Provides a single instance of a class, and global point of access to it.

You need to have these conditions fulfilled:

1. Presence of a static member variable – as a placeholder for the instance of that class
2. Locked down the constructor of that class – simply by making it's visibility private
3. Prevent any object or instance of that class to be cloned, that is to prevent any entity within the system to make copy of this object – make it private.
4. Have a single globally accessible static method to access/retrieve the instance of that class – a public static method

Common Uses:

1. Logging, Caching, Configuration, Registry, Filesystem and Database Access, Shared resource (such as an event queue).
2. The Abstract Factory, Builder, and Prototype patterns can use Singletons in their implementation.
3. Facade objects are often singletons because only one Facade object is required.
4. State objects are often singletons.
5. Singletons are often preferred to global variables because:
    1. They do not pollute the global namespace (or, in languages with namespaces, their containing namespace) with unnecessary variables.
    2. They permit lazy allocation and initialization, whereas global variables in many languages will always consume resources.
6.

Cons :

1. Consider it to be an anti-pattern breaks the Single Responsibility Principle.
    - Singleton objects are responsible of both their purpose and controlling the number of instances the produces.
2. Introduces global state into your application.
3. Might be a problem for Unit Testing; since it tests individual objects. Hidden dependencies
    - Visible dependencies - Any parameters accepted by a function
    - Hidden dependencies - Without taking a look at the actual function implementation
4. Performance issues.