

정적 제한자 static





static의 전반적인 내용

사용 제한자(Usage Level modifier)

* static

- static 제한자는 **변수, 메서드**에 적용되는 자바의 키워드입니다.
- static 메서드나 변수는 해당 클래스의 **객체 없이도 참조**할 수 있습니다.
- static 블록(static 메서드, 정적 초기화자) 안에는 static 변수만 사용해야하고, static 메서드만 호출할 수 있습니다. 즉 static 블록에서 non-static 멤버를 객체 생성 없이 직접 참조할 수 없습니다.
- static 제한자는 지정된 변수와 메서드를 객체와 무관하게 만들어주기 때문에 this를 가질 수 없습니다.
- static 메서드는 non-static 메서드로 재정의(Overriding) 될 수 없습니다.
- 대표적인 static 메서드는 애플리케이션의 main() 메서드입니다.
- static에 단순히 블록({ })을 사용한 경우에는 정적 초기화자라고 부르며, static 변수를 초기화하는 역할을 가지고 클래스가 로딩될 때 main() 메서드가 있더라도 그보다 앞서 딱 한 번 실행됩니다.

static 변수

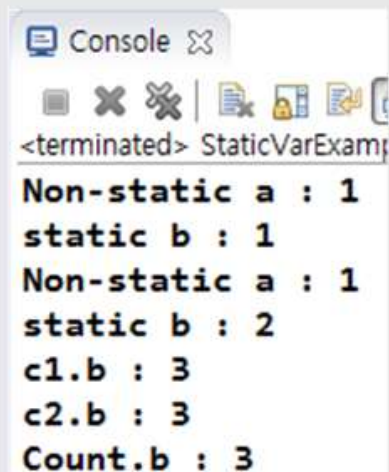
정적 변수(static field)

- static 변수는 모든 객체들이 공유하는 **공유변수**가 됩니다.
- 그리고 객체 생성 없이 **클래스 이름만으로 참조**가 가능합니다.

static_/StaticVarExample.java

```
1: package static_;
2:
3: public class StaticVarExample {
4:
5:     public static void main(String[] args) {
6:         Count c1 = new Count();
7:         c1.a++;
8:         c1.b++;
9:         System.out.println("Non-static a : " + c1.a);
10:        System.out.println("static b : " + c1.b);
11:
12:        Count c2 = new Count();
13:        c2.a++;
14:        c2.b++;
15:        System.out.println("Non-static a : " + c2.a);
16:        System.out.println("static b : " + c2.b);
17:
18:        Count.b++;
19:        System.out.println("c1.b : " + c1.b);
20:        System.out.println("c2.b : " + c2.b);
21:        System.out.println("Count.b : " + Count.b);
22:    }
23: }
```

- 정적 변수는 객체를 만들어 참조할 수도 있지만, 객체를 만들지 않고 **클래스 이름만으로도 참조**가 가능하기 때문에 이를 "클래스 변수"라고도 부릅니다.



```
<terminated> StaticVarExampl
Non-static a : 1
static b : 1
Non-static a : 1
static b : 2
c1.b : 3
c2.b : 3
Count.b : 3
```

static 메서드

정적 메서드(static method)

- static 메서드는 static 변수와 마찬가지로 해당 클래스의 객체 생성 없이도 참조가 가능하게 해줍니다.
- static 메서드에서 멤버를 참조할 때 주의해야 할 사항은 "**static 메서드 안에서는 non-static 멤버를 객체 생성 없이 직접 참조할 수 없다**"는 것입니다.
- static 메서드 안에서는 static 변수를 선언할 수 없습니다.

static_/Count.java

```
1: package static_;
2:
3: public class Count {
4:     public int a=0;
5:     public static int b=0;
6:
7:     public static int doIt() {
8:         // return ++a; //Cannot make a static reference to the non-static field a
9:         return ++b;
10:    }
11: }
```

static_/StaticMethodExample.java

```
1: package static_;
2:
3: public class StaticMethodExample {
4:     public static void main(String[] args) {
5:         System.out.println("Count.doIt() : " + Count.doIt());
6:         System.out.println("Count.doIt() : " + Count.doIt());
7:         System.out.println("Count.doIt() : " + Count.doIt());
8:     }
9: }
```

