

## 인스턴스 멤버

필드와 메소드는 선언 방법에 따라 인스턴스 멤버와 정적 멤버로 분류할 수 있다. 인스턴스 멤버로 선언되면 객체 생성 후 사용할 수 있고, 정적 멤버로 선언되면 객체 생성 없이도 사용할 수 있다.

구분	설명
인스턴스 멤버	객체에 소속된 멤버(객체를 생성해야만 사용가능)
정적(static) 멤버	클래스에 고정된 멤버(객체 없이 사용가능)

- 인스턴스 멤버 선언 및 사용

인스턴스 멤버 : 필드, 메소드

```
public class Car {
    // 인스턴스 필드 선언
    int gas;


    // 인스턴스 메소드 선언
    void setSpeed(int speed) {}
}
```

외부 클래스에서 필드와 메소드에 접근하려면 참조변수를 통해 접근해야 한다.

```
Car car = new Car();
car.gas = 10;
car.setSpeed(1);

Car ucar = new Car();
ucar.gas = 20;
ucar.setSpeed(1);
```

위 코드를 실행하면 gas 필드는 객체마다 생성되고 setSpeed(1); 메소드는 메소드 영역에 따로 저장된다. 메소드는 코드 덩어리이므로 객체마다 저장되면 중복 저장으로 인해 메모리 효율이 떨어진다. 따라서 메소드 코드는 메소드 영역에 두되 공유해서 사용하고, 객체 없이 사용하지 못하도록 제한을 걸어둔다.

 메소드 영역

### this 키워드

객체 내부에서 인스턴스 멤버에 접근하기 위해 this 키워드를 사용한다.

```
public class Car {
    // 필드 선언
    String model;
    int speed;
```

```

// 생성자 선언
Car(String model) {
    this.model = model;
}

// 메소드 선언
void setSpeed(int speed) {
    this.speed = speed;
}

void run() {
    this.setSpeed(100);
    System.out.println(this.model);
}
}

// CarExample.java
package ch06.sec09;

public class CarExample {
    public static void main(String[] args) {
        Car car = new Car("포르쉐");
        Car ucar = new Car("벤츠");

        car.run();
        ucar.run();
    }
}

```

## 정적 멤버 (제한자)

필드와 메소드 모두 정적 멤버가 될 수 있다. 정적 멤버를 사용하려면 static 키워드를 추가한다.

자바는 클래스 로더를 이용해 클래스를 메소드 영역에 저장하고 사용한다. 정적 멤버란 메소드 영역의 클래스에 고정적으로 위치하는 멤버를 말한다. 때문에 정적 멤버를 객체를 생성할 필요 없이 클래스를 통해 바로 사용이 가능하다.

\*클래스 로더(Class Loader) : 클래스 로더는 자바 가상 머신(JVM)에서 사용되는 핵심 컴포넌트 중 하나이다. 클래스 로더는 클래스 파일(.class)을 로드하고, 이를 JVM에 적재한다. 클래스 로더는 다양한 클래스 로딩 전략을 사용하여 클래스를 로드하며, 이는 클래스 로더 계층 구조에 의해 관리된다.

\*메소드 영역(Method Area): 메소드 영역은 JVM의 메모리 영역 중 하나로, 클래스 로더에 의해 로드된 클래스 파일, 메서드, 정적 변수, 상수 풀 등의 정보를 저장한다. 클래스 바이트코드와 클래스 수준의 정보가 메소드 영역에 저장되며, 모든 스레드가 공유한다.

- 정적 멤버 선언

정적 멤버(static)은 공용적인 필드를 정적 필드로 선언하거나, 인스턴스 필드를 사용하지 않는 메소드를 정적 메소드로 사용한다. 즉, 불변의 값을 가지는 것을 정적으로 사용한다.

```

public class 클래스 {
    // 정적 필드 선언
    static 타입 필드 [= 초기값];

    // 정적 메소드
    static 리턴타입 메소드(매개변수, ...) {}
}

public class 클래스 {
    // 정적 필드 선언
    static double pi = 3.14159;

    // 정적 메소드 선언
    static int plus(int x) {
        return x;
    }
}

```

- 정적 멤버 사용

클래스가 메모리로 로딩되면 정적 멤버를 사용할 수 있다. 클래스 이름과 도트(.) 연산자로 접근한다. 인스턴스 접근도 가능하지만 권장되지 않는다.

```

public class Cal {
    static int plus(int a, int b) {
        return x + y;
    }
}

// Example.java
public class Example {
    public static void main(String[] args) {
        int result = Cal.plus(1, 2);
    }
}

