필드와 접근제어

- 변수의 종류(4개)

- 필드
  - 필드의 선언
  - 필드의 범위
  - 필드의 초기화
    - getter/setter
    - 생성자

- 접근제어(접근지정자)
  - 정보 공개 범위 지정
  - private
  - public
  - protected(상속관계)

- this
  - 자기자신

필드와 변수의
차이점을
이해하고
메서드의
종류에 대해
알아본다

지역 변수

- 메서드 안에 선언
- 블락 안에 선언
- 매개 변수도 지역 변수의 일종

```
class Box {
    int width=0, length=0, height=0;
        ...
    public int getVolume()
    {
        int volume;
        volume = width*length*height;
        return volume;
    }
}
```

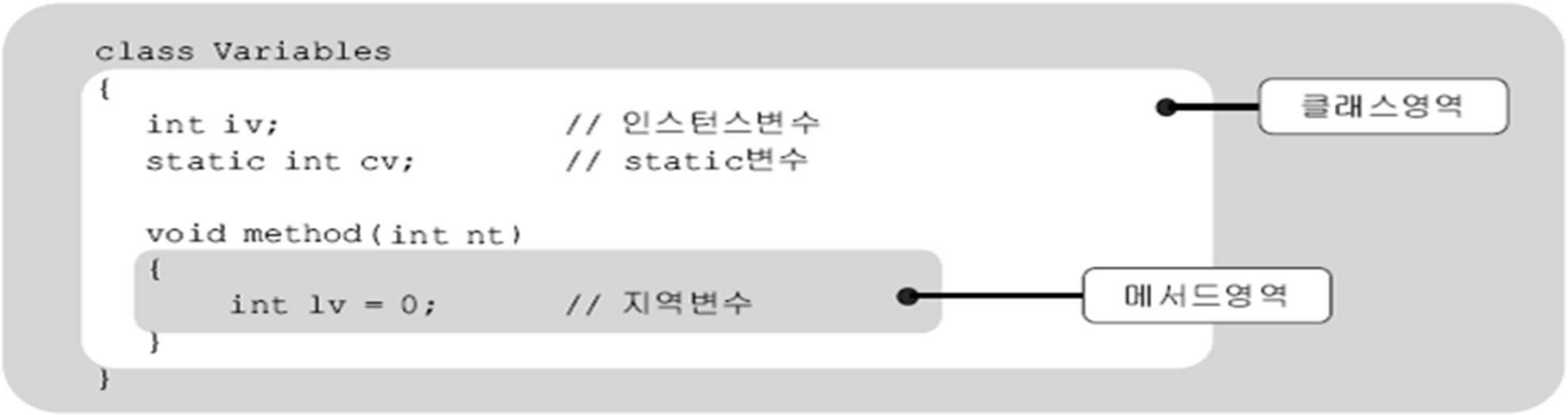
