클래스

안 화 수

개체 지향 프로그래밍

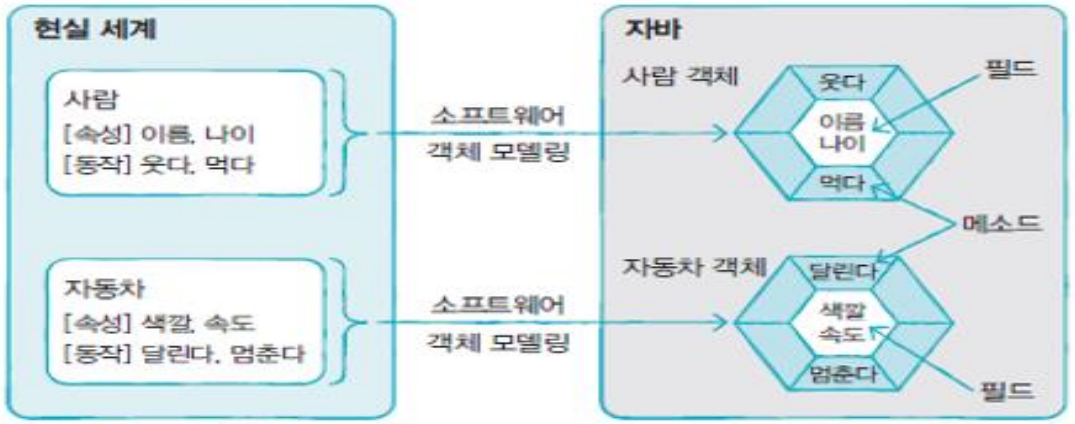
- 객체 지향 프로그래밍

현실 세계에서 어떤 제품을 만들 때 부품을 먼저 개발하고 이 부품들을 하나씩 조립해서 제품을 완성하듯이 소프트웨어를 개발할 때에도 부품에 해당하는 객체를 먼저 만든다. 그리고 객체를 하나씩 조립해서 완성된 프로그램을 만드는 기법을 객체 지향 프로그래밍(OOP: Object Oriented Programming)이라고 한다.

개체 지향 프로그래밍

객체(Object)

- 객체(object)란 물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지고 있으면서 식별 가능한 것을 말한다.
- 객체의 예로는 물리적으로 존재하는 것(자동차,사람,책)과 추상적인 것(회사,날짜) 중에서 자신의 속성과 동작을 모두 가진 것
- 현실 세계의 속성과 동작을 자바에서는 필드(속성)와 메소드(동작)로 표현한다.



- 현실 세계의 객체를 소프트웨어 객체로 설계하는 것을 객체 모델링 이라고 한다.
- 객체 모델링은 현실 세계 객체의 속성과 동작을 추려내어 소프트웨어 객체의 필드와 메소드로 정의하는 과정이다.

