SWE-4501: Design Pattern

Singleton Design Pattern

Md. Nazmul Haque
Lecturer, IUT

Department of Computer Science and Engineering
Islamic University of Technology

July 29, 2021

Contents

- Motivation
- Solution

Real life examples

- Printer
- Database connection
- Logger
- Cache memory

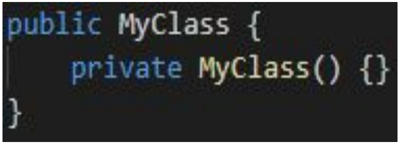
Creation of a single instance

new myObject();

Can we instantiate it one or more times?

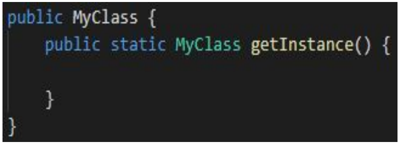
Yes, if it is a public class. If not, classes in the same package can instantiate it one or more times.

Implementation of a class

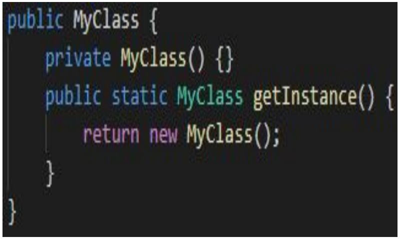

```
public MyClass {
    private MyClass() {}
}
```

Can we instantiate it one or more times?

The code **in the MyClass** can instantiate it.

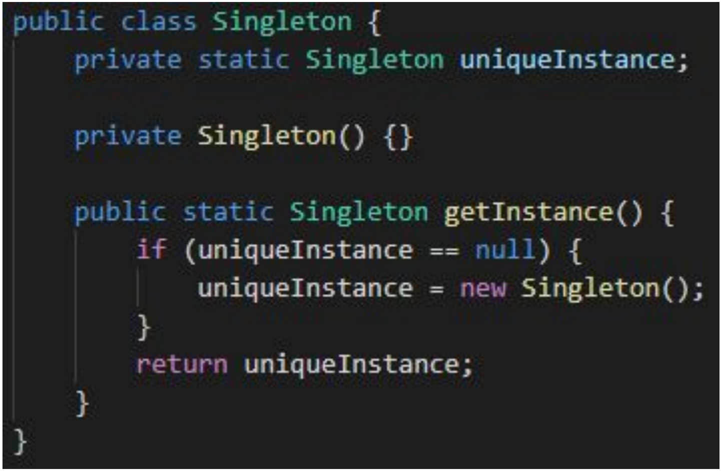

```
public MyClass {
    public static MyClass getInstance() {

    }
}
```

Call static method: **MyClass.getInstance()**


```
public MyClass {
    private MyClass() {}
    public static MyClass getInstance() {
        return new MyClass();
    }
}
```

Call CLASS method to instantiate an object: **MyClass.getInstance()**

Implementation of Singleton class



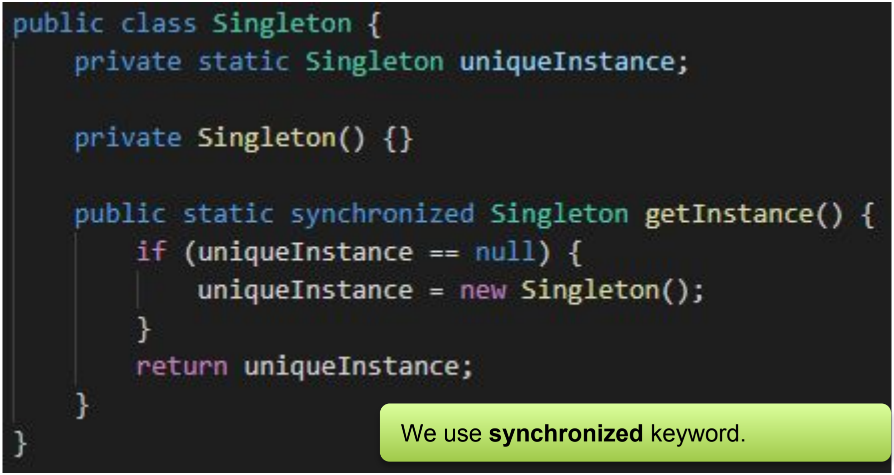
```java
public class Singleton {
    private static Singleton uniqueInstance;

    private Singleton() {}

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

The Singleton Pattern ensures a class has only one instance, and provides a global point of access to it.

MultiThreading of Singleton class

Thread ONE	Thread TWO	Value of uniqueInstance
public static Singleton getInstance() {		null
	public static Singleton getInstance() {	null
if(uniqueInstance==null) {		null
	if(uniqueInstance==null) {	null
uniqueInstance = new Singleton()		<object**1**>
return uniqueInstance		<object**1**>
	uniqueInstance = new Singleton()	<object**2**>
	return uniqueInstance	<object**2**>

Dealing with Multithreading of Singleton class



```java
public class Singleton {
    private static Singleton uniqueInstance;

    private Singleton() {}

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

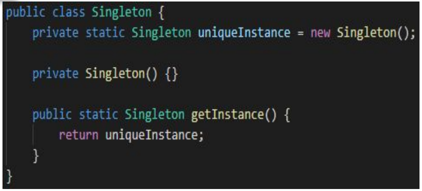
We use **synchronized** keyword.

After the first time through, synchronization is totally unneeded overhead!

Improvement of multithreading

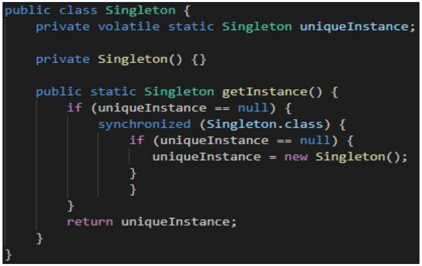
i. Do nothing if the performance of getInstance() is not critical to your application.

ii. Move to an eagerly created instance rather than a lazily created one



```java
public class Singleton {
    private static Singleton uniqueInstance = new Singleton();

    private Singleton() {}

    public static Singleton getInstance() {
        return uniqueInstance;
    }
}
```

iii. Use "double-checked locking" to reduce the use of synchronization in getInstance()



```java
public class Singleton {
    private volatile static Singleton uniqueInstance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            synchronized (Singleton.class) {
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```

ANY QUESTION ?
THANK YOU !

Acknowledgements

[1] Gamma, Erich. Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional, 1 edition, 1994.

[2] Freeman, Eric, et al. Head first design patterns. " O'Reilly Media, Inc.", 2008.

[3] TutorialsPoint

[4] GeeksforGeeks