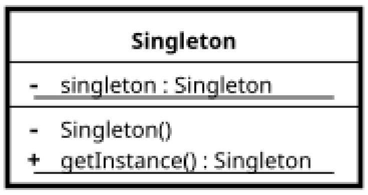
WIKIPEDIA
The Free Encyclopedia
WIKIPEDIA

Singleton pattern

In object-oriented programming, the **singleton pattern** is a software design pattern that restricts the instantiation of a class to a singular instance. It is one of the well-known "Gang of Four" design patterns, which describe how to solve recurring problems in object-oriented software.[1] The pattern is useful when exactly one object is needed to coordinate actions across a system.



A class diagram exemplifying the singleton pattern.

More specifically, the singleton pattern allows classes to:[2]

- Ensure they only have one instance
- Provide easy access to that instance
- Control their instantiation (for example, hiding the constructors of a class)

The term comes from the mathematical concept of a singleton.

Common uses

Singletons are often preferred to global variables because they do not pollute the global namespace (or their containing namespace). Additionally, they permit lazy allocation and initialization, whereas global variables in many languages will always consume resources.[1][3]

The singleton pattern can also be used as a basis for other design patterns, such as the abstract factory, factory method, builder and prototype patterns. Facade objects are also often singletons because only one facade object is required.

Logging is a common real-world use case for singletons, because all objects that wish to log messages require a uniform point of access and conceptually write to a single source.[4]

Implementations

Implementations of the singleton pattern ensure that only one instance of the singleton class ever exists and typically provide global access to that instance.

Typically, this is accomplished by:

- Declaring all constructors of the class to be private, which prevents it from being instantiated by other objects
- Providing a static method that returns a reference to the instance

The instance is usually stored as a private static variable; the instance is created when the variable is initialized, at some point before when the static method is first called.

This C++11 implementation is based on the pre C++98 implementation in the book .

```
#include <iostream>

class Singleton {
public:
  // defines an class operation that lets clients access its unique instance.
  static Singleton& get() {
    // may be responsible for creating its own unique instance.
    if (nullptr == instance) instance = new Singleton;
    return *instance;
  }
  Singleton(const Singleton&) = delete; // rule of three
  Singleton& operator=(const Singleton&) = delete;
  static void destruct() {
    delete instance;
    instance = nullptr;
  }
  // existing interface goes here
  int getValue() {
    return value;
  }
  void setValue(int value_) {
    value = value_;
  }
private:
  Singleton() = default; // no public constructor
  ~Singleton() = default; // no public destructor
  static Singleton* instance; // declaration class variable
  int value;
};

Singleton* Singleton::instance = nullptr; // definition class variable

int main() {
  Singleton::get().setValue(42);
  std::cout << "value=" << Singleton::get().getValue() << '\n';
  Singleton::destruct();
}
```

The program output is

```
value=42
```

This is an implementation of the Meyers singleton[5] in C++11. The Meyers singleton has no destruct method. The program output is the same as above.

```
#include <iostream>

class Singleton {
public:
  static Singleton& get() {
    static Singleton instance;
    return instance;
  }
  int getValue() {
    return value;
  }
  void setValue(int value_) {
    value = value_;
  }
private:
  Singleton() = default;
  ~Singleton() = default;
  int value;
};

int main() {
  Singleton::get().setValue(42);
```

```
    std::cout << "value=" << Singleton::get().getValue() << '\n';
}
```