개체 지향 프로그래밍

- 객체지향 프로그래밍의 특징

1. 캡슐화

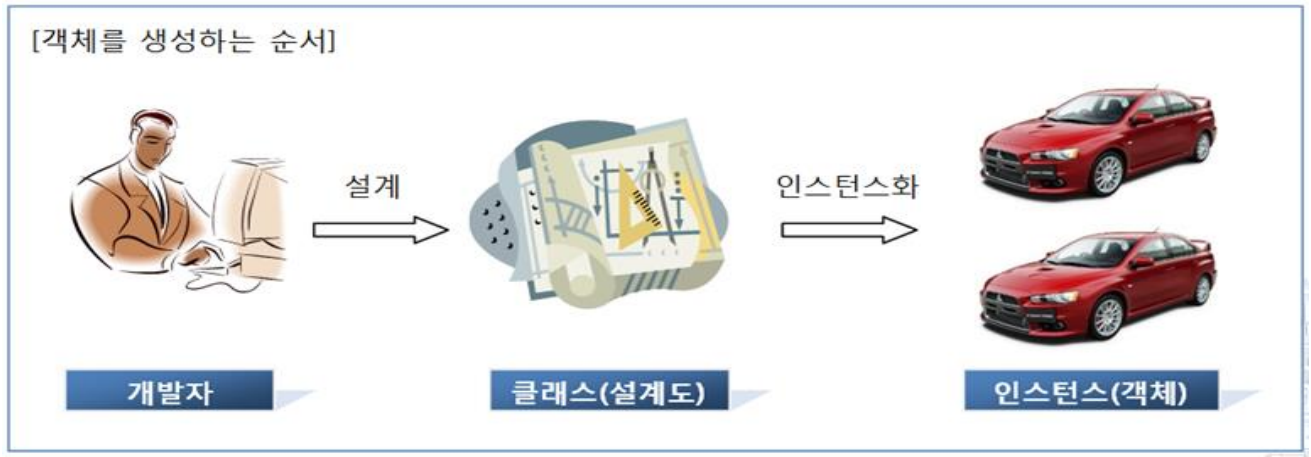
2. 상속

3. 다형성

개체 지향 프로그래밍

❖ 클래스와 객체

- 현실세계 : 설계도 -> 객체
  자바 : 클래스 -> 객체
- 클래스에는 객체를 생성하기 위한 필드와 메소드로 정의되어 있다.
- 클래스로부터 만들어진 객체를 해당 클래스의 인스턴스라고 한다.
- 자동차 객체는 자동차 클래스의 인스턴스이다.
- 한 개의 클래스로 여러 개의 객체(인스턴스)를 만들 수 있다.

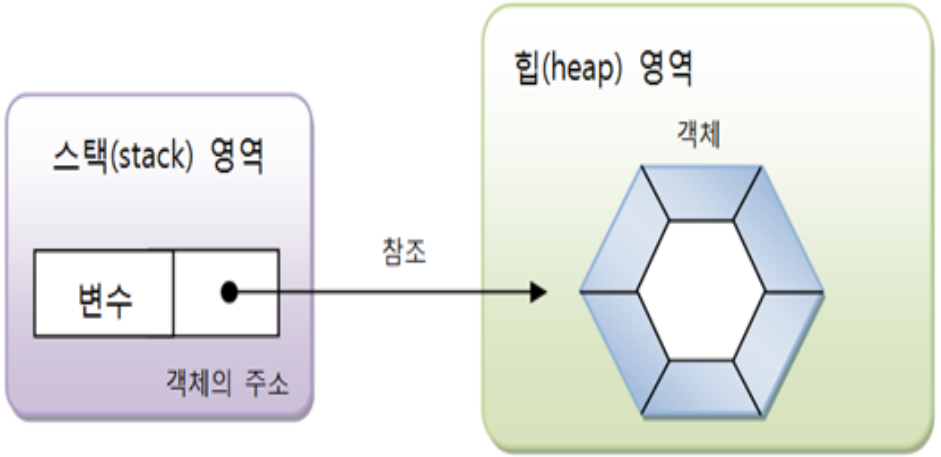


개체 지향 프로그래밍

❖ 클래스로 객체 생성

- 클래스를 이용해서 객체를 생성한다.
- new 연산자로 힙(heap) 메모리 영역에 객체를 저장하기 위한 공간을 할당 받는다.

