# 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.kh.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.jw.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.kh.smart;
public class Circle {
    double rad;
    final double PI;

    public Circle(double r) { . . . }
    public double getArea() { . . . }
}
```

www.kh.com의 Circle.java

```
package com.jw.simple;
public class Circle {
    double rad;
    final double PI;

    public Circle(double r) { . . . }
    public double getPerimeter() { . . . }
}
```

www.jw.com의 Circle.java

패키지 이름은 모두 소문자로 구성

인터넷 도메인 이름의 역순으로 이름을 구성

이름 끝에 클래스를 정의한 주체 또는 팀의 이름 추가

```
com.kh.smart.Circle c1 = new com.kh.smart.Circle(3.5);
com.jw.simple.Circle c2 = new com.jw.simple.Circle(5.5);
```

-d 옵션을 주고 컴파일 하면 패키지 디렉토리도 자동 생성

# 클래스 하나에 대한 import 선언

```
import com.kh.smart.Circle;
```

동일 이름의 두 클래스에 대한 import 선언은 컴파일 오류

```
import com.kh.smart.Circle;
import com.jw.simple.Circle;
```

# 패키지 전체에 대한 import 선언

```
import com.kh.smart.*;
```

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

# 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());
}
```

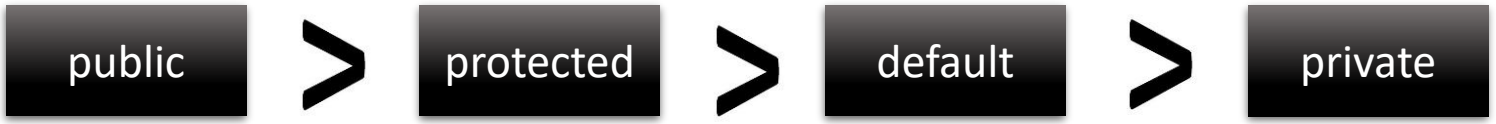
# 정보의 은닉을 위한 private 선언

```
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());
}
```

# 접근 수준 지시자
# (접근 제한자)

# 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();
   }                          OK!

   public void makeDuck() {
      zoo.Duck quack = new
zoo.Duck();                   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("야용");
   }

   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 | ○ | ○ | ○ | ○ |

# 캡슐화

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

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

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

class SnuffleCap {    // 코 막힘 처치용 캡슐
   void take() {
      System.out.println("코가 뻥 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 처치용 캡슐
    void take() {
        System.out.println("콧물이 싹~ 납니다.");
    }
}

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

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

```
class SinusCap {
    void sniTake() {
        System.out.println("콧물이 싹~ 납니다.");
    }
    void sneTake() {
        System.out.println("재채기가 멎습니다.");
    }
    void snuTake() {
        System.out.println("코가 뻥 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 처치용 캡슐
   void take() {
      System.out.println("콧물이 싹~ 납니다.");
   }
}
```

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

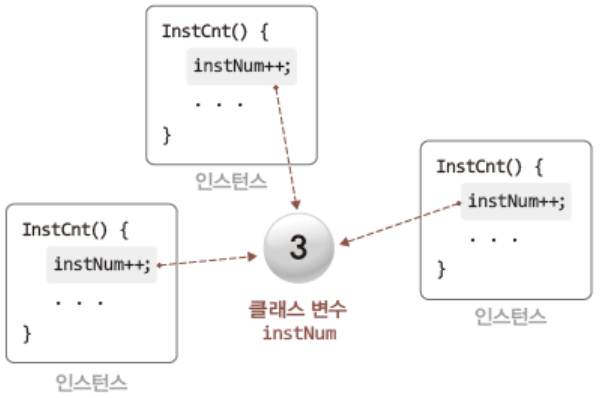
```
class SnuffleCap {   // 코 막힘 처치용 캡슐
   void take() {
      System.out.println("코가 뻥 뚫립니다.");
   }
}
```

