

Java

제 8 강

함수 & 객체지향개념 I-2

2. 함수(array)

2.1 함수의 개요?

2.2 함수의 종류

2.3 함수는 변수다

2.4 리턴과 매개변수

2.5 기본 데이터 타입의 값복사

2.6 값복사와 매개변수 전달

2.7 계산기 예제

2.1 함수의 개요?

■ 함수를 사용하지 않은 예

```
◆ double a = 10.0, b = 20.0, c=30.0;  
◆ double m = (a + b + c)/3.0F;  
◆ double d = (a-m) * (a-m) + (b-m) * (b-m) + (c-m) * (c-m);  
◆ double f = d/3.0; //결과
```

```
◆ double x = 5.0, y= 25.0, z = 50.0;  
◆ double s =( x + y + z)/3.0;  
◆ double t = (x-s) * (x-s) + (y-s) * (y-s) + (z-s) * (z-s);  
◆ double w = t/3.0; //결과
```

함수(Function)란?

특정 작업이 반복되는 경우, 이 작업을 따로 정의해두고 필요할 때마다 사용할 수 있는 프로그램 내의 독립된 작업의 단위이다.


두 부분의 로직이 중복된다.

■ 함수화

중복되는 로직을 함수로 만들어두면 함수의 이름으로 재사용가능하다. 

```
◆ double Variance (double x1, double x2, double x3){  
◆     double m = (x1 + x2 + x3)/3.0;  
◆     double d = (x1-m) * (x1-m) + (x2-m) * (x2-m) + (x3-m) * (x3-m);  
◆     double f = d/3.0; //결과  
◆     return f;  
◆ }
```

■ 함수의 사용



```
◆ double d1 = Variance (10.0, 20.0, 30.0);  
◆ double d2 = Variance (5.0, 25.0, 50.0);
```

함수를 사용하는 이유?

특정 작업을 반복해야 하는 경우에 언제든지 사용할 수 있다.
함수단위로 관리하면 작업을 관리하기 편리하다.

2.2 함수의 종류

■ 함수의 종류

- ◆ 일만 하는 함수
- ◆ 일을 한 후 값을 리턴하는 함수

■ 일만 하는 함수

- ◆ `void SumA(int x, int y){`
- ◆ `int c;`
- ◆ `c = x + y;`
- ◆ `System.out.println("c=" + c);`
- ◆ `return; //값을 리턴하지 않고 단순히 끝나 버림`
- ◆ `}`



→ 일만 하는 함수의 호출
SumA(3, 4);

■ 값을 리턴하는 함수

- ◆ `int SumB(int x, int y){`
- ◆ `int c;`
- ◆ `c = x + y;`
- ◆ `return c; //c의 값을 리턴`
- ◆ `}`



→ 값을 리턴하는 함수의 호출
int s = SumB(3, 4);

2.3 함수는 변수다

■ 변수와 함수의 할당

- ◆ `int a = 7;`
- ◆ `SumB(3, 4);`

■ 변수와 함수의 할당

- ◆ `int a = 7;`
- ◆ `int b = a;`
- ◆ `int c = SumB(3, 4);`

■ 변수의 재할당

- ◆ `int a = 7;`
- ◆ `a = 10;`

■ 함수의 재할당

- ◆ `int c = SumB(3, 4);`
- ◆ `c = SumB(5, 5);`

■ 기본 데이터 타입 변수와 함수의 차이점

- ◆ 기본 데이터 타입 변수는 직접할당을 원칙으로 한다.
- ◆ 함수는 간접할당을 원칙으로 한다.

2.4 리턴과 매개변수

■ 기본 데이터 타입 변수의 선언

- ◆ `int a;`

■ `int a`의 분해

- ◆ 데이터 타입 : `int`
- ◆ 변수 : `a`

■ `SumB()` 함수의 선언

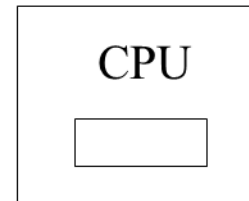
- ◆ `int SumB(int x, int y){`
- ◆ `int c;`
- ◆ `c = x + y;`
- ◆ `return c;`
- ◆ `}`

`int c = SumB(5, 5);`

x
y
c

■ `SumB()`의 분해

- ◆ 리턴형 : `int`
- ◆ 함수이름 : `SumB`
- ◆ 매개변수 : `(int x, int y)`
- ◆ 작업의 내용 : `{ }`
- ◆ 종료키워드 : `return`
- ◆ 리턴값 : `c`



■ 참고

- ◆ 함수 내부의 변수 `c`는 지역 변수이며, 매개변수 `x, y` 또한 지역 변수이다.
- ◆ 변수 `x, y`는 함수 외부와 연결된 지역변수에 해당한다.

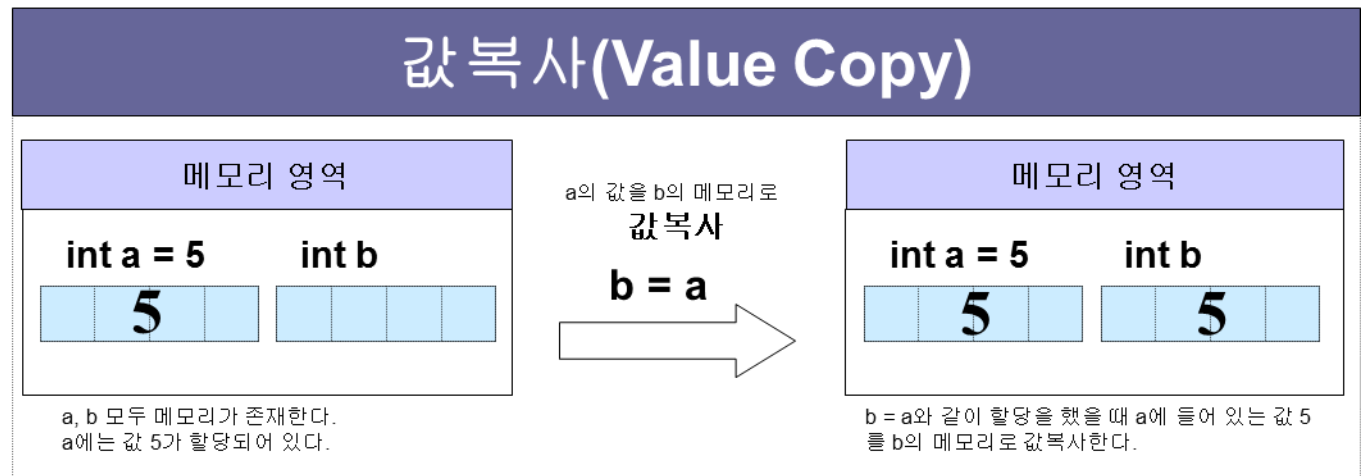
2.5 기본 데이터 타입의 값복사

■ 값복사(Value Copy)란?

