



java 3강-메서드

양 명 속

[now4ever7@gmail.com]



목차

- 메서드
- 메서드 오버로딩(overloading)
- 재귀 호출



함수 (메서드)

■ 메서드

- 자주 반복하여 사용하는 내용에 대해 특정 이름으로 정의한 묶음
- 클래스 내부에 존재하면서 **특정기능(Function)**을 수행하는 최소 실행 단위

- 반환값, 메서드 이름, 매개변수(인수)로 구성

int AAA() => 반환타입 이름(월넣어야하나)

java는 js와 다르게 매개변수가 필수임
반환타입은 return값에 맞춤

- 필요할 때마다 호출하여 원하는 작업을 시킬 수 있음
- 불필요한 반복을 제거
- 코드의 재사용성을 높이는 역할을 함

- 객체지향기법 - 함수는 클래스에 소속됨
 - 클래스에 소속되는 함수를 메서드라고 부름

메서드

```
class MethodTest{
    public static void main(String[] args){
        for (int i=0;i<5 ;i++ )
        {
            System.out.print("*");
        }
        System.out.println("\nHello");

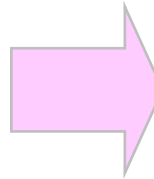
        for (int i=0;i<5 ;i++ )
        {
            System.out.print("*");
        }
        System.out.println("\nJAVA");

        for (int i=0;i<5 ;i++ )
        {
            System.out.print("*");
        }
        System.out.println("\n!!!!");

        for (int i=0;i<5 ;i++ )
        {
            System.out.print("*");
        }

    } //main
} //class
```

```
*****
Hello
*****
JAVA
*****
!!!!
*****
```



```
public static void main(String[] args)
{
    X
    System.out.println("\nHello");

    X
    System.out.println("\nJAVA");

    X
    System.out.println("\n!!!!");

    X

} //main
```

```
X()
{
    for (int i=0;i<5 ;i++ )
    {
        System.out.print("*");
    }
}
```



메서드

```
class MethodTest
{
    public static void X(){
        for (int i=0;i<5 ;i++ )
        {
            System.out.print("*");
        }
    }

    public static void main(String[] args){
        X();
        System.out.println("\nHello");

        X();
        System.out.println("\nJAVA");

        X();
        System.out.println("\n!!!!");

        X();
    } //main
} //class
```



메서드

- 메서드의 종류

- 기본적으로 제공되는 메서드(내장 메서드)

- `System.out.println()`, `Integer.parseInt()`,
`Character.toUpperCase()` 등

- 사용자 정의 메서드

- 매개변수가 없는 메서드
 - 매개변수가 있는 메서드
 - 반환값이 있는 메서드
 - 반환값이 없는 메서드



메서드 정의

- main도 메서드

- 새로운 메서드도 main과 같은 형식으로 정의

```
class 샘플클래스
{
    public static void 샘플메서드( )
    {
        ....
    }

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

메서드(Method) 형식

결과값 형태

입력값

접근제한자 반환형 메서드이름 (매개변수)

{

블록;
return 반환값;

}

```
class 샘플클래스
{
    public static void 샘플메서드( )
    {
        .....
    }
}
```


메서드를 구성하는 것들

- 메서드의 이름
- 메서드의 입력 - 매개변수
- 메서드의 결과 - 결과 반환, 반환타입(반환형)
- 메서드의 기능

반환형

메서드 이름

매개변수 선언

public static

double

calcInterest

(int money)

{

메서드 몸체의 시작

double interest = money*0.016;

return interest;

n의 값 반환

}

메서드 몸체의 끝



메서드

- 매개변수(parameter, 인수, 인자)
 - 입력 값
- 반환형(return type)
 - 작업을 마친 후의 결과를 호출자에게 돌려주는 값의 형태
- 접근 제한자(접근 제어자)
 - 메서드의 속성
 - 메서드를 외부에 있는 클래스에 노출시킬 것인지, 숨길 것인지 등을 결정
 - public, private, protected, default



메서드 호출

- 메서드 호출

- 같은 클래스 안에 있는 메서드 호출

- 메서드명()

- 메서드 이름으로 호출 (괄호 안에 매개변수 목록을 사용)

- 다른 클래스 안에 있는 (public) 메서드 호출

- 클래스명.메서드명()

- 호출되는 메서드는 반드시 **public** 키워드로 선언되어야 함

- 중첩 호출 가능

- 메서드는 중첩 호출이 가능함.

- 즉, 메서드가 또 다른 메서드를 호출할 수 있음.

메서드의 호출