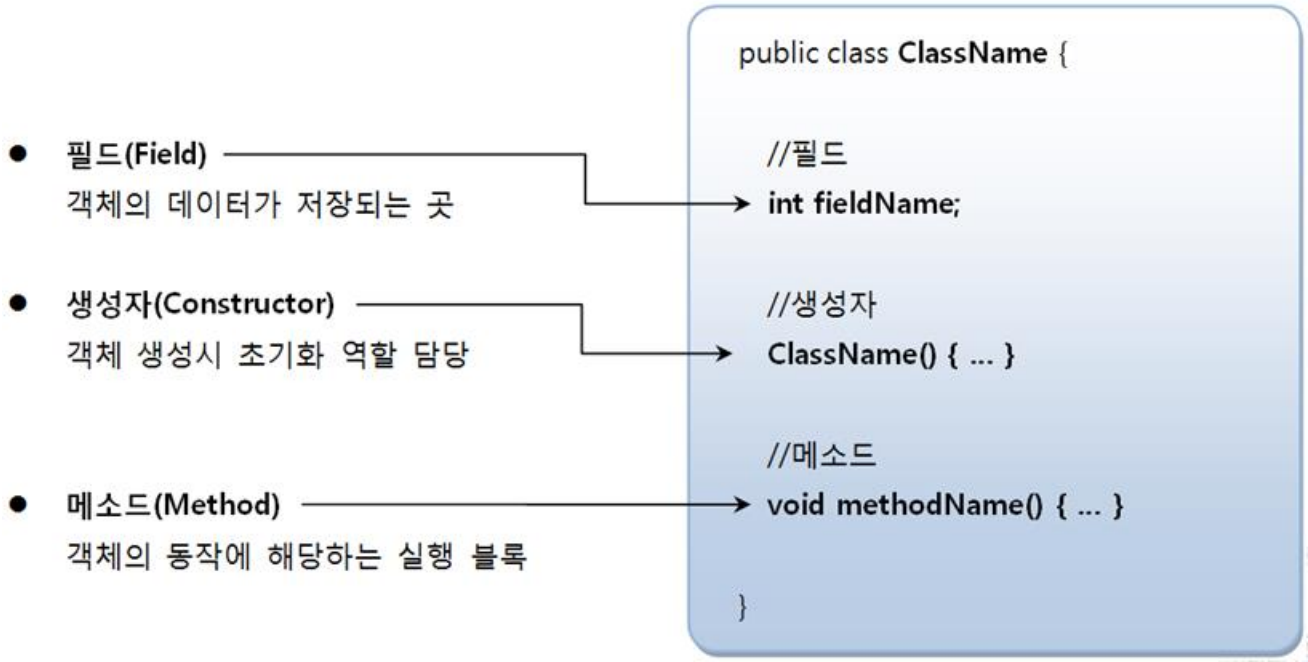
```
클래스 변수;
변수 = new 클래스();

클래스 변수 = new 클래스();
```



