



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());  
}
```

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());
}
```

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.

Dog.java

```
package zoo;

class Duck {
    public void makeSound() {
        System.out.println("quack~");
    }
}

public class Dog {
    public void makeSound() {
        System.out.println("bowwow~");
    }
}
```

Animal.java

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

        zoo.Dog dog = new zoo.Dog();
        dog.makeSound();    OK!

        zoo.Duck duck = new zoo.Duck();
        duck.makeSound();    ERROR!
    }
}
```

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");
    }
}
```

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 땡 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.)

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 {

```
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();
    }
}
```

The End

Chapter 09의 강의를 마칩니다.