필드: 전체 클래스 안에서 사용가능

지역 변수

지역 변수 volume의 사용 범위

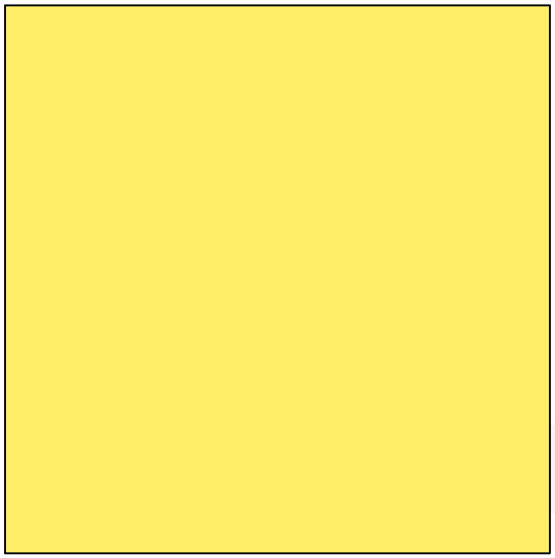
변수의 종류

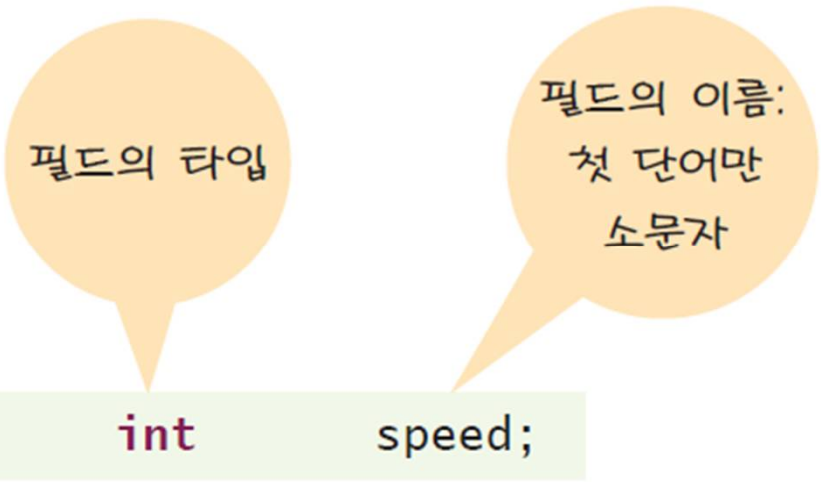
"변수의 선언 위치가 변수의 종류와 범위(scope)을 결정한다."



변수의 종류	선언위치	생성시기
필드(인스턴스변수)	클래스 영역	클래스 안에서 선언되는 멤버 변수. **개별 데이터** Getter/Setter를 이용
static필드		클래스가 메모리에 올라갈 때. **공유 데이터**
매개변수 : (,) 사이에서	메서드 영역	메서드 선언시 사용된 변수
지역변수 : { , } 사이에서	블락 영역	블록 안에서 선언되는 변수

필드의 선언

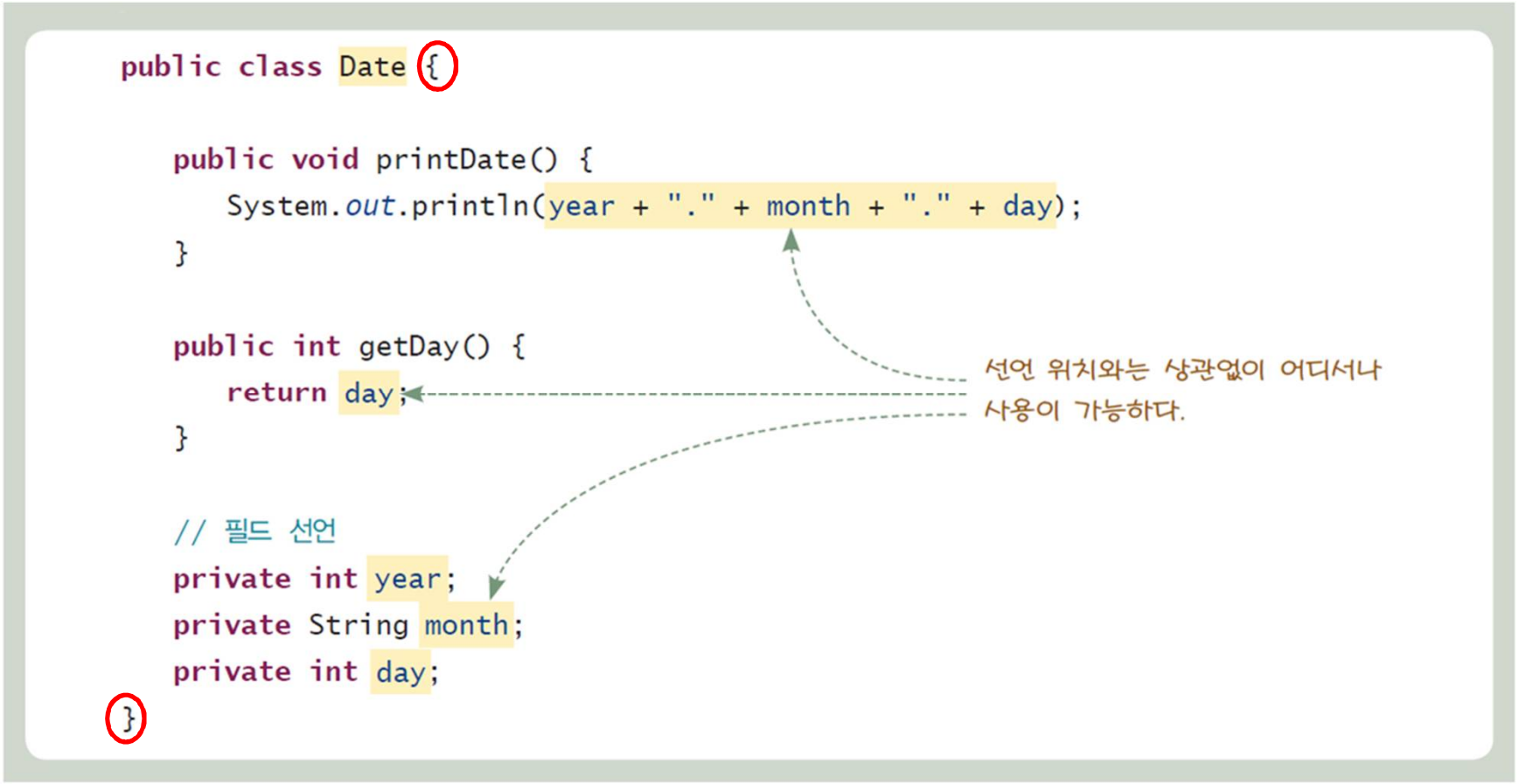




필드의 접근 지정자는 어떤 클래스가 필드에 접근할 수 있는지를 표시한다.

