Exercise 1: Implementing the Singleton Pattern

# 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.

# Steps:

## Step 1: Create a New Java Project

Create a new Java project named SingletonPatternExample.

## Step 2 & 3: Define and Implement the Singleton Class

// File: Logger.java  
package singleton;  
  
public class Logger {  
 // Step 1: Private static instance of the class  
 private static Logger instance;  
  
 // Step 2: Private constructor to prevent instantiation  
 private Logger() {  
 System.out.println("Logger Initialized");  
 }  
  
 // Step 3: Public static method to provide access to the instance  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 // A simple logging method  
 public void log(String message) {  
 System.out.println("LOG: " + message);  
 }  
}

## Step 4: Test the Singleton Implementation

// File: LoggerTest.java  
package singleton;  
  
public class LoggerTest {  
 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.");  
  
 // Check if both loggers are the same instance  
 if (logger1 == logger2) {  
 System.out.println("Both logger instances are the same (Singleton works).");  
 } else {  
 System.out.println("Logger instances are different (Singleton failed).");  
 }  
 }  
}