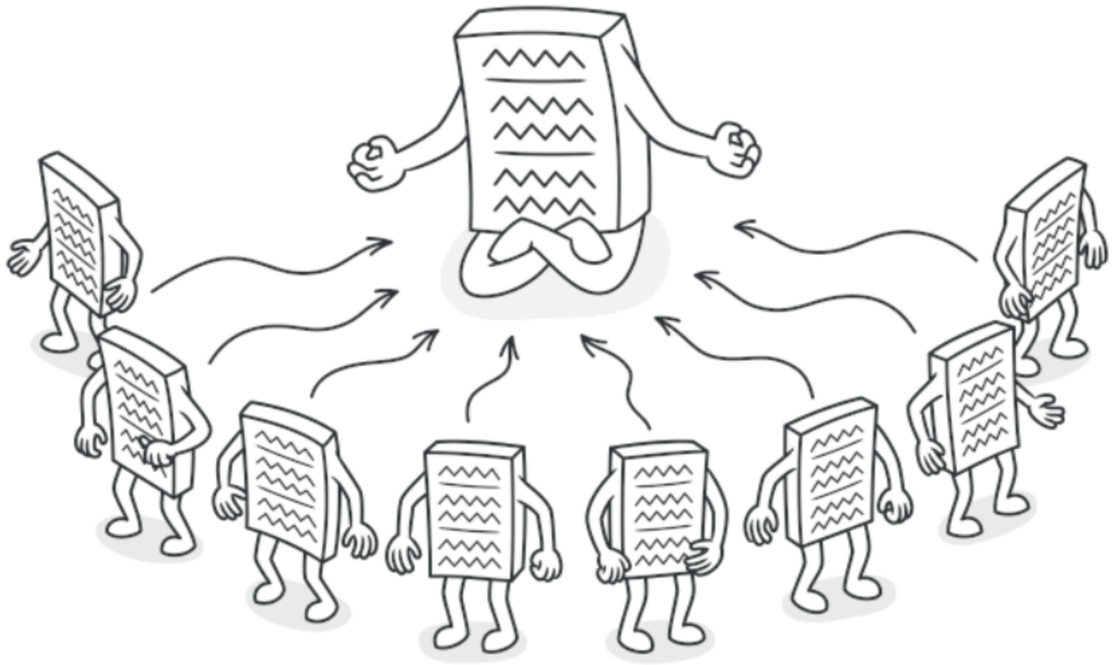
18주차 발표_민서연

# 디자인 패턴
# : Singleton

# 01 Singleton



클래스에 인스턴스가 하나만 있도록 하면서 이 인스턴스에 대한
전역 접근 지점을 제공하는 생성 디자인 패턴
**즉, 단 하나의 유일한 객체를 만들기 위한 패턴**

# 01 Singleton

## 필요 이유

### 메모리 절약을 위해

인스턴스가 필요할 때 인스턴스를 새로 만드는 것이 아닌 기존의 인스턴스를 가져와 활용한다.

### 속도 향상을 위해

이미 생성된 인스턴스를 사용하기 때문에 새 인스턴스를 만드는 시간을 줄이고 속도를 향상 시킬 수 있다.

### 쉬운 데이터 공유를 위해

인스턴스는 전역으로 사용되기 때문에 여러 클래스에서 데이터를 공유하며 사용할 수 있다. (동시성 문제를 조심해야한다!)

# 01 Singleton

```java
package com.example.studytest;

12 usages
public class Singleton {
    3 usages
    private static Singleton singleton = new Singleton();

    2 usages
    private Singleton() {
    }

    public static Singleton getInstance() {
        if(singleton == null){
            singleton = new Singleton();
        }
        return singleton;
    }

    no usages
    public void singleton() {
        System.out.println("Singleton");
    }
}
```

1개만 있어야하는 객체의 인스턴스이기 때문에 static 으로 선언

private 생성자로 외부에서 객체 생성을 막음

외부에서는 getInstance()로 instance를 반환

# 01 Singleton

```java
package com.example.studytest;

12 usages
public class Singleton {
    3 usages
    private static Singleton singleton = new Singleton();

    2 usages
    private Singleton() {
    }

    4 usages
    public static Singleton getInstance() {
        if(singleton == null){
            singleton = new Singleton();
        }
        return singleton;
    }

    no usages
    public void singleton() {
        System.out.println("Singleton");
    }
}
```

```java
@SpringBootApplication
public class StudyTestApplication {

    public static void main(String[] args) {
        Singleton singleton1 = Singleton.getInstance();
        Singleton singleton2 = Singleton.getInstance();

        System.out.println(singleton1);
        System.out.println(singleton2);

    }

}
```

```
StudyTest [:StudyTestApplication.main()]
Console   Actuator
Stuc 10 sec, 157 ms
오후 1:43:55: Executing ':StudyTestApplication.main()'...

> Task :compileJava
> Task :processResources
> Task :classes

> Task :StudyTestApplication.main()
com.example.studytest.Singleton@646d64ab
com.example.studytest.Singleton@646d64ab
```

# 01 Singleton

## 문제점

싱글톤 패턴은 multi-thread 환경에서는 **동시성 문제**가 발생 할 수 있다.

multi-thread
: 하나의 프로그램에게 동시에 여러개의 일을 수행 할 수 있도록 해주는 것

# 01 Singleton

## 문제점: 동시성 문제

```
4 usages
public static Singleton getInstance() {
    if(singleton == null){
        singleton = new Singleton();
    }
    return singleton;
}
```

멀티 스레드 환경에서는 두개 이상의 스레드가 getInstance()를 하게 될 경우 두개의 인스턴스가 생성되는 문제가 발생한다.