- **public**: 이 필드는 모든 클래스로부터 접근가능하다.
- **private**: 클래스 내부에서만 접근이 가능하다.

필드의 범위(Scope) : { .. }



클래스 안의 모든 메서드는 필드를 언제, 어디서나 사용할 수 있다.

필드에 값을 대입하는 방법

- 필드 선언과 동시에 초기화 가능

```
public class ClassRoom {
    public static int capacity = 60;        // 60으로 초기화
    private boolean use = false;            // false로 초기화
}
```

- getter / setter 를 이용한 초기화
- 생성자를 이용한 초기화

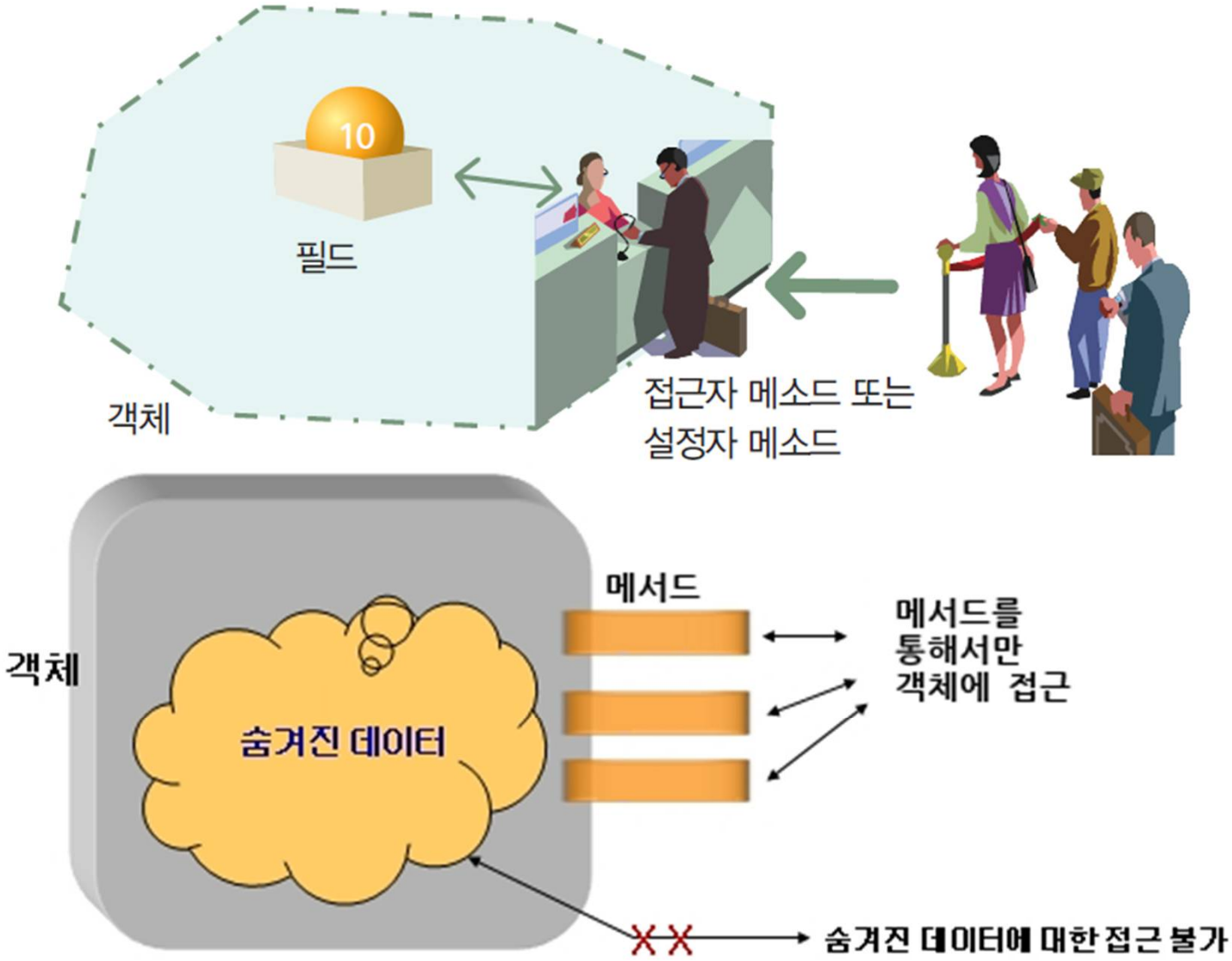
getter란? setter란?

