# Singleton Design Pattern

Singleton Design Pattern is a creational design pattern that restricts the instantiation of a class and ensures that only one instance of the class exists throughout the application. In other words, the singleton class must provide a global access point to get the instance of the class.

**Use cases of Singleton design pattern:**

* Singleton pattern is used for [**logging**](https://www.digitalocean.com/community/tutorials/logger-in-java-logging-example), drivers objects, caching, and [thread pool](https://www.digitalocean.com/community/tutorials/threadpoolexecutor-java-thread-pool-example-executorservice).
* Singleton design pattern is also used in other design patterns like [Abstract Factory](https://www.digitalocean.com/community/tutorials/abstract-factory-design-pattern-in-java), [Builder](https://www.digitalocean.com/community/tutorials/builder-design-pattern-in-java), [Prototype](https://www.digitalocean.com/community/tutorials/prototype-design-pattern-in-java), [Facade](https://www.digitalocean.com/community/tutorials/facade-design-pattern-in-java), etc.
* Singleton design pattern is used in core Java classes also (for example, java.lang.Runtime, java.awt.Desktop).

**How to implement a singleton class?**

To implement a singleton pattern, we have different ways or types, but all of them have the following common concepts.

* Private constructor to restrict instantiation of the class from other classes.
* Private static variable of the same class that is the only instance of the class.
* Public static method that returns the instance of the class, this is the global access point for the outer world to get the instance of the singleton class.

**Some types of Singleton design patterns:**

* 1. [Eager initialization](https://www.digitalocean.com/community/tutorials/java-singleton-design-pattern-best-practices-examples#1-eager-initialization)
* public class MyLogger {
* private static final MyLogger instance = new MyLogger();
* // immediately initialize when the class is created
* private EagerInitializedSingleton(){}
* public static EagerInitializedSingleton getInstance() {
* return instance;
* }
* }
  1. Lazy Initialization
* public class MyLogger {
* private static MyLogger instance;
* private MyLogger(){}
* public static MyLogger getInstance() {
* If(instance == null){ // create the instance in the global access method after checking if it exists or not.
* Instance = new MyLogger();
* }
* return instance;
* }
* }