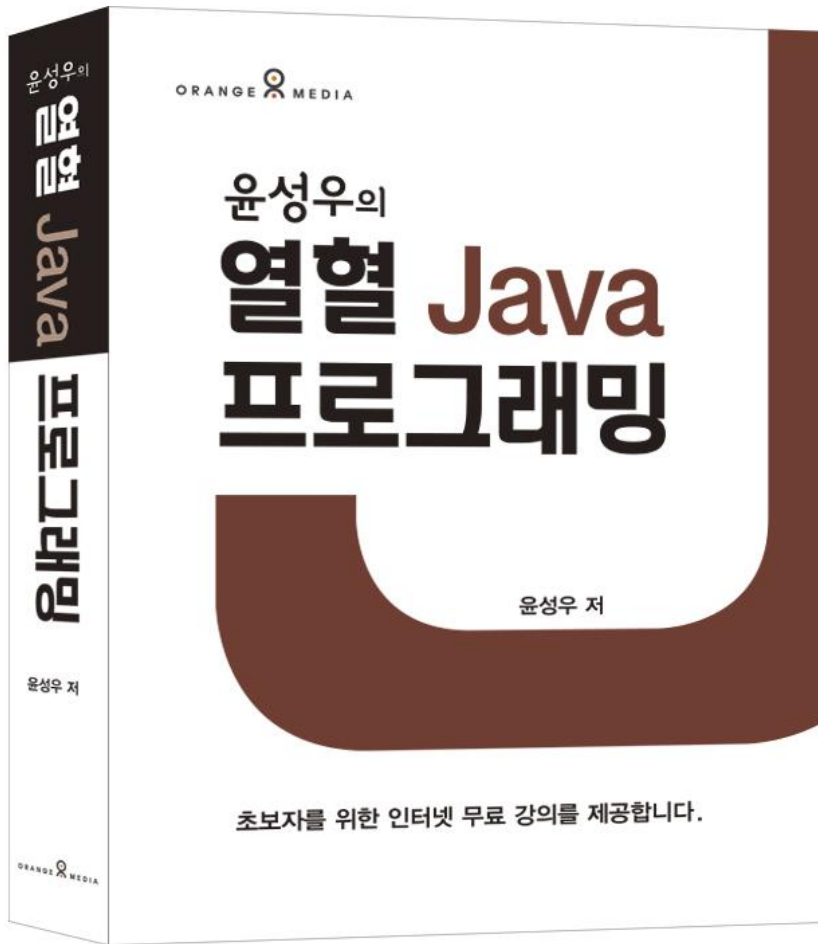


2022년 봄학기

JAVA

나사렛대학교
IT융합학부
김광기



열혈 Java 프로그래밍

Chapter 08. 패키지와 클래스 패스

08-1. 클래스 패스

현재 디렉토리에 대한 이해

현재 디렉토리: 실행 중인 프로그램의 작업 디렉토리

C:\PackageStudy>

현재 디렉토리는 C:\PackageStudy

C:\PackageStudy>javac WhatYourName.java

현재 디렉토리 C:\PackageStudy를 기준으로 자바 파일을
찾는다.

WhatYourName.java

```
class AAA { ... }  
class ZZZ { ... }  
class WhatYourName {  
    public static void main(String args[]) {  
        AAA aaa = new AAA();  
        aaa.showName();  
  
        ZZZ zzz = new ZZZ();  
        zzz.showName();  
    }  
}
```

디렉토리 구조 변경 및 컴파일

WhatYourName.class

C:\PackageStudy에 위치시킨다.

AAA.class

C:\PackageStudy\MyClass에 위치시킨다.

ZZZ.class

C:\PackageStudy\MyClass에 위치시킨다.

C:\PackageStudy>java WhatYourName

실행이 가능하게 하려면?

WhatYourName.java

```
class AAA { ... }  
class ZZZ { ... }  
class WhatYourName {  
    public static void main(String args[]) {  
        AAA aaa = new AAA();  
        aaa.showName();  
  
        ZZZ zzz = new ZZZ();  
        zzz.showName();  
    }  
}
```

클래스 패스란?

