

제어문과 메서드

백석대학교 강윤희

메서드

■ 개요

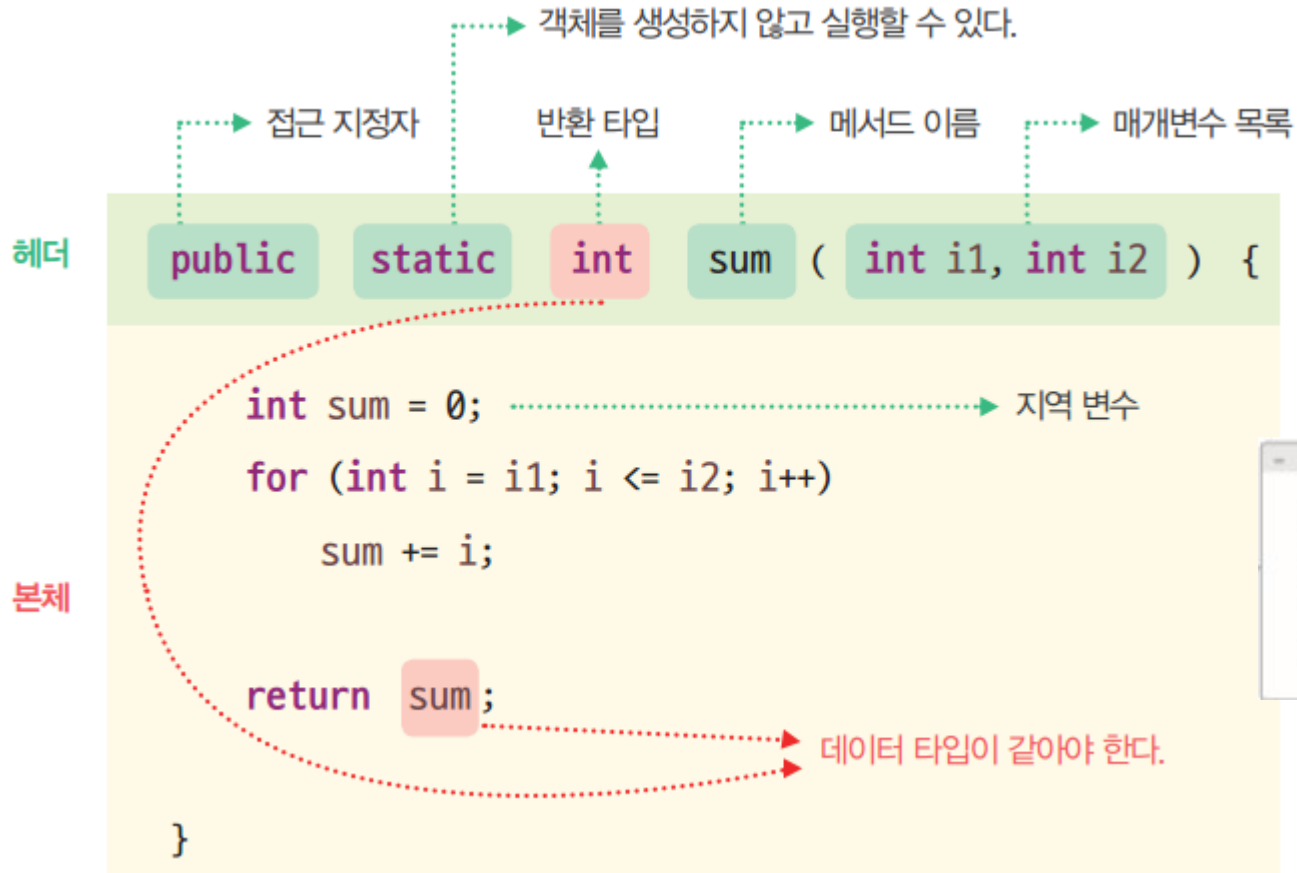
- 특정 연산을 수행하는 실행문 블록
- 함수 또는 프로시저라고 함
- 자바의 메소드는 클래스 내부에서 만 정의됨

■ 메서드를 이용하면 얻을 수 있는 장점

- 중복 코드를 줄이고 코드를 재사용(reuse)할 수 있음
- 코드를 모듈화하여 가독성을 높이므로 프로그램의 품질을 향상시킴

메서드

■ 메서드의 구조



```
합(1~10) : 55
합(10~100) : 5005
합(100~1000) : 495550
```

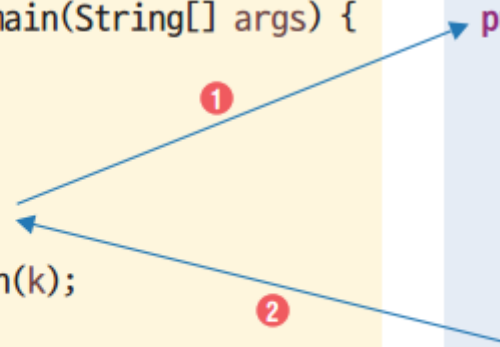
메서드

■ 메서드의 호출과 반환

- 메서드를 호출하면 제어가 호출된 메서드(callee)로 넘어갔다가 호출된 메서드의 실행을 마친 후 호출한 메서드(caller)로 다시 돌아온다
- 단, return 문을 사용하면 다음과 같이 메서드의 실행 도중에도 호출한 메서드로 제어를 넘길 수 있다

```
public static void main(String[] args) {  
    int i = 1, j = 10;  
  
    int k = sum(i, j);  
    system.out.println(k);  
}
```

```
public static int sum(int i1, int i2) {  
    int sum = 0;  
    for (int i = i1; i <= i2; i++)  
        sum += i;  
  
    return sum;  
}
```