- ◆ 값복사란 두 개의 메모리가 존재하고 한쪽의 메모리에 들어 있는 값을 다른 쪽의 메모리로 그 값만을 복사하는 행위를 말한다.

■ 값복사의 예

- ◆ `int a = 5;`
- ◆ `int b;`
- ◆ `b = a;`



2.6 값복사와 매개변수 전달

■ 매개변수의 전달

- ◆ 값타입에서 매개변수의 전달은 값복사의 기법만을 사용한다.
- ◆ 이것을 값에 의한 호출(Call By Value) 또는 값복사에 의한 호출이라고 한다.

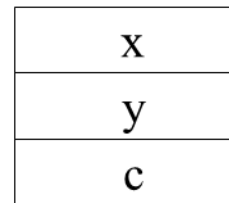
■ SumB() 함수의 선언

- ◆ `int SumB(int x, int y){`
- ◆ `int c;`
- ◆ `c = x + y;`
- ◆ `return c;`
- ◆ `}`



■ 함수의 호출

- ◆ `int a = 3;`
- ◆ `int b = 4;`
- ◆ `int c = SumB(a,b);`

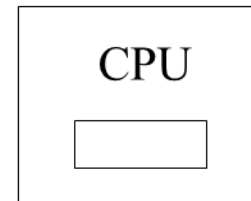


■ 값복사의 예 I

- ◆ `x = a;` //함수를 호출할 때 값복사 발생
- ◆ `y = b;` //함수를 호출할 때 값복사 발생
- ◆ `c = SumB(a,b);` //함수의 리턴값을 c의 메모리에 값복사

■ 값복사의 예 II

- ◆ `int a = 3;`
- ◆ `int b = 4;`
- ◆ `c = x + y;`



2.7 계산기 예제

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

    Scanner reader = new Scanner(System.in);
    System.out.println("두수를 계산하는 계산기 프로그램");

    // 실수값을 키보드로 입력
    System.out.print("첫번째 수를 입력하시오->");
    double first = reader.nextDouble();
    System.out.print("두번째 수를 입력하시오->");
    double second = reader.nextDouble();

    System.out.print("연산자를 선택하시오(+, -, *, /)->");
    char operator = reader.next().charAt(0);

    double result = 0.0d;
```

8강 함수 & 객체지향개념 I

```
switch(operator)
{
    case '+':
        result = first + second;
        break;
    case '-':
        result = first - second;
        break;
    case '*':
        result = first * second;
        break;
    case '/':
        result = first / second;
        break;
    // 연산자가 맞지 않는 경우 (+, -, *, /)
    default:
        System.out.printf("오류: 연산자가 잘못되었습니다");
        return;
}
System.out.printf("%.1f %c %.1f = %.1f", first, operator, second, result);
}
```

사칙연산 부분을 함수로 작성하시오

- 1. 객체지향언어란?
- 2. 클래스와 객체

객체지향개념 I-1

- 3. 변수와 메서드
- 4. 메서드 오버로딩

객체지향개념 I-2

- 5. 생성자
- 6. 변수의 초기화

객체지향개념 I-3

3. 변수와 메서드

- 3.1 선언위치에 따른 변수의 종류
- 3.2 클래스변수와 인스턴스변수
- 3.3 메서드
- 3.4 return문
- 3.5 메서드 호출
- 3.6 JVM의 메모리구조
- 3.7 기본형 매개변수와 참조형 매개변수
- 3.8 재귀호출
- 3.9 클래스 메서드와 인스턴스 메서드
- 3.10 멤버간의 참조와 호출
- 3.11 랜덤 그래프 예제

4. 메서드 오버로딩(method overloading)

- 4.1 메서드 오버로딩이란?
- 4.2 오버로딩의 조건
- 4.3 오버로딩의 예

3. 변수와 메서드

3.1 선언위치에 따른 변수의 종류

“변수의 선언위치가 변수의 종류와 범위(scope)을 결정한다.”

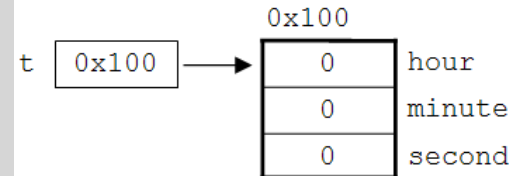
```
class Variables
{
    int iv;           // 인스턴스변수
    static int cv;    // 클래스변수 (static변수, 공유변수)

    void method()
    {
        int lv = 0;  // 지역변수
    }
}
```

클래스영역

메서드영역

```
class Time {
    int hour;
    int minute;
    int second;
}
```



변수의 종류	선언위치	생성시기
클래스변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스변수		인스턴스 생성시
지역변수	메서드 영역	변수 선언문 수행시

3.1 선언위치에 따른 변수의 종류

▶ 인스턴스변수(instance variable)

- 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장가능
- 인스턴스 생성 후, '참조변수.인스턴스변수명'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지컬렉터에 의해자동 제거됨

▶ 클래스변수(class variable)

- 같은 클래스의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
- 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

▶ 지역변수(local variable)

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

3.2 클래스변수와 인스턴스변수

“인스턴스변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 각기 다른 값을 유지할 수 있지만, 클래스변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다.”



속성	무늬 숫자
	폭 높이
기능	...

인스턴스변수

클래스변수

