# 제8장
# 객체와 클래스 - 3

# 1. 클래스(static)메서드와 인스턴스 메서드

▣ 클래스 메서드

- 인스턴스 생성 없이도 '클래스명.클래스메서드명()' 으로 호출이 가능
- 메서드 내에서는 인스턴스 멤버 사용이 안됨
- 같은 클래스 내라면, 클래스명은 생략해도 됨.

▣ 인스턴스 메서드

- 인스턴스를 반드시 생성한 후, '참조변수.메서드명()' 으로 호출이 가능
- 메서드 내에서 인스턴스 멤버 및 정적 멤버들 사용이 가능

(이유 : 인스턴스 메서드를 호출할 수 있을 때는, 이미 인스턴스가 만들어져 있기 때문)

# 2. 멤버간 참조와 호출

같은 클래스의 클래스 멤버간에 인스턴스 생성이나, 참조변수 없이 참조할 수 있지만,

**static멤버들은 인스턴스 멤버들을 참조할 수 없다.**

```
public class Test {

    int iv;         //인스턴스 변수
    static int cv; //클래스 변수(정적 변수)

    //인스턴스 메서드
    public void instanceMethod() {
        System.out.println(this.iv);  //인스턴스 변수 사용 가능
        System.out.println(cv);       //클래스 변수 사용 가능
        staticMethod();               //클래스 메서드 사용 가능
    }
    //클래스(정적) 메서드
    public static void staticMethod() {
        System.out.println(this.iv);  //인스턴스 변수 사용불가
        System.out.println(cv);       //클래스 변수 사용 가능
        this.instanceMethod();        //인스턴스 메서드 사용불가
    }
}
```

# 3. 메서드 오버로딩(method overloading)

▣ 오버로딩

- 하나의 클래스에 같은 이름의 메서드를 여러 개 정의하는 것을 메서드 오버로딩, 간단히 오버로딩이라고 함.

  * overload - vt. 과적하다. 부담을 많이 지우다.(ex. 멀티플레이어)

▣ 오버로딩의 조건

- 메서드의 이름이 같아야 한다.
- 매개변수의 개수 또는 타입, 순서가 달라야 한다.
- 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다. (리턴타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다.)

* 즉, 다시 말해 메서드들은 매개변수의 개수와 타입에 의해서만 구별됨

# 4. System.out.println()의 진실

## ▣ System.out.println()메서드

- 하나의 메서드 이름으로 모든 매개변수를 받아 String형태로 출력할 수 있도록 만들어짐.

| | |
|---|---|
| void | **println**()<br>Terminates the current line by writing the line separator string. |
| void | **println**(boolean x)<br>Prints a boolean and then terminate the line. |
| void | **println**(char x)<br>Prints a character and then terminate the line. |
| void | **println**(char[] x)<br>Prints an array of characters and then terminate the line. |
| void | **println**(double x)<br>Prints a double and then terminate the line. |
| void | **println**(float x)<br>Prints a float and then terminate the line. |
| void | **println**(int x)<br>Prints an integer and then terminate the line. |
| void | **println**(long x)<br>Prints a long and then terminate the line. |
| void | **println**(Object x)<br>Prints an Object and then terminate the line. |
| void | **println**(String x)<br>Prints a String and then terminate the line. |

# 5. 오버로딩의 예(사용자 정의 메서드)-1

▣ 매개변수 이름이 다르다고 오버로딩이 성립하는 것은 아님에 주목.
(데이터 타입이 달라야 함을 기억하자)

▣ 리턴타입이 다르다고 오버로딩이 성립하는 것 또한, 아님에 주목.
(리턴 타입은 오버로딩 성립조건 관여 無)

```
public class Test {

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

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

[매개변수 이름이 다를 경우]

```
public class Test {

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

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

[리턴타입이 다를 경우]

# 5. 오버로딩의 예(사용자 정의 메서드)-2

▣ 매개변수 타입이 다르므로, 오버로딩이 성립함.

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

▣ 오버로딩의 올바른 예제

매개변수는 다르지만 같은 의미의 기능을 수행함.

```
int add(int x, int y) {
    return x + y;
}
long add(long x, int y) {
    return x + y;
}
int add(int[] arr) {
    int sum = 0;
    for(int i : arr) {
        sum += i;
    }
    return sum;
}
```

# 6. 오버로딩의 장점

▣ 변수처럼 메서드도 이름으로 구분이 된다면, 여러 개의 이름을 가진 메서드를 구현해야 하는 번거로움이 발생함.

▣ 사용자는 그 많은 메서드를 외워야 하고, 개발자들은 메서드를 작성함에 있어 메서드 이름을 짓기에 상당히 힘들 것이다.

ex) println()의 경우 하나의 메서드 명만 기억하고 있다면, 10가지 메서드를 사용할 수 있으며, 외우기도 쉽고 기능도 쉽게 이해할 수 있다.

아울러, 메서드 명을 줄일 수 있어 개발자도 편할 것이다.

# 7. final 필드

▣ **최종적인 값을 갖고 있는 멤버변수 = 값을 변경할 수 없는 필드**

▣ Class 앞에 붙으면 더 이상 확장되지 않는다.**(상속 불가)**

▣ final 필드의 딱 한번의 초기값 지정 방법

- 멤버 변수로 선언시 초기화를 함.
- 생성자에서 단 한번, 초기화 가능함.

```
public class Student {

    final int MAX = 250;
    final String name;
    int age;

    //매개변수가 있는 생성자
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

# 8. static final 필드(상수)

▣ 상수 = static final 필드

- final키워드 : 인스턴스 생성 시 독립적으로 인스턴스마다 가지는 **불변의 인스턴스 필드**
- **static final(상수) : 메모리 클래스 영역에서 클래스별로 관리가 되는 불변의 필드이며, 공용 데이터로 사용한다.**

▣ 상수이름은 전부 대문자로 작성함.

▣ 다른 단어가 결합하면 _로 연결하여 작성함.

감사합니다.