PRESENTATION

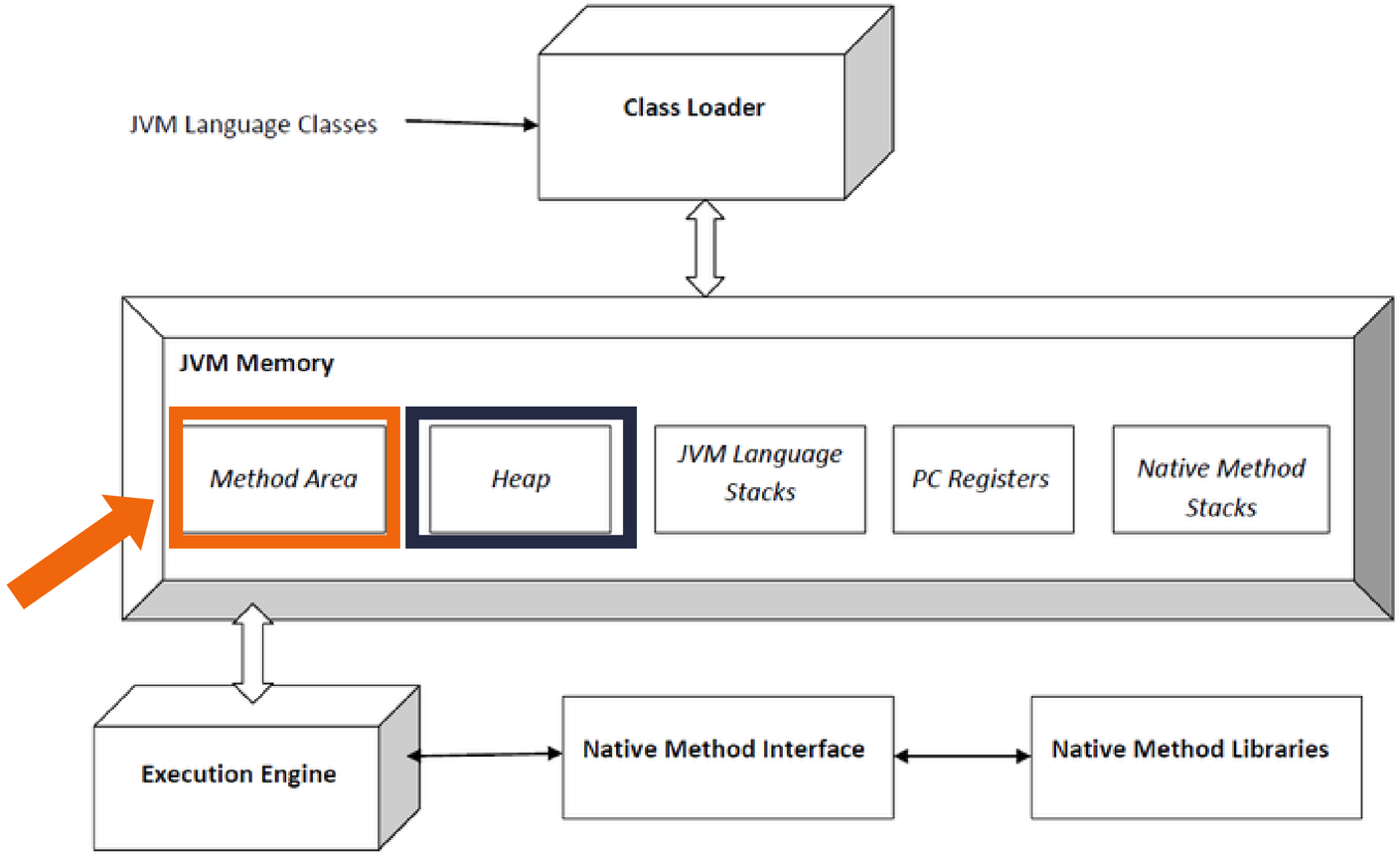
39주차 주제 static

~static의 장점과 단점~

by mun

static

- 자바에서 클래스의 멤버(변수, 메서드 등)가 객체의 인스턴스가 아닌 클래스 자체에 종속되도록 하는 키워드



- **Heap**:
  객체가 저장되는 공간
  인스턴스를 생성할 때 이 공간에 객체가 할당됨
- **Stack**:
  각 메서드 호출 시 지역 변수와 메서드 정보가 저장되는 공간
- **Method Area(또는 Class Area)**:
  클래스의 구조(정적 변수, 정적 메서드, 상수, 클래스 메타데이터 등)가 저장되는 공간