- setter(setter)
  - 필드의 값을 설정하는 메서드
  - setXXX() 형식

- getter(getter)
  - 필드의 값을 가져오는 메서드
  - getXXX() 형식

실생활의 getter와 setter

왜 getter와 setter를 쓰는가?



setter와 getter는 왜 사용하는가?

- 필드에 대한 제한 조건을 추가할 때 유용.

- setter에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.
- 필요할 때마다 필드값을 계산하여 반환할 수 있다.
- getter만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있다.

```
public void setSpeed(int s)
{
    if( s < 0 )
        speed = 0;
    else
        speed = s;
}
```

속도가 음수이면 0으로 만든다.

예제. setter와 getter

CarTest1.java

```
01  class Car {
02      private String color;          // 색상
03      private int speed;             // 속도
04      private int gear;              // 기어
05
06
07
08
09
10
11
12
13
14
15  }
```

필드가 모두 private로 선언되었다.
클래스 내부에서만 사용이 가능하다.

1. getter/setter 만들기
2. toString() 만들기
3. 생성자 만들기

이런 식으로 간략하게
표기하기도 한다.

예제.setter를 이용한 필드 초기화

```
16  public class CarTest1 {
17      public static void main(String[] args) {
18          // 객체 생성
19          Car myCar = new Car();
20
21
22
23
24
25          System.out.println("현재 자동차의 색상은 " + myCar.getColor());
26          System.out.println("현재 자동차의 속도는 " + myCar.getSpeed());
27          System.out.println("현재 자동차의 기어는 " + myCar.getGear());
28      }
29  }
```

객체 생성

setter를 이용하여 값을 설정
color="red", speed=100, gear = 3

예제.setter를 이용한 필드 초기화

- Speed가 0보다 작다면? 0
  Speed가 250보다 크다면? 250

```
if (speed < 0) {
    this.speed = 0;
} else if (speed > 250) {
    this.speed = 250;
} else {
    this.speed = speed;
}
```

- Gear가 0보다 작다면? 0
  Gear가 5보다 크다면? 5

```
if (gear < 0) {
    this.gear = 0;
} else if (gear > 5) {
    this.gear = 5;
} else {
    this.gear = gear;
}
```

실습-사칙연산 클래스 만들기

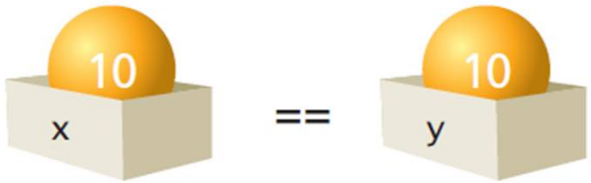
- 두 개의 정수를 입력 받고, 두 수의덧셈, 뺄셈, 곱셈, 나눗셈 결과를 출력하는 프로그램을 작성하시오.
  - Oper 클래스를 만들고 그 클래스 안에 Add(), Minus(), Mul(), Div() 메서드만을 만드시오. 입력 부분을 Oper 클래스안에 넣지 마시오.
  - 테스트 클래스에서 각각의 연산 결과를 출력하시오.
  - 나눗셈의 결과는 실수가 되도록 한다.
- 패키지명: java12 클래스명: Oper , OperTest

▪ 실행결과예시

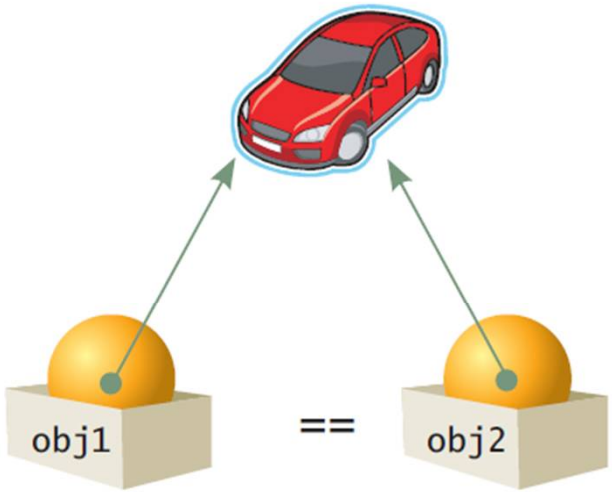
```
First num : 2
Second num : 4
Add : 6          ---> Add() 메서드를 사용하시오
Minus : -2       ---> Minus() 메서드를 이용하시오
Mul : 8          ---> Mul() 메서드를 이용하시오
Div : 0.500000   ---> Div() 메서드를 이용하시오
```

