07장. 메소드

1. 메소드

클래스의 주요 요소 중 하나가 메소드이다. 메소드는 클래스가 가진 기능을 정의하는 것으로 여러가지 명령을 하나의 이름으로 정의하여 호출되어지는 명령의 실행 구조이다. 메소드는 멤버 변수(필드, 속성)과 함께 클래스의 멤버로 꼽히는 요소로 클래스를 인스턴스화하여 만든 개체의 기능으로 개체를 사용하는 방법이라 할 수 있다. 구체적으로는 입력, 처리, 출력 (Input / Process / Output)를 작성한다.

메소드의 입력과 출력은 **메소드 인수**와 **반환 값**이라고 한다. 메소드 선언 부분과 본체로 나눌 수 있다. 선언 부분은 형식의 "반환 형식 메소드 이름 (인수 형 인수 이름)"에 해당하는 부분으로 그 메소드에 대한 다른 구성 요소와의 관련 특성을 결정한다. 본체는 형식의 '{메소드 본체}'에 해당하는 부분으로 그 메소드의 실제 명령을 기술한다.

[메소드 선언 형식]

```
[한정자] [반환 값 형] 메소드 이름 (<인수 목록>) {
 명령 ;
}
```

- 한정자는 필요에 따라 사용할 수 있다.
- 반환 데이터 형식은 해당 메소드가 종료 될 때 반환 값의 데이터 형식이다.
- 메소드 이름과 인수 목록을 맞게 시그니처라고 한다. 이 메소드를 식별한다.

한정자는 메소드의 선언 부분에 여러 가지 한정자를 부여하고 방법을 가진 다른 구성 요소와 관련된 속성을 지정할 수 있다. 한정자는 메소드 뿐만 아니라 필드 나 클래스에 붙이는 것으로, 그 메소드가 어디에서 액세스 할 수 있는지를 나타내는 액세스 한정자라는 것과 다른 수식자가 존재한다.

다음은 메소드앞에 선언될 한정자의 접근에 대한 액세스 수준과 종류이다.

액세스 수준	동일 클래스	동일 패키지	서브 클래스	모든
public	○	○	○	○
protected	○	○	○	×
지정 없음 (기본)	○	○	×	×
private	○	×	×	×

액세스 제한을 기술하는 수식자는 public , protected , private 이며 액세스 제한적인 것부터, private> default(생략)> protected> public 이다.

메소드에 한정자와 함께 사용되는 수식자는 다음과 같다.

수식	설명
액세스 수준	액세스 수준을 지정하여 메소드를 다른 클래스에서 참조 할 수없는 등을 지정할 수 있다.
static	메소드가 인스턴스 메소드가 아닌 클래스 메소드임을 나타낸다.
abstract	처리를 구현하지 않는 메소드 (추상 메소드)을 생성하고 싶을 때 사용한다.
final	final 을 사용하면 그 메소드는 서브 클래스에서 재정의 할 수 없다.
native	C 언어 등 다른 언어로 작성된 프로그램을 Java 프로그램에서 사용하고 싶을 때 사용한다. 사용하려는 프로그램에 native 를 붙여 선언한다.
synchronized	스레드를 동시에 여러 달리게 때 동일한 데이터에 대해 여러 메소드가 호출 될 수 있다. . 그런 경우 안전하게 메소드가 실행되기 위하여 synchronized 를 사용한다.
반환형	메소드를 실행 한 결과를 메소드 호출자에게 반환 할 수 있다.여기에서는 반환 값의 데이터 형식을 지정한다. 반환 값이없는 경우는 void 형을 사용한다. 데이터 타입은 선언시 필수 항목이다.
메소드 이름	메소드 이름을 선언한다. 메소드 이름은 관례 적으로 소문자로 시작 단어 구분은 대문자를 사용한다. 또한 메소드 이름으로 동사가 자주 사용된다. 메소드 이름은 선언시 필수 항목이다.
(인수 형 인수 이름)	메소드에 값을 전달하고 싶을 때, 전달하려는 값의 인수 형 인수 이름을 선언하며 전달하려는 값이없는 경우는 () 만 기재한다. (인수 형 인수 이름) 선언시 필수 항목이다.
throws 예외 이름	메소드에서 어떤 예외를 던질 때 사용한다.