Console

<terminated> StaticMethodExample

Count.doIt() : 1
Count.doIt() : 2
Count.doIt() : 3

정적 초기화자 static initializer

정적 초기화자(static initializer)

- 정적 초기화자는 **static** 변수들의 초기화에 사용됩니다. 일반 멤버변수는 생성자에서 초기화하지만 static 변수는 객체 생성 없이도 사용해야하므로 생성자를 통해 초기화할 수 없습니다.
- 그래서 **static** 변수는 정적초기화자를 통해 초기화를 합니다.
- 정적 초기화자는 클래스가 로딩될 때 생성자와 main() 메서드에 앞서 오직 단 한번만 실행되기 때문에 애플리케이션 실행 중 반드시 한번만 실행되어야 할 로직이 있다면 이곳에 기술하여 사용될 수 있습니다.

static_/StaticInit.java

```
1: package static_;
2:
3: public class StaticInit {
4:     static {
5:         System.out.println("static initializer가 수행됨");
6:     }
7:     public StaticInit() {
8:         System.out.println("Constructor 호출됨");
9:     }
10: }
```

static_/StaticInitExample.java

```
1: package static_;
2:
3: public class StaticInitExample {
4:     public static void main(String[] args) {
5:         StaticInit s1 = new StaticInit();
6:         System.out.println(s1);
7:         StaticInit s2 = new StaticInit();
8:         System.out.println(s2);
9:         System.out.println("main() 메서드 종료");
10:    }
11: }
```

Console

<terminated> StaticInitExample [Java Applicati

static initializer가 수행됨

Constructor 호출됨

static_.StaticInit@15db9742

Constructor 호출됨

static_.StaticInit@6d06d69c

main() 메서드 종료

자바에서 static에 대해 알아둬야 할 것

자바에서 static이 갖는 의미는 굉장히 중요하다 반드시 외우자

1. static멤버는 객체 생성 없이 클래스명.이름 으로 참조 가능하다
2. static변수는 객체간 값의 공유의 의미
3. static메서드는 같은 static멤버만 참조가능하다.
클래스명.이름 으로 참조한다

*static메서드의 대표적인 사용

```
Math.random();  
Arrays.toString(배열명);  
Integer.parseInt(문자열);
```

Singleton Design Pattern

싱글톤 패턴(Singleton Pattern)

- 싱글톤 패턴은 어떤 클래스의 객체는 오직 하나임을 보장하며, 이 객체에 접근할 수 있는 전역적인 접근점을 제공하는 패턴입니다.
- 클래스 객체를 유일하게 하나만 생성하여 모든 곳에서 하나의 객체에 접근하게 하여, 전역의 개념으로 객체를 사용할 수 있습니다.
- 싱글톤 패턴은 **객체의 생성을 제한**하기 위해 사용합니다. (자바를 이용한 프로그래밍에서 사용됨)

static_/Company.java

```
1: package static_;
2:
3: public class Company{
4:     private String str;
5:
6:     private static Company c = new Company();
7:
8:     private Company() {
9:         str = "company";
10:        System.out.println(str);
11:    }
12:
13:    public static Company getCompany() {
14:        return c;
15:    }
16: }
```

getter메서드를 이용해서 객체를 반환한다
메서드의 반환 타입을 확인하자

static_/TestSingleton.java

```
1: package static_;
2:
3: public class SingletonExample {
4:     public static void main(String [] args) {
5:         Company c1 = Company.getCompany();
6:         Company c2 = Company.getCompany();
7:
8:         System.out.println(c1);
9:         System.out.println(c1 == c2);
10:        System.out.println(c2);
11:    }
12: }
```

Console

<terminated> SingletonExample [Java App]

company
static_.Company@15db9742
true
static_.Company@15db9742



Chapter 15

수고하셨습니다