



SUMMER SALE



😊 Solution

All implementations of the Singleton have these two steps in common:

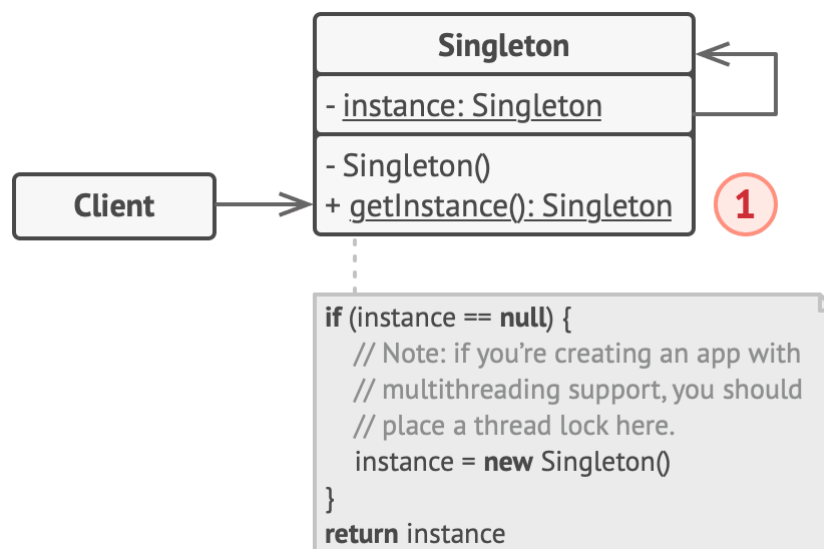
- Make the default constructor private, to prevent other objects from using the `new` operator with the Singleton class.
- Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following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

If your code has access to the Singleton class, then it's able to call the Singleton's static method. So whenever that method is called, the same object is always returned.

🚗 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





SUMMER SALE



of its own class.

The Singleton's constructor should be hidden from the client code. Calling the `getInstance` method should be the only way of getting the Singleton object.

Pseudocode

In this example, the database connection class acts as a **Singleton**. This class doesn't have a public constructor, so the only way to get its object is to call the `getInstance` method. This method caches the first created object and returns it in all subsequent calls.

```
// The Database class defines the `getInstance` method that lets
// clients access the same instance of a database connection
// throughout the program.
class Database is
    // The field for storing the singleton instance should be
    // declared static.
    private static field instance: Database

    // The singleton's constructor should always be private to
    // prevent direct construction calls with the `new`
    // operator.
    private constructor Database() is
        // Some initialization code, such as the actual
        // connection to a database server.
        // ...

    // The static method that controls access to the singleton
    // instance.
    public static method getInstance() is
        if (Database.instance == null) then
            acquireThreadLock() and then
                // Ensure that the instance hasn't yet been
                // initialized by another thread while this one
                // has been waiting for the lock's release.
                if (Database.instance == null) then
                    Database.instance = new Database()
            return Database.instance

    // Finally, any singleton should define some business logic
    // which can be executed on its instance.
    public method query(sql) is
        // For instance, all database queries of an app go
        // through this method. Therefore, you can place
        // throttling or caching logic here.
        // ...
```