정적 멤버와 static

정적 멤버와 static

❖ 정적(static) 멤버란?

- 인스턴스와 무관하게 클래스에 고정된 필드와 메소드
  - 정적 필드, 정적 메소드
- 정적 멤버는 클래스에 소속된 멤버
  - 객체 내부(Heap)에 존재하지 않고, 메소드 영역에 존재
  - 정적 멤버는 객체를 생성하지 않고 클래스를 통해 바로 접근해 사용

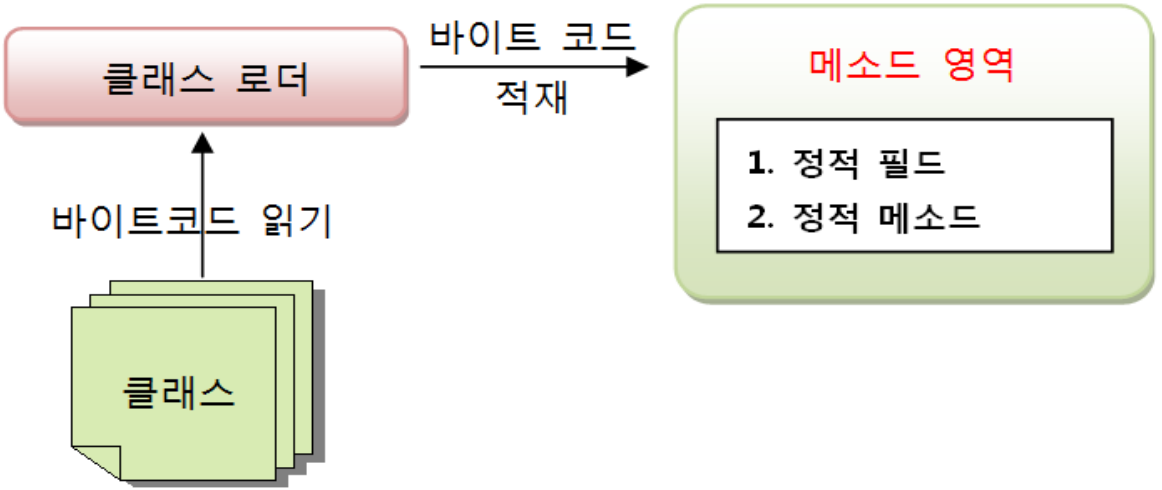
정적 멤버와 static

❖ 정적 멤버 선언

○ 필드 또는 메소드 선언할 때 **static** 키워드 붙임

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

   //정적 메소드
   static 리턴타입 메소드( 매개변수선언, ... ) { ... }
}
```



❖ 정적 멤버 사용

- 클래스 이름과 함께 도트(.) 연산자로 접근

```
클래스.필드;
클래스.메소드( 매개값, ... );
```

```
public class Calculator {
    static double pi = 3.14159;
    static int plus(int x, int y) { ... }
    static int minus(int x, int y) { ... }
}
```

[바람직한 사용]

```
double result1 = 10 * 10 * Calculator.pi;
int result2 = Calculator.plus(10, 5);
int result3 = Calculator.minus(10, 5);
```

[바람직하지 못한 사용]

```
Calculator myCalcu = new Calculator();
double result1 = 10 * 10 * myCalcu.pi;
int result2 = myCalcu.plus(10, 5);
int result3 = myCalcu.minus(10, 5);
```

정적 멤버와 static

❖ 인스턴스 멤버 선언 vs 정적 멤버 선언의 기준

- 필드
  - 객체마다 가지고 있어야 할 데이터 → 인스턴스 필드
  - 공용적인 데이터 → 정적 필드

```
public class Calculator {
   String color;                    //계산기 별로 색깔이 다를 수 있다.
   static double pi = 3.14159;   //계산기에서 사용하는 파이( π)값은 동일하다.
}
```

❖ 인스턴스 멤버 선언 vs 정적 멤버 선언의 기준

- 메소드
  - 인스턴스 필드로 작업해야 할 메소드 → 인스턴스 메소드
  - 인스턴스 필드로 작업하지 않는 메소드 → 정적 메소드

❖ 정적 멤버 사용: Calculator.java

```
public class Calculator {
    static double pi = 3.14159;

    static int plus(int x, int y) {
        return x + y;
    }

    static int minus(int x, int y) {
        return x - y;
    }
}
```

❖ 정적 멤버 사용: CalculatorExample.java

```
public class CalculatorExample {
    int a = 10;
    public static void main(String[] args) {
        System.out.println(“a : " + a);
        // System.out.println(“a : " + this.a);



        double result1 = 10 * 10 * Calculator.pi;
        int result2 = Calculator.plus(10, 5);
        int result3 = Calculator.minus(10, 5);

        System.out.println("result1 : " + result1);
        System.out.println("result2 : " + result2);
        System.out.println("result3 : " + result3);
    }
}
```

❖ 정적 초기화 블록

- 클래스가 메소드 영역으로 로딩될 때 자동으로 실행하는 블록

