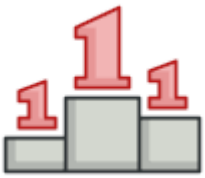

Singleton in Java

Singleton is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code.

Singleton has almost the same pros and cons as global variables. Although they're super-handy, they break the modularity of your code.

You can't just use a class that depends on a Singleton in some other context, without carrying over the Singleton to the other context. Most of the time, this limitation comes up during the creation of unit tests.

Learn more about Singleton →

Navigation

OutputDemoMultiThread

Want more?

Complexity: ★☆☆

Popularity: ★★☆

Usage examples: A lot of developers consider the Singleton pattern an antipattern. That's why its usage is on the decline in Java code.

Despite this, there are quite a lot of Singleton examples in Java core libraries:

- `java.lang.Runtime#getRuntime()`
- `java.awt.Desktop#getDesktop()`
- `java.lang.System#getSecurityManager()`

Identification: Singleton can be recognized by a static creation method, which returns the same cached object.

Naïve Singleton (single-threaded)

It's pretty easy to implement a sloppy Singleton. You just need to hide the constructor and implement a static creation method.

Singleton.java: Singleton

```
package refactoring_guru.singleton.example.non_thread_safe;

public final class Singleton {
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // The following code emulates slow initialization.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
```

```
    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }
}
```

DemoSingleThread.java: Client code

```
package refactoring_guru.singleton.example.non_thread_safe;

public class DemoSingleThread {
    public static void main(String[] args) {
        System.out.println("If you see the same value, then singleton was reused (yay!)" +
                "If you see different values, then 2 singletons were created (booo!!)" +
                "RESULT:" + "\n");
        Singleton singleton = Singleton.getInstance("FOO");
        Singleton anotherSingleton = Singleton.getInstance("BAR");
        System.out.println(singleton.value);
        System.out.println(anotherSingleton.value);
    }
}
```

OutputDemoSingleThread.txt: Execution result

```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)

RESULT:

FOO
FOO
```

Naïve Singleton (multithreaded)

The same class behaves incorrectly in a multithreaded environment. Multiple threads can call the creation method simultaneously and get several instances of Singleton class.