😀

자바static

자바의 `static` 키워드: 메모리와 사용법

이번 글에서는 자바의 `static` 키워드에 대해 알아보겠습니다. **`static`**은 자바에서 메모리와 변수, 메서드에 대해 다루는 중요한 개념으로, 객체를 생성하지 않고도 클래스명으로 직접 접근할 수 있게 만들어줍니다. 이 글에서는 **`static`**이 어떻게 작동하는지, 메모리에서 어떻게 관리되는지, 그리고 언제 사용하면 좋은지에 대해 설명드리겠습니다.

1. `static`이란 무엇인가?

자바에서 `static` 키워드를 사용하면, **객체를 생성하지 않고도 클래스명으로 직접 변수나 메서드에 접근할 수 있습니다.** 이는 프로그램 전체에서 공유되는 변수를 만들거나, 특정 작업을 수행하는 유틸리티 메서드를 정의할 때 유용합니다.

```
class MyClass {
    static int count = 0;  // static 변수
    static void increment() {  // static 메서드
        count++;
    }
}
```

위 코드에서 `count`와 `increment()`는 **`static`**으로 선언되었기 때문에 객체를 생성하지 않고도 `MyClass.count` 또는 `MyClass.increment()`로 접근할 수 있습니다.

2. `static` 변수의 메모리 관리

`static` 변수는 **메서드 영역(Method Area)**에 저장되며, 프로그램이 실행되는 동안 **JVM 메모리에 계속 남아있습니다**. 즉, **프로그램이 종료될 때까지** 해당 변수는 메모리에서 사라지지 않습니다. 이는 메모리 낭비로 이어질 수 있기 때문에 신중하게 사용해야 합니다.

- **스택(Stack)**: 메서드 호출 시 지역 변수와 참조 변수가 저장됨.
- **힙(Heap)**: 객체가 생성될 때 저장되는 메모리 공간.
- **메서드 영역(Method Area)**: 클래스 정보와 **static 변수**들이 저장되는 영역.

```
MyClass.increment();  // 객체 생성 없이 메서드 호출
System.out.println(MyClass.count);  // 출력: 1
```

3. `static`의 특징

- **프로그램이 종료될 때까지 메모리 유지**: `static`으로 선언된 변수는 메모리에서 계속 남아 있기 때문에, 프로그램이 끝날 때까지 어디서든 접근할 수 있습니다.
- **객체 생성 없이 접근 가능**: 일반적인 변수는 객체를 생성한 후 접근해야 하지만, `static` 변수는 클래스명으로 바로 접근 가능합니다.
- **모든 객체가 공유**: `static` 변수는 클래스의 모든 객체가 공유하므로, 하나의 객체가 값을 변경하면 다른 객체도 그 값을 공유합니다.

4. `static`의 사용 사례

- *`static`*은 다음과 같은 경우에 사용됩니다:
- **유틸리티 메서드**: `Math.sqrt()`, `Collections.sort()`처럼 객체 생성 없이 호출되는 메서드.
- **상수 선언**: 변경되지 않는 값들을 `static final`로 선언하여 사용.

```
public class Constants {
    public static final String RED = "RED";
    public static final int MAX_USERS = 100;
}
```

5. `static`의 주의사항

- *`static`*을 남발하면 **메모리 낭비**와 성능 저하를 초래할 수 있습니다. 프로그램이 종료될 때까지 메모리에 남아있기 때문에, 너무 많은 `static` 변수를 사용하면 시스템 자원이 낭비될 수 있습니다. 따라서 반드시 필요할 때만 사용하는 것이 좋습니다.

결론

- *`static`*은 자바에서 매우 유용한 기능이지만, 메모리 사용에 신경 쓰지 않으면 문제가 발생할 수 있습니다. 이를 이해하고, 필요한 경우에만 적절히 사용하면 더욱 효율적인 코드를 작성할 수 있습니다.
