# Exercise 1 – Singleton Pattern

## 1. Objective

To implement a logging utility using the Singleton design pattern, ensuring only one instance is created and shared across the application.

## 2. Problem Statement / Scenario

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

## 3. Approach / Steps

1. Create a class `Logger` with:  
 - A private static instance of itself.  
 - A private constructor.  
 - A public static method to get the instance.  
  
2. Use the `Logger` class in a `Main` class to test:  
 - Only one instance is created.  
 - Same instance is used for multiple log messages.

## 4. Code

Logger.java

public class Logger {  
 private static Logger logger;  
  
 private Logger() {  
 System.out.println("Logger Object Created");  
 }  
  
 public static Logger getLoggerInstance() {  
 if (logger == null) {  
 logger = new Logger();  
 }  
 return logger;  
 }  
  
 public void logMessage(String message) {  
 System.out.println("[LOG] " + message);  
 }  
}

Main.java

public class Main {  
 public static void main(String[] args) {  
 Logger log1 = Logger.getLoggerInstance();  
 Logger log2 = Logger.getLoggerInstance();  
  
 log1.logMessage("This is the first message.");  
 log2.logMessage("This is the second message.");  
  
 if (log1 == log2) {  
 System.out.println("Only one logger instance is used.");  
 } else {  
 System.out.println("Different logger instances are created.");  
 }  
 }  
}

## 5. Output

Logger Object Created  
[LOG] This is the first message.  
[LOG] This is the second message.  
Only one logger instance is used.

## 6. Conclusion

The Singleton pattern ensures that only one instance of a class is created and used across the application. This pattern is ideal for classes such as loggers, config managers, and database connection pools.

## 5. Output (Screenshot)

