Java Programming II Lab 6

교 과 명 : 자바프로그래밍 2
담당교수명 : 박경신 교수님
학 과 : 컴퓨터공학과
학 번 : 32185010
성 명 : 홍찬희
제 출 일 : 2021. 10. 28

DKU
단국대학교
DANKOOK UNIVERSITY

Lab 6

싱글톤 패턴이란? 전역변수를 사용하지 않고, 객체를 하나만 생성하여 어디서든 객체를 참조할 수 있게 해주는 패턴이다.

Classic

```
public class Singleton {
    private static Singleton inst = null;
    private Singleton() { System.out.println("classic Singleton Constructor"); }

    public static Singleton getInstance(){
        if(inst == null){
            inst = new Singleton();
        }
        return inst;
    }
    public void print() { System.out.println("classic singleton instatnce hashcod
}
```

```
classic Singleton Constructor
classic Singleton Constructor
classic Singleton Constructor
classic Singleton Constructor
classic singleton instatnce hashcode : 1527989115
5b13437b
classic singleton instatnce hashcode : 1085194098
40aebf72
classic singleton instatnce hashcode : 1492981044
58fd1534
classic singleton instatnce hashcode : 1824636871
6cc1bfc7
classic singleton instatnce hashcode : 1196409528
474fc2b8
```

가장 기본적인 싱글톤 패턴의 모습이다. 하지만, 이 방법의 경우 다중스레드 환경에서 Singleton클래스의 getInstance 메소드의 if문이 아직 실행되지도 않았을때, 다른 스레드의 getInstance 메소드가 실행되어 Singleton 객체가 생성되지 않았다고 판단하여 여러개의 객체가 생성된다. 따라서, 위 경우에는 서로다른 Singleton객체가 5개가 생성되었다.

Synchronized

```
public class Singleton {
    private static Singleton inst = null;
    private Singleton() { System.out.println("Threadsafe Singleton Constructor"); }

    public static synchronized Singleton getInstance(){
        if(inst == null){
            inst = new Singleton();
        }
        return inst;
    }
    public void print() { System.out.println("Threadsafe Singleton instatnce hashco
}
```

```
Threadsafe Singleton Constructor
Threadsafe Singleton instatnce hashcode : 1085194098
Threadsafe Singleton instatnce hashcode : 1085194098
Threadsafe Singleton instatnce hashcode : 1085194098
Threadsafe Singleton instatnce hashcode : 1085194098
40aebf72
40aebf72
40aebf72
Threadsafe Singleton instatnce hashcode : 1085194098
40aebf72
40aebf72
78ms
```

synchronized 방법의 경우는 컴파일시점에 객체를 생성하는 것이 아닌, 객체가 필요한 시점에 요청하여 동적바인딩을 통해 객체를 생성합니다. 이런 synchronized 방식은 Thread-Safe 하다는 장점이 있지만, 객체가 생성되었든, 안되었든 무조건 synchronized 블록을 거친다는 단점이 있다.

Eage Initialization

```
public class Singleton {
    private static Singleton inst = new Singleton();
    private Singleton(){
        System.out.println("Eager Initialization Constructor");
    }

    public static Singleton getInstance(){
        return inst;
    }
    public void print() { System.out.println("Eager Initialization hashcode
}
```

```
Eager Initialization Constructor
Eager Initialization hashcode : 415216288
Eager Initialization hashcode : 415216288
18bfb2a0
Eager Initialization hashcode : 415216288
18bfb2a0
18bfb2a0
Eager Initialization hashcode : 415216288
18bfb2a0
Eager Initialization hashcode : 415216288
18bfb2a0
87ms
```

Eager Initialization 방식은, static 키워드의 특징을 이용해서 클래스가 초기화하는 시점에서 정적 바인딩을 통

해 해당 객체를 메모리에 등록해서 사용하는 방법이다. 클래스가 초기화되는 시점에 객체가 생성되기때문에 Thread-safe하다.