다음은 메소드의 선언 예이다.

static boolean methodA () { -→반환 형식으로 boolean 형을 돌려주는 클래스 메소드 methodA 선언

return true; 또는 return false;

}

public void methodB (double d1, double d2) {

//명령

}

---→ 액세스 수준이 public (모든 클래스에서 참조 할 수있다)에서 double 형의 인수 d1, d2 를 가지고 리턴 타입이없는 (void) 메소드 methodB 를 선언

abstract double methodC (double d);

➔ 메소드 본문 구현을 가지지 않는 추상 메소드 (abstract)에서 리턴 타입이 double 형 인수에 double 형의 인수 d 를 가지고 메소드 methodC 을 선언

메소드는 클래스 안에 기술한다. .프로그램이 실행될 때 처음으로 호출되는 메소드 인 main 메소드가 클래스에 이미 기술되어있는 것을 볼 수 있다. 새롭게 정의하는 메소드도 클래스 "{"에서 "}"에서 정의한다. 이때 메소드가 선언하는 순서는 중요하지 않다.

2. 클래스 메소드의 void형 메소드

리턴 값이 없는 메소드를 정의할 때 사용하는 void형 메소드는 명령 구문에 return 키워드를 이용하여 값을 리턴 하는 구조가 아닌 명령을 수행하는 코드로만 작성되어 있다.

메소드의 사용 방법은 다음과 같이 "."(도트 연산자)를 사용합니다.
메소드의 사용은 다음과 같다.

클래스 . 메소드 (인수 목록);

메소드를 선언하고 호출해 보자.

```
public class MethodTest01 {
        public static void main(String[] args) {
            System.out.println ( "안녕하세요");
            test (); //---------   MethodTest01.test()라고 호출해되 된다.
                                   //같은 클래스에 정의되어 메소드 이름만 호출 해도 된다.
            System.out.println ( "안녕");
          }
          private static void test () {
            System.out.println ( "어떻게 지내세요");
          }
        }
```

[실행결과]
안녕하세요
어떻게 지내세요
안녕

프로그램의 흐름은 다음과 같다.

```
프로그램 실행 시작되면 main 메소드의 첫 문장에서 실행되는  "안녕하세요"가 화면에 출력된다
test 메소드가 호출되고 test 메소드의 첫 문장에서 실행되는 "어떻게 지내세요 "가 화면에 출력된다
test 메소드 블록의 끝까지 도달하게 되면 test 메소드를 호출 한 다음 문장에서 실행되는
"안녕"이 화면에 출력되고  main 메소드 블록의 끝까지 도달한 후  프로그램이  종료된다.
```

메소드의 호출은 main 메소드에서도 할 수 있으며, 메소드에서 또 다른 메소드를 호출 할 수 있다.

매개 인자가 있는 즉, 인수가 있는 void형 메소드를 호출할 때는 매개 인수의 데이터 타입에 따른 값을 지정하면서 메소드를 호출한다.

숫자를 입력 받아 1에서 원하는 수만큼 출력하고 합을 구하는 HapResult() 메소드를 만들어 보자.

```
import java.util.Scanner; //--  스캐너 클래스가 있는 팩키지를 임포트 한다

public class MethodTest02 {
        public static void main (String args []) {
                System.out.print("Input su: ");
                    Scanner   sc=new Scanner(System.in);
                    int su= sc.nextInt();
                HapResult(su);//--  입력한 값을 대입한 su변수를 호출하면 입력한 값이
리턴되어  메소드에 대입된다.
                    }
                    public static void HapResult(int su){// su=10이 되입 된다.
                        int sum=0;
                            for(int i=1;i<=su;i++)   { // 1에서 10까지의 반복을 한다
                                    sum+= i; // 증가된 값이 대입되며 합을
구한다.
                        }
                        System.out.println("   sum="+ sum);
                    }
}
```

[실행결과]

Input su : 10

 sum =55

[참고]

java.util.Scanner : java.util 팩키지에 있는 스캐너 클래스는 자바에서 콘솔 또는 파일로 값을 입력 받을 수 있는 메소드를 제공하는 클래스이다.