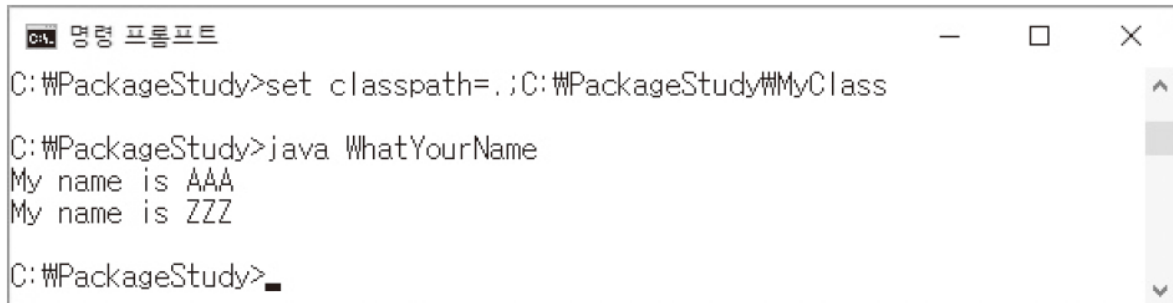
클래스 패스: 자바 가상머신의 클래스 탐색 경로

C:\PackageStudy>set classpath



```
명령 프롬프트
C:\PackageStudy>set classpath
classpath 환경 변수가 정의되지 않았습니다.
C:\PackageStudy>
```

C:\PackageStudy>set classpath=.;C:\PackageStudy\MyClass



```
명령 프롬프트
C:\PackageStudy>set classpath=.;C:\PackageStudy\MyClass
C:\PackageStudy>java WhatYourName
My name is AAA
My name is ZZZ
C:\PackageStudy>
```

절대 경로 vs 상대 경로

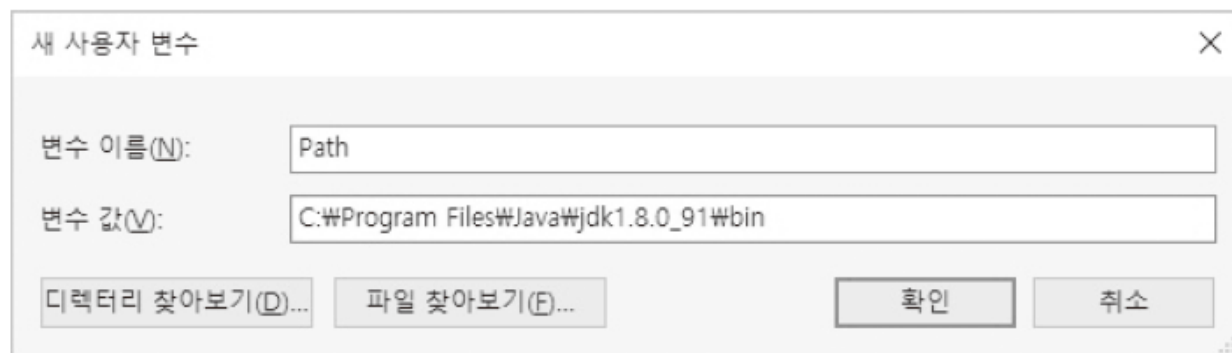
```
C:\PackageStudy>set classpath=.;C:\PackageStudy\MyClass
```

루트 디렉토리를 시작으로 지정한 '절대 경로'

```
C:\PackageStudy>set classpath=.;.\MyClass
```

현재 디렉토리를 기준으로 지정한 '상대 경로'

클래스 패스를 고정시키는 방법



변수의 이름으로 classpath, 변수 값으로 경로 정보 전달하면 클래스 패스가 시스템 전체에 적용이 된다.

But! 좋은 방법 아니므로, 이렇듯 클래스 패스를 고정시키는 일이 가능하다는 사실 정도만 알고 있자.