장점 1: 메모리 절약

static을 사용하지 않을 경우

```
public class NonStaticExample {
    public int counter = 0; // 인스턴스마다 별도의 변수

    public NonStaticExample() {
        counter++;
    }

    public static void main(String[] args) {
        NonStaticExample obj1 = new NonStaticExample();
        NonStaticExample obj2 = new NonStaticExample();
        System.out.println("obj1's counter: " + obj1.counter); // 출력: 1
        System.out.println("obj2's counter: " + obj2.counter); // 출력: 1
    }
}
```

객체마다 별도로 저장되기 때문에, 두 인스턴스가 각각 메모리를 차지

= 인스턴스가 많아질수록 메모리 사용이 늘어나게 됨

장점 1: 메모리 절약

```
public class NonS
    public int cc

    public NonSta
        counter++
    }

    public static
        NonStatic
        NonStatic
        System.ou
        System.ou
    }
}
```

static을 사용할 경우

```
public class StaticExample {
    public static int counter = 0; // 클래스 레벨에서 공유

    public StaticExample() {
        counter++;
    }

    public static void main(String[] args) {
        StaticExample obj1 = new StaticExample();
        StaticExample obj2 = new StaticExample();
        System.out.println("Shared counter: " + StaticExample.counter); // 출력: 2
    }
}
```

클래스 로드 시 한 번만 메모리에 할당
= 모든 인스턴스가 동일한 메모리 공간을 공유
= 인스턴스가 아무리 많아져도 추가 메모리 할당 없이 공유된 값을 참조하게 됨

= 메모리 사용량이 크게 줄어든다

장점 1: 메모리 절약

```
public class NonS
    public int cc

    public NonSta
        counter++
    }

    public static
        NonStatic
        NonStatic
        System.ou
        System.ou
    }
}
```

static을 사용할 경우

```
public class StaticExample {
    public static int counter = 0; // 클래스 레벨에서 공유

    public StaticExample() {
        counter++;
    }

    public static void main(String[] args) {
        StaticExample obj1 = new StaticExample();
        StaticExample obj2 = new StaticExample();
        System.out.println("Shared counter: " + StaticExample.counter); // 출력: 2
    }
}
```

클래스 로드 시 한 번만 메모리에 할당
= 모든 인스턴스가 동일한 메모리 공간을 공유
= 인스턴스가 아무리 많아져도 추가 메모리 할당 없이 공유된 값을 참조하게 됨

= 메모리 사용량이 크게 줄어든다

동일한 데이터가 여러 인스턴스에서 공유되어야 할 때 유리

장점 2: 전역 접근성

static을 사용하지 않을 경우

```java
public class NonStaticGlobalAccess {
    public String companyName = "Tech Corp"; // 인스턴스마다 별도의 값

    public void displayCompanyName() {
        System.out.println("Company Name: " + companyName);
    }

    public static void main(String[] args) {
        NonStaticGlobalAccess obj1 = new NonStaticGlobalAccess();
        NonStaticGlobalAccess obj2 = new NonStaticGlobalAccess();
        obj1.displayCompanyName(); // 출력: Company Name: Tech Corp
        obj2.displayCompanyName(); // 출력: Company Name: Tech Corp
    }
}
```

인스턴스 변수

메서드를 호출하려면 객체를 먼저 생성해야 한다

= 자주 사용하는 전역 데이터를 접근할 때 불편함

장점 2: 전역 접근성

static을 사용할 경우

객체를 생성하지 않고도 클래스명으로 바로 접근 가능

= 공용 데이터나 메서드를 전역적으로 관리할 때 편리

```java
public class StaticGlobalAccess {
    public static String companyName = "Tech Corp"; // 클래스 레벨에서 공유

    public static void displayCompanyName() {
        System.out.println("Company Name: " + companyName);
    }

    public static void main(String[] args) {
        StaticGlobalAccess.displayCompanyName(); // 출력: Company Name: Tech Corp
    }
}
```

