**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:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**Source Code :**

|  |
| --- |
| **package week1.designpatterns;**  **class Logger {**  **private static Logger uniqueLogger;**  **private Logger() {**  **System.out.println("Logger object initialized.");**  **}**  **public static Logger getInstance() {**  **if (uniqueLogger == null) {**  **uniqueLogger = new Logger();**  **}**  **return uniqueLogger;**  **}**  **public void log(String msg) {**  **System.out.println("Log Message: " + msg);**  **}**  **}**  **public class Singlepattern\_week1 {**  **public static void main(String[] args) {**  **Logger logRef1 = Logger.getInstance();**  **Logger logRef2 = Logger.getInstance();**  **logRef1.log("Logging first info.");**  **logRef2.log("Logging second info.");**  **if (logRef1 == logRef2) {**  **System.out.println("Both references point to the same Logger instance.");**  **} else {**  **System.out.println("Logger instances are different.");**  **}**  **}**  **}** |

Output :

