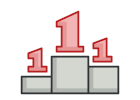
-

-

-

 / [Design Patterns](https://refactoring.guru/design-patterns) / [Singleton](https://refactoring.guru/design-patterns/singleton) / [C#](https://refactoring.guru/design-patterns/csharp)



**Singleton in C#**

**Singleton** is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code.

Singleton has almost the same pros and cons as global variables. Although they’re super-handy, they break the modularity of your code.

You can’t just use a class that depends on a Singleton in some other context, without carrying over the Singleton to the other context. Most of the time, this limitation comes up during the creation of unit tests.

[Learn more about Singleton](https://refactoring.guru/design-patterns/singleton)

**Complexity:**

**Popularity:**

**Usage examples:** A lot of developers consider the Singleton pattern an antipattern. That’s why its usage is on the decline in C# code.

**Identification:** Singleton can be recognized by a static creation method, which returns the same cached object.

[Naïve Singleton](https://refactoring.guru/design-patterns/singleton/csharp/example#example-0)[Thread-safe Singleton](https://refactoring.guru/design-patterns/singleton/csharp/example#example-1)[Want more?](https://refactoring.guru/design-patterns/singleton/csharp/example#example-2)

**Naïve Singleton**

It’s pretty easy to implement a sloppy Singleton. You just need to hide the constructor and implement a static creation method.

**The same class behaves incorrectly in a multithreaded environment. Multiple threads can call the creation method simultaneously and get several instances of Singleton class.**

-

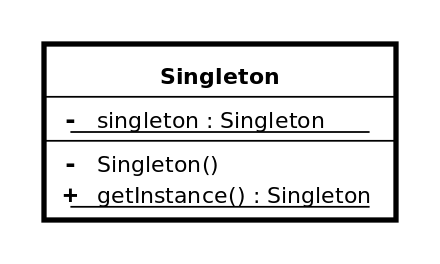
-

-

# Singleton pattern

#### *Connected to:*

[Design Patterns](https://www.wikiwand.com/en/Design_Patterns)[Software design pattern](https://www.wikiwand.com/en/Software_design_pattern)[Class diagram](https://www.wikiwand.com/en/Class_diagram)



A [class diagram](https://www.wikiwand.com/en/Class_diagram) exemplifying the singleton pattern.

In [software engineering](https://www.wikiwand.com/en/Software_engineering), the **singleton pattern** is a [software design pattern](https://www.wikiwand.com/en/Software_design_pattern) that restricts the [instantiation](https://www.wikiwand.com/en/Instantiation_(computer_science)) of a [class](https://www.wikiwand.com/en/Class_(computer_programming)) to a singular instance. One of the well-known ["**Gang of Four**" design patterns](https://www.wikiwand.com/en/Design_Patterns), which describe how to solve recurring problems in [object-oriented software](https://www.wikiwand.com/en/Object-oriented_programming),[[1]](https://www.wikiwand.com/en/Singleton_pattern#citenoteGoF1) the pattern is useful when exactly one object is needed to coordinate actions across a system.

More specifically, the singleton pattern allows objects to:[[2]](https://www.wikiwand.com/en/Singleton_pattern#citenote2)

* Ensure they only have one instance
* Provide easy access to that instance
* **Control their instantiation (for example, hiding the**[**constructors**](https://www.wikiwand.com/en/Constructor_(object-oriented_programming))**of a**[**class**](https://www.wikiwand.com/en/Class_(computer_programming))**)**

The term comes from the [mathematical concept of a singleton](https://www.wikiwand.com/en/Singleton_(mathematics)).

## Common uses

Singletons are often preferred to [global variables](https://www.wikiwand.com/en/Global_variables) because they do not pollute the global [namespace](https://www.wikiwand.com/en/Namespace) (or their containing namespace).