08-2. 패키지의 이해

패키지 선언이 필요한 상황의 연출

```
public class Circle {  
    double rad;  
    final double PI;  
  
    public Circle(double r) {  
        rad = r;  
        PI = 3.14;  
    }  
    public double getArea() {  
        return (rad * rad) * PI;  
    }  
}
```

www.wxfox.com의 Circle.java

```
public class Circle {  
    double rad;  
    final double PI;  
  
    public Circle(double r) {  
        rad = r;  
        PI = 3.14;  
    }  
    public double getPerimeter() {  
        return (rad * 2) * PI;  
    }  
}
```

www.fxmx.com의 Circle.java

공간에서의 충돌

동일 이름의 클래스 파일을 같은 위치에 둘 수
없다.

접근 방법에서의 충돌

인스턴스 생성 방법에서 두 클래스에 차이가
없다.

공간적, 접근적 충돌 해결을 위한 패키지 선언

클래스 접근 방법의 구분

- 서로 다른 패키지의 두 클래스는 인스턴스 생성 시 사용하는 이름이 다르다.

클래스의 공간적인 구분

- 서로 다른 패키지의 두 클래스 파일은 저장되는 위치가 다르다.

컴파일 과정에서, 클래스 파일이 저장되어야 하는 위치를 상대적으로 결정이 된다.

그리고 이렇게 결정된 위치는 컴파일 이후에 바꿀 수 없다.

패키지 선언에 따른 문제 해결

```
package com.wxfx.smart;
```

```
public class Circle {  
    double rad;  
    final double PI;  
  
    public Circle(double r) { ... }  
    public double getArea() { ... }  
}
```

www.wxfx.com의 Circle.java

```
package com.fxmx.simple;
```

```
public class Circle {  
    double rad;  
    final double PI;  
  
    public Circle(double r) { ... }  
    public double getPerimeter() { ... }  
}
```

www.fxmx.com의 Circle.java

패키지 이름은 모두 소문자로 구성

인터넷 도메인 이름의 역순으로 이름을 구성

이름 끝에 클래스를 정의한 주체 또는 팀의 이름 추가

```
com.wxfx.smart.Circle c1 = new com.wxfx.smart.Circle(3.5);
```

```
com.fxmx.simple.Circle c2 = new com.fxmx.simple.Circle(5.5);
```

-d 옵션을 주고 컴파일 하면 패키지 디렉토리도 자동 생성

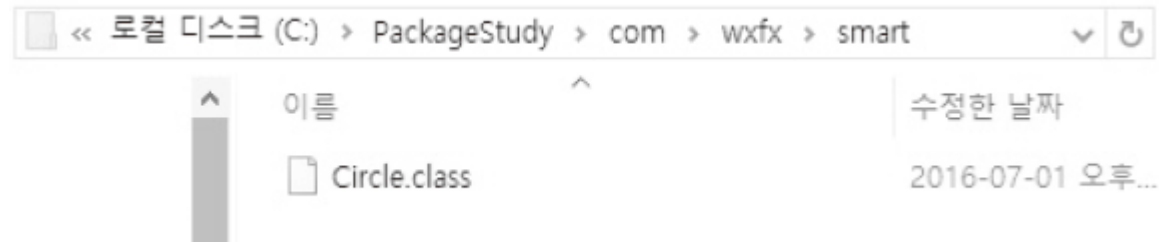
패키지 선언이 된 소스파일 컴파일 방법

```
C:\PackageStudy>javac -d <directory> <filename>
```

<directory> 패키지를 생성할 위치 정보

<filename> 컴파일할 파일의 이름

```
C:\PackageStudy>javac -d . src\circle1\Circle.java
```



패키지로 묶인 클래스의 접근

```
com.wxfx.smart.Circle c1 = new com.wxfx.smart.Circle(3.5);
```