개체 지향 프로그래밍

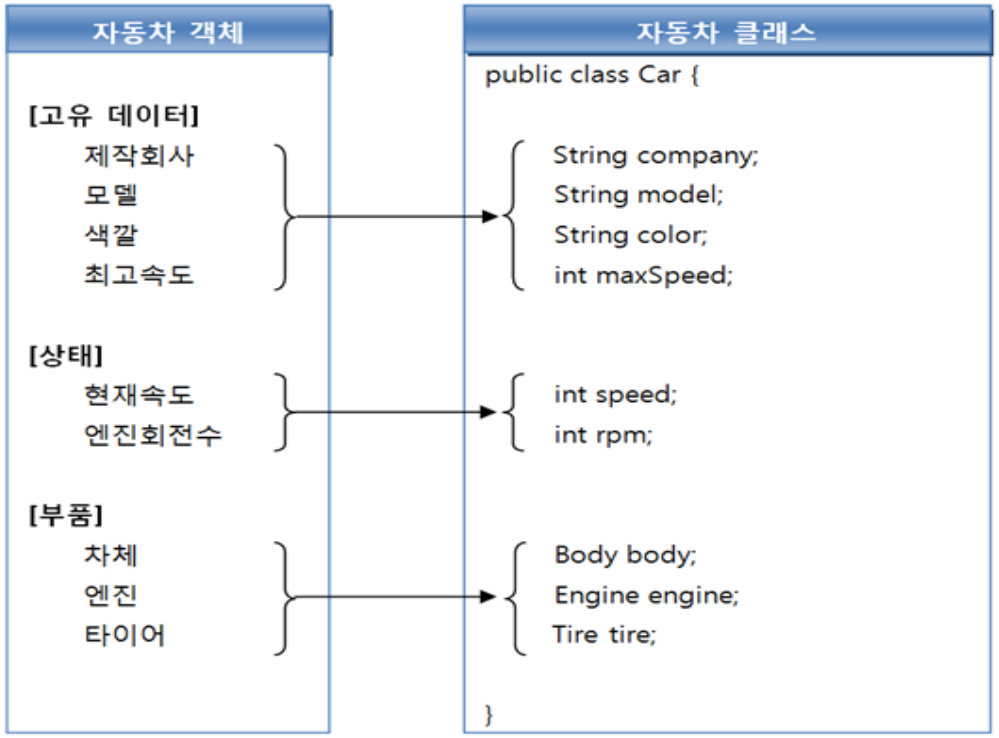
❖ 클래스의 구성 멤버

- 필드(Field) : 객체의 속성을 heap 메모리에 저장하는 역할(멤버변수)
- 생성자(Constructor) : 객체를 생성할 때 호출되면서 필드를 초기화 시켜주는 역할
- 메소드(Method) : 객체의 동작에 해당하는 실행 역할



필드

- 필드(field)
- 필드는 한 개의 클래스를 구성하는 멤버 중 하나이다.
- 필드는 객체의 속성을 저장하는 멤버변수 이다.
- 필드는 클래스로 객체를 생성할 때 heap 메모리 영역에 값을 저장하는 변수이다.



필드

- 필드의 초기화
- 필드는 객체를 생성할 때 자동으로 초기화 된다.

분류		데이터 타입	초기값
기본 타입	정수 타입	byte	0
		char	₩u0000 (빈 공백)
		short	0
		int	0
		long	0L
	실수 타입	float	0.0F
		double	0.0
	논리 타입	boolean	false
참조 타입		배열	null
		클래스(String 포함)	null
		인터페이스	null

생성자

생성자(constructor)

1. 생성자는 클래스명과 동일한 이름으로 생성해야 한다.

```
public class Animal{
    public   Animal(){                    //기본 생성자(매개변수가 없는 생성자)
    }
}
```

2. 생성자는 객체가 생성될 때 호출되며, 필드를 초기화하는 역할을 한다.

3. 생성자는 객체가 생성될 때 호출된다.

```
Animal   an  = new  Animal();
                    생성자 호출
```

4. 기본 생성자는 객체가 생성될 때 컴파일러가 자동으로 생성 해준다.

생성자

❖ 생성자 오버로딩(overloading)

- 클래스 내에 매개변수가 다른 생성자를 여러 개 선언하는 것
- 생성자 오버로딩의 조건
  매개변수의 타입(자료형), 개수, 순서를 서로 다르게 선언 해야 한다.

```
public class 클래스 {
    클래스 ( [타입 매개변수, ...] ) {
        ...
    }
    클래스 ( [타입 매개변수, ...] ) {
        ...
    }
}
```

[생성자의 오버로딩]
매개변수의 타입, 개수, 순서가 다르게 선언

```
Car(String model, String color) { ... }
Car(String color, String model) { ... } //오버로딩이 아님
```

```
public class Car {
    Car() { ... }
    Car(String model) { ... }
    Car(String model, String color) { ... }
    Car(String model, String color, int maxSpeed) { ... }
}
```

```
Car car1 = new Car();
Car car2 = new Car("그랜저");
Car car3 = new Car("그랜저", "흰색");
Car car4 = new Car("그랜저", "흰색", 300);
```

this

- this

  자기 자신을 의미하는 내부 레퍼런스 변수

- this. : 생성자와 메소드 안에서 멤버변수와 매개변수 이름이 동일한 경우에 주로 사용한다.

```
public Test(int a, int b){
        this.a = a;
        this.b = b;
 }
```

- this() : 같은 클래스안에 있는 생성자를 호출할 때 사용한다.