```
class Card {
```

```
    String kind; // 무늬
```

```
    int number; // 숫자
```

```
    static int width = 100; // 폭
```

```
    static int height = 250; // 높이
```

```
}
```


3.3 메서드(method)

▶ 메서드란?

- 작업을 수행하기 위한 명령문의 집합
- 어떤 값을 입력받아서 처리하고 그 결과를 돌려준다.
(입력받는 값이 없을 수도 있고 결과를 돌려주지 않을 수도 있다.)

▶ 메서드의 장점과 작성지침

- 반복적인 코드를 줄이고 코드의 관리가 용이하다.
- 반복적으로 수행되는 여러 문장을 메서드로 작성한다.
- 하나의 메서드는 한 가지 기능만 수행하도록 작성하는 것이 좋다.
- 관련된 여러 문장을 메서드로 작성한다.

3.3 메서드(method)

```
public static void main(String args[]) {  
    while(true) {  
        switch(displayMenu()) { // 화면에 메뉴를 출력한다.  
            case 1 :  
                inputRecord();    // 데이터를 입력받는다.  
                break;  
            case 2 :  
                deleteRecord();    // 데이터를 삭제한다.  
                break;  
            case 3 :  
                sortRecord();      // 데이터를 정렬한다.  
                break;  
            case 4 :  
                System.out.println("프로그램을 종료합니다.`");  
                System.exit(0);  
        }  
    } // while(true)  
} // main메서드의 끝
```

3.3 메서드(method)

- ▶ 메서드를 정의하는 방법 – 클래스 영역에만 정의할 수 있음

리턴타입 메서드이름 (타입 변수명, 타입 변수명, ...)

선언부

{

// 메서드 호출시 수행될 코드

구현부

}

int add(int a, int b)

선언부

{

int result = a + b;

return result; // 호출한 메서드로 결과를 반환한다.

구현부

}

void power() { // 반환값이 없는 경우 리턴타입 대신 void를 사용한다.

power = !power;

}

3.4 return문

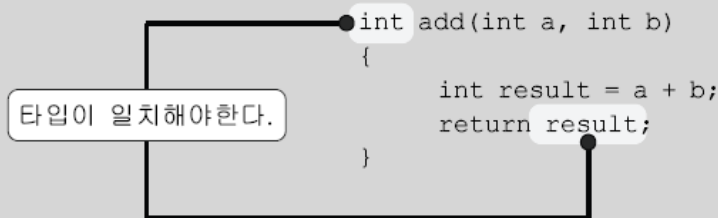
- ▶ 메서드가 정상적으로 종료되는 경우
 - 메서드의 블록{}의 끝에 도달했을 때
 - 메서드의 블록{}을 수행 도중 return문을 만났을 때
- ▶ return문
 - 현재 실행 중인 메서드를 종료하고 호출한 메서드로 되돌아간다.

1. 반환값이 없는 경우 - return문만 써주면 된다.

```
return;
```

2. 반환값이 있는 경우 - return문 뒤에 반환값을 지정해 주어야 한다.

```
return 반환값;
```



3.4 return문 - 주의사항

- ▶ 반환값이 있는 메서드는 모든 경우에 return문이 있어야 한다.

```
int max(int a, int b) {  
    if(a > b)  
        return a;  
}
```

```
int max(int a, int b) {  
    if(a > b)  
        return a;  
    else  
        return b;  
}
```

- ▶ return문의 개수는 최소화하는 것이 좋다.

```
int max(int a, int b) {  
    if(a > b)  
        return a;  
    else  
        return b;  
}
```

```
int max(int a, int b) {  
    int result = 0;  
    if(a > b)  
        result = a;  
    else  
        result = b;  
    return result;  
}
```

3.5 메서드의 호출

▶ 메서드의 호출방법

참조변수.메서드 이름();

// 메서드에 선언된 매개변수가 없는 경우

참조변수.메서드 이름(값1, 값2, ...);

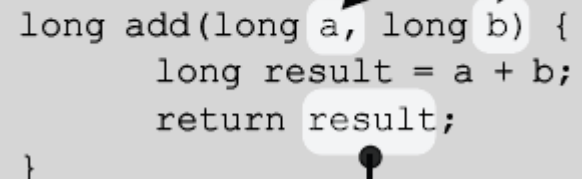
// 메서드에 선언된 매개변수가 있는 경우

```
class MyMath {  
    long add(long a, long b) {  
        long result = a + b;  
        return result;  
    }  
    //  
    return a + b;  
    ...  
}
```

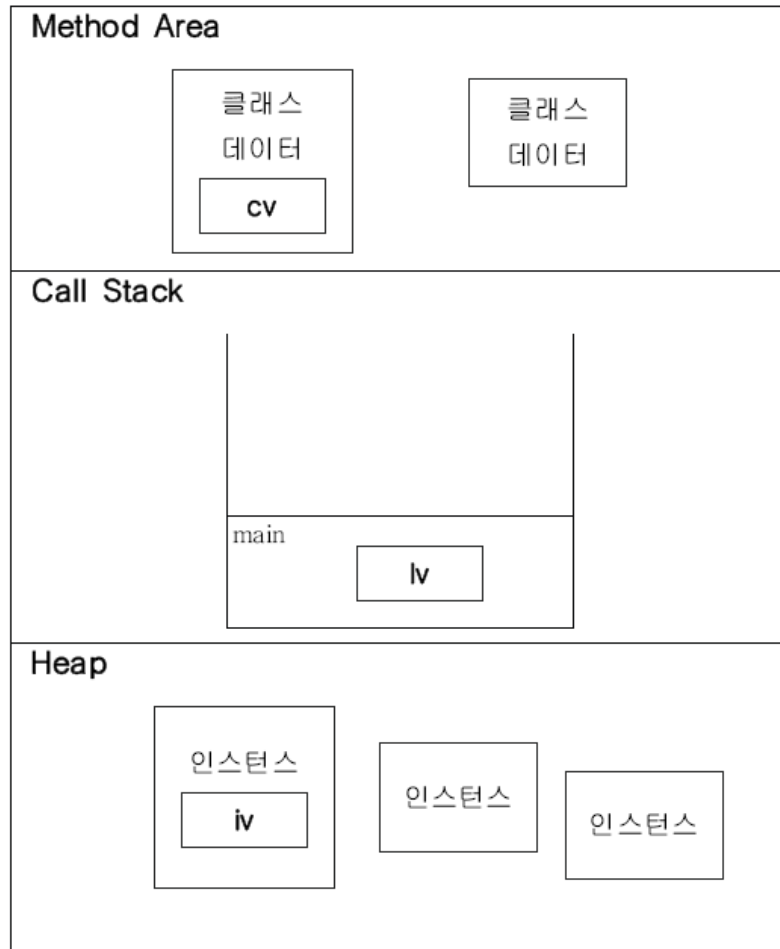
```
MyMath mm = new MyMath();
```