장점 3: 간편함

static을 사용하지 않을 경우

```
public class NonStaticMathUtility {
    public int add(int a, int b) {
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public static void main(String[] args) {
        NonStaticMathUtility math = new NonStaticMathUtility();
        System.out.println("Sum: " + math.add(5, 10));          // 출력: Sum: 15
        System.out.println("Product: " + math.multiply(5, 10)); // 출력: Product: 50
    }
}
```

인스턴스를 생성한 후에야 메서드를 호출할 수 있음

= 자주 사용하는 유틸리티 메서드라면
불필요한 객체 생성으로 인해 코드가 복잡해질 수 있다

장점 3: 간편함

static을 사용할 경우

```java
public class StaticMathUtility {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int multiply(int a, int b) {
        return a * b;
    }

    public static void main(String[] args) {
        System.out.println("Sum: " + StaticMathUtility.add(5, 10));         // 출력: Sum: 15
        System.out.println("Product: " + StaticMathUtility.multiply(5, 10)); // 출력: Product: 50
    }
}
```

매번 객체를 생성하지 않고도 바로 사용 가능

단점 1: 객체 지향성 훼손

```java
public class ObjectOrientedProblem {
    public static int sharedValue = 0; // 모든 인스턴스가 공유

    public int instanceValue = 0; // 인스턴스마다 고유

    public ObjectOrientedProblem() {
        sharedValue++;
        instanceValue++;
    }

    public static void main(String[] args) {
        ObjectOrientedProblem obj1 = new ObjectOrientedProblem();
        ObjectOrientedProblem obj2 = new ObjectOrientedProblem();

        // 인스턴스별 값
        System.out.println("obj1's instanceValue: " + obj1.instanceValue); // 출력: 1
        System.out.println("obj2's instanceValue: " + obj2.instanceValue); // 출력: 1

        // 공유된 static 값
        System.out.println("Shared value: " + ObjectOrientedProblem.sharedValue); // 출력: 2
    }
}
```

모든 객체가 같은 값을 공유하게 됨
= 객체가 고유한 상태를 가지는 객체 지향의 원칙을 위반할 수 있다

= 객체의 고유 상태 관리가 어렵다

단점 2: 메모리 관리의 어려움

```java
public class MemoryLeakExample {
    public static List<String> dataList = new ArrayList<>(); // static으로 메모리에 남음

    public static void addData(String data) {
        dataList.add(data);
    }

    public static void main(String[] args) {
        addData("Test1");
        addData("Test2");

        // dataList는 프로그램 종료 전까지 메모리에 유지
        System.out.println("Data List: " + dataList); // 출력: [Test1, Test2]
    }
}
```

클래스가 메모리에서 해제되지 않는 한 계속 메모리에 남아 있음
= 메모리 누수의 원인이 될 수 있음

단점 3: 테스트 어려움

```java
public class TestingDifficultyExample {
    public static int counter = 0;

    public static void incrementCounter() {
        counter++;
    }

    public static void main(String[] args) {
        incrementCounter();
        System.out.println("Counter: " + counter); // 출력: Counter: 1
    }
}
```

메서드나 변수가 클래스에 직접 종속되어 있음
= 유닛 테스트 시 객체 간의 의존성을 분리하기가 어려움
= mocking 어려움

다른 테스트 케이스에서도 이 값이 변경될 수 있어
테스트의 독립성이 깨질 수 있다

= 테스트 및 유지보수가 어렵다

static의 장점과 단점을 이해하고
적절하게 사용해 보아요!

감사합니다

참고

- https://techblog.woowahan.com/9232/
- https://velog.io/@jungmyeong96/JAVA%EA%B8%B0%EC%B4%88-JVM%EC%9D%B4%EB%9E%80
- ChatGPT