생성자

```
class MyDate{
    private int year;                   // 필드
    private int month;
    private int day;

    // 생성자는 클래스로 객체를 생성할때 호출되며, 필드값을 초기화 시키는 역할을 한다.
    public MyDate() {                   // 기본 생성자
        year = 2024;
        month = 7;
        day = 11;
    }
    public MyDate(int year, int month, int day) {
        this.year = year;               // 2024
        this.month = month;             // 12
        this.day = day;                 // 25
    }
    public void print() {               // 메소드
        System.out.println(year+"/"+month+"/"+day);
    }
}
public class ClassTest {
    public static void main(String[] args) {

        MyDate d = new MyDate();
//                          생성자 호출
        // private 접근 제어자는 외부 클래스의 접근을 허용하지 않기 때문에 직접 접근 할 수 없다.
        // System.out.println(d.year);    오류발생
        d.print();

        MyDate d2 = new MyDate(2024,12,25);
        d2.print();
    }
}
```

메소드

- 메소드(method)

1. 메소드는 여러 코드를 모아놓은 집합체이다.
2. 메소드는 자바에서 클래스를 구성하는 멤버 중 하나이다.
   클래스 = 필드 + 생성자 + 메소드
3. 클래스에서 메소드를 사용하면 중복되는 코드의 사용을 줄일 수 있다.
4. 클래스에서 메소드를 사용하면 코드를 재사용 할 수 있다.
5. 프로그램에서 문제가 발생하거나 기능의 변경이 필요할 때 손쉽게 유지보수를 할 수 있다.

메소드

- 메소드(method) 선언

선언부 { 접근제어자 리턴타입 메소드명(매개변수 선언) {

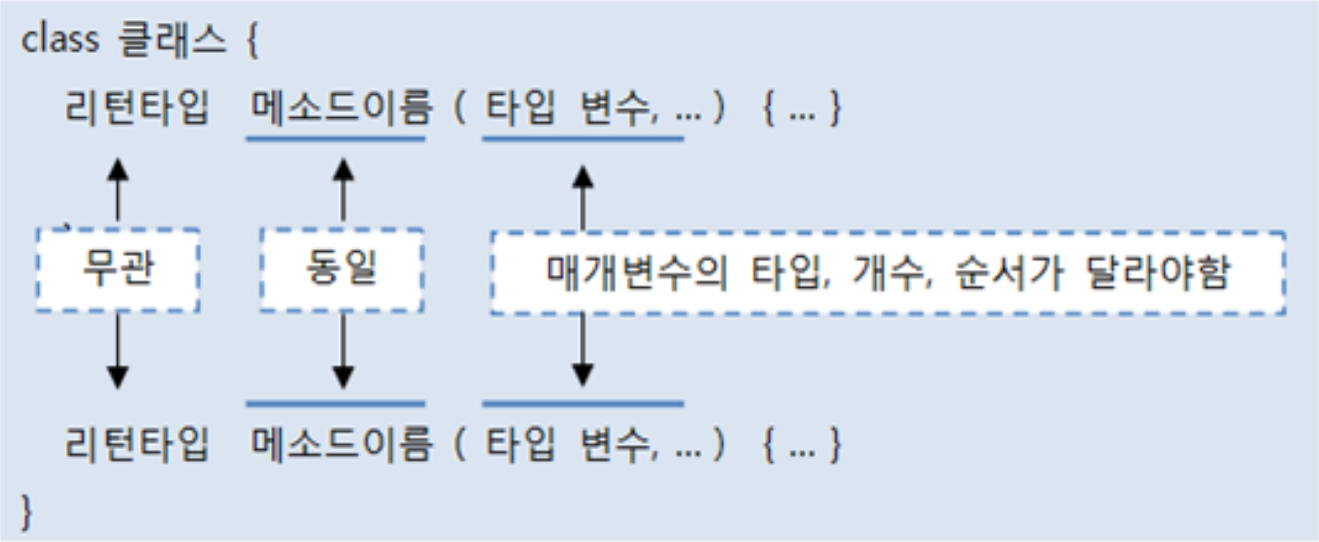
구현부 {
```
        // 실행할 코드 작성
        return 반환값;
    }
```

```
public    static    int    check(int a){

        return    값;
}
```

메소드

❖ 메소드 오버로딩(overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값 받기 위해 메소드 오버로딩
- 메소드 오버로딩의 조건
  매개변수의 타입(자료형), 개수, 순서가 서로 달라야 한다.