동시성 문제 (thread unsafe)

# 01 Singleton

## 해결법: 인스턴스를 조건을 주지 않고 생성

```
11 usages
public class Singleton {
    1 usage
    private static Singleton singleton = new Singleton();

    1 usage
    private Singleton() {
    }

    4 usages
    public static Singleton getInstance() {
        return singleton;
    }
```

이러한 경우 불필요한 경우에도 인스턴스가 생성 될 수 있다.

# 01 Singleton

## 해결법: synchronized 키워드 사용

```
4 usages
public synchronized static Singleton getInstance(){
    if(singleton == null){
        singleton = new Singleton();
    }
    return singleton;
}
```

synchronized 키워드를 사용하면 getInstance() 메서드를 동기화 시키면서 thread-safe하게 만들 수 있음

**성능 저하**의 문제점이 있다.

# 01 Singleton

## 해결법: LazyHolder 사용

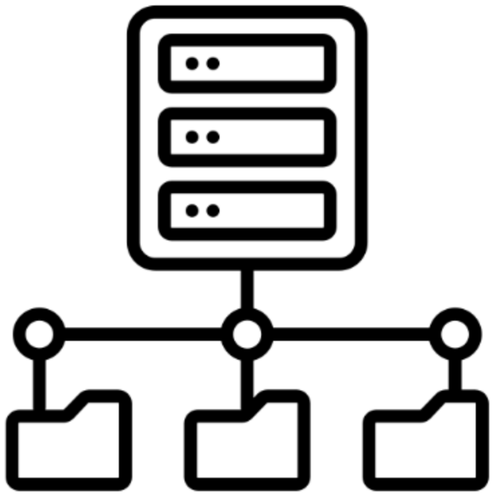
```
4 usages
public static Singleton getInstance(){
    return LazyHolder.INSTANCE;
}

1 usage
private static class LazyHolder {
    1 usage
    public static final Singleton INSTANCE = new Singleton();
}
```

LazyHolder.INSTANCE를 호출하는 순간 Class가 로딩되며 초기화가 진행되고 thread-safe를 보장한다.

가장 많이 사용되고 있는 해결방법

# 02 스프링의 싱글톤 레지스트리



스프링은 하나의 서버에 초당 수많은 요청이 도착하고 처리되어야 하기 때문에
요청이 올 때마다 서버에서 관련 객체를 생성하는 것은 굉장히 큰 오버헤드일 수 있다.

따라서 하나의 객체만 생성해서 공유할 수 있는 싱글톤 패턴을 이용한다!

# 02 스프링의 싱글톤 레지스트리

## 싱글톤 레지스트리

싱글톤 패턴의 단점을 보완하기 위해 나온 레지스트리

스프링 컨테이너가 싱글톤 레지스트리의 역할을 해서 빈을 싱글톤으로 관리한다.

private 생성자로 객체의 생성을 막는 것이 아닌 일반 자바 클래스로 싱글톤을 활용할 수 있다.

생성, 관계설정, 사용에 대한 제어권이 컨테이너에게 있기 때문에 일반 자바 클래스도 싱글톤으로 관리가 가능하다.

# 02 스프링의 싱글톤 레지스트리

## 주의할 점

객체가 상태를 가지게 설계하면 안됨

즉, 클라이언트에게 의존적인 필드가 있으면 안되며

클라이언트가 수정이 가능한 필드가 있으면 안된다.

싱글톤 내부의 필드들은 읽기 전용 필드만 존재하거나
지역 변수나 파라미터를 이용해야한다.

# 03 출처

https://velog.io/@seongwon97/%EC%8B%B1%EA%B8%80%ED%86%A4Singleton-%ED%8C%A8%ED%84%B4%EC%9D%B4%EB%9E%80
https://inpa.tistory.com/entry/GOF-%F0%9F%92%A0-%EC%8B%B1%EA%B8%80%ED%86%A4Singleton-%ED%8C%A8%ED%84%B4-%EA%BC%BC%EA%BC%BC%ED%95%98%EA%B2%8C-%EC%95%8C%EC%95%84%EB%B3%B4%EC%9E%90
https://refactoring.guru/ko/design-patterns/singleton
https://goodgid.github.io/What-is-Multi-Thread/
https://ynzu-dev.tistory.com/entry/JAVA-%EC%8B%B1%EA%B8%80%ED%86%A4-%ED%8C%A8%ED%84%B4Singleton-Pattern-%EB%A9%80%ED%8B%B0-%EC%8A%A4%EB%A0%88%EB%93%9C-%ED%99%98%EA%B2%BD%EC%97%90%EC%84%9C%EC%9D%98-%EB%AC%B8%EC%A0%9C%EC%A0%90
https://yjksw.github.io/spring-singleton-registry/