클래스 하나에 대한 import 선언

◆ ImportCircle.java

```
1. import com.wxfx.smart.Circle;
2.
3. class ImportCircle {
4.     public static void main(String args[]) {
5.         Circle c1 = new Circle(3.5);
6.         System.out.println("반지름 3.5 원 넓이: " + c1.getArea());
7.         Circle c2 = new Circle(5.5);
8.         System.out.println("반지름 5.5 원 넓이: " + c2.getArea());
9.     }
10. }
```

```
import com.wxfx.smart.Circle;
import com.fxmx.simple.Circle;
```

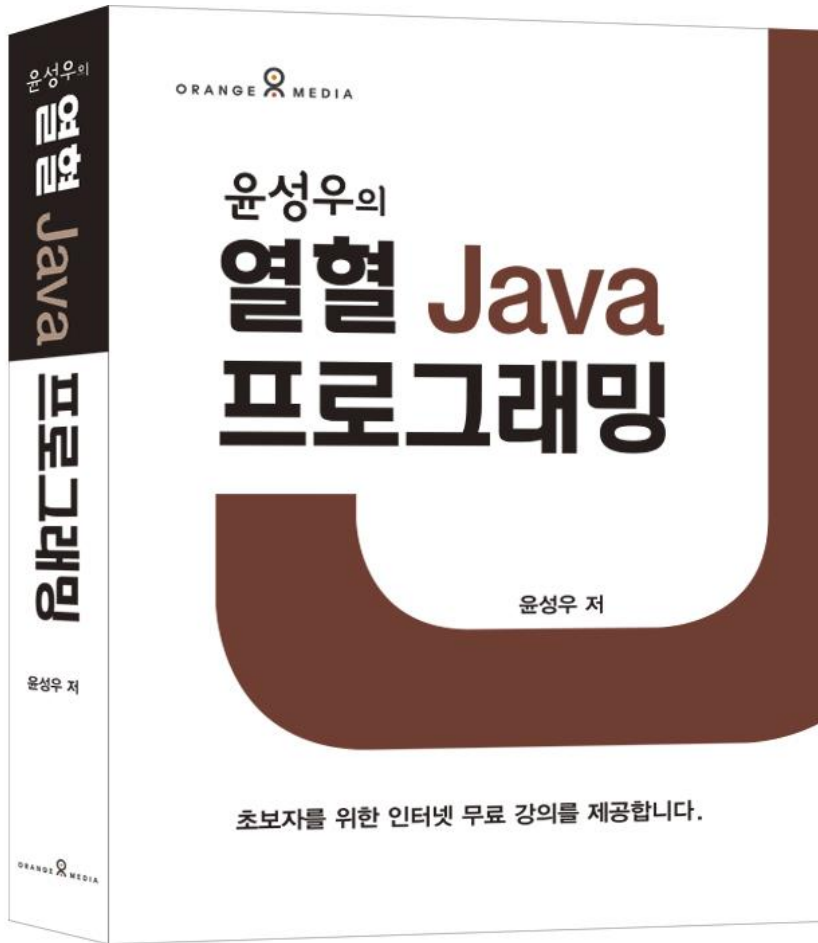
동일 이름의 두 클래스에 대한 import 선언은 컴파일 오류

동일 이름은 X

패키지 전체에 대한 import 선언

```
import com.wxfx.smart.*;
```

com.wxfx.smart 패키지로 묶인 전체 클래스에 대한 패키지 선언



열혈 Java 프로그래밍

Chapter 09. 정보 은닉 그리고 캡슐화

09-1. 정보 은닉

정보를 은닉해야 하는 이유

```
class Circle {  
    double rad = 0;        // 원의 반지름  
    final double PI = 3.14;  
  
    public Circle(double r) {  
        setRad(r);  
    }  
  
    public void setRad(double r) {  
        if(r < 0) {  
            rad = 0;  
            return;  
        }  
        rad = r;  
    }  
  
    public double getArea() {  
        return (rad * rad) * PI;  
    }  
}
```

```
public static void main(String args[]) {  
    Circle c = new Circle(1.5);  
    System.out.println(c.getArea());  
  
    c.setRad(2.5);  
    System.out.println(c.getArea());  
    c.setRad(-3.3);  
    System.out.println(c.getArea());  
    c.rad = -4.5;    // 컴파일 오류 발생 안함  
    System.out.println(c.getArea());  
}
```

