

JAVA

- 메서드

메서드와 변수의 스코프

메서드에 대한 이해와 메서드의 정의

main 메서드, 아는 것과 모르는 것

아는 것

- 메서드 이름은 main
- 메서드 중괄호 내에 존재하는 문장들이 위에서 아래로 순차적 실행

모르는 것

- public, static, void
- 왜? 항상 main인가?
- String[] args

```
public static void main(String[] args)
{
    int num1=5, num2=7;
    System.out.println("5+7="+ (num1+num2));
}
```

자바 프로그램의 시작은 main이라는 이름의 메서드를 실행하는 데서부터 시작한다!

메서드

▶ 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.

메서드

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;
```

```
}
```

메서드

메서드의 호출방법

참조변수.메서드 이름();

// 메서드에 선언된 매개변수가 없는 경우

참조변수.메서드 이름(값1, 값2, ...);

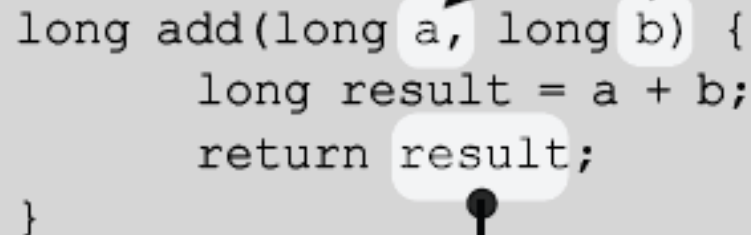
// 메서드에 선언된 매개변수가 있는 경우

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

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

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

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



다른 이름의 메서드 만들기

```
public static void main(String[ ] args)
{
    System.out.println("프로그램의 시작");
    hiEveryone(12);
    hiEveryone(13);
    System.out.println("프로그램의 끝");
}
```

1

```
public static void hiEveryone(int age)
{
    System.out.println("좋은 아침입니다.");
    System.out.println("제 나이는 .... ");
}
```

2

12

값의 전달

3

다른 이름의 메서드 만들기

```
class MethodDefAdd
{
    public static void main(String[] args)
    {
        System.out.println("프로그램의 시작");
        hiEveryone(12);
        hiEveryone(13);
        System.out.println("프로그램의 끝");
    }

    public static void hiEveryone(int age)
    {
        System.out.println("좋은 아침입니다.");
        System.out.println("제 나이는 "+ age+"세입니다.");
    }
}
```

프로그램의 시작
좋은 아침입니다.
제 나이는 12세입니다.
좋은 아침입니다.
제 나이는 13세입니다.
프로그램의 끝

매개변수가 두 개인 형태의 메서드

```
class Method2Param {  
    public static void main(String[] args) {  
        double myHeight=175.9;  
        hiEveryone(12, 12.5);  
        hiEveryone(13, myHeight);  
        byEveryone();  
    }  
  
    public static void hiEveryone(int age, double height) {  
        System.out.println("제 나이는 "+ age+"세 입니다.");  
        System.out.println("저의 키는 "+ height+"cm 입니다.");  
    }  
  
    public static void byEveryone() {  
        System.out.println("다음에 뵙겠습니다.");  
    }  
}
```

제 나이는 12세 입니다.
저의 키는 12.5cm 입니다.
제 나이는 13세 입니다.
저의 키는 175.9cm 입니다.
다음에 뵙겠습니다.

예제

문제1

두 개의 정수를 전달받아서, 두수의 사칙연산 결과를 출력하는 메서드와 이 메서드를 호출하는 main메서드를 정의해보자.

단, 나눗셈은 몫과 나머지를 각각 출력해야 한다.

문제2.

두 개의 정수를 전달 받아서, 두수의 절대값을 계산하여 출력하는 메서드와 이 메서드를 호출하는 main메서드를 정의해 보자. 단 메서드 호출 시 전달되는 값의 순서에 상관없이 절대값이 계산되어서 출력되어야 한다.

return

▶ 메서드가 정상적으로 종료되는 경우

- 메서드의 블록{}의 끝에 도달했을 때
- 메서드의 블록{}을 수행 도중 return문을 만났을 때

▶ return문

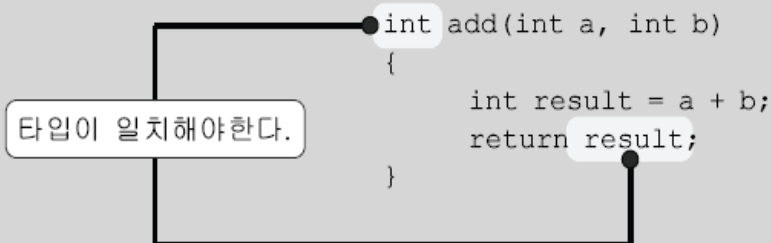
- 현재 실행 중인 메서드를 종료하고 호출한 메서드로 되돌아간다.

1. 반환값이 없는 경우 - return문만 써주면 된다.