메소드

- getter, setter 메소드 (1/2)

```
class MyDate2{
    private int year;                    // 필드, 멤버변수
    private int month;
    private int day;

    public MyDate2(int year, int month, int day) {
        this.year = year;                // 2024
        this.month = month;              // 7
        this.day = day;                  // 11
    }
    // 1. 필드값을 메소드를 호출한곳에 돌려주는 역할
    public int getYear() {               // getter 메소드
        return year;
    }
    public int getMonth() {
        return month;
    }
    public int getDay() {
        return day;
    }
    // 2. 필드값을 수정하는 역할
    public void setYear(int year) {   // setter 메소드
        this.year = year;
    }
    public void setMonth(int month) {
        this.month = month;
    }
    public void setDay(int day) {
        this.day = day;
    }
    // 3. 필드값을 출력하는 역할
    public void print() {
        System.out.println(year+"/"+month+"/"+day);
    }
}
```

메소드

- getter, setter 메소드 (2/2)

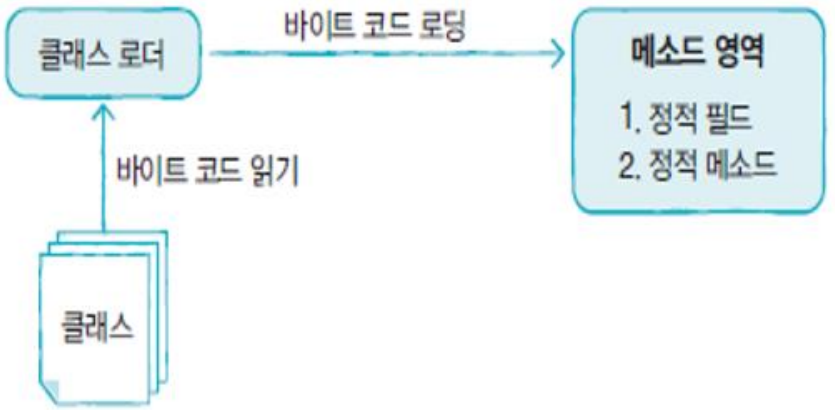
```
public class ClassTest2 {

    public static void main(String[] args) {

        // 매개변수가 있는 생성자가 있을 경우에는 컴파일러가 기본 생성자를 자동으로 생성해주지 않기 때문에 에러발생한다.
//      MyDate2 d = new MyDate2();    // 오류 발생

        MyDate2 d = new MyDate2(2024, 7, 11);
        d.print();

        d.setYear(2025);     // year값을 2025로 수정
        d.setMonth(12);      // month값을 12로 수정
        d.setDay(25);        // day값을 25로 수정
        d.print();

        int year = d.getYear();
        int month = d.getMonth();
        int day = d.getDay();
        System.out.println("돌려받은 year:"+ year);
        System.out.println("돌려받은 month:"+ month);
        System.out.println("돌려받은 day:"+ day);
    }
}
```

정적 멤버

- 정적 멤버
  1. 정적 필드와 정적 메소드는 공유를 목적으로 누구나 쉽게 접근할 경우에 사용한다.
  2. 클래스에 고정된 필드와 메소드이다. – 정적 필드, 정적 메소드
  3. 정적 필드와 정적 메소드는 static을 붙여서 만든다.
  4. 정적 필드와 정적 메소드는 heap영역에 저장되지 않고 메소드 영역(공유영역)에 저장된다.
  5. 정적 필드와 정적 메소드는 객체를 생성하지 않고 클래스명으로 직접 접근한다.