```
public static void main(String[] args )  
{  
    int m = 1000000;  
    ...  
    double i = calcInterest(m);  
    ....  
}
```

m = 1000000;

2

3

```
public static double calcInterest(int money)  
{  
    double interest=money*0.016;  
    return interest;  
}
```

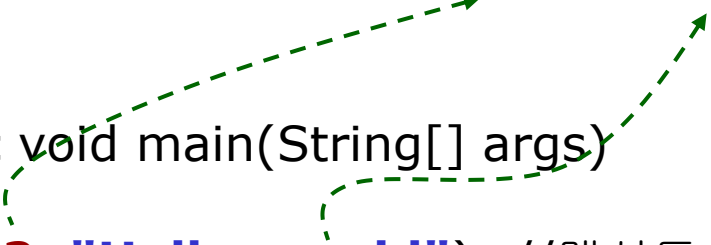
16000

매개변수(parameter, 인수, 인자)

- 매개변수 정의
 - 메서드명 다음에 나오는 괄호안에 위치
 - 각 매개변수에 자료형과 이름 지정
 - 메서드 외부에서 내부로 값을 전달
- 매개변수를 사용한 메서드 호출
 - 각 매개변수에 값이 전달됨

```
public static void methodA(int n, String y) //메서드 정의
{ ... }

public static void main(String[] args)
{
    methodA(2, "Hello, world"); //메서드 호출
}
```

A diagram with two green dashed arrows. One arrow starts from the number '2' in the methodA call and points to the 'int n' parameter in the methodA definition. The other arrow starts from the string '"Hello, world"' in the methodA call and points to the 'String y' parameter in the methodA definition.

return 구문 사용

```
public static double calcInterest(int money)
{
    double interest=money*0.016;
    return interest;
}
```

■ return

- 메서드의 실행을 종료하고 호출원으로 복귀함
- 메서드의 처리 결과를 호출원으로 돌려주는 기능도 함

■ 즉시 리턴

■ 조건문을 사용한 리턴

```
public static void methodA( )
{
    int num;
    //...

    System.out.println("Hello");
    if (num < 10)
        return;
    System.out.println("World");
}
```

```
public static void methodB( )
{
    System.out.println("Hello");
    System.out.println("World");
    // return; 생략가능
}
```

```
public static void main(String[] args)
{
    methodA();
    methodB();
    int val=calcInterest(300000);
}
```

- 메서드 내부에서 **return**을 만나면 메서드의 내부에서 실행하던 것은 멈추고 **자신을 호출한 곳으로 감**
- **return**이 없는 메서드에서는 메서드 내부의 마지막 행까지 실행되고 나서 자신을 호출한 곳으로 감



반환 값

- 반환값이 있는 메서드
 - 반환값의 type이 무엇인지에 따라 메서드를 정의할 때 메서드 이름 앞에 반환형이 와야 함
 - 반환값의 type이 int이면 메서드 정의시에도 반환형이 int가 되어야 함
 - int 메서드이름()
- 반환값이 없는 메서드 : **void**
- return 키워드를 구문에 추가
 - 반환값 설정
 - 호출한 곳으로 반환값 반환
 - **한 개의 값만 반환**해 줌
- 반환값이 있는 메서드는 반드시 값을 리턴해야 함



반환 값

```
public static void main(String[] args)
{
    int x = plus(2, 3);
    System.out.println(x);
}
```



```
int x = 5;
```

```
public static int plus(int a, int b)
{
    int c = a+b;
    return c;
}
```


입력(매개변수), 결과(반환값)가 모두 없는 메서드

```
class MethodTest2
{
```

```
// 1. 매개변수도 없고, 반환값도 없는 메서드인 경우
```

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

```
{
```

```
    //메서드 호출하기
```

```
    func1();
```

```
}
```

void인 메서드는 절대 바로 sys.out으로 바로 찍을 수 없다. void는 return이 없기 때문이다.

