String[] args) {
         Webtoon levelUpUp = new Webtoon("나 혼자만 레벨업업", "판타지",
                                                 "추공님");
         Webtoon sweetHouse = new Webtoon("스위트 집","스릴러","칸비님");

         Webtoon itaewonClassic = new Webtoon("이태원 클래식",드라마",
                                                    "광진님");
      }
}
```

실습해보기

과정 3

- 인스턴스 메소드 toStr()과 클래식 메소드 getCount()를 작성

```
class Webtoon {
        private String title; // 제목
        private String genre; // 장르
        private String author; // 저자
        private static int count = 0; // 생성된 웹툰 객체의 수

        public Webtoon(String t, String g, String a) {
                …
        }
        public String toStr() {
                return String.format("Webtoon { title: %s,
                genre: %s, author: %s }",title, genre, author);
        }
        public String toString() {//Object 클래스의 메소드를 재정의하여 사용하는 경우
                return String.format("Webtoon { title: %s,
                genre: %s, author: %s }",title, genre, author);
        }
        public static int getCount() {
                return Webtoon.count;
        }
}
```

실습해보기

과정 3

- 객체의 정보와 개수를 출력

```
public class WebtoonTest {
	public static void main(String[] args) {
		Webtoon levelUpUp = new Webtoon("나 혼자만 레벨업업", "판타지",
                                                "추공님");
		Webtoon sweetHouse = new Webtoon("스위트 집","스릴러","칸비님");

		Webtoon itaewonClassic = new Webtoon("이태원 클래식",드라마",
                                                    "광진님");


		System.out.println(levelUpUp.toStr());
		System.out.println(sweetHouse.toStr());
		System.out.println(itaewonClassic.toString());
		System.out.printf("웹툰 객체의 총 수: %d", Webtoon.getCount());
	}
}
```

실습해보기

과정 4

- 객체 배열을 활용한 코드로 변경하여 출력

```
public class WebtoonTest {
	public static void main(String[] args) {
		Webtoon levelUpUp = new Webtoon("나 혼자만 레벨업업", "판타지",
                                                "추공님");
		Webtoon sweetHouse = new Webtoon("스위트 집","스릴러","칸비님");

		Webtoon itaewonClassic = new Webtoon("이태원 클래식",드라마",
                                                        "광진님");

		Webtoon[] webtoons = {levelUpUp, sweetHouse, itaewonClassic};

		for (int i = 0; i < webtoons.length; i++) {
			System.out.println(webtoons[i].toStr());
		}
		System.out.printf("웹툰 객체의 총 수: %d", Webtoon.getCount());
	}
}
```

Thank you!