Software Design Patterns

Lecture 5

Singleton

Dr. Fan Hongfei

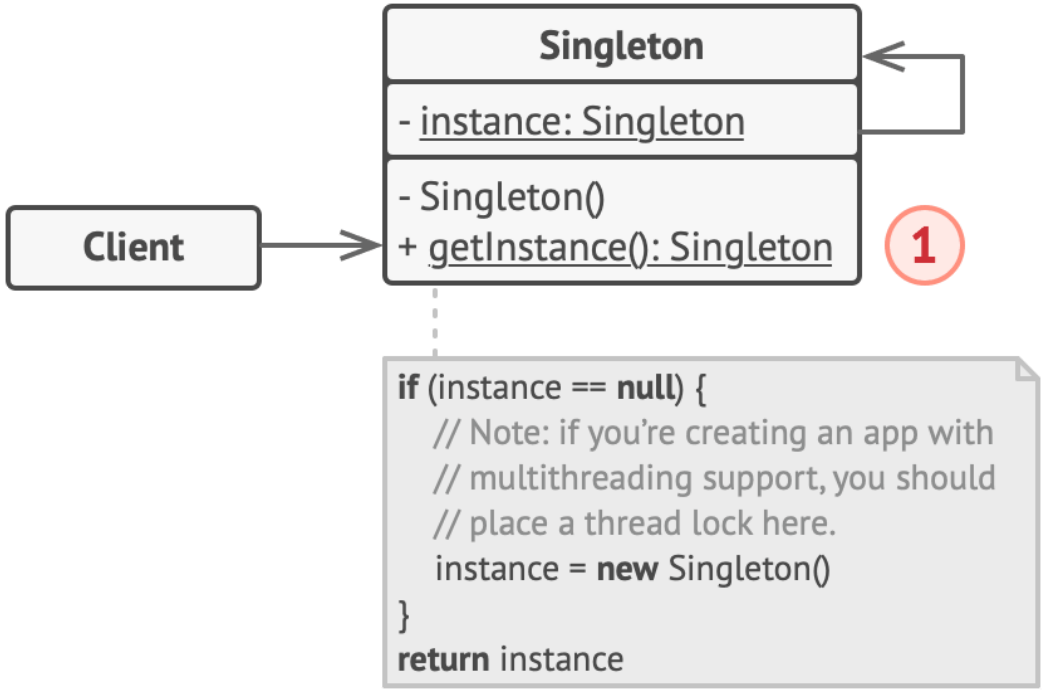
10 October 2024

Singleton: Problem

1. **Ensure that a class has just one single instance**
   - Common reason: control access to a shared resource
   - Impossible to implement with a regular constructor
2. **Provide a global access point to that instance**
   - A global variable (storing essential objects) is unsafe, since any code can potentially overwrite the contents
   - The code that solves the first problem should not be scattered all over the program

Singleton: Solution and Structure

1. Make the **default constructor private**, to prevent other objects from using the new operator
2. Create a **static creation method** that acts as a constructor
   - This method calls the private constructor to create an object and saves it in a static field
3. All following calls to this method **return the cached object**



- Note: multithreading issue

[Figures in this slide are extracted from https://refactoring.guru/design-patterns/singleton]

Singleton: Applicability

- When a class in your program should have just a single instance available to all clients
  - The Singleton pattern disables all other means of creating objects
- When you need stricter control over global variables
  - The Singleton pattern guarantees that there is just one instance of a class
- Note: you can always adjust this limitation and allow creating **any number of Singleton instances**, by changing the body of the getInstance method

Singleton: Implementation

1. Add **a private static field** for storing the singleton instance
2. Declare a **public static creation method** for getting the singleton instance
3. Implement "**lazy initialization**" inside the static method
4. Make the **constructor private**
   - Only the static creation method of the class will be able to call the constructor, but not others
5. Go over the client code and replace all direct calls to the singleton's constructor with calls to its static creation method

Singleton: Pros and Cons

- **Pros**
  - You can be sure that a class has only a single instance
  - You gain a global access point to that instance
  - The singleton object is initialized only when it is requested for the first time
- **Cons**
  - Violates the Single Responsibility Principle
  - The Singleton pattern can mask bad design, for instance, when the components of the program know too much about each other
  - The pattern requires special treatment in a multithreaded environment
  - It may be difficult to unit test the client code of the Singleton because many test frameworks rely on inheritance when producing mock objects