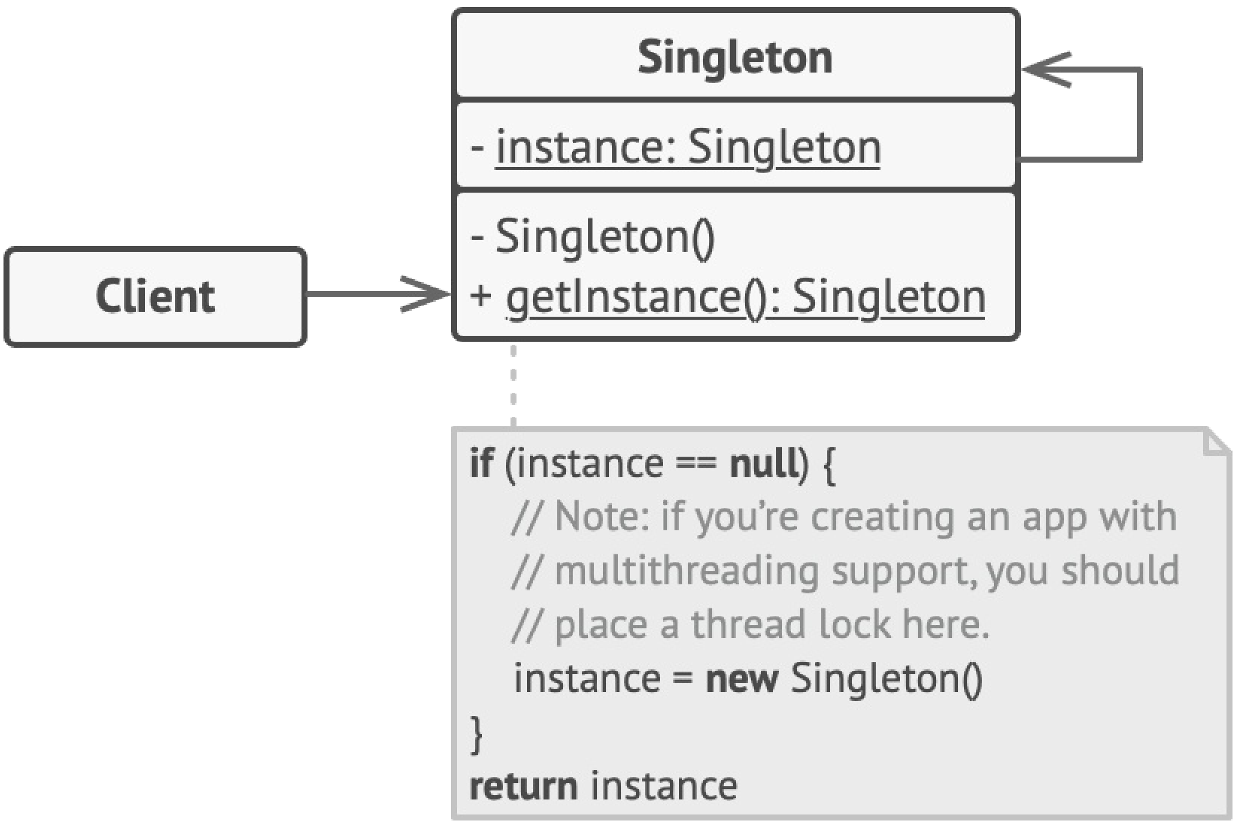
Singleton Design Pattern

- 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.
- Main Problem this pattern solves is to ensure that only a single instance of this class exists
- Any state you add in your singleton, becomes part of "global state" of your application

UML diagram



```
if (instance == null) {
    // Note: if you're creating an app with
    // multithreading support, you should
    // place a thread lock here.
    instance = new Singleton()
}
return instance
```

Implementation Consideration

1. Controlling instance creating
    1. Class constructor must not be accessible globally
    2. Subsclassing/inheritance must not be allowed
2. Keeping tract of instance
    1. Class itself is a good place to keep track of instance
3. Giving access to the singleton instance
    1. A public static method is a good choice
    2. Can expose instance as final public static field but it won't work for all singletom implementations.
4. Two options for implementing a singleton

1. Eager Singleton : Create a singleton as soon as class is loaded
2. Lazy Singleton: Create a singleton when it is first required

Design Considerations

1. Singleton creation does not need any parameter. If you find yourself in need of support for constructor arguments, you need a simple factory or factory method pattern instead.
2. Make Sure singletons are not carrying a lot of mutable global state.

Applicability

1. 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.
2. Use the Singleton pattern when you need stricter control over global variables.

Pros and Cons

Pros	Cons
You can be sure that a class has only a single instance.	Violates the Single Responsibility Principle since the objects control how they are created and manage their life-cycle.
You gain a global access point to that instance.	The pattern requires special treatment in a multithreaded environment so that multiple threads won't create a singleton object several times.
The singleton object is initialized only when it's requested for the first time.	It may be difficult to unit test the client code of the Singleton because many test frameworks rely on inheritance when producing mock objects.

Reference

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