메서드

■ 메서드의 호출과 반환

```
public static void main(String[] args) {  
    printScore(99);  
    printScore(120);  
}
```

```
public static void printScore(int score) {  
    if (score <= 0 || score >= 100) {  
        System.out.println("잘못된 점수 : " + score);  
        return;  
    }  
    System.out.println("점수 : " + score);  
}
```



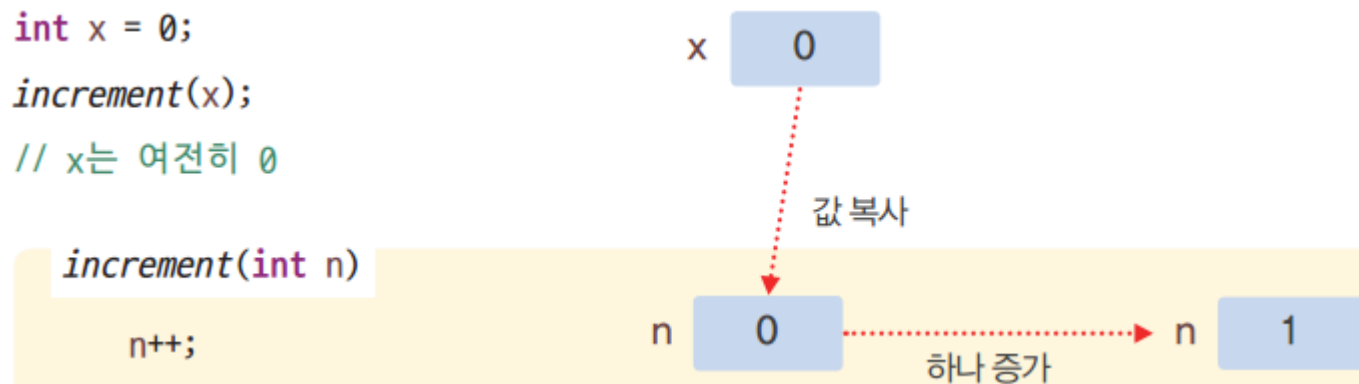
메서드

■ 값 전달(call by value)

```
1
2 public class IncrementDemo {
3     public static void main(String[] args) {
4         int x = 0;
5         System.out.println("increment() 메서드를 호출하기 전의 x는 " + x);
6         increment(x);
7         System.out.println("increment() 메서드를 호출한 후의 x는 " + x);
8     }
9
10    public static void increment(int n) {
11        System.out.println("increment() 메서드를 시작할 때의 n은 " + n);
12        n++;
13        System.out.println("increment() 메서드가 끝날 때의 n은 " + n);
14    }
15 }
16
```

increment() 메서드를 호출하기 전의 x는 0
increment() 메서드를 시작할 때의 n은 0
increment() 메서드가 끝날 때의 n은 1
increment() 메서드를 호출한 후의 x는 0

```
int x = 0;
increment(x);
// x는 여전히 0
```

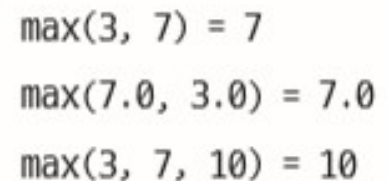


메서드

■ 메서드 오버로딩

- 메서드 시그니처(Method Signature) : 메서드 이름과 매개변수 개수, 데이터 타입, 순서를 의미
- 메서드 이름은 같지만 메서드 시그니처가 다른 메서드를 정의하는 것을 메서드 오버로딩(Method Overloading)이라고 함

```
1 public class OverloadDemo {  
2     public static void main(String[] args) {  
3         int i1 = 3, i2 = 7, i3 = 10;  
4         double d1 = 7.0, d2 = 3.0;  
5  
6         System.out.printf("max(%d, %d) = %d\n", i1, i2, max(i1, i2));  
7         System.out.printf("max(%.1f, %.1f) = %.1f\n", d1, d2, max(d1, d2));  
8         System.out.printf("max(%d, %d, %d) = %d\n", i1, i2, i3, max(i1, i2, i3));  
9     }  
10  
11     public static int max(int n1, int n2) {  
12         int result = n1 > n2 ? n1 : n2;  
13         return result;  
14     }  
15  
16     public static double max(double n1, double n2) {  
17         double result = n1 > n2 ? n1 : n2;  
18         return result;  
19     }  
20  
21     public static int max(int n1, int n2, int n3) {  
22         return max(max(n1, n2), n3);  
23     }  
24 }
```



```
max(3, 7) = 7  
max(7.0, 3.0) = 7.0  
max(3, 7, 10) = 10
```



Q & A