→ Private로 선언

정보의 은닉을 위한 private 선언

```
class Circle {  
    private double rad = 0;  
    final double PI = 3.14;  
  
    public Circle(double r) {  
        setRad(r);  
    }  
    public void setRad(double r) { // Setter  
        if(r < 0) {  
            rad = 0;  
            return;  
        }  
        rad = r;  
    }  
    public double getRad() { // Getter  
        return rad;  
    }  
    public double getArea() {...}  
}
```

```
public static void main(String args[]) {  
    Circle c = new Circle(1.5);  
    System.out.println(c.getArea());  
  
    c.setRad(2.5);  
    System.out.println(c.getArea());  
    c.setRad(-3.3);  
    System.out.println(c.getArea());  
    c.rad = -4.5; // 컴파일 오류로 이어짐  
    System.out.println(c.getArea());  
}
```

→ 뭘든 메서드로!

09-2. 접근 수준 지시자

네 가지 종류의 접근 수준 지시자



클래스 정의 대상: public, default

인스턴스 변수와 메소드 대상: public, protected, default, private

클래스 정의 대상의 public과 default 선언이 갖는 의미

```
public class AAA {
```

```
    ....
```

```
} public으로 선언된 AAA 클래스
```

```
class ZZZ {
```

```
    ....
```

```
} default로 선언된 ZZZ 클래스
```

public 어디서든 인스턴스 생성이 가능하다.

default 동일 패키지로 묶인 클래스 내에서만 인스턴스 생성을 허용한다.

Cat.java

```
package zoo;

class Duck {
    // 빈 클래스
}

public class Cat {
    public void makeCat() {
        Duck quack = new Duck();
    }
}
```

Dog.java

```
package animal;

public class Dog {
    public void makeCat() {
        zoo.Cat yaong = new zoo.Cat();
    }
    public void makeDuck() {
        zoo.Duck quack = new zoo.Duck();
    }
}
```

OK!

ERROR!

간단하게!

클래스의 public, default 선언 관련 예

인스턴스 멤버 대상의 접근 수준 지시자 선언

```
class AAA {  
    public int num1;  
    protected int num2;  
    private int num3;  
    int num4;    // default 선언  
  
    public void md1() {...}  
    protected void md2() {...}  
    private void md3() {...}  
    void md4() {...}    // default 선언  
}
```

public 어디서든 접근 가능

default 동일 패키지로 묶인 클래스 내에서만 접근
가능

Cat.java

```
package zoo;

public class Cat {
    public void makeSound() {
        System.out.println("야옹");
    }
    void makeHappy() {
        System.out.println("스마일");
    }
}
```

default

Dog.java

```
package animal;

public class Dog {
    public void welcome(zoo.Cat c) {
        c.makeSound(); OK!

        c.makeHappy(); ERROR!
    }
}
```

인스턴스 멤버의 public, default 선언 관련 예

인스턴스 멤버의 private 선언이 갖는 의미

```
class Duck {  
    private int numLeg = 2;    // 클래스 내부에서만 접근 가능  
  
    public void md1() {  
        System.out.println(numLeg);    // 접근 가능  
        md2();    // 호출 가능  
    }  
  
    private void md2() {  
        System.out.println(numLeg);    // 접근 가능  
    }  
  
    void md3() {  
        System.out.println(numLeg);    // 접근 가능  
        md2();    // 호출 가능  
    }  
}
```

상속에 대한 약간의 설명: protected 선언의 의미 이해를 위한

AAA.java

```
public class AAA {  
    int num;  
}
```

디폴트 패키지

ZZZ.java

```
// extends AAA는 AAA 클래스의 상속을 의미함  
public class ZZZ extends AAA {  
    public void init(int n) {  
        num = n;    // 상속된 변수 num의 접근!  
    }  
}
```