Lazy Initialization Double Checking Locking

```
public class Singleton {
    private static volatile Singleton inst = null;
    private Singleton() { System.out.println("LazyInitialization Constructor"); }

    public static Singleton getInstance(){
        if(inst == null){
            synchronized (Singleton.class){
                if(inst == null){
                    inst = new Singleton();
                }
            }
        }
        return inst;
    }
    public void print() { System.out.println("LazyInitialization singleton instatn
}
```

```
LazyInitialization Constructor
LazyInitialization singleton instatnce hashcode : 1085194098
LazyInitialization singleton instatnce hashcode : 1085194098
40aebf72
40aebf72
LazyInitialization singleton instatnce hashcode : 1085194098
40aebf72
LazyInitialization singleton instatnce hashcode : 1085194098
40aebf72
LazyInitialization singleton instatnce hashcode : 1085194098
40aebf72
87ms
```

Lazy Initialization Double Checking Locking 방식은 객체가 생성되지 않은 경우에만 동기화 블럭이 실행되게끔 구현한 방식이다. 멀티스레드 환경에서는 스레드가 변수 값을 읽어올 때 각각의 CPU Cache에 저장된 값이 다르기때문에 변수 값 불일치 문제가 발생하게 되는데, volatile 키워드를 통해 이 문제를 해결한다.

InnerStatic

```
public class Singleton {
    private static class SingletonHolder{
        private static final Singleton inst = new Singleton();
    }

    private Singleton() { System.out.println("InnerStatic Singleton Constructor"); }

    public static Singleton getInstance() { return SingletonHolder.inst; }
    public void print(){
        System.out.println("InnerStatic singleton instatnce hashcode : "+SingletonHo
    }
}
```

```
InnerStatic Singleton Constructor
InnerStatic singleton instatnce hashcode : 1150362559
InnerStatic singleton instatnce hashcode : 1150362559
InnerStatic singleton instatnce hashcode : 1150362559
InnerStatic singleton instatnce hashcode : 1150362559
449123bf
InnerStatic singleton instatnce hashcode : 1150362559
449123bf
449123bf
449123bf
449123bf
85ms
```

InnerStatic 방식은 volatile이나 synchronized 키워드 없이 동시성 문제를 해결하기 때문에 성능이 뛰어나다. 싱글톤 클래스에는 SingletonHolder 클래스의 변수가 없기 때문에, static 멤버 클래스라도, 클래스 초기화 과정을 진행할 때 SingletonHolder를 초기화하지 않고, getInstance( ) 메소드를 호출할 때 초기화 된다. 동적바인딩의 특징을 이용하였기 때문에 Thread-safe한 동시에 성능이 뛰어나다. 또 final 키워드를 썼기때문에 객체가 다시 할당되지 않는다.

[추가코드]

Enum

```
public enum Singleton {
    INSTANCE;

    public static Singleton getInstance(){
        return INSTANCE;
    }
    public void print(){
        System.out.println("Enum singleton instatnce hashcode : "+INSTANCE.hashCode
    }
}
```

```
Enum singleton instatnce hashcode : 1085194098
Enum singleton instatnce hashcode : 1085194098
Enum singleton instatnce hashcode : 1085194098
40aebf72
Enum singleton instatnce hashcode : 1085194098
40aebf72
Enum singleton instatnce hashcode : 1085194098
40aebf72
40aebf72
40aebf72
77ms
```

enum 방식은 enum 객체가 생성될때 기본적으로 Thread-Safe하다. 따라서 스레드에 관련된 코드가 사라져 코

드가 간단해진다. 또한, 아주복잡한 직렬화 상황이나, 리플렉션 공격에 제2의 객체가 생성되는 것을 막아준다고 한다. 하지만, enum 내의 다른 메소드가 있는 경우에 해당 메소드가 Thread-safe한지는 불분명하고, 만들려는 싱글톤이 enum외의 다른 클래스를 상속해야하는 경우에는 사용할 수 없다.