사용하는 방법은 먼저 클래스 파일 선언부에 import java.util.Scanner; 이라고 선언하게

메소드 안에 코드를 구현할 때 Scanner sc=new Scanner(System.in); 이라고 객체를 생성한다.

System.in은 콘솔로 입력하는 입력장치를 의미하며 생성된 객체를 sc의 주소 번지가 받아 입력받는 값에 따른 자료형에 매핑된 메소드로 값을 읽어와 리턴해 준다.

Scanner 클래스는 입력 값(문자열, 파일, 입력 스트림)을 정규 표현식으로 구분하여 문자열이나 기본 데이터 타입으로 토큰 할 수 있는 클래스이다.

주요 메소드는 다음과 같다.

next() : 문자열을 입력받아 리턴
nextInt():정수를 입력받아 리턴
nextDouble(): 실수를 입력 받아 리턴
nextLine():문자열을 라인단위로 입력받아 리턴

3. 클래스 메소드의 return 형 메소드

return 형 메소드를 정의하면 프로그램 안에서 호출 할 수 있다. 정의되는 메소드 명을 지정하여 호출합니다. 메소드에 값을 전달할 경우에는 괄호 안에 쉼표 (,)로 구분하여 나열한다. 형식은 다음과 같다. .

```
변수 = 메소드 이름 (값 1, 값 2, ....);
```

반환 값으로 값이 호출자에게 반환 될 때 메소드를 호출 한 문장 자체가 반환 값으로 리턴된다. 따라서 출력하거나 변수에 할당 할 수 있다

```
public static void main (String args []) {
  / * 반환 값을 직접 출력하는 경우 * /
  System.out.println (test ()); --------→ 10 이 리턴 되어 출력

  / * 반환 값을 변수에 할당하는 경우 * /
  int num = test ();  -------→10 이 리턴 되어 num 에 대입
}

private static int test () {
  return 10;
}
```

다음과 같이 숫자를 넘기면 *2의 결과를 리턴 하는 메소드를 만들어 호출해 보자.

```
public class MethodTest03 {
        public static void main(String[] args) {
            int Res;

            Res = getResult(9);
```

```
            System.out.println (Res);

        Res = getResult(5);
            System.out.println (Res);
        }

        private static int getResult(int n) {
            return n * 2;
        }
    }
```

[실행 결과]

18

10

숫자를 입력 받아 "홀수 " 인지 "짝수"인지를 구현하는 메소드를 작성해 보자.

```
public class MethodTest04 {
    public static void main(String[] args) {
        int num;
        String res;

        num = 9;
        res = getRes(num);
        System.out.println(num + "는" + res);

        num = 6;
        res = getRes(num);
        System.out.println(num + "는" + res);
    }

    private static String getRes(int n) {
        if (n % 2 == 0) {
            return "짝수";
        } else {
            return "홀수";
        }
    }
```

}
[실행 결과]
9 는 홀수
6 는 짝수

가변인수를 받아 처리할 수 있습니다. 점 세개 "..."를 이용해서 매개인수 앞에 선언되면 메소드를 호출 할 때 인수를 임의의 수만큼 대입하여 호출합니다.

예를 들어 다음과 같이 사용합니다.

```
public static void main (String args []) {
  disp (10, 7, 8); -----→ 10 7 8 이 대입되어 출력된다.
}

private static void disp (int ... num) {
  for (int i = 0; i <num.length; i + +) {
    System.out.println (num [i]);
  }
}
```

메소드를 호출 할 때 3 개의 인수를 지정하고 있다. 메소드를 받는 측에서는 배열의 요소에 순서대로 인수가 포함 된 것처럼 취급하기 때문에 메소드 내에서 요소를 참조하여 인수를 사용할 수 있게 된다.