기본형 vs 참조형

- "변수1 == 변수2"의 의미



기초형 변수의 경우
값이 같으면 true

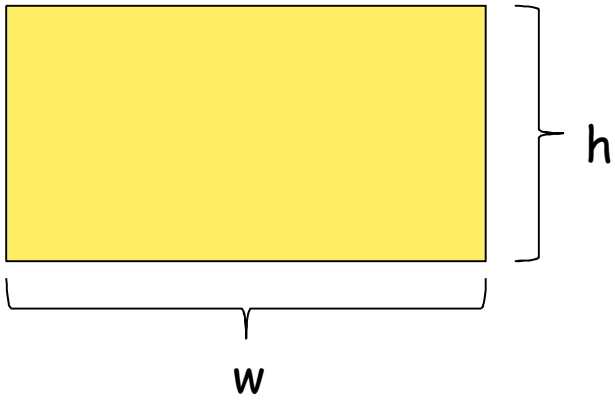


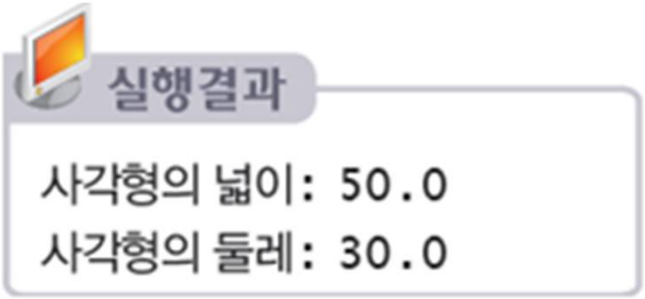
참조형 변수의 경우
같은 객체를 가리키면 true

- 참조형 변수의 경우
  - ==는 객체의 내용이 같다는 의미가 아니다.
  - ==는 객체의 참조값(주소값)이 같다는 의미다.
- 참조형 변수에서 내용이 같은지를 검사하려면 equals() 사용

실습: 면적 구하기 예제

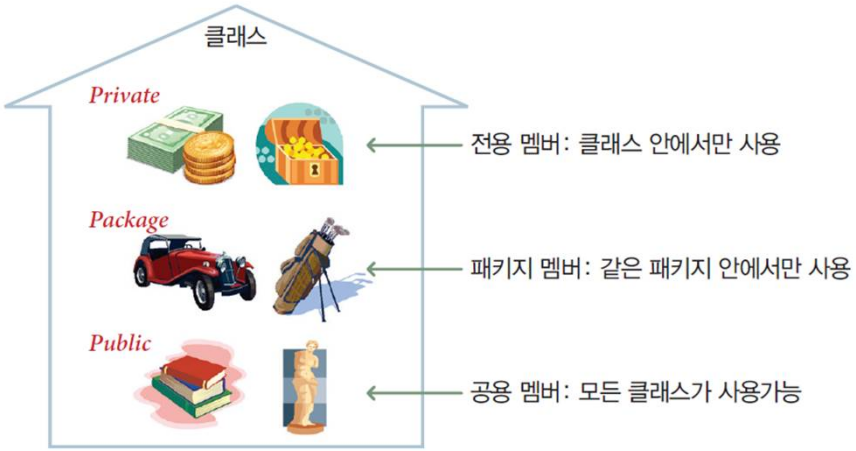
- 직사각형의 둘레와 면적을 구하는 프로그램을 작성하여 보자. 직사각형의 가로와 세로를 각각 w와 h라고 라고 하고 **width 값과 heigh 값은 표준 입력(Scanner)을 이용하여 입력 받도록 하시오.**
  - 필드 : width (가로) , height (세로)
  - 메서드 : getArea (면적) , getPerimeter (둘레)
  - area = width*height;
  - perimeter = 2*(width+height);
  - 패키지명: java12. 클래스명: Rect, RectTest





접근 제어

- 접근 제어(access control): 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 제어하는 것

