**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 {  // instance variable  private static Logger instance;   // private constructor for restricting the access  private Logger() {  System.out.println("Logger instance created.");  }   // method to get an instance  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();  Logger logger2 = Logger.getInstance();   logger1.log("This is the first message.");  logger2.log("This is the second message.");   if (logger1 == logger2) {  System.out.println("logger instances are the same");  } else {  System.out.println("Different logger instances");  }  } }** |
| --- |

Output :

