COMP2511

Creational Pattern:

Singleton Pattern

Prepared by

Dr. Ashesh Mahidadia

Creational Patterns

Creational patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

- Factory Method
  - provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
- Abstract Factory
  - let users produce families of related objects without specifying their concrete classes.
- Singleton
  - Let users ensure that a class has only one instance, while providing a global access point to this instance.
- Builder
  - let users construct complex objects step by step. The pattern allows users to produce different types and representations of an object using the same construction code.

Singleton Pattern

Singleton Pattern

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: A client wants to,

- ensure that a class has just a single instance, and
- provide a global access point to that instance

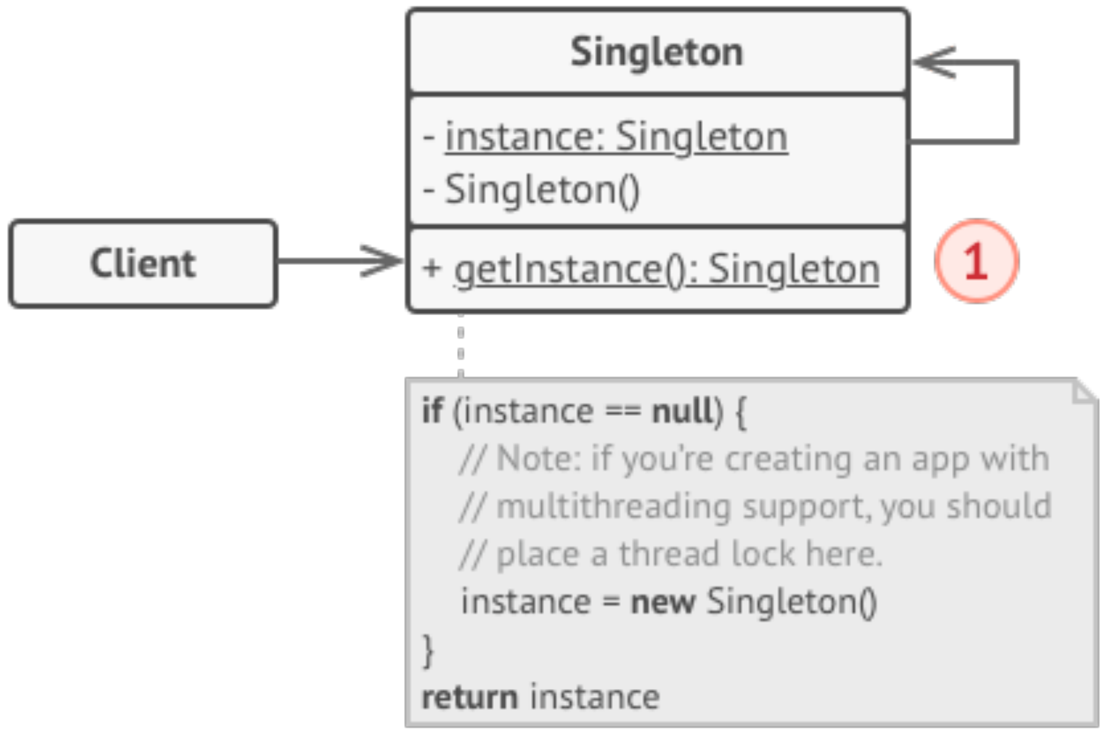
Solution:

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.
- Whenever Singleton's static method is called, the same object is always returned.

Singleton: Structure

- The Singleton class declares the static method *getInstance* (1) that returns the same instance of its own class.
- The Singleton's constructor should be hidden from the client code.
- Calling the *getInstance* (1) method should be the only way of getting the Singleton object.



```
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
```

Singleton: How to Implement

- Add a private static field to the class for storing the singleton instance.
- Declare a public static creation method for getting the singleton instance.
- Implement “lazy initialization” inside the static method.
  - It should create a new object on its first call and put it into the static field.
  - The method should always return that instance on all subsequent calls.
- Make the constructor of the class private.
  - The static method of the class will still be able to call the constructor, but not the other objects.
- In a client, call singleton’s static creation method to access the object.

Example in **Java (MUST read)**:

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

Singleton Pattern

For more information, read:

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

End