```
long value = mm.add(1L, 2L);
```

```
long add(long a, long b) {  
    long result = a + b;  
    return result;  
}
```



3.6 JVM의 메모리 구조

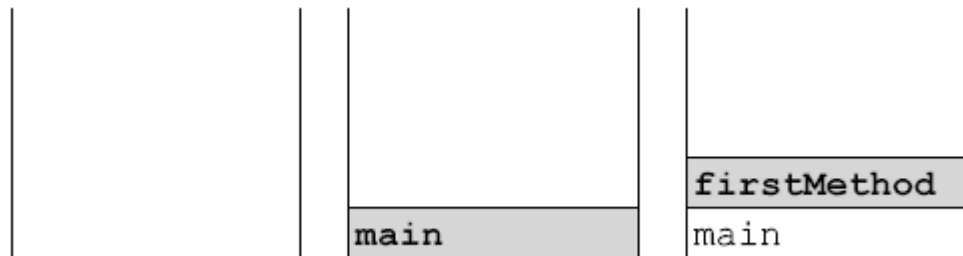


- ▶ 메서드영역(Method Area)
 - 클래스 정보와 클래스변수가 저장되는 곳
- ▶ 호출스택(Call Stack)
 - 메서드의 작업공간. 메서드가 호출되면 메서드 수행에 필요한 메모리공간을 할당받고 메서드가 종료되면 사용하던 메모리를 반환한다.
- ▶ 힙(Heap)
 - 인스턴스가 생성되는 공간. new연산자에 의해서 생성되는 배열과 객체는 모두 여기에 생성된다.

3.6 JVM의 메모리 구조 - 호출스택

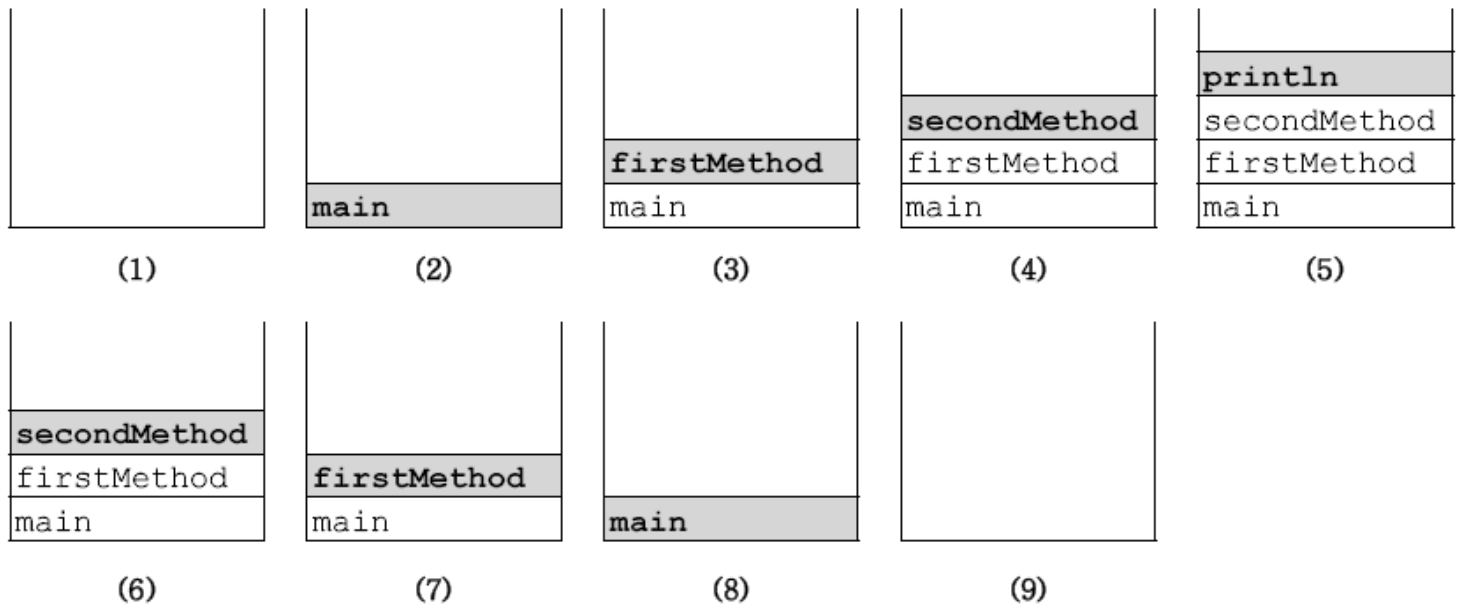
▶ 호출스택의 특징

- 메서드가 호출되면 수행에 필요한 메모리를 스택에 할당받는다.
- 메서드가 수행을 마치면 사용했던 메모리를 반환한다.
- 호출스택의 제일 위에 있는 메서드가 현재 실행중인 메서드다.
- 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드다.



3.6 JVM의 메모리 구조 - 호출스택

