# Exercise 1: Implementing the Singleton Pattern

## What I Learned

In this exercise, I learned how to implement the Singleton design pattern in Java. The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. This is particularly useful in scenarios like logging where a consistent, shared instance is required across the application.

## Concepts Covered

- Singleton Design Pattern

- Static instance and method in Java

- Private constructor to prevent instantiation

- Consistent logging through a single instance

## Java Code

class Logger {  
 private static Logger instance;  
  
 private Logger() {  
 System.out.println("Logger Initialized");  
 }  
  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void log(String message) {  
 System.out.println("Log: " + message);  
 }  
}  
  
public class SingletonPatternExample {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 logger1.log("This is the first log message.");  
  
 Logger logger2 = Logger.getInstance();  
 logger2.log("This is the second log message.");  
  
 System.out.println("Are both instances same? " + (logger1 == logger2));  
 }  
}

## Conclusion

The Singleton pattern successfully restricts the instantiation of the Logger class to a single object. Using '==' comparison, we confirmed that both logger1 and logger2 refer to the same instance. This pattern is widely used in logging, database connections, and configuration settings where a single shared resource is essential.