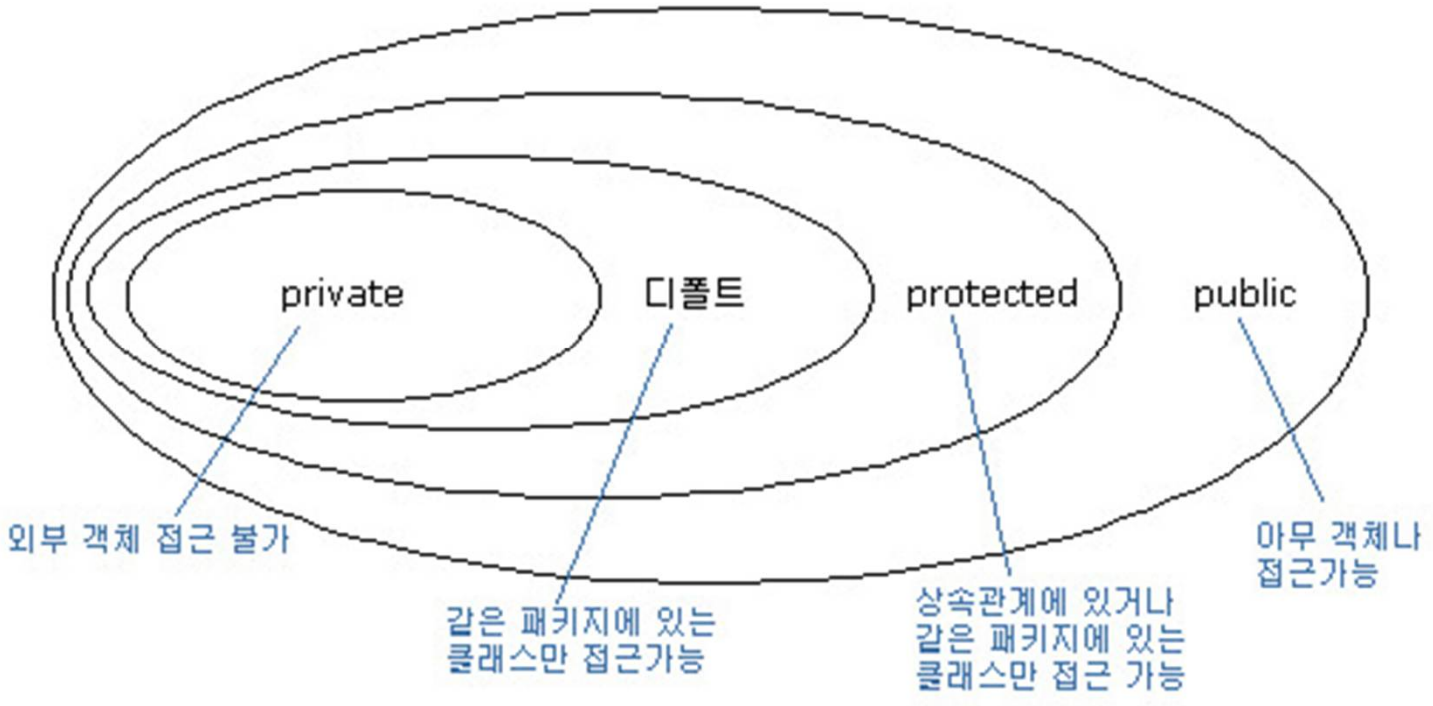


- private == 비공개
- public == 공개
- protected == 조건부 공개

- package == 패키지

접근 제어

분류	접근 지정자	클래스 내부	같은 패키지 내의 클래스	다른 모든 클래스
전용 멤버	private	O	X	X
패키지 멤버	없음	O	O	X
공용 멤버	public	O	O	O



중간 점검 문제

1. 필드의 경우, private로 만드는 것이 바람직한 이유는 무엇인가?
2. 필드를 정의할 때 아무런 접근 제어 수식자를 붙이지 않으면 어떻게 되는가?

this 참조

- 자기 자신을 참조하는 키워드
- 나는 , 내가

```
public void setSpeed(int speed)
{
    this.speed = speed;      // speed는 매개변수, this.speed는 필드
}
```

필드 speed와 매개변수 speed를 구별하기 위하여 this 사용

- 생성자를 호출할 때도 사용된다.

```
// 두 번째 생성자
public Time(int h, int m, int s) {
    this.setTime(h, m, s);    // this는 없어도 된다.
}
```

중간 점검 문제

1. this의 주된 용도는 무엇인가?
2. this()와 같이 표기하면 무엇을 의미하는가??