- 정적 멤버 선언

```
public class 클래스 {
  //정적 필드
  static 타입 필드 [= 초기값];

  //정적 메소드
  static 리턴타입 메소드( 매개변수선언, ... ) { ... }
}
```



정적 멤버

- 정적 메소드 사용시 유의할 점

1. 정적 메소드에서는 this 레퍼런스 변수를 사용할 수 없다.

2. 정적 메소드에서는 일반적인 변수를 사용할 수 없다.
   정적 메소드에서는 정적 멤버변수만 사용 할 수 있다.

3. 정적 메소드는 메소드 오버라이딩 되지 않는다.

정적 멤버

- 정적 멤버 예제

```
public class MathEx {

    public static void main(String[] args) {

        // Math 클래스
        // Math 클래스 = 정적 필드 + 정적 메소드

        // Math 클래스는 생성자가 제공되지 않기 때문에, Math클래스로 직접 객체를 생성할 수 없다.
        // Math m = new Math();                    // 오류 발생

        System.out.println("E="+ Math.E);    // 오일러 상수
        System.out.println("PI="+ Math.PI); // 원주율

        System.out.println("abs()="+ Math.abs(-10));        //절대값 : 10
        System.out.println("ceil()="+ Math.ceil(3.14));    //올림기능 : 4.0
        System.out.println("round()="+ Math.round(10.5)); //반올림기능 : 11
        System.out.println("floor()="+ Math.floor(10.9)); //내림기능 : 10.0
        System.out.println("max()="+ Math.max(10,20));     //최대값 : 20
        System.out.println("min()="+ Math.min(10,20));     //최소값 : 10
        System.out.println("pow()="+ Math.pow(2,3));       //2의 3승 : 8.0

        //       0.0 <= Math.random() < 1.0
        System.out.println("random()="+ Math.random());

        System.out.println("sqrt()="+ Math.sqrt(5));   // 제곱근
    }
}
```

정적 멤버

- 싱글톤(singleton)

  하나의 애플리케이션 내에서 단 하나만 생성되는 객체를 의미한다.

정적 멤버

- 싱글톤 예제: SingletonEx.java (1/2)

```
class Singleton{

    // 싱글톤(singleton) : 객체 생성을 1번만 수행하는것.
    private static Singleton s = new Singleton();    // 정적 필드

    private Singleton() {}; // 직접 객체 생성을 막아주는 역할

    public static Singleton getInstance() {          // 정적 메소드
        return s;
    }

    public void check() {
        System.out.println("메소드 호출 성공1");
    }

    public void check1() {
        System.out.println("메소드 호출 성공2");
    }

}
```

정적 멤버

- 싱글톤 예제: SingletonEx.java (2/2)

```
public class SingletonEx {

    public static void main(String[] args) {

        // private 접근제어자 때문에 외부 클래스에서 접근할 수 없다.
        // System.out.println(Singleton.s);   // 오류 발생

        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();
        System.out.println(obj1);
        System.out.println(obj2);

        if(obj1 == obj2) {                          // 주소값 비교
            System.out.println("같은 주소");
        }else {
            System.out.println("다른 주소");
        }

        // Singleton s = new Singleton();    오류발생

        obj1.check();
        obj1.check1();
        obj2.check();
        obj2.check1();
    }
}
```

패키지

- 패키지(package) : java api 클래스
  관련 있는 클래스를 묶어 놓은 폴더

  - java.lang 패키지 - 기본 패키지
    - 자바에서 가장 사용 빈도가 높은 클래스를 묶어 놓은 패키지
    ex) java.lang.String
    java.lang.System
    java.lang.Integer
  - 기본 패키지(java.lang)의 클래스는 import를 생략할 수 있다.
  - 기본 패키지(java.lang)외의 패키지의 클래스는 import하고 사용 해야 한다.

```
ex)  import  java.util.Date;
     import  java.util.Random;
     import  java.util.Scanner;

     import  java.util.*;
```