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

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

# static 선언을 붙여서 선언하는 클래스 변수

# 선언된 클래스의 모든 인스턴스가 공유하는 클래스 변수

```
class InstCnt {
   static int instNum = 0;    // 클래스 변수(static 변수)

   InstCnt() {
      instNum++;
      System.out.println("인스턴스 생성: " + instNum);
   }
}
class ClassVar {
   public static void main(String[] args) {
      InstCnt cnt1 = new InstCnt();
      InstCnt cnt2 = new InstCnt();
      InstCnt cnt3 = new InstCnt();
   }
}
```




```
명령 프롬프트
C:₩JavaStudy>java ClassVar
인스턴스 생성: 1
인스턴스 생성: 2
인스턴스 생성: 3

C:₩JavaStudy>
```

# 클래스 변수의 접근 방법

## 클래스 내부 접근

- static 변수가 선언된 클래스 내에서는 이름만으로 직접 접근 가능

## 클래스 외부 접근

- private으로 선언되지 않으면 클래스 외부에서도 접근 가능
- 접근 수준 지시자가 허용하는 범위에서 접근 가능
- 클래스 또는 인스턴스의 이름을 통해 접근

# 클래스 변수 접근의 예

```
class AccessWay {
   static int num = 0;

   AccessWay() { incrCnt(); }
   void incrCnt() {
      num++;   // 클래스 내부에서 이름을 통한 접근
   }
}

class ClassVarAccess {
   public static void main(String[] args) {
      AccessWay way = new AccessWay();
      way.num++;    // 외부에서 인스턴스의 이름을 통한 접근
      AccessWay.num++;    // 외부에서 클래스의 이름을 통한 접근
      System.out.println("num = " + AccessWay.num);
   }
}
```

# 클래스 변수의 초기화 시점과 초기화 방법

```
class InstCnt {
   static int instNum = 100;
                클래스 변수의 적절한 초기화 위치
   InstCnt() {
      instNum++;
      System.out.println("인스턴스 생성: " + instNum);
   }
}

class OnlyClassNoInstance {
   public static void main(String[] args) {
      InstCnt.instNum -= 15;   // 인스턴스 생성 없이 instNum에 접근
      System.out.println(InstCnt.instNum);
   }
}
```

클래스 변수는 생성자 기반 초기화 하면 안된다!

이 경우 인스턴스 생성시마다 값이 리셋!

# 클래스 변수의 활용의 예

```
class Circle {
   static final double PI = 3.1415;
   private double radius;

   Circle(double rad) {
      radius = rad;
   }
   void showPerimeter() {
      double peri = (radius * 2) * PI;
      System.out.println("둘레: " + peri);
   }
   void showArea() {
      double area = (radius * radius) * PI;
      System.out.println("넓이: " + area);
   }
}
```

인스턴스 별로 가지고 있을 필요가 없는 변수

- 값의 참조가 목적인 변수
- 값의 공유가 목적인 변수

그리고 그 값이 외부에서도 참조하는 값이라면 public으로 선언한다.

# static 선언을 붙여서 정의하는 클래스 메소드

# 클래스 메소드의 정의와 호출

```
class NumberPrinter {
   private int myNum = 0;
   static void showInt(int n) { System.out.println(n); }
   static void showDouble(double n) {System.out.println(n); }

   void setMyNumber(int n) { myNum = n; }
   void showMyNumber() { showInt(myNum); }
}

class ClassMethod {
   public static void main(String[] args) {
      NumberPrinter.showInt(20);
      NumberPrinter np = new NumberPrinter();
      np.showDouble(3.15);
      np.setMyNumber(75);
      np.showMyNumber();
   }
}
```

내부 접근

외부 접근

외부 접근

클래스 메소드의 성격 및 접근 방법이 클래스 변수와 동일하다.

# 클래스 메소드로 정의하는 것이 옳은 경우

```
class SimpleCalculator {
   static final double PI = 3.1415;

   static double add(double n1, double n2) {
      return n1 + n2;
   }
   static double min(double n1, double n2) {
      return n1 - n2;
   }
   static double calCircleArea(double r) {
      return PI * r * r;
   }
   static double calCirclePeri(double r) {
      return PI * (r * 2);
   }
}
```

단순 기능 제공이 목적인 메소드들, 인스턴스 변수와 관련 지을 이유가 없는 메소드들은 static으로 선언하는 것이 옳다.

# 클래스 메소드에서 인스턴스 변수에 접근이 가능할까?

```
class AAA {
    int num = 0;
    static void addNum(int n) {
        num += n;
    }
}
```

논리적으로 이 문장이 유효할 수 있는지를 생각해보자.