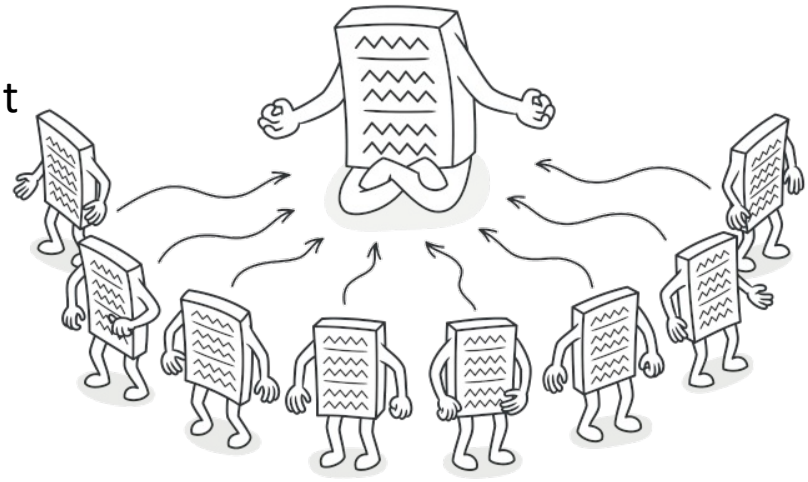
Advanced Software Engineering Part 06 – Singleton Pattern

Dr. Amjad AbuHassan

Intent

- **Singleton** is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.



Problem

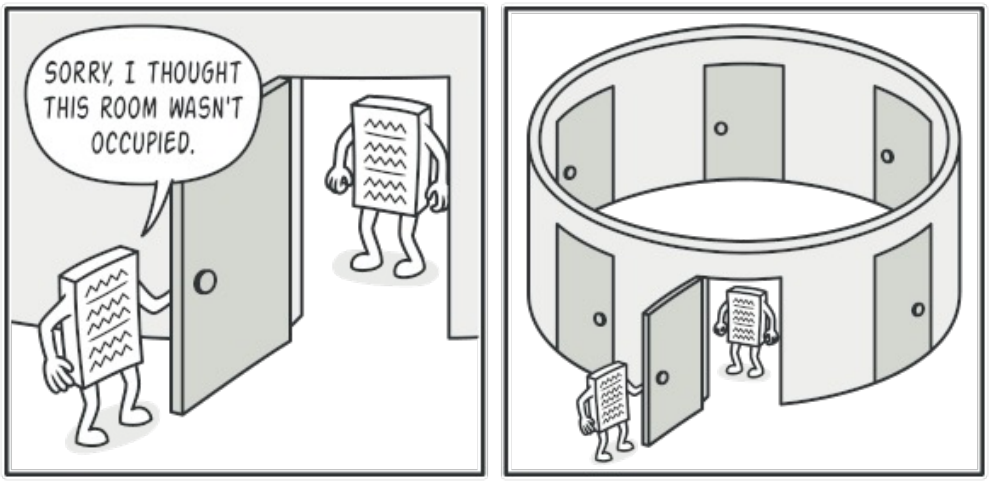
- The Singleton pattern solves two problems at the same time, *violating the Single Responsibility Principle*:
  - Ensure that a class has just a single instance
  - Provide a global access point to that instance

Single Instance

- Why would anyone want to control how many instances a class has?
  - The most common reason for this is to control access to some shared resource—for example, a database or a file.
- Imagine that you created an object, but after a while decided to create a new one. Instead of receiving a fresh object, you'll get the one you already created.
- Note that this behavior is impossible to implement with a regular constructor since a constructor call must always return a new object by design.

Single Instance cont.

Clients may not even realize that they’re working with the same object all the time.



Global Access

- Just like a global variable, the Singleton pattern lets you access some object from anywhere in the program.
  - However, it also protects that instance from being overwritten by other code.

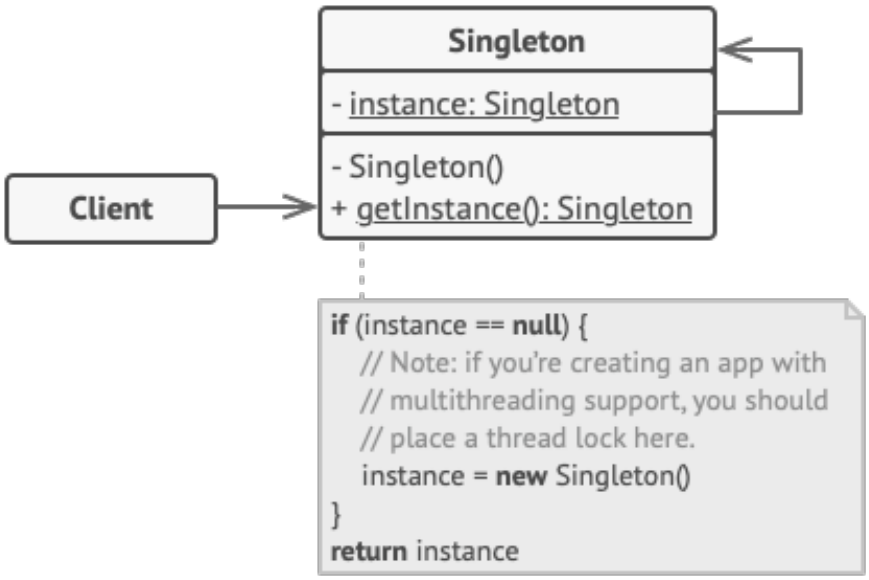
Solution

- Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
- Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

Real-World Analogy

The government is an excellent example of the Singleton pattern. A country can have only one official government. Regardless of the personal identities of the individuals who form governments, the title, “The Government of X”, is a global point of access that identifies the group of people in charge.

Structure



Pseudocode

```
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;
    }
}
```

Pseudocode cont.

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

```
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
```


Applicability

Use the Singleton pattern when a class in your program should have just a single instance available to all clients; for example, a single database object shared by different parts of the program.

- The Singleton pattern disables all other means of creating objects of a class except for the special creation method. This method either creates a new object or returns an existing one if it has already been created.

Applicability cont.

Use the Singleton pattern when you need stricter control over global variables.

- Unlike global variables, the Singleton pattern guarantees that there's just one instance of a class. Nothing, except for the Singleton class itself, can replace the cached instance.
- Note that you can always adjust this limitation and allow creating any number of Singleton instances. The only piece of code that needs changing is the body of the *getInstance* method.