```

- 정적 블록/필드

복잡한 초기화 작업이 필요한 경우 정적 블록을 사용한다. 정적 블록은 클래스가 메모리로 로딩될 때 자동으로 실행된다. 정적 블록이 클래스 내부에 여러 개가 선언되어 있을 경우에는 선언된 순서대로 실행된다.

```

public class MyClass {
    // 정적 필드
    static int pi = 3.14;

    static {

```

```

        // 정적 블록 - 클래스 로딩 시에 실행
        // 클래스 초기화 및 설정 작업을 수행
    }
}

```

**\*정적 필드:** 정적 필드는 객체 생성 없이도 사용할 수 있기 때문에 생성자에서 초기화 작업을 하지 않는다. 생성자는 객체 생성 후 실행되기 때문이다.

**\*정적 블록(Static Block):**

1. 정적 블록은 클래스가 메모리로 로딩될 때 실행되는 코드 블록이다. 이 블록은 클래스 로딩 시에 한 번만 실행되며, 클래스의 초기화 및 설정 작업을 수행하는 데 사용된다.
2. 정적 블록은 주로 클래스 변수(static 변수)의 초기화, 외부 리소스의 로딩, 설정 파일의 읽기 등과 관련된 작업을 수행할 때 활용됩니다.
3. 클래스가 처음으로 로딩될 때 한 번만 실행되므로, 정적 블록의 내용은 프로그램의 라이프 사이클 동안 유지된다.

메소드 내부 블록 안에도 블록을 중첩시킬 수 있다.

```

public void myMethod() {
    // 외부 블록 시작
    int outerVariable = 10;

    {
        // 내부 블록 1 시작
        int innerVariable1 = 20;
        // innerVariable1을 사용하는 코드
        // 내부 블록 1 끝
    }

    {
        // 내부 블록 2 시작
        int innerVariable2 = 30;
        // innerVariable2를 사용하는 코드
        // 내부 블록 2 끝
    }

    // outerVariable과 다른 변수들을 사용하는 코드
    // 외부 블록 끝
}

```

중첩된 블록을 사용할 때 주의해야 할 점은 변수의 스코프와 블록 간 변수의 가시성이다. 중첩된 블록 내에서 선언된 변수는 해당 블록 내에서만 사용할 수 있으며, 외부 블록 내에서는 접근할 수 없다. 또한 중첩된 블록 내부 변수와 외부 변수의 이름이 같은 경우, 중첩된 블록 내부 변수가 우선한다.

- 인스턴스 멤버 사용 불가

정적 메소드와 정적 블록은 객체가 없어도 실행된다는 특징 때문에 내부에 인스턴스 필드나 인스턴스 메소드를 사용할 수 없다. 또한 객체 자신의 참조인 `this`도 사용할 수 없다.

```

public class ClassName {
    // 인스턴스 필드와 메소드 선언
    int field;
    void method() {}

    // 정적 필드와 메소드 선언
    static int field2;
    static void method2() {}

    // 정적 블록 선언
    static {
        field = 10; // 컴파일 에러
        method(); // 컴파일 에러
        field2 = 10;
        method2();
    }

    static void method3() {
        this.field = 10; // 컴파일 에러
        this.method(); // 컴파일 에러
        field2 = 10;
        method2();
    }

    // 정적멤버에서 인스턴스 멤버를 사용하고 싶다면
    // 객체 생성 후 참조 변수로 접근해야 한다.
    static void method4() {
        // 객체 생성
        ClassName obj = new ClassName();
        // 인스턴스 멤버 사용
        obj.field = 10;
        obj.method();
    }
}

```

## 정리

### 클래스 변수(정적 멤버, static)

- static 키워드가 붙음
- 모든 인스턴스가 공유. 즉, 클래스의 모든 객체가 동일한 값을 가짐.
- 인스턴스를 생성하지 않아도 클래스 이름으로 사용 가능
- 클래스가 로딩될 때 한 번 생성되어 프로그램이 종료될 때까지 메모리에 유지
- 자동 초기화, 값이 명시되지 않으면 기본값을 가짐

### 인스턴스 변수

- 클래스 내에 선언되며, 각 인스턴스 생성시 별도의 변수가 생성됨 (독립적). 즉, 한 인스턴스 변수를 변경해도 다른 인스턴스에 영향이 없음.
- 객체(인스턴스)가 생성될 때 자동 초기화되며, 각 인스턴스는 다른 값을 가질 수 있음.