```
class CallStackTest {  
    public static void main(String[] args) {  
        firstMethod();  
    }  
    static void firstMethod() {  
        secondMethod();  
    }  
    static void secondMethod() {  
        System.out.println("secondMethod()");  
    }  
}
```



3.7 기본형 매개변수와 참조형 매개변수

- ▶ 기본형 매개변수 – 변수의 값을 읽기만 할 수 있다.(read only)
- ▶ 참조형 매개변수 – 변수의 값을 읽고 변경할 수 있다.(read & write)

한글 MS-DOS

8 x 12

```
C:\wj2sdk1.4.1\work>javac ParameterTest.java
C:\wj2sdk1.4.1\work>java ParameterTest
```

ParameterTest.java

```
1  class Data { int x; }
2  class ParameterTest {
3  public static void main(String[] args) {
4
5      Data d = new Data();
6      d.x=10;
7      System.out.println("main() : x = " + d.x);
8
9      change(d.x);
10     System.out.println("After change(d.x)");
11     System.out.println("main() : x = " + d.x);
12 }
13
14 static void change(int x) {
15     x = 1000;
16     System.out.println("change() : x = " + x);
17 }
18 }
19
```

Method Area

ParameterTest
클래스

Call Stack

main

Heap

한글 MS-DOS

8 x 12

```
C:\wj2sdk1.4.1\work>javac ParameterTest.java
C:\wj2sdk1.4.1\work>java ParameterTest
```

ParameterTest.java

```
1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19
```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

main

d 0x100

Heap

Data인스턴스

0x100

x

0

2. Data클래스가 메모리에 로드되고, Data타입의 참조변수 d가 main메서드의 지역변수로 생성된다.
Data클래스의 인스턴스가 생성되고, 생성된 인스턴스의 주소가 참조변수 d에 저장된다.



한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest
main() : x = 10

ParameterTest.java

```

1  class Data { int x; }
2  class ParameterTest {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x=10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d.x);
10         System.out.println("After change(d.x)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(int x) {
15         x = 1000;
16         System.out.println("change() : x = " + x);
17     }
18 }
19

```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

println

main

d 0x100

Heap

Data인스턴스

0x100

x

10

한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest
main() : x = 10

ParameterTest.java

```

1  class Data { int x; }
2  class ParameterTest {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x=10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d.x);
10         System.out.println("After change(d.x)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(int x) {
15         x = 1000;
16         System.out.println("change() : x = " + x);
17     }
18 }
19

```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

change

x

main

d

0x100

Heap

Data인스턴스

0x100

x

10

5.change메서드를 호출하면서 매개변수로 참조변수 d가 가리키고 있는 인스턴스의 멤버변수 x(d.x)의 값을 넘겨준다. d.x의 값인 10이 change의 매개변수 x에 복사된다.



MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest
main() : x = 10

ParameterTest.java

```

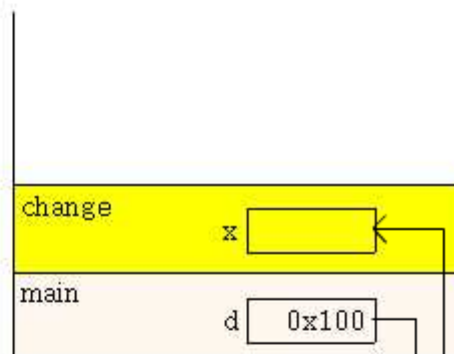
1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19

```

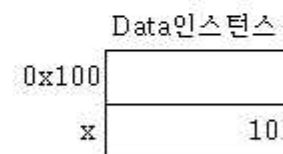
Method Area

ParameterTest
클래스Data
클래스

Call Stack



Heap



5.change메서드를 호출하면서 매개변수로 참조변수 d가 가리키고 있는 인스턴스의 멤버변수 x(d.x)의 값을 넘겨준다. d.x의 값인 10이 change의 매개변수 x에 복사된다.

한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest
main() : x = 10

ParameterTest.java

```
1 class Data { int x; }  
2 class ParameterTest {  
3     public static void main(String[] args) {  
4  
5         Data d = new Data();  
6         d.x=10;  
7         System.out.println("main() : x = " + d.x);  
8  
9         change(d.x);  
10        System.out.println("After change(d.x)");  
11        System.out.println("main() : x = " + d.x);  
12    }  
13  
14    static void change(int x) {  
15        x = 1000;  
16        System.out.println("change() : x = " + x);  
17    }  
18 }  
19
```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

change

x 1000

main

d 0x100

Heap

Data인스턴스

0x100

x 10

MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest

main() : x = 10

change() : x = 1000

ParameterTest.java

```

1  class Data { int x; }
2  class ParameterTest {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x=10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d.x);
10         System.out.println("After change(d.x)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(int x) {
15         x = 1000;
16         System.out.println("change() : x = " + x);
17     }
18 }
19

```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

println

change

x 1000

main

d 0x100

Heap

Data인스턴스

0x100

x 10

MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest

main() : x = 10

change() : x = 1000

ParameterTest.java

```
1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19
```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

main

d 0x100

Heap

Data인스턴스

0x100

x

10

MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest.java

C:\wj2sdk1.4.1\work>java ParameterTest

main() : x = 10

change() : x = 1000

After change(d.x)

main() : x = 10

ParameterTest.java

```
1 class Data { int x; }
2 class ParameterTest {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x=10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d.x);
10        System.out.println("After change(d.x)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(int x) {
15        x = 1000;
16        System.out.println("change() : x = " + x);
17    }
18 }
19
```

Method Area

ParameterTest
클래스Data
클래스

Call Stack

main

d 0x100

Heap

Data인스턴스

0x100

x

10

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java

C:\Wj2sdk1.4.1\work>java ParameterTest2

ParameterTest2.java

```
1 class Data { int x; }
2 class ParameterTest2 {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x = 10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d);
10        System.out.println("After change(d)");
11        System.out.println("main() : x = " + d.x);
12
13    }
14    static void change(Data d) {
15        d.x = 1000;
16        System.out.println("change() : x = " + d.x);
17    }
18 }
```

Method Area

ParameterTest2
클래스

Call Stack

main

Heap

MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac ParameterTest2.java

C:\wj2sdk1.4.1\work>java ParameterTest2

ParameterTest2.java

