**Intent**

* Ensure a class has only one instance, and provide a global point of access to it.
* Encapsulated "just-in-time initialization" or "initialization on first use".

**Problem**

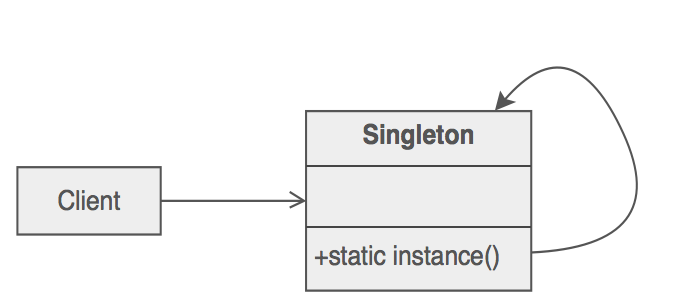
Application needs one, and only one, instance of an object. Additionally, lazy initialization and global access are necessary.

**Discussion**

Singleton should be considered only if all three of the following criteria are satisfied:

* Ownership of the single instance cannot be reasonably assigned
* Lazy initialization is desirable
* Global access is not otherwise provided for

**Structure**

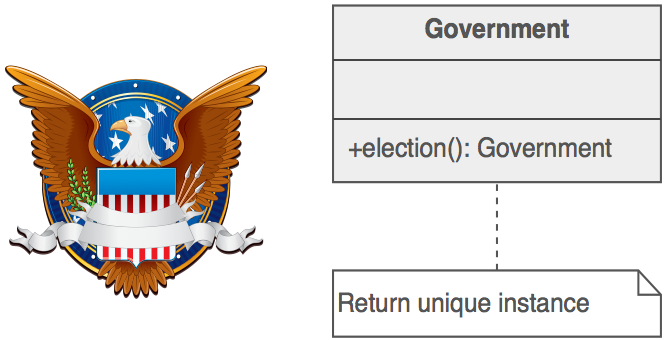


Make the class of the single instance responsible for access and "initialization on first use". The single instance is a private static attribute. The accessor function is a public static method.



**Example**

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. It is named after the singleton set, which is defined to be a set containing one element. The office of the President of the United States is a Singleton. The United States Constitution specifies the means by which a president is elected, limits the term of office, and defines the order of succession. As a result, there can be at most one active president at any given time. Regardless of the personal identity of the active president, the title, "The President of the United States" is a global point of access that identifies the person in the office.



**Check list**

1. Define a private static attribute in the "single instance" class.
2. Define a public static accessor function in the class.
3. Do "lazy initialization" (creation on first use) in the accessor function.
4. Define all constructors to be protected or private.
5. Clients may only use the accessor function to manipulate the Singleton.

**Rules of thumb**

* Abstract Factory, Builder, and Prototype can use Singleton in their implementation.
* Facade objects are often Singletons because only one Facade object is required.
* State objects are often Singletons.
* The advantage of Singleton over global variables is that you are absolutely sure of the number of instances when you use Singleton, and, you can change your mind and manage any number of instances.
* The Singleton design pattern is one of the most inappropriately used patterns. Singletons are intended to be used when a class must have exactly one instance, no more, no less. Designers frequently use Singletons in a misguided attempt to replace global variables. A Singleton is, for intents and purposes, a global variable. The Singleton does not do away with the global; it merely renames it.
* When is Singleton unnecessary? Short answer: most of the time. Long answer: when it's simpler to pass an object resource as a reference to the objects that need it, rather than letting objects access the resource globally. The real problem with Singletons is that they give you such a good excuse not to think carefully about the appropriate visibility of an object. Finding the right balance of exposure and protection for an object is critical for maintaining flexibility.