또한 인수의 수는 선택 사항이지만 인수의 데이터 형은 동일해야 한다. 또한 인수는 몇 개 건네오는지는 메소드가 불릴 때까지 모르기 때문에 인수의 수를 알기 위해서는 "배열 변수 이름 .length"를 사용하여 배열의 길이를 리터 받아 사용한다. 또한 가변 인수의 경우 인수는 0 이어도 상관 없다.

가변 인수를 이용하여 대입하는 인수 값 만큼 합을 구하여 리턴하는 메소드를 작성해 보자.

```
public class MethodTest05 {
        public static void main(String[] args) {
            System.out.println (sum (4, 10));// -------    4+7를 리턴 하는 메소드
            System.out.println (sum (7, 2, 8)); //---------    7+2+8 를 리턴 하는 메소드
            System.out.println (sum ());
          }

          private static int sum (int ... nums) {//-----    가변 인수를 사용하여 선언
```

```
        int sum = 0;
        for (int i = 0; i <nums.length; i ++) {
            sum += nums [i];
        }
        return sum;
    }
}
```

[실행결과]

14

17

0

4. 메소드 오버로딩(Method Overloading)

>>메소드 오버로딩이란?

메소드는 변수와 마찬가지로 같은 클래스 내에서 서로 구별될 수 있어야 하기 때문에 각기 다른 이름을 가져야 한다. 하지만, 자바에서는 한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메소드가 있더라도 매개변수의 개수 또는 타입이 다르면, 같은 이름을 사용해서 메소드를 정의할 수 있도록 했다.

이처럼, 한 클래스 내에 같은 이름의 메소드를 여러 개 정의하는 것을 메소드 오버로딩(Method Overloading) 또는 간단히 오버로딩(Overloading)이라 한다.

. 오버로딩의 조건

오버로딩이 성립하기 위해서는 다음과 같은 조건을 만족해야 한다.

① 메소드 이름이 같아야 한다.
② 매개변수의 개수 또는 타입이 달라야 한다.
③ 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다.
④ (리턴타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다.)

오버로딩의 장점은 같은 기능을 하는 메소드들을 효율적으로 작성 및 사용 할 수 있다는 점이다.

>> 오버로딩의 예

오버로딩의 예로 가장 대표적인 것은 println메소드다.
PrintStream클래스에는 어떤 종류의 매개변수를 지정해도 출력할 수 있도록 아래와 같이 10개의 오버로딩된 println메소드를 정의해놓고 있다.

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)
println메소드를 호출할 때 매개변수로 넘겨주는 값의 타입에 따라서 위의 오버로딩된 메소드들 중의 하나가 선택되어 실행되는 것이다.

다음과 같이 오버로드를 선언할 수 있다.

○인수의 수가 다른 경우 :

```
private static void test (int n1) { }
private static void test (int n1, int n2) {   }
```

○인수의 종류가 다른 경우 :

```
private static void test (int n1) { }
private static void test (double d1) { }
```

○인수의 데이터 형과 수는 같아도 순서가 다른 경우 :

```
private static void test (int n1, double d1) {}
private static void test (double d1, int n1) { }
```

이 중 하나의 조건 만 충족하면 동일한 메소드 이름도 같은 클래스에 정의 할 수 있다.

다음 오버로드 할 수 없는 경우이다

메소드 정의 순간 인수의 변수 이름 만 다른 같은 것은 오버로드 할 수 없다. 또한 반환 값의 데이터 형이 다른 것도 오버로드 할 수 없다.

×인수의 변수 이름 만 다른 경우 :

```
private static void test (int a) {   }
private static void test (int b) {   }
```

×반환 값 만 다른 경우 :

```
private static void test (int n) { }
private static int test (int n) {   }
```

이와 같이 변수 이름 만 다른 경우 나 반환 값 만 다르다 메소드를 선언하게 되면 컴파일 에러가 발생 "(메소드 이름)는 (클래스 이름)에 정의되어 있습니다."와 같은 오류 메시지가 표시된다.