Additionally, they permit [lazy](https://www.wikiwand.com/en/Lazy_evaluation) allocation and initialization, whereas global variables in many languages will always consume resources.[[1]](https://www.wikiwand.com/en/Singleton_pattern#citenoteGoF1)[[3]](https://www.wikiwand.com/en/Singleton_pattern#citenotedevin3)

**The singleton pattern can also be used as a basis for other design patterns, such as the**[**abstract factory**](https://www.wikiwand.com/en/Abstract_factory_pattern)**,**[**factory method**](https://www.wikiwand.com/en/Factory_method_pattern)**,**[**builder**](https://www.wikiwand.com/en/Builder_pattern)**and**[**prototype**](https://www.wikiwand.com/en/Prototype_pattern)**patterns.**

[**Facade**](https://www.wikiwand.com/en/Facade_pattern)**objects are also often singletons because only one facade object is required.**

[Logging](https://www.wikiwand.com/en/Log_file) 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]](https://www.wikiwand.com/en/Singleton_pattern#citenoterainsberger4)

## Implementations

Implementations of the singleton pattern ensure that only one instance of the singleton class ever exists and typically provide [global access](https://www.wikiwand.com/en/Global_scope) to that instance.

Typically, this is accomplished by:

* Declaring all [constructors](https://www.wikiwand.com/en/Constructor_(object-oriented_programming)) of the class to be [private](https://www.wikiwand.com/en/Private_member), which prevents it from being instantiated by other objects
* Providing a [static method](https://www.wikiwand.com/en/Static_method) that returns a [reference](https://www.wikiwand.com/en/Reference_(computer_science)) to the instance

The instance is usually stored as a private [static variable](https://www.wikiwand.com/en/Static_variable); the instance is created when the variable is initialized, at some point before when the static method is first called.

The following demonstrates an example implementation in [Java](https://www.wikiwand.com/en/Java_(programming_language)):[[5]](https://www.wikiwand.com/en/Singleton_pattern#citenoteghanchigulnaz5)

**public** **class** Coin {

**private** **static** final **int** ADD\_MORE\_COIN = 10;

**private** **int** coin;

**private** **static** Coin instance = **new** Coin(); *// eagerly loads the singleton*

**private** **Coin**() {

***//******private to prevent anyone else from instantiating***

}

**public** **static** Coin **getInstance**() {

**return** instance;

}

**public** **int** **getCoin**() {

**return** coin;

}

**public** **void** **addMoreCoin**() {

coin += ADD\_MORE\_COIN;

}

**public** **void** **deductCoin**() {

coin--;

}

}

### **Lazy initialization**

A singleton implementation may use [**lazy initialization**](https://www.wikiwand.com/en/Lazy_initialization) in which the instance is created **when the static method is first invoked**.

**In**[**multithreaded**](https://www.wikiwand.com/en/Multithreading_(software))**programs, this can cause**[**race conditions**](https://www.wikiwand.com/en/Race_condition)**that result in the creation of multiple instances.**

**The following Java example is a**[**thread-safe**](https://www.wikiwand.com/en/Thread_safety)**implementation, using lazy initialization with**[**double-checked locking**](https://www.wikiwand.com/en/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**](https://www.wikiwand.com/en/Anti-pattern)**that introduces**[**global state**](https://www.wikiwand.com/en/Global_variables)**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.[[6]](https://www.wikiwand.com/en/Singleton_pattern#citenotegoogle6)

This increased [coupling](https://www.wikiwand.com/en/Coupling_(computer_programming)) can introduce **difficulties with**[**unit testing**](https://www.wikiwand.com/en/Unit_testing).[[7]](https://www.wikiwand.com/en/Singleton_pattern#citenotebutton7) In turn, this places restrictions on any abstraction that uses the singleton, such as preventing [concurrent](https://www.wikiwand.com/en/Concurrency_(computer_science)) use of multiple instances.[[7]](https://www.wikiwand.com/en/Singleton_pattern#citenotebutton7)[[8]](https://www.wikiwand.com/en/Singleton_pattern#citenote8)[[9]](https://www.wikiwand.com/en/Singleton_pattern#citenotegoogletestingblogspotcom9)[[10]](https://www.wikiwand.com/en/Singleton_pattern#citenotehackernoon10)

**Singletons also violate the**[**single-responsibility principle**](https://www.wikiwand.com/en/Single-responsibility_principle) **because they are responsible for enforcing their own uniqueness along with performing their normal functions.**[**[7]**](https://www.wikiwand.com/en/Singleton_pattern#citenotebutton7)

-

-

-