Lazy initialization

A singleton implementation may use lazy initialization in which the instance is created when the static method is first invoked. In multithreaded programs, this can cause race conditions that result in the creation of multiple instances. The following Java 5+ example[6] is a thread-safe implementation, using lazy initialization with double-checked locking.

```
public class Singleton {

    private static volatile Singleton instance = null;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized(Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

Criticism

Some consider the singleton to be an anti-pattern that introduces global state into an application, often unnecessarily. This introduces a potential dependency on the singleton by other objects, requiring analysis of implementation details to determine whether a dependency actually exists.[7] This increased coupling can introduce difficulties with unit testing.[8] In turn, this places restrictions on any abstraction that uses the singleton, such as preventing concurrent use of multiple instances.[8][9][10]

Singletons also violate the single-responsibility principle because they are responsible for enforcing their own uniqueness along with performing their normal functions.[8]

See also

- Initialization-on-demand holder idiom
- Multiton pattern
- Software design pattern

References

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software* (https://archive.org/details/designpatternsel00gamm/page/127). Addison Wesley. pp. 127ff (https://archive.org/details/designpatternsel00gamm/page/127). ISBN 0-201-63361-2.

2. "The Singleton design pattern - Problem, Solution, and Applicability" (http://w3sdesign.com/?gr=c05&ugr=proble). *w3sDesign.com*. Retrieved 2017-08-16.
3. Soni, Devin (31 July 2019). "What Is a Singleton?" (https://betterprogramming.pub/what-is-a-singleton-2dc38ca08e92). *BetterProgramming*. Retrieved 28 August 2021.
4. Rainsberger, J.B. (1 July 2001). "Use your singletons wisely" (https://web.archive.org/web/20210224180356/https://www.ibm.com/developerworks/library/co-single/). IBM. Archived from the original (https://www.ibm.com/developerworks/library/co-single/) on 24 February 2021. Retrieved 28 August 2021.
5. Scott Meyers (1997). *More Effective C++*. Addison Wesley. pp. 146 ff. ISBN 0-201-63371-X.
6. Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates (October 2004). "5: One of a Kind Objects: The Singleton Pattern" (https://books.google.com/books?id=GGpXN9SMELMC&pg=PA182). *Head First Design Patterns* (First ed.). O'Reilly Media, Inc. p. 182. ISBN 978-0-596-00712-6.
7. "Why Singletons Are Controversial" (https://web.archive.org/web/20210506162753/https://code.google.com/archive/p/google-singleton-detector/wikis/WhySingletonsAreControversial.wiki). *Google Code Archive*. Archived from the original (https://code.google.com/archive/p/google-singleton-detector/wikis/WhySingletonsAreControversial.wiki) on 6 May 2021. Retrieved 28 August 2021.
8. Button, Brian (25 May 2004). "Why Singletons are Evil" (https://web.archive.org/web/20210715184717/https://docs.microsoft.com/en-us/archive/blogs/scottdensmore/why-singletons-are-evil). *Being Scott Densmore*. Microsoft. Archived from the original (https://docs.microsoft.com/en-us/archive/blogs/scottdensmore/why-singletons-are-evil) on 15 July 2021. Retrieved 28 August 2021.
9. Steve Yegge. Singletons considered stupid (http://steve.yegge.googlepages.com/singleton-considered-stupid), September 2004
10. Hevery, Miško, "Global State and Singletons (http://googletesting.blogspot.com/2008/11/clean-code-talks-global-state-and.html)", *Clean Code Talks*, 21 November 2008.

External links

- Complete article "Java Singleton Pattern Explained (https://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/)"
- Four different ways to implement singleton in Java "Ways to implement singleton in Java (https://web.archive.org/web/20150709155148/http://www.javaexperience.com/design-patterns-singleton-design-pattern/)"
- Book extract: Implementing the Singleton Pattern in C# (https://csharpindepth.com/Articles/Singleton) by Jon Skeet
- Singleton at Microsoft patterns & practices Developer Center (https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650849(v=pandp.10))
- IBM article "Double-checked locking and the Singleton pattern (https://www.ibm.com/developerworks/library/j-dcl/)" by Peter Haggar
- Geary, David (April 25, 2003). "How to navigate the deceptively simple Singleton pattern" (https://www.infoworld.com/article/2073352/core-java-simply-singleton.html). Java Design Patterns. *JavaWorld*. Retrieved 2020-07-21.
- Google Singleton Detector (https://code.google.com/archive/p/google-singleton-detector/) (analyzes Java bytecode to detect singletons)