```
public static void func1()    //사용자 정의 메서드
```

```
{
```

```
    //특정 기능 처리하기
```

```
    System.out.println("*****");
```

```
}
```

```
}
```

반환값이 없으면 void



매개변수는 있고, 반환값은 없는 메서드

```
class MethodTest2
{
    // 2. 매개변수는 있고, 반환값은 없는 메서드인 경우
    public static void main(String[] args)
    {
        int count=7;
        func3(count);    //매개변수가 있으므로 형식에 맞게 매개변수값을 전달
    }

    public static void func3(int cnt)
    {
        //입력 받은 개수만큼 별을 출력하는 메서드
        for (int i=0;i<cnt ;i++ ) //매개변수로 받은 값으로 특정한 처리하기
        {
            System.out.print("*");
        }
    }
}
```



매개변수는 없고, 반환값은 있는 메서드

```
class MethodTest3
{
    // 3. 매개변수는 없고, 반환값은 있는(return을 써서 값을 전달한다는 것) 메서드인 경우
    public static void main(String[] args)
    {
        //매개변수는 없지만, func2에서의 반환값이 있으므로 그 값을 변수 temp가 받자
        double temp = func2();
        System.out.println(temp);
    }

    public static double func2()
    {
        //1~10까지의 합
        int sum=0;
        for(int i=1;i<=10;i++)
        {
            sum += i; //최종 결과값 산출
        }
        double avg = sum/10.0;
        return avg; //avg변수의 값을 전달, avg변수가 double이므로 func2 메서드의 반환형은
double
    }
}
```



매개변수도 있고, 반환값도 있는 메서드

```
class MethodTest4
{
    // 4. 매개변수도 있고, 반환값도 있는 메서드인 경우
    // 일반적으로 가장 많이 사용하는 메서드 형식
    public static void main(String[] args)
    {
        int c=5, d=7;
        int temp = func4(c, d); //매개변수값 넘기고, 함수가 전달해준 반환값 받고
        System.out.println("결과값=" + temp);
    }

    public static int func4(int a, int b)
    {
        int c = a + b; //매개변수값a, b를 받아왔으므로 그 값을 가지고 처리

        return c; //이 메서드에서의 최종 결과값인 c의 값을 전달, c변수가 int이므로
        func3메서드의 반환형은 int
    }
}
```



예제

```
3 + 4 = 7
10 - 30 = -20
```

```
class Calculate
{
    public static int plus(int a, int b) //매개변수가 있는 메서드
    {
        int c = a + b;
        return c;
    }

    public static int minus(int a, int b)
    {
        int c = a - b;
        return c;
    }

    public static void main(String[] args)
    {
        int value = plus(3, 4); //매개변수값 지정
        System.out.println("3 + 4 = " + value);

        int x=10, y=30;
        value = minus(x, y);
        System.out.println(x + "-" + y + "=" + value);
    }
}
```

<비교>

```
int n = Integer.parseInt("120");
```

```
String str = "470";
int num = Integer.parseInt(str);
```

실습

- 1. 매개변수, 반환값이 모두 없는 경우
 - Hello Java를 출력하는 메서드를 만들고, main() 에서 호출하기
- 2. 매개변수가 있는 경우
 - 매개변수로 횟수를 받아서 그 횟수만큼 Hello Java를 출력하는 메서드를 만들고, main() 에서 호출하기
 - main()에서는 임의의 횟수를 매개변수로 넘겨줌
- 3. 반환값이 있는 경우
 - 1부터 100까지의 홀수의 합을 구하여 그 값을 반환해주는 메서드를 만들고, main()에서 그 값을 받아서 출력
- 4. 매개변수, 반환값이 모두 있는 경우
 - 메서드에서 매개변수로 두 실수를 받아서 두 수의 곱을 구하고, 그 결과값을 반환해주는 메서드 만들기
 - main()에서는 임의의 두 실수를 매개변수로 넘겨주고, 결과값을 받아서 출력

실습1
Hello java

실습2
Hello java
Hello java
Hello java

실습3
1~100까지 홀수의 합=2500

실습4
10.0 * 2.5 = 25.0



지역 변수 사용

- 변수

- 지역변수(Local variables)

- 메서드 내부에서만 사용 가능한 지역변수, 메서드 내에서 선언되는 변수
 - 메서드가 시작될 때 생성
 - 메서드를 빠져나갈 때 사라짐
- ※ 블록변수 – 메서드내의 또 다른 블록(if, for문등)내에서 선언된 변수

- 멤버변수(instance변수)

- 클래스에서 선언된 변수
- 클래스의 멤버역할을 하는 member **field**
- 클래스 내의 여러 메서드에서 공통으로 사용 가능, 클래스 외부에서도 접근 가능하게 할 수 있음

- 클래스 변수(static변수)



예제