실습 예제-날짜 클래스 만들기

테스트는 아래 2개의 방법을 모두 사용하도록 프로그램하시오.
setter를 이용한 필드값 설정, 결과 출력
setDate() 메서드를 이용해서 필드값을 설정, 결과 출력
생성자를 이용한 필드값 설정, 결과 출력

java12.Date
-year : int -month : String -day : int
+setDate(year, month, day) : void +printDate() : void

- : private
+ : public

java12.DateTest
+main() : void

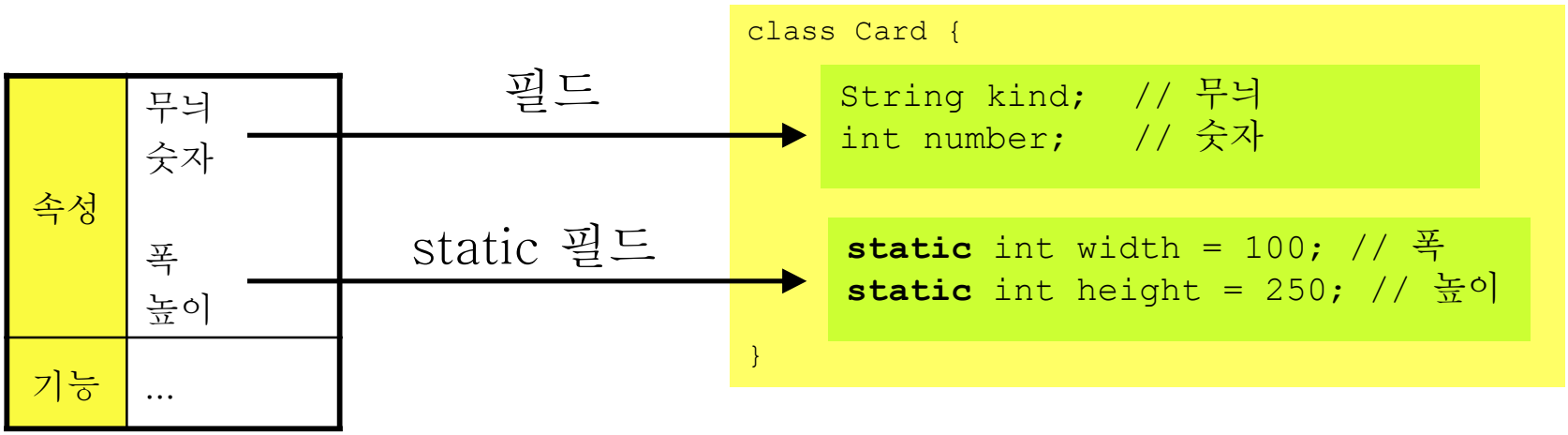
실행결과

```
2010년 3월 2일
```

static 필드(정적 변수)

static 필드와 인스턴스 필드

- 필드(필드)는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 **개별** 값을 갖지만,
- static 필드는 프로그램이 시작될 때 저장공간(메모리)가 할당되어 인스턴스들이 저장공간을 **공유**하므로 항상 공통된 값을 갖는다.



static 필드의 예제

```
class Car {
    private String color;
    private int speed;
    private int gear;
    // 자동차의 시리얼 번호
    private int id;
    // 실체화된 Car 객체의 개수를 위한 정적 변수
    private static int numberOfCars = 0;

    public Car(String c, int s, int g) {
        color = c;
        speed = s;
        gear = g;
        // 자동차의 개수를 증가하고 id 번호를 할당한다.
        id = ++numberOfCars;
    }
    // 정적 메소드
    public static int getNumberOfCars() {
        return numberOfCars; // OK!
    }
}
```

정적 변수

정적 메소드에서는 인스턴스 변수와 인스턴스 메소드에 접근할 수 없다.

static 필드의 예제

```
public class CarTest  {
   public static void main(String args[]) {
      Car c1 = new Car("blue", 100, 1);      // 첫 번째 생성자 호출
      Car c2 = new Car("white", 0, 1);       // 첫 번째 생성자 호출
      int n = Car.getNumberOfCars();         // 정적 메소드 호출
      System.out.println("지금까지 생성된 자동차 수 = " + n);
   }
}
```

변수의 종류

▶ 필드(instance variable)

- 각 인스턴스의 **개별**적인 저장공간. 인스턴스마다 다른 값 저장가능
- 인스턴스 생성 후, '**인스턴스명.필드이름**'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조하는 인스턴스가 없을 때 자동 제거됨

▶ **static 필드**(static variable)

- 같은 클래스의 인스턴스들이 **공유**하는 변수
- 인스턴스 생성 없이 '**클래스명.필드이름**'으로 바로 접근
- 프로그램 실행시 자동으로 static변수가 메모리가 할당되고 종료될 때 소멸
  new 없이 사용 가능

▶ 지역변수(local variable)

- 메서드 내에 선언되며, 메서드의 종료와 함께 소멸
- 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸

static 메서드(정적 변수)

- 정적 메소드(static method)
  객체를 생성하지 않고 사용할 수 있는 메소드

- (예) Math 클래스에 들어 있는 각종 수학 메소드 들

```
double value = Math.sqrt(9.0);
```

정적 메서드를 외부에서 사용할 때

- "클래스이름.static메서드명" 형식을 사용한다.

