static

주차	39주차
스터디 일자	@2024/08/30

상수

보통 평소 사용하는 상수 형태는 다음과 같다.

```
private static final int MAX = 1000;
```

상수란?

- 일반적으로 상수는 **불변의 값**을 의미한다. (변하지 않는 값)

🤔 그런데, 상수를 선언할 때 static과 final은 왜 같이 쓰는거지?

불변의 값이면 처음 초기값을 지정하면 다시는 수정할 수 없는 **final** 만 쓰면 되지 않나 라는 의문이 든다.

final 이란?

> final은 **최종적**이라는 뜻을 가진다.

> 초기값이 저장되면, **수정할 수 없는 값이 된다.**

그렇다면 final이 변수, 메소드, 클래스에 붙으면?

- final **변수**

  생성자나 대입 연산자(=)를 통해 한 번만 초기화 할 수 있다.
- final **메소드**

  해당 메소드를 오버 라이드하거나 숨길 수 없다.
- final **클래스**

  해당 클래스는 상속할 수 없다.

그러면 final만 있어도 상수 아닌가?

상수(불변의 값)의 조건은 다음과 같다.

1. 객체마다 저장할 필요가 없어, **공통으로 사용**할 수 있어야 한다.
2. **여러가지 값**으로 **초기화** 될 수 없다.

final이 이 조건에 만족하는지 알아보자.

```
public class User {
    private final String name; // final 변수

    public User(String name) { // 생성자
        this.name = name;
    }
}
```

위 코드를 보면,

객체마다 name 변수가 저장되고, 생성자를 통해 객체마다 **여러가지 값**을 가질 수 있다.

👉 **final만으로는 상수라고 할 수 없다!**

그렇다면 static은 무엇일까?

> static은 **정적**인 의미를 가진다.

- static **변수**

  해당 필드는 컴파일 타임에 메모리를 할당하여, 주로 공유하기 위해 사용한다.

- static **메소드**

  해당 메소드는 객체를 생성하지 않아도 접근이 가능하여, 주로 유틸리티 성 메소드를 위해 사용한다.

  (예를 들어 `연산`, `오늘 날짜 구하기` 등)

다시 상수(불변의 값)의 조건을 보면

1. 객체마다 저장할 필요가 없어, **공통으로 사용**할 수 있어야 한다.
2. **여러가지 값**으로 **초기화** 될 수 없다.

static을 사용하면 컴파일 타임에 메모리를 할당하여 공통으로 사용할 수 있기 때문에 **1번 조건을 만족**하는 것을 확인할 수 있다.

결론

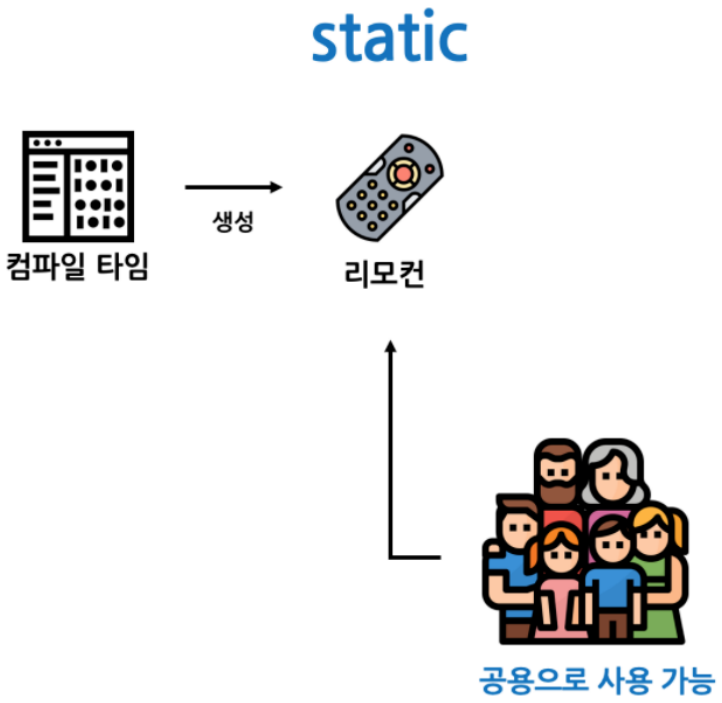


그림 1번

final

그림 2번

그림 1번처럼 static 키워드를 사용하면

컴파일 타임에 리모컨을 생성해서 하루가 지나도 같은 리모컨을 가족들끼리 공통으로 사용할 수 있는 것이고, (상수의 1번 조건 만족)

그림 2번처럼 final 키워드를 사용하면

거실에 있는 TV 리모컨이 안방에 있는 TV 리모컨으로 조작할 수 없어지므로, (상수의 2번 조건 만족)

💡 **static**과 **final**을 같이 쓰면 **상수**를 만들어 쓸 수 있다!

... 에서 의문이 하나 들었는데,

위 이야기는 클래스 인스턴스를 여러개 생성하는 환경에서의 상수를 의미하는 것 같았다.

그렇다면 Spring에서 사용하는 상수는 **static** 할 필요가 있을까..?????