```
class MethodTest3
{
    public static int getSum(int count) //int count => 매개변수
    {
        int sum=0; //지역변수
        for (int i=1;i<=count ;i++ ) //int i => 블록변수
        {
            sum += i;
        }
        return sum;
    }
    public static void main(String[] args)
    {
        int count=10;
        int sum = getSum(count);
        System.out.printf("1~ " + count + "까지의 합=" + sum);
    }
}
```


실습

```
두 수를 입력하세요
57
12
더 큰수는 57
```

```
두 수를 입력하세요
75
3
나머지 연산 결과 : 75%3 = 0
두 수를 입력하세요
13
8
나머지 연산 결과 : 13%8 = 5
두 수를 입력하세요
0
0
계속하려면 아무 키나 누르십시오 . . .
```

■ 메서드 만들기

■ 1. 두 수 중 더 큰 수를 구하는 메서드

- 매개변수로 두 수를 받아서 두 수 중 더 큰 수를 구한 후 그 결과값을 리턴해 주는 메서드
- main() 메서드에서는 사용자로부터 2개의 숫자를 입력 받아서 메서드를 호출하고, 그 결과를 출력

■ 2. 두 정수의 나머지를 구하는 메서드

- 매개변수로 두 정수를 받아서 나머지 연산을 한 후 그 결과값을 리턴해 주는 메서드를 정의
- main() 메서드에서는 사용자로부터 2개의 숫자를 입력 받아서 메서드를 호출하고, 그 결과를 출력
- 이러한 작업은 사용자가 0 을 1개 입력할 때까지 계속 반복되도록 한다

실습

to cm

to Inch

3. inch를 cm로 변환하는 메서드, cm를 inch로 변환하는 메서드

- inch 단위의 데이터를 매개변수(float)로 받아서, cm 단위의 데이터로 변환하여 그 결과값을 리턴해주는 메서드와 cm 단위의 데이터를 inch 단위의 데이터로 변환하여 그 결과값을 리턴해주는 메서드를 정의
- 1 inch = 2.54 cm
- main()에서는 변환하려는 값을 사용자로부터 입력 받아 매개변수로 넘겨주고, 그 결과값을 출력

```
변환하려는 inch를 입력하세요
2
2.0inch => 5.08cm
변환하려는 cm를 입력하세요
10
10.0cm => 3.937008inch
```

4. 삼각형의 면적을 구하는 메서드 만들기

- 매개변수로 밑변의 길이, 높이를 받기
- 면적을 구한 후 리턴해주기
 - 면적 = 밑변길이 * 높이 / 2
- main()에서 밑변의 길이와 높이를 매개변수로 넘겨주고, 메서드 호출하여 삼각형의 면적을 구한 후, 화면 출력

```
삼각형의 밑변의 길이, 높이를 입력하세요.
20
7
삼각형의 면적 : 70
```



실습

- 1. 윤년인지 평년인지 구하는 메서드 만들기
 - main()에서 년도를 매개변수로 넘겨주고, 메서드 호출하여 윤년인지 평년인지 구한 후, 화면 출력
 - 윤년
 - 1) 400으로 나누어 떨어지면 윤년
 - 또는
 - 2) 4로 나누어 떨어지고, 100으로 나누어 떨어지지 않으면 윤년
- 2. 홀수인지 짝수인지 구하는 메서드 만들기
- 3. 주민번호의 성별 자리를 이용하여 남자인지, 여자인지 성별 구하는 메서드
 - 1 이나 3 이면 남자,
 - 2 나 4 이면 여자



실습

```
0~9 사이의 숫자를 입력하세요 : 7
7=>7
7* 100 = 700
```

- 문자 '0'이 전달되면 정수 0을, 문자 '7'이 전달되면 정수 7을 반환하는 메서드 만들기
 - 즉, 문자를 정수로 변환하는 메서드
 - 이름 : convToInt()
 - 0 이상 9 이하의 정수만 입력된다고 가정하자
- 입력에 따른 출력을 다음과 같이 보이는 메서드 만들기