```

1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x = 10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d);
10         System.out.println("After change(d)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(Data d) {
15         d.x = 1000;
16         System.out.println("change() : x = " + d.x);
17     }
18 }

```

Method Area

ParameterTest2
클래스Data
클래스

Call Stack

main

d 0x100

Heap

Data인스턴스

0x100

x

0

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java

C:\Wj2sdk1.4.1\work>java ParameterTest2

main() : x = 10

ParameterTest2.java

```

1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x = 10;
7      System.out.println("main() : x = " + d.x);
8
9          change(d);
10         System.out.println("After change(d)");
11         System.out.println("main() : x = " + d.x);
12     }
13
14     static void change(Data d) {
15         d.x = 1000;
16         System.out.println("change() : x = " + d.x);
17     }
18 }

```

Method Area

ParameterTest2
클래스Data
클래스

Call Stack

println

main

d 0x100

Heap

Data인스턴스

0x100

x

10

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java

C:\Wj2sdk1.4.1\work>java ParameterTest2
main() : x = 10

ParameterTest2.java

```

1 class Data { int x; }
2 class ParameterTest2 {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x = 10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d);
10        System.out.println("After change(d)");
11        System.out.println("main() : x = " + d.x);
12    }
13 }
14 static void change(Data d) {
15     d.x = 1000;
16     System.out.println("change() : x = " + d.x);
17 }
18 }

```

Method Area

ParameterTest2
클래스Data
클래스

Call Stack

change

d 0x100

main

d 0x100

Heap

Data인스턴스

0x100

x

10

5. change에서d를 호출하면서 매개변수로 참조변수 d를 넘겨준다.
main에서d의 참조변수 d의 값(Data인스턴스의 주소)은 change에서d의 매개변수 d에 복사된다.



MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java

C:\Wj2sdk1.4.1\work>java ParameterTest2

main() : x = 10

change() : x = 1000

ParameterTest2.java

```
1 class Data { int x; }
2 class ParameterTest2 {
3     public static void main(String[] args) {
4
5         Data d = new Data();
6         d.x = 10;
7         System.out.println("main() : x = " + d.x);
8
9         change(d);
10        System.out.println("After change(d)");
11        System.out.println("main() : x = " + d.x);
12    }
13
14    static void change(Data d) {
15        d.x = 1000;
16        System.out.println("change() : x = " + d.x);
17    }
18 }
```

Method Area

ParameterTest2
클래스Data
클래스

Call Stack

println

change

d 0x100

main

d 0x100

Heap

Data인스턴스

0x100

x

1000

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac ParameterTest2.java

C:\Wj2sdk1.4.1\work>java ParameterTest2

main() : x = 10

change() : x = 1000

After change(d)

main() : x = 1000

ParameterTest2.java

```

1  class Data { int x; }
2  class ParameterTest2 {
3      public static void main(String[] args) {
4
5          Data d = new Data();
6          d.x = 10;
7          System.out.println("main() : x = " + d.x);
8
9          change(d);
10         System.out.println("After change(d)");
11         System.out.println("main() : x = " + d.x);
12
13     }
14     static void change(Data d) {
15         d.x = 1000;
16         System.out.println("change() : x = " + d.x);
17     }
18 }

```

Method Area

ParameterTest2
클래스Data
클래스

Call Stack

main

d 0x100

Heap

Data인스턴스

0x100

x

1000

9. println메서드를 호출하여 d.x의 값을 출력한다. d.x의 값은 1000이므로 "main() : x = 1000"을 출력한다.



3.8 재귀호출(recursive call)

▶ 재귀호출이란?

- 메서드 내에서 자기자신을 반복적으로 호출하는 것
- 재귀호출은 반복문으로 바꿀 수 있으며 반복문보다 성능이 나쁨
- 이해하기 쉽고 간결한 코드를 작성할 수 있다

▶ 재귀호출의 예(例)

- 팩토리얼, 제곱, 트리은행, 폴더목록표시 등

*팩토리얼 (factorial)

$5! = 5 * 4 * 3 * 2 * 1$

$f(n) = n * f(n-1)$ 단, $f(1) = 1$



```
long factorial(int n) {  
    long result = 0;  
    if(n==1) {  
        result = 1;  
    } else {  
        result = n * factorial(n-1);  
    }  
    return result;  
}
```

3.8 재귀호출(recursive call)

The screenshot shows a Java IDE with a DOS window at the top displaying the compilation and execution of `FactorialTest.java`. The main editor shows the source code for `FactorialTest.java`, which includes a `main` method that calls `factorial(4)` and a static `factorial` method that uses recursion. The `Call Stack` on the right illustrates the sequence of recursive calls, showing frames for `factorial` with `n` values 1, 2, 3, and 4, and a `main` frame. A status bar at the bottom explains the recursive process: `5, n * factorial(n-1)` is calculated in the current frame, which then calls `factorial` with `n-1`, repeating until `n` reaches 1.

Method Area

FactorialTest 클래스

Call Stack

Method	n	result
factorial	1	0
factorial	2	0
factorial	3	0
factorial	4	0
main		

5, `n * factorial(n-1)`를 수행하는 과정에서 다시 `factorial`메서드가 호출된다. 매개변수로는 `n-1`, 현재 `n`의 값이 4이므로 3이 넘겨진다, `n`이 1이 될 때까지 이 과정이 계속 반복된다.

MS 한글 MS-DOS

8 x 12

```
C:\Wj2sdk1.4.1\work>javac FactorialTest.java
C:\Wj2sdk1.4.1\work>java FactorialTest
```

FactorialTest.java

```
1 class FactorialTest {
2     public static void main(String args[]) {
3         long result = factorial(4);
4         System.out.println(result);
5     }
6
7     static long factorial(int n) {
8         long result=0;
9         if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18
```

Method Area

FactorialTest
클래스

Call Stack

main

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```
1 class FactorialTest {  
2     public static void main(String args[]) {  
3         long result = factorial(4);  
4         System.out.println(result);  
5     }  
6  
7     static long factorial(int n) {  
8         long result=0;  
9         if ( n == 1) {  
10             result = 1;  
11         } else {  
12             result = n * factorial(n-1);  
13         }  
14  
15         return result;  
16     }  
17 }  
18
```

Method Area

FactorialTest
클래스

Call Stack

main

result

2. long타입의 변수 result가 main메서드의 지역변수로 생성된다.
factorial메서드를 호출하면서 매개변수로 4를 넘겨준다.



MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```
1 class FactorialTest {
2     public static void main(String args[]) {
3         long result = factorial(4);
4         System.out.println(result);
5     }
6
7     static long factorial(int n) {
8         long result=0;
9         if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18
```

Method Area

FactorialTest
클래스

Call Stack

factorial	n	4	result	0
main	result			

5. $n * \text{factorial}(n-1)$ 을 수행하는 과정에서 다시 factorial메서드가 호출된다. 매개변수로는 $n-1$, 현재 n 의 값이 4이므로 3이 넘겨진다. n 이 1이 될 때까지 이 과정이 계속 반복된다.

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```
1 class FactorialTest {
2     public static void main(String args[]) {
3         long result = factorial(4);
4         System.out.println(result);
5     }
6
7     static long factorial(int n) {
8         long result=0;
9         if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18
```

Method Area

FactorialTest
클래스

Call Stack

factorial	n	3	result	0
factorial	n	4	result	0
main	result			

5. $n * \text{factorial}(n-1)$ 을 수행하는 과정에서 다시 factorial메서드가 호출된다. 매개변수로는 $n-1$, 현재 n 의 값이 4이므로 3이 넘겨진다. n 이 1이 될 때까지 이 과정이 계속 반복된다.

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```

1  class FactorialTest {
2      public static void main(String args[]) {
3          long result = factorial(4);
4          System.out.println(result);
5      }
6
7      static long factorial(int n) {
8          long result=0;
9          if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18

```

Method Area

FactorialTest
클래스

Call Stack

factorial	n	1	result	1
factorial	n	2	result	0
factorial	n	3	result	0
factorial	n	4	result	0
main	result			

7. 지역변수 result의 값 1을 자신을 호출한 메서드에게 반환하면서 현재 수행중인 메서드는 종료된다.



MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```

1  class FactorialTest {
2      public static void main(String args[]) {
3          long result = factorial(4);
4          System.out.println(result);
5      }
6
7      static long factorial(int n) {
8          long result=0;
9          if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18

```

result = n * factorial(n-1)
 result = 2 * 1
 result = 2

Method Area

FactorialTest
클래스

Call Stack

factorial	n	2	result	0
factorial	n	3	result	0
factorial	n	4	result	0
main	result			

8. factorial메서드를 호출했던 곳으로 돌아가서 다시 수행을 계속한다. factorial(n-1) 대신 factorial(n-1)을 호출한 결과로 반환받은 값 1이 사용되어 계산된다.



MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```

1  class FactorialTest {
2      public static void main(String args[]) {
3          long result = factorial(4);
4          System.out.println(result);
5      }
6
7      static long factorial(int n) {
8          long result=0;
9          if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18

```

result = n * factorial(n-1)
 result = 3 * 2
 result = 6

Method Area

FactorialTest
클래스

Call Stack

factorial	n	3	result	0
factorial	n	4	result	0
main	result			

MS 한글 MS-DOS

8 x 12

C:\wj2sdk1.4.1\work>javac FactorialTest.java

C:\wj2sdk1.4.1\work>java FactorialTest

FactorialTest.java

```

1  class FactorialTest {
2      public static void main(String args[]) {
3          long result = factorial(4);
4          System.out.println(result);
5      }
6
7      static long factorial(int n) {
8          long result=0;
9          if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18

```

result = n * factorial(n-1)
 result = 4 * 6
 result = 24

Method Area

FactorialTest
클래스

Call Stack

factorial	n	4	result	0
main	result			

MS 한글 MS-DOS

8 x 12

```
C:\Wj2sdk1.4.1\work>javac FactorialTest.java
C:\Wj2sdk1.4.1\work>java FactorialTest
```

FactorialTest.java

```
1 class FactorialTest {
2     public static void main(String args[]) {
3         long result = factorial(4);
4         System.out.println(result);
5     }
6
7     static long factorial(int n) {
8         long result=0;
9         if ( n == 1) {
10             result = 1;
11         } else {
12             result = n * factorial(n-1);
13         }
14
15         return result;
16     }
17 }
18
```

result = factorial(4)
result = 24

Method Area

FactorialTest
클래스

Call Stack

main

result

MS 한글 MS-DOS

8 x 12

C:\Wj2sdk1.4.1\work>javac FactorialTest.java

C:\Wj2sdk1.4.1\work>java FactorialTest
24

FactorialTest.java

```
1 class FactorialTest {  
2     public static void main(String args[]) {  
3         long result = factorial(4);  
4         System.out.println(result);  
5     }  
6  
7     static long factorial(int n) {  
8         long result=0;  
9         if ( n == 1) {  
10            result = 1;  
11        } else {  
12            result = n * factorial(n-1);  
13        }  
14  
15        return result;  
16    }  
17 }  
18
```

Method Area

FactorialTest
클래스

Call Stack

println

main

result

24

3.9 클래스메서드(static메서드)와 인스턴스메서드

▶ 인스턴스메서드

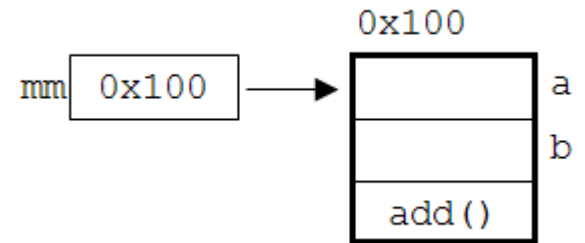
- 인스턴스 생성 후, '참조변수.메서드이름()'으로 호출
- 인스턴스변수나 인스턴스메서드와 관련된 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 사용가능

▶ 클래스메서드(static메서드)

- 객체생성없이 '클래스이름.메서드이름()'으로 호출
- 인스턴스변수나 인스턴스메서드와 관련없는 작업을 하는 메서드
- 메서드 내에서 인스턴스변수 사용불가
- 메서드 내에서 인스턴스변수를 사용하지 않는다면 static을 붙이는 것을 고려한다.

3.9 클래스메서드(static메서드)와 인스턴스메서드