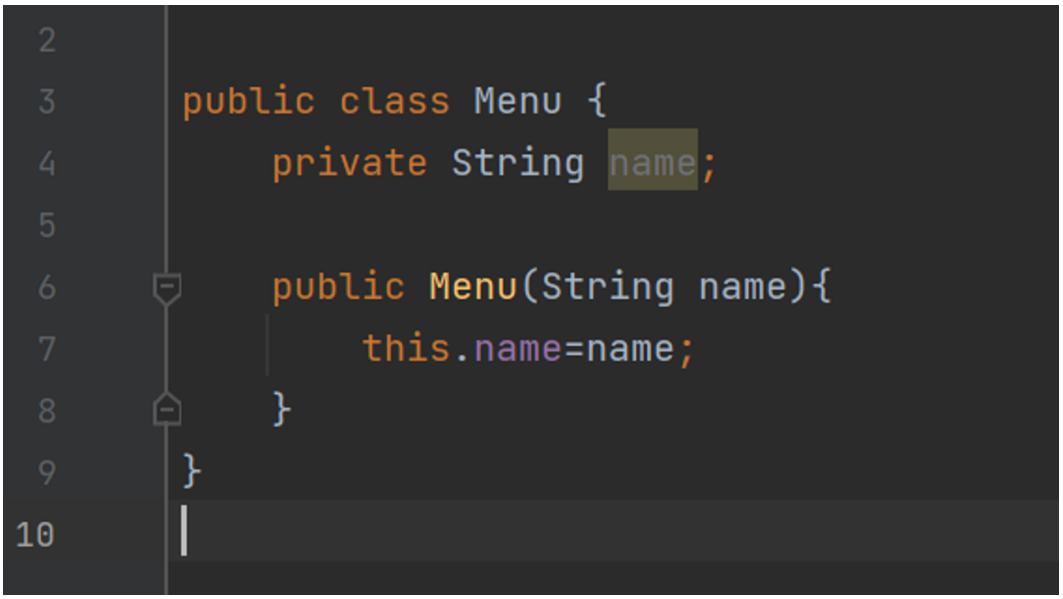
Spring과 static

먼저 아주 중요하게 알아야 할 것은, Spring은 **싱글톤 디자인패턴**을 사용한다는 것이다. (기본적으로)

싱글톤 패턴?

> 객체의 **인스턴스를 한개만 생성**되게 하는 패턴이다.

- 싱글톤 **X**



```java
public class SiEx {

    @Test
    @DisplayName("싱글톤 없이 객체 생성")
    void notSingleton(){
        Menu menu1 = new Menu( name: "자장면");
        Menu menu2 = new Menu( name: "자장면");

        Assertions.assertThat(menu1).isEqualTo(menu2);
    }
}
```

→ 결과는 **menu1과 menu2는 같지 않다** 가 나온다.

두 인스턴스가 다른 것이기 때문!

위 코드를 싱글톤 형식으로 바꿨다면?

- 싱글톤 O

```java
public class SiEx {

    @Test
    @DisplayName("싱글톤을 활용한 객체 생성")
    void notSingleton(){
        Menu menu1 = Menu.getInstance();
        Menu menu2 = Menu.getInstance();

        menu1.setName("자장면");
        menu2.setName("자장면");

        Assertions.assertThat(menu1).isEqualTo(menu2);
    }
}
```

→ 결과는 **menu1과 menu2가 같다** 고 나온다.

두 인스턴스처럼 보이지만 Menu 클래스의 getInstance 메서드 내에서 처음 한 번만 static한 인스턴스를 생성하고 ,

그 이후 호출부터는 기존 인스턴스를 재사용하도록 설계해두었기 때문에~

그러면 !!

```
49  @Service
50  @RequiredArgsConstructor
51  @Transactional
52  public class ExService {
53
        3 usages
54      private static final String DEFAULT_CATEGORY = "all";
        2 usages
55      private static final int DEFAULT_PAGE_NUMBER = 0;
56
```

static + final

```
49  @Service
50  @RequiredArgsConstructor
51  @Transactional
52  public class ExService {
53
        3 usages
54      private final String DEFAULT_CATEGORY = "all";
        2 usages
55      private final int DEFAULT_PAGE_NUMBER = 0;
56
```

final만

위 같은 경우,

Service 클래스(`ExService`) 를 몇개의 컨트롤러에서 선언하던간에,

하나의 `ExService` 인스턴스만 생성되어 컨테이너에 존재할 것이고,

그러면 그 안에 선언된 변수도 그대로 있을 것이다.

그래서 싱글톤으로 동작하는 Spring의 위와 같은 상황에서는 두번째 코드처럼 값이 변하지 않게만 해주면

같은 변수를 사용한다는 것이다..

결론

💡 Spring에서 특정한 상황에서 빈으로 등록하는 클래스의 상수는 static 없이 final만 써도 똑같이 쓸 수 있다!

출처

https://velog.io/@seongwon97/싱글톤Singleton-패턴이란

https://hyojaedev.tistory.com/5

https://taehoung0102.tistory.com/73