```
반복 횟수를 입력하세요 : 5
B
AB
AAB
AAAB
AAAAAB
```

두 수를 입력 받아서, 두 수 사이의 홀수의 합을
구하여 리턴하는 메서드 만들기

실습

- 판매수량이 200 이상이면 판매금액의 30%, 100개 이상이면 20%, 100 미만이면 10% 로 성과급을 계산하는 메서드 만들기

```
판매수량, 판매금액을 입력하세요  
120  
250000  
성과급 : 50000.0
```

- 입력 받은 문자열이 숫자인지를 판별하는 메서드 만들기
 - 반복문과 `charAt(int i)`를 이용해서 문자열의 문자를 하나씩 읽어서 검사한다.

```
값을 입력하세요  
123A7  
123A7는 숫자가 아닙니다.
```

```
값을 입력하세요  
12345  
12345는 숫자입니다.
```



예제-숫자 맞추기 게임

- 컴퓨터는 1~100 중 random하게 하나의 숫자를 뽑는다 => 메서드로 만들기
- 사용자가 숫자를 입력
 - 맞추면 “정답입니다”
 - 틀리면 “좀 더 작은 수를 입력하세요”, “큰 수를 입력하세요”
- 입력기회를 4회로 제한

```
//랜덤값을 뽑는 메서드를 만든다.  
int num=(int)(Math.random()*100+1);  
//1~100 중에서 뽑는다
```

Math 클래스 (java.lang.Math)

■ Math.random() 메서드

- 0.0 ~ 1.0 범위의 임의의 double 값을 반환함
- 0.0 은 범위에 포함, 1.0 은 포함되지 않음
- $0.0 \leq x < 1.0$

1. 발생시키려는 수의 개수를 각 변에 곱한다
2. 시작값을 더한다
3. int로 형변환한다

public static double **random()**

- 예) 1과 10 사이의 임의의 정수 구하기
- 1. 각 변에 10을 곱한다

```
0.0*10 <= Math.random() * 10 < 1.0*10  
0.0 <= Math.random() * 10 < 10.0
```

- 2. 각 변에 1을 더한다

```
0.0+1 <= Math.random() * 10 + 1 < 10.0+1  
1.0 <= Math.random() * 10 + 1 < 11.0
```

- 3. 각 변을 int형으로 형변환한다

```
(int)1.0 <= (int)(Math.random() * 10+1) < (int)11.0  
1 <= (int)(Math.random() * 10+1) < 11
```

1 부터 100 사이의 숫자를 입력하세요

55

너무 큼니다. 작은 수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

45

너무 큼니다. 작은 수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

32

좀 더 큰수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

35

정답입니다

1 부터 100 사이의 숫자를 입력하세요

52

좀 더 큰수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

68

너무 큼니다. 작은 수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

57

너무 큼니다. 작은 수를 입력하세요

1 부터 100 사이의 숫자를 입력하세요

54

실패!! 다음 기회에!!



예제

```
import java.util.*;
class CompareNumber
{
    public static void main(String[] args) {
        //랜덤값을 뽑는다.
        int answer= rnd();
        Scanner sc = new Scanner(System.in);

        int i=0;
        while(i<4)
        {
            System.out.print("1 부터 100 사이의 숫자를 입력하세요 : ");
            int temp=sc.nextInt();
            if (temp == answer)
            {
                System.out.println("정답입니다");
                break;
            }
            else if(temp > answer)
            {
                System.out.println("너무 큼니다. 작은 수를 입력하세요\n");
            }
        }
    }
}
```



예제

```
        else
        {
            System.out.println("좀 더 큰 수를 입력하세요\n");
        }//if
        i++;
    }//while
}
static int rnd() //랜덤값을 뽑는 메서드를 만듦
{
    int num = (int)(Math.random()*100+1); //1~100까지 뽑는다
    return num;
}//rnd()
}//class
```



예제-보완

