**EXERCISE1: IMPLEMENTING THE SINGLETON PATTERN**

# Singleton Pattern

The Singleton Pattern ensures that a class has only one instance and provides a global point of access to it. This pattern is commonly used for resources like loggers, configuration managers, and database connections.  
  
Benefits:  
- Ensures a single instance across the application  
- Reduces memory usage  
- Useful for shared resources  
  
Common Pitfalls:  
- Thread safety (consider synchronized or eager initialization for multithreaded applications)

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:**

# 1. Create a New Java Project

Create a new Java project named `SingletonPattern`.

# 2. Define a Singleton Class

Create a class named `Logger`:  
- It should have a private static instance of itself.  
- The constructor should be private to prevent external instantiation.  
- Provide a public static method `getInstance()` to access the single instance.

# 3. Implement the Singleton Pattern

public class Logger {  
 private static Logger instance;  
  
 private Logger() {  
 System.out.println("Logger instance created");  
 }  
  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void log(String message) {  
 System.out.println("Log: " + message);  
 }  
}

# 4. Test the Singleton Implementation

Create a test class to verify that only one instance of Logger is created:  
  
public class TestLogger {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 Logger logger2 = Logger.getInstance();  
  
 logger1.log("This is the first log message.");  
 logger2.log("This is the second log message.");  
  
 System.out.println("logger1 and logger2 are the same instance: " + (logger1 == logger2));  
 }  
}

# Sample Output

# 