디폴트 패키지

디폴트 패키지는 패키지 선언이 되어 있지 않은 클래스들을 하나의 패키지로 묶기 위한 개념

인스턴스 멤버의 protected 선언이 갖는 의미

AAA.java

```
package alpha;  
public class AAA {  
    protected int num;  
}
```

alpha 패키지

ZZZ.java

```
public class ZZZ extends alpha.AAA {  
    public void init(int n) {  
        num = n;    // 상속된 변수 num의 접근!  
    }  
}
```

디폴트 패키지

protected 선언으로 인해 상속 관계에서 접근,
가능 동일 패키지로 묶이지 않았더라도

인스턴스 멤버 대상 접근 수준 지시자 정리

지시자	클래스 내부	동일 패키지	상속 받은 클래스	이외의 영역
private	○	×	×	×
default	○	○	×	×
protected	○	○	○	×
public	○	○	○	○

09-3. 캡슐화

캡슐화 무너진 예(가정: 코감기는 콧물, 재채기, 코 막힘을 늘 동반한다.)

```
class SinivelCap {    // 콧물 처리용 캡슐
    void take() {
        System.out.println("콧물이 싹~ 납니다.");
    }
}

class SneezeCap {    // 재채기 처리용 캡슐
    void take() {
        System.out.println("재채기가 멎습니다.");
    }
}

class SnuffleCap {    // 코 막힘 처리용 캡슐
    void take() {
        System.out.println("코가 뽕 뚫립니다.");
    }
}
```

```
class ColdPatient {
    void takeSinivelCap(SinivelCap cap) {
        cap.take();
    }

    void takeSneezeCap(SneezeCap cap) {
        cap.take();
    }

    void takeSnuffleCap(SnuffleCap cap) {
        cap.take();
    }
}
```

약의 복용 순서가 중요하다면?

클래스 SinivelCap, SneezeCap, SnuffleCap의 적용 및 사용 방법이 별도로 존재한다면?

무너진 캡슐화의 결과

```
class BadEncapsulation {  
    public static void main(String[] args) {  
        ColdPatient suf = new ColdPatient();  
  
        // 콧물 캡슐 구매 후 복용  
        suf.takeSinivelCap(new SinivelCap());  
  
        // 재채기 캡슐 구매 후 복용  
        suf.takeSneezeCap(new SneezeCap());  
  
        // 코막힘 캡슐 구매 후 복용  
        suf.takeSnuffleCap(new SnuffleCap());  
    }  
}
```

캡슐화가 무너지면 이렇듯 클래스 사용 방법과 관련하여
알아야 할 사항들이 많이 등장한다.

- 복용해야 할 약의 종류
- 복용해야 할 약의 순서

결론적으로, 코드가 복잡해진다.

적절한 캡슐화의 예 (가정: 코감기는 콧물, 재채기, 코 막힘을 늘 동반한다.)

클래스 3 메서드 3

```
class SinivelCap { // 콧물 처리용 캡슐
```

```
void take() {  
    System.out.println("콧물이 싹~ 납니다.");  
}
```

```
class SneezeCap { // 재채기 처리용 캡슐
```

```
void take() {  
    System.out.println("재채기가 멎습니다.");  
}
```

```
class SnuffleCap { // 코 막힘 처리용 캡슐
```

```
void take() {  
    System.out.println("코가 땡 뚫립니다.");  
}
```