```
class MyMath2 {  
    long a, b;  
  
    long add() {    // 인스턴스메서드  
        return a + b;  
    }  
  
    static long add(long a, long b) { // 클래스메서드(static메서드)  
        return a + b;  
    }  
}
```



```
class MyMathTest2 {  
    public static void main(String args[]) {  
        System.out.println(MyMath2.add(200L,100L)); // 클래스메서드 호출  
        MyMath2 mm = new MyMath2(); // 인스턴스 생성  
        mm.a = 200L;  
        mm.b = 100L;  
        System.out.println(mm.add()); // 인스턴스메서드 호출  
    }  
}
```

3.10 멤버간의 참조와 호출(1/2) – 메서드의 호출

“같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 static멤버들은 인스턴스멤버들을 참조할 수 없다.”

```
class TestClass {  
    void instanceMethod() {}          // 인스턴스메서드  
    static void staticMethod() {}    // static메서드  
  
    void instanceMethod2() {          // 인스턴스메서드  
        instanceMethod();            // 다른 인스턴스메서드를 호출한다.  
        staticMethod();              // static메서드를 호출한다.  
    }  
  
    static void staticMethod2() {     // static메서드  
        instanceMethod();            // 에러!!! 인스턴스메서드를 호출할 수 없다.  
        staticMethod();              // static메서드는 호출 할 수 있다.  
    }  
} // end of class
```


3.10 멤버간의 참조와 호출(2/2) - 변수의 접근

“같은 클래스의 멤버간에는 객체생성이나 참조변수 없이 참조할 수 있다. 그러나 static 멤버들은 인스턴스 멤버들을 참조할 수 없다.”

```
class TestClass2 {  
    int iv;           // 인스턴스변수  
    static int cv;    // 클래스변수  
  
    void instanceMothod() {           // 인스턴스에서드  
        System.out.println(iv);      // 인스턴스변수를 사용할 수 있다.  
        System.out.println(cv);      // 클래스변수를 사용할 수 있다.  
    }  
  
    static void staticMethod() {      // static에서드  
        System.out.println(iv);      // 에러!!! 인스턴스변수를 사용할 수 없다.  
        System.out.println(cv);      // 클래스변수를 사용할 수 있다.  
    }  
} // end of class
```

3.11 램덤 그래프 예제 (GraphRandom.java)

```
public static void main(String[] args)
{
    int[] number = new int[100];
    int[] counter = new int[10];

    for (int i=0; i < number.length ; i++ ) {
        System.out.print(number[i] = (int)(Math.random() * 10));
    }
    System.out.println();

    for (int i=0; i < number.length ; i++ ) {
        counter[number[i]]++;
    }

    for (int i=0; i < counter.length ; i++ ) {
        System.out.println( i +"의 개수 :"+
                           printGraph('■',counter[i]) + " " + counter[i]);
    }
}
```

```
public static String printGraph(char ch, int value) {  
    String result = "";  
    for(int i=0; i < value;i++) {  
        result += ch;  
    }  
  
    return result;  
}
```

4. 메서드 오버로딩

4.1 메서드 오버로딩(method overloading)이란?

“하나의 클래스에 같은 이름의 메서드를 여러 개 정의하는 것을 메서드 오버로딩, 간단히 오버로딩이라고 한다.”

* overload - vt. 과적하다. 부담을 많이 지우다.

4.2 오버로딩의 조건

- 메서드의 이름이 같아야 한다.
- 매개변수의 개수 또는 타입이 달라야 한다.
- 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다.

(리턴타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다.)

4.3 오버로딩의 예(1/3)

▶ System.out.println메서드

- 다양하게 오버로딩된 메서드를 제공함으로써 모든 변수를 출력할 수 있도록 설계

```
void println()  
void println(boolean x)  
void println(char x)  
void println(char[] x)  
void println(double x)  
void println(float x)  
void println(int x)  
void println(long x)  
void println(Object x)  
void println(String x)
```

4.3 오버로딩의 예(1/2)

- ▶ 매개변수의 이름이 다른 것은 오버로딩이 아니다.

[보기1]

```
int add(int a, int b) { return a+b; }  
int add(int x, int y) { return x+y; }
```

- ▶ 리턴타입은 오버로딩의 성립조건이 아니다.

[보기2]

```
int add(int a, int b) { return a+b; }  
long add(int a, int b) { return (long) (a + b); }
```

4.3 오버로딩의 예(1/3)

- ▶ 매개변수의 타입이 다르므로 오버로딩이 성립한다.

[보기3]

```
long add(int a, long b) { return a+b; }  
long add(long a, int b) { return a+b; }
```

- ▶ 오버로딩의 올바른 예 – 매개변수는 다르지만 같은 의미의 기능수행

[보기4]

```
int add(int a, int b) { return a+b; }  
long add(long a, long b) { return a+b; }  
int add(int[] a) {  
    int result =0;  
  
    for(int i=0; i < a.length; i++) {  
        result += a[i];  
    }  
    return result;  
}
```


4.4 오버로딩 연습

```
2
3 public class DrawLineTest {
4
5     public static void main(String[] args) {
6
7         // drawLine 클래스에 함수 오버로딩 정의
8
9         // ===== 선그리기 (20)
10        line();
11        // 특정문자로 선그리기
12        line('*');
13        line('-');
14        // 특정문자로 선그리기 및 길이입력
15        line('+', 40);
16    }
17
18 }
```

감사합니다.