```
static {
    …
}
```

- 정적 필드의 복잡한 초기화 작업과 정적 메소드 호출 가능
- 클래스 내부에 여러 개가 선언되면 선언된 순서대로 실행

정적 멤버와 static

❖ 정적 초기화 블록

```
public class Television {

    static String company = "Samsung";
    static String model = "LCD";
    static String info;

    static {
        info = company + model;
    }
}
```

❖ 정적 초기화 블록: Television.java

```
public class Television {
    static String company = "Samsung";
    static String model = "LCD";
    static String info;

    static {
        info = company + "-" + model;
    }
}
```

❖ 정적 초기화 블록: TelevisionExample.java

```
public class TelevisionExample {
    public static void main(String[] args) {
        System.out.println(Television.info);
    }
}
```

정적 멤버와 static

❖ 정적 메소드와 정적 블록 작성시 주의할 점

- 객체가 없어도 실행 가능
- 정적 블록 내부/정적메서드에서 인스턴스 필드나 인스턴스 메소드 사용 불가
  - 지역 변수만 사용 가능
- 객체 자신의 참조인 `this` 사용 불가
  - EX) `main()`

정적 멤버와 static

❖ 정적 메소드와 정적 블록 작성시 주의할 점

```
//정적 블록
static {
    field1 = 10;        (x)
    method1();          (x)
    field2 = 10;        (o)
    method2();          (o)
}

//정적 메소드
static void Method3 {
    this.field1 = 10;       (x)
    this.method1();         (x)
    field2 = 10;            (o)
    method2();              (o)
}
```

컴파일 에러

컴파일 에러

```
static void Method3() {
    ClassName obj = new ClassName();
    obj.field1 = 10;
    obj.method1();
}
```

❖ 싱글톤(Singleton)

- 하나의 애플리케이션 내에서 단 하나만 생성되는 객체

❖ 싱글톤을 만드는 방법

- 외부에서 new 연산자로 생성자를 호출할 수 없도록 막기
  - private 접근 제한자를 생성자 앞에 붙임

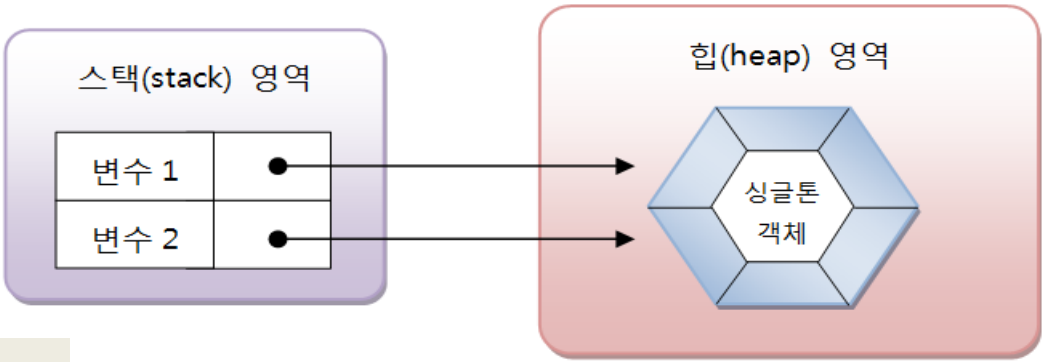
- 클래스 자신의 타입으로 정적 필드 선언
  - 자신의 객체를 생성해 초기화
  - private 접근 제한자 붙여 외부에서 필드 값 변경 불가하도록

- 외부에서 호출할 수 있는 정적 메소드인 getInstance() 선언
  - 정적 필드에서 참조하고 있는 자신의 객체 리턴

정적 멤버와 static

❖ 싱글톤 얻는 방법

```
클래스 변수 1 = 클래스.getInstance();
클래스 변수 2 = 클래스.getInstance();
```




```
/*
Singleton obj1 = new Singleton();  //컴파일 에러
Singleton obj2 = new Singleton();  //컴파일 에러
*/

Singleton obj1 = Singleton.getInstance();
Singleton obj2 = Singleton.getInstance();

if(obj1 == obj2) {
   System.out.println("같은 Singleton 객체 입니다.");
} else {
   System.out.println("다른 Singleton 객체 입니다.");
}
```

❖ 싱글톤 객체 : Singleton.java

```
public class Singleton {
    private static Singleton singleton = new Singleton();

    private Singleton() {
    }

    static Singleton getInstance() {
        return singleton;
    }
}
```

❖ 싱글톤 객체 사용: SingletonExample.java

```
public class SingletonExample {
    public static void main(String[] args) {
        /*
         * Singleton obj1 = new Singleton(); //컴파일 에러
         * Singleton obj2 = new Singleton(); //컴파일 에러
         */

        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();

        if (obj1 == obj2) {
            System.out.println("같은 Singleton 객체 입니다.");
        } else {
            System.out.println("다른 Singleton 객체 입니다.");
        }
    }
}
```