패키지

- 패키지(package) : 사용자 정의 클래스

1. 같은 패키지 안에 들어 있는 클래스

src - p2024_07_20 - Called.java : check()메소드

- Calling.java : main() 메소드

1) 같은 패키지 안에 들어 있는 클래스에 접근 하기 위해서는 접근 제어제가 public이나 default 접근 제어자로 되어 있어야 한다.
2) 같은 패키지 안에 들어 있는 클래스에 접근 하기 위해서는 import 를 하지 않아도 된다.

패키지

- 패키지(package) : 사용자 정의 클래스

2. 다른 패키지 안에 들어 있는 클래스

src - a - b - Called.java : check()메소드

c - Calling.java : main() 메소드

1) 다른 패키지 안에 들어 있는 클래스에 접근 하기 위해서는 접근 제어제가 public 접근 제어자로 되어 있어야 한다.
2) 다른 패키지 안에 들어 있는 클래스에 접근 하기 위해서는 해당 클래스를 import 를 해야 된다.

접근 제한자

- 접근 제한자(Access Modifier)

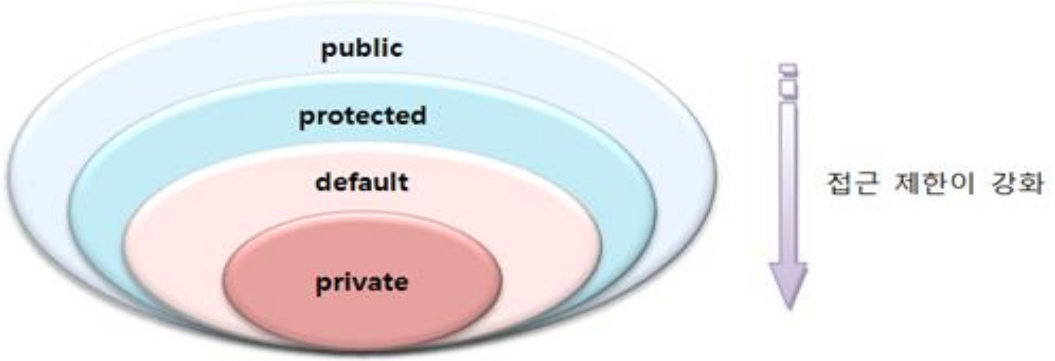
1. 접근 제한자는 클래스와 클래스를 구성하는 멤버에 대한 접근을 제한하는 역할을 한다.

2. 접근 제한자는 클래스와 클래스를 구성하는 필드, 생성자, 메소드 앞에 붙일 수 있다.

접근 제한자

❖ 접근 제한자의 종류

- public : 외부 클래스에서 자유롭게 사용할 수 있다.
- default : 같은 패키지에 소속된 클래스에서만 사용할 수 있다. - 생략된 형태
- protected : 같은 패키지 또는 자식 클래스에서 사용할 수 있다. - 상속에서 주로 사용
- private : 외부 클래스에서 사용할 수 없다. (클래스 내부에서만 사용가능)



접근 제한	적용 대상	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

접근 제한자

- 접근 제한자(Access Modifier)

접근제한자	자신의클래스	같은패키지	하위클래스	다른패키지
private	O	X	X	X
생략(default)	O	O	X	X
protected	O	O	O	X
public	O	O	O	O

final

- final
  1. final이 멤버변수(필드)에 사용될 경우
     상수 - 값을 수정할 수 없다
     final int a=10;

  2. final이 메소드에 사용될 경우
     메소드 오버라이딩을 허용하지 않는다는 의미
     public final void setStr(String s) { }

  3. final이 클래스에 사용될 경우
     상속을 허용하지 않는다는 의미
     final class FinalClass{ }