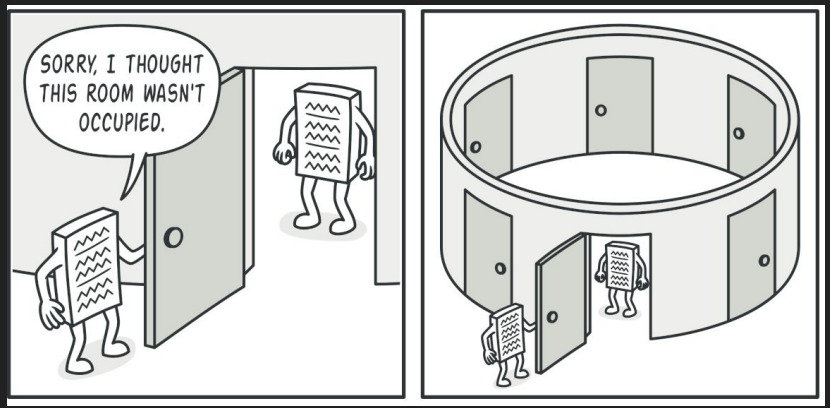
Singleton Design Pattern

Owen Forsyth, William Huynh, and Tarun Kannan

What is the Singleton design pattern?

Singleton is a design pattern that ensures that a class has only one instance while providing a global access point to it.



What is it used for?

- Logging
- Driver objects
- Caching
- Thread pool
- Database connections

Why use Singleton?

- Reduces memory usage and initialization time
- Makes testing and debugging code easier
- Control object creation

How to implement it

1. Create a class
2. Declare a private static object to store the instance
3. Create a private constructor to prevent it from being instantiated by other objects outside the class
4. Create a public static method to check if there is an instance of that class
5. If object has no value, make a new instance, but if object has value, return its reference

Java Example

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

    // private constructor to force use of getInstance() to create Singleton object
    private Singleton() {
    }

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

    public void helloWorld() {
        System.out.println("Hello, World!");
    }
}
```

Java Example

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

    // private constructor to force use of
    // getInstance() to create Singleton object
    private Singleton() {
    }

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

    public void helloWorld() {
        System.out.println("Hello, World!");
    }
}
```

You will not be able to make a new instance of it using the “new” keyword:

~~Singleton single = new Singleton();~~

Instead, you can declare it by:

Singleton single = Singleton.getInstance();

Calling a method:

single.helloWorld();

Pros and Cons

Pros

- Makes sure only one instance is contained in an object for when exactly one object is needed to coordinate actions in a system
- Controls an object's instantiation
- Gives easy access to given instance from anywhere in application
- Instance is only created after accessed once
- Lets multiple threads access instance without leading to concurrency problems

Cons

- Introduces global state into application, many times unnecessarily
- Other objects rely on it often
- Difficulties with unit testing due to this coupling
- Violates the single-responsibility principle because of enforcing their uniqueness and performing their functions

How Design Can be Used in Real Life

- Logging: records events and debugs, makes sure only one insurance of logger and it can be easily accessed by any object anywhere in the application
- Database connection: making a single, shared connection for multiple components, gives centralized point for managing the connection
- Configuration manager: loads and manages configuration data so all parts of application can access same data
- Cache manager: makes sure only one cache instance exists and can be accessed from anywhere in application
- Thread pool: manage creation, allocation, execution of threads, giving shared resource for handling concurrent tasks

Sources

https://www.oracle.com/technical-resources/articles/java/singleton.html

https://www.geeksforgeeks.org/singleton-design-pattern/

https://en.wikipedia.org/wiki/Singleton_pattern

https://refactoring.guru/design-patterns/singleton

https://www.linkedin.com/advice/0/how-does-singleton-pattern-improve-your-code-skills-design-patterns

https://en.wikipedia.org/wiki/Software_design_pattern