주의할 점

```
public class Test {
    public static void main(String args[]) {
        add(10,20);      // 오류!! 정적 메소드 안에서 인스턴스 메소드 호출
    }
    int add(int x, int y) {
        return x + y;
    }
}
```

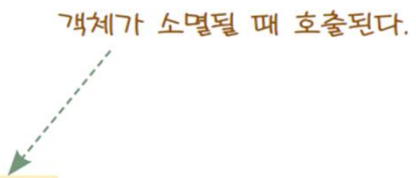
객체가 생성되지 않았으므로
인스턴스 메소드는 존재하지 않는다.

정적 메소스에서 인스턴스 메소드를 호출할 수 없음. Why?

정적 변수의 예

EmployeeTest.java

```
01  import java.util.*;
02
03  class Employee {
04      private String name;
05      private double salary;
06
07      private static int count = 0;   // 정적 변수
08
09      // 생성자
10      public Employee(String n, double s) {
11          name = n;
12          salary = s;
13          count++; // 정적 변수인 count를 증가
14      }
15
16      // 객체가 소멸될 때 호출된다.
17      protected void finalize() {
18          count--; // 직원이 하나 줄어드는 것이므로 count를 하나 감소
19      }
```



예제

```
20
21      // 정적 메소드
22      public static int getCount() {
23          return count;
24      }
25  }
26
27  public class EmployeeTest {
28      public static void main(String[] args) {
29          Employee e1,e2,e3;
30          e1 = new Employee("김철수", 35000);
31          e2 = new Employee("최수철", 50000);
32          e3 = new Employee("김철호", 20000);
33
34          int n = Employee.getCount();
35          System.out.println("현재의 직원수=" + n);
36      }
37  }
```

중간 점검 문제

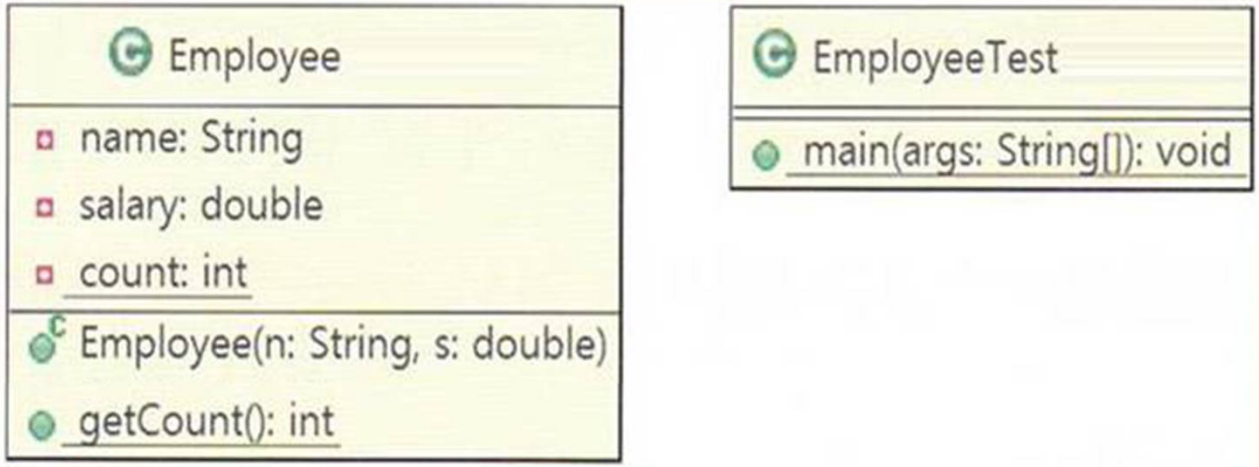
1. static 필드는 어떤 경우에 사용하면 좋은가?

2. static 필드나 static 메소드를 사용할 때, 클래스 이름을 통하여 접근하는 이유는 무엇인가?

3. main() 안에서 인스턴스 메소드를 호출할 수 없는 이유는 무엇인가?

실습 예제- Employee 클래스 만들기

직원을 나타내는 클래스에서 직원들의 수를 카운트하는 예를 살펴보자. 직원의 수를 정적 변수로 나타낸다. 객체가 하나씩 생성될 때마다 생성자에서 정적 변수 count를 증가한다.



또 현재 직원의 수를 출력하는 정적 메소드인 getCount()를 작성하여 보자.

```
현재의 직원수=3
```