```
return;
```

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

```
return 반환값;
```



return

- ▶ 반환 값이 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;  
}
```

메서드

```
class ReturnTest {  
    public static void main(String[] args) {  
        ReturnTest r = new ReturnTest();  
  
        int result = r.add(3,5);  
        System.out.println(result);  
  
        int[] result2 = {0};  
        r.add(3,5,result2);  
        System.out.println(result2[0]);  
    }  
  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    void add(int a, int b, int[] result) {  
        result[0] = a + b;  
    }  
}
```

값을 반환하는 메서드

```
class MethodReturns
```

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

```
        int result=add(4, 5);  
        System.out.println("4와 5의 합: " + result);  
        System.out.println("3.5의 제곱: " + square(3.5));  
    }
```

```
    public static int adder(int num1, int num2)  
    {  
        int addResult=num1+num2;  
        return addResult;  
    }
```

```
    public static double square(double num)  
    {  
        return num*num;  
    }
```

```
}
```

4와 5의 합 : 9

3.5의 제곱 : 12.25

int result = adder(4, 5) ;



int result = 9 ;

키워드 return이 지니는 두 가지 의미

```
class OnlyExitReturn
```

```
{
```

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

```
    {
```

```
        divide(4, 2);
```

```
        divide(6, 2);
```

```
        divide(9, 0);
```

```
    }
```

```
    public static void divide(int num1, int num2)
```

```
    {
```

```
        if(num2==0)
```

```
        {
```

```
            System.out.println("0으로는 값을 나눌 수 없습니다.");
```

```
            return;
```

```
        }
```

```
        System.out.println("나눗셈 결과: " + (num1/num2));
```

```
    }
```

```
}
```

**값의 반환, 메서드의 종료,
이렇게 두 가지의 의미를 지님**

나눗셈 결과 : 2

나눗셈 결과 : 3

0으로는 값을 나눌 수 없습니다.

문제3

원의 반지름 정보를 전달하면, 원의 넓이를 계산해서 반환하는 메서드와 원의 둘레를 계산해서 반환하는 메서드를 각각 정의하고, 이를 호출하는 main 메서드를 정의하자. $(2*\pi*r)$, $(\pi*r^2)$

문제4.

전달되는 값이 소수(prime number)인지 아닌지를 판단하여, 소수인 경우 true를, 소수가 아닌 경우 false를 반환하는 메서드를 정의하고, 이를 이용해서 1이상 100 이하의 소수를 전부 출력할 수 있도록 main 메서드를 정의하자.

소수 : 1과 자기 자신만으로 나누어 떨어지는 1보다 큰 양의 정수

변수의 스코프

가시성: 여기서는 저변수가 보여요.

```
class LocalVariable
{
    public static void main(String[] args)
    {
        boolean scope=true;
        if(scope)
        {
            int num=1;
            num++;
            System.out.println(num);
        }
        else
        {
            int num=2;
            System.out.println(num);
        }
        simple();
    }
    public static void simple()
    {
        int num=3;
        System.out.println(num);
    }
}
```

```
for( int num=0 ; num<5; num++)
{
    /* 추가적인
       변수 num 선언 불가 지역 */
}
```

변수 num의
접근 가능지역

```
public static void myFunc( int num )
{
    /* 추가적인
       변수 num 선언 불가 지역 */
}
```

변수 num의
접근 가능지역

“지역변수”

**선언된 지역을 벗어나면 변수는
자동 소멸된다.**

자바의 이름 규칙 (Naming Rule)

클래스, 메서드, 상수의 이름 규칙

클래스 이름

- `class MyClass`
- `class ProgrammingBook`

Camel Case

- 대문자로 시작
- 둘 이상의 단어가 묶여서 하나의 이름 구성 시, 새 단어는 대문자로 시작

인스턴스 변수, 메소드 이름

- `int addYourMoney(int money)`
- `int yourAge`

변형된 Camel Case

- 소문자로 시작
- 둘 이상의 단어가 묶여서 하나의 이름 구성 시, 새 단어는 대문자로 시작

상수 이름

- `final int COLOR=7`
- `final int COLOR_RAINBOW=7`

- 전부 대문자로 표현
- 둘 이상의 단어가 묶여서 하나의 이름 구성 시, 두 단어 사이에 `_` 삽입