클래스 1 메서드 3

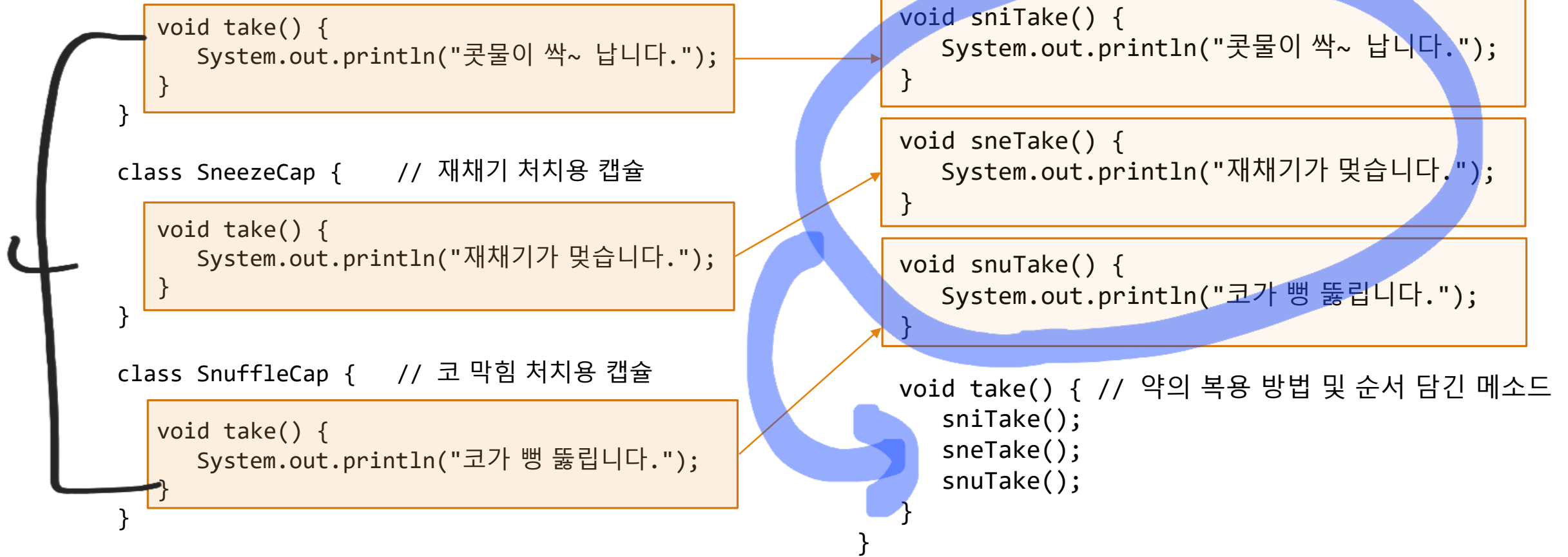
```
class SinusCap {
```

```
void sniTake() {  
    System.out.println("콧물이 싹~ 납니다.");  
}
```

```
void sneTake() {  
    System.out.println("재채기가 멎습니다.");  
}
```

```
void snuTake() {  
    System.out.println("코가 땡 뚫립니다.");  
}
```

```
void take() { // 약의 복용 방법 및 순서 담긴 메소드  
    sniTake();  
    sneTake();  
    snuTake();  
}
```



적절한 캡슐화로 인한 코드 수준의 향상

```
class ColdPatient {  
    void takeSinus(SinusCap cap) {  
        cap.take();  
    }  
}  
  
class OneClassEncapsulation {  
    public static void main(String[] args) {  
        ColdPatient suf = new ColdPatient();  
        suf.takeSinus(new SinusCap());  
    }  
}
```

코감기 관련해서 알아야 할 사실들이 많이 줄었다.

SinivelCap, SneezeCap, SnuffleCap 클래스들은 몰라도 된다.
SinusCap 클래스 하나만 알면 된다.

복용 순서 몰라도 된다.

take 메소드를 통해 복용 과정이 모두 자동화 된다.

포함 관계로 캡슐화 완성하기

```
class SinivelCap {    // 콧물 처리용 캡슐
    void take() {
        System.out.println("콧물이 싹~ 납니다.");
    }
}
```

```
class SneezeCap {    // 재채기 처리용 캡슐
    void take() {
        System.out.println("재채기가 멎습니다.");
    }
}
```

```
class SnuffleCap {    // 코 막힘 처리용 캡슐
    void take() {
        System.out.println("코가 땡 뚫립니다.");
    }
}
```

```
class SinusCap {
    SinivelCap siCap = new SinivelCap();
    SneezeCap szCap = new SneezeCap();
    SnuffleCap sfCap = new SnuffleCap();

    void take() {
        siCap.take(); szCap.take(); sfCap.take();
    }
}
```