**Singleton Design Pattern**

**Define Singleton design pattern:**

To ensures that only one instance of a particular class is going to be created and then provide simple global access to that instance for the entire application.

**Advantages:**

* The first and most important advantage of using the singleton design pattern in C# is that it takes care of concurrent access to the shared resource. That means if we are sharing a resource with multiple clients simultaneously, then concurrent access to that resource is well managed by the singleton design pattern.
* It can be lazy-loaded and also has Static Initialization.
* To share common data i.e. master data and configuration data which is not changed that frequently in an application. In that case, we need to cache the objects in memory.
* As it provides a single global point of access to a particular instance, so it is easy to maintain.
* To reduce the overhead of instantiating a heavy object again and again.

**Real-time scenarios:**

* **Service Proxies:** As we know invoking a Service API is an extensive operation in an application. The process that taking most of the time is creating the Service client in order to invoke the service API. If you create the Service proxy as Singleton then it will improve the performance of your application.
* **Facades:** You can also create the Database connections as Singleton which can improve the performance of the application.
* **Logs:** In an application, performing the I/O operation on a file is an expensive operation. If you create your Logger as Singleton then it will improve the performance of the I/O operation.
* **Data sharing:** If you have any constant values or configuration values then you can keep these values in Singleton So that these can be read by other components of the application.
* **Caching:** As we know fetching the data from a database is a time-consuming process. In your application, you can cache the master and configuration in memory which will avoid the DB calls. In such situations, the Singleton class can be used to handle the caching with thread synchronization in an efficient manner which drastically improves the performance of the application.

**Lazy Initialization in the Singleton Design Pattern:**

In our example, until and unless we invoke the GetInstance Property of the Singleton class, the Singleton instance is not created. That means when we invoke the GetInstance Property of the Singleton class then only the Singleton object is created. This Delay of Singleton Instance creation is called Lazy Initialization.

**How to use Multi threads in Singleton:**

The lazy initialization i.e. the on-demand object creation of the singleton class works fine when we invoke the GetInstance property in a Single-threaded environment. But in a multi-threaded environment, it will not work as expected. In a multi-thread environment, the lazy initialization may end up creating multiple instances of the singleton class when multiple threads invoke the GetInstance property parallelly at the same time which we already discussed in our previous example.

**Disadvantages:**

* Unit testing is very difficult because it introduces a global state into an application.
* It reduces the potential for parallelism within a program because to access the singleton instance in a multi-threaded environment, you need to serialize the object by using locking.

**Eager Loading:**

The Eager loading in singleton design pattern is nothing a process in which we need to initialize the singleton object at the time of application start-up rather than on-demand and keep it ready in memory to be used in the future. The advantage of using Eager Loading in the Singleton design pattern is that the CLR (Common Language Runtime) will take care of object initialization and thread-safety. That means we will not require to write any code explicitly for handling the thread-safety for a multithreaded environment.

**Lazy or Deferred Loading:**

The Lazy or Deferred Loading is a design pattern or you can say its a concept which is commonly used to delay the initialization of an object until the point at which it is needed. So the main objective of Lazy loading is to load the object on-demand or you can say object when needed. The most important point that you need to keep in mind is that, you need to use the Lazy loading when the cost of the object creation is very high as well as the use of that object is very rare. The lazy loading improves the performance of an application if it is used properly. We can use the Lazy keyword to make the singleton instance as lazy loading.

The most important point that you need to remember is the Lazy<T> objects are by default thread-safe. In a multi-threaded environment, when multiple threads are trying to access the same Get Instance property at the same time, then the lazy object will take care of thread safety.

**Singleton vs Static**

|  |  |
| --- | --- |
| **Singleton** | **Static** |
| Clone Possible | Clone Impossible |
| Save🡪 Heap | Save 🡪 Stack |
| Support OOP Concept | Not support |
| Can implement Interface | Can’t implement interface |