```
import java.util.*;
```

```
class CompareNumber3
```

```
{
```

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

```
    {
```

```
        //랜덤값을 뽑는다.
```

```
        int answer, i=0;
```

```
        answer = rnd();
```

```
        Scanner sc = new Scanner(System.in);
```

```
        while(i<4)
```

```
        {
```

```
            System.out.println("1 부터 100사이의 숫자를 입력하세요");
```

```
            int temp=sc.nextInt();
```

```
            String result="";
```

```
            if (temp == answer)
```

```
            {
```

```
                System.out.println("정답입니다");
```

```
                break;
```

```
            }
```



예제-보완

```
        if (i==3)
        {
            result = "실패!! 다음 기회에!! 정답 : " + answer;
        }
        else
        {
            if(temp > answer)
            {
                result = "너무 큼니다 작은 수를 입력하세요Wn";
            }
            else
            {
                result = "좀더 큰 수를 입력하세요Wn";
            }
        }
        System.out.println(result);
        i++;
    }
}

public static int rnd() //랜덤값을 뽑는 메서드를 만듬
{
    int num = (int)(Math.random()*100+1); //1-100까지 뽑는다
    return num;
}
}

}
}
```

오버로드된 메서드 선언(Overloading Method)

- 클래스 안에서 메서드 이름 공유
 - 매개변수의 **개수**나 매개변수의 **자료형**(type)이 **다를** 경우

```
class OverloadTest
{
    static int add(int a, int b)
    {
        return a + b;
    }
    static int add(int a, int b, int c)
    {
        return a + b + c;
    }
    static void main( )
    {
        System.out.println(add(1,2) + add(1,2,3));
    }
}
```





오버로딩(Overloading)

■ 오버로딩

- 하나의 클래스내에서 동일한 이름을 가진 여러 개의 메서드
- 메서드 이름이 동일하더라도 매개변수의 개수나 매개변수의 자료형(type)이 다를 경우, 별개의 메서드로 인정하는 것
- 하나의 이름으로 비슷비슷한 기능을 중복 정의하는 것
- 리턴타입(Return type)으로는 구별하지 않음
- 각 자료형에 따라 다른 처리를 해야 할 필요가 있을 때 사용
 - 받아들이는 매개변수의 타입은 다르지만 하는 작업은 동일할 때 사용



오버로딩

- `void func(int a)`
- `void func(double a)`
- `void func(int a, int b)`
- `void func(int a, string b)`

예제

```
public class OverloadCalc {
    public static int Plus(int a, int b) {
        return a+b;
    }
    public static float Plus(float a, float b) {
        return a+b;
    }
    public static double Plus(double a, double b) {
        return a+b;
    }
    public static String Plus(String a, String b) {
        return a+b;
    }
    public static void main(String[] args) {
        int i=Plus(3, 5);
        float j=Plus(0.1f, 0.2f);
        double k=Plus(0.5, 0.7);
        String l = Plus("Hello", "java");
        System.out.println("int 합:" + i);
        System.out.println("float 합:" + j);
        System.out.println("double 합:" + k);
        System.out.println("string 합:" + l);
    }
}
```

```
public static String Plus(int a, int b) {
    String s= Integer.toString(a+b);
    return s;
}
//오버로딩 아님-리턴타입으로 구별 안함
```

```
int 합:8
float 합:0.3
double 합:1.2
string 합:Hellojava
```




예제

```
Java으로 검색, 결과 : 클릭하세요 Java  
92123456으로 검색, 결과 : 자바 프로그래밍  
홍길동,2009년,3월로 검색, 결과 : JSP
```

```
class OverloadingMethod2  
{  
    public static String findBook(String name)  
    {  
        String result = name + "으로 검색, ";  
        result += "결과 : 클릭하세요 Java";  
        return result;  
    }  
    public static String findBook(int isbn)  
    {  
        String result = isbn + "으로 검색, ";  
        result += "결과 : 자바 프로그래밍";  
        return result;  
    }  
    public static String findBook(String author, int year, int month)  
    {  
        String result = author + ", " + year + "년," + month + "월로 검색, ";  
        result += "결과 : JSP";  
        return result;  
    }  
}
```



예제

```
public static void main(String[] args)
{
    String book = findBook("Java");
    System.out.println(book);

    book = findBook(92123456);
    System.out.println(book);

    book = findBook("홍길동", 2009, 3);
    System.out.println(book);
}
}
```



실습

```
가로, 세로, 반지름을 입력하세요.  
10  
20  
5  
사각형의 면적 : 200  
원의 면적 : 78.5
```

- 원의 면적, 사각형의 면적을 구하는 메서드를 오버로딩 메서드로 만들기
 - 원의 면적 : 반지름 * 반지름 * 3.14
 - 사각형의 면적 : 가로 * 세로
- main()에서 사용자로부터 가로, 세로, 반지름을 입력 받고, 각 메서드를 호출하여 면적을 구한 후, 화면 출력하기



실습

- 원의 면적을 구하는 메서드 만들기
 - 원의 면적 : 반지름 * 반지름 * 3.14
- main()에서 사용자로부터 반지름을 입력 받고, 메서드를 호출하여 면적을 구한 후, 화면 출력하기

반지름을 입력하세요.

17

원의 면적 : **907.46**



재귀 호출(recursive call)

- 재귀 호출
 - 메서드의 내부에서 메서드 자기 자신을 다시 호출하는 것
 - 반복적인 작업을 해야 하는 메서드에 반복문 대신 재귀 호출을 이용하면, 메서드를 훨씬 간단하게 할 수 있는 경우가 있음
 - 재귀 호출은 다소 효율이 떨어진다는 단점이 있음
 - 재귀 호출은 반복적으로 메서드를 호출하는 것이기 때문에 메서드를 호출하는데 드는 비용이 추가적으로 발생하기 때문



재귀 호출(recursive call)

- 대표적인 재귀 호출의 예 : 팩토리얼(factorial)을 구하는 것
 - 팩토리얼 : 한 숫자가 1이 될 때까지 1씩 감소시켜가면서 계속 해서 곱해 나가는 것
 - $n!$ 로 표현
 - 예) $5! = 5*4*3*2*1$
 - 팩토리얼을 수학적 함수로 표현하면
 - $f(n) = n * f(n-1)$, 단 $f(1)=1$

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

$$4! = 4 * 3!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1$$

$$3! = 3 * 2 * 1$$

예제

팩토리얼을 계산하는 메서드를 구현
-재귀호출은 반복문이 무한반복에 빠지는 것처럼
무한호출이 되기 쉬우므로 주의

24

```
class FactorialTest {  
    public static void main(String args[]) {  
        System.out.println(factorial(4)); // FactorialTest.factorial(4)  
    }  
  
    public static long factorial(int n) {  
        long result=0;  
  
        if ( n == 1) {  
            result = 1;  
        } else {  
            result = n * factorial(n-1);    // 다시 메서드 자신을 호출한다.  
        }  
  
        return result;  
    }  
}
```

• 삼항 연산자를 이용하면 **factorial()** 메서드를 다음과 같이 더욱 간결하게 표현할 수 있음

```
public static long factorial(int n){  
    return (n==1)?1:n*factorial(n-1);  
}
```



예제

```
Exception in thread "main" java.lang.StackOverflowError
  at MainTest.main(MainTest.java:3)
  at MainTest.main(MainTest.java:3)
  at MainTest.main(MainTest.java:3)
```

```
class MainTest {
    public static void main(String args[]) {
        main(null);        // 자기 자신을 다시 호출한다.
    }
}
```

- **main** 메서드도 자기 자신을 호출하는 것이 가능하며, 아무 조건도 없이 계속해서 자기 자신을 다시 호출하기 때문에 무한호출에 빠지게 됨
- **main** 메서드가 종료되지 않고 호출스택에 계속해서 쌓이게 되므로 결국 호출스택의 메모리 한계를 넘게 되고 **StackOverflowError**가 발생하여 프로그램은 비정상적으로 종료됨

실습

- 피보나치 수열의 값을 계산해서 반환하는 메서드 정의하기
 - 피보나치 수열은 0과 1에서부터 시작함
 - 앞에 있는 두 개의 값을 더하는 형태로 전개됨
 - 0, 1, 1, 2, 3, 5, 8, 13
 - 첫 번째 피보나치 수열의 값은 0 이고, 두 번째와 세 번째 피보나치 수열의 값은 1 임

fib(n) = 0	n=0
1	n=1
fib(n-2) + fib(n-1)	n=2 이상

fib(0) + fib(1)	n=2
fib(1) + fib(2)	n=3
fib(2) + fib(3)	n=4
fib(3) + fib(4)	n=5

실습

