**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.

Code:

package Design\_Patterns\_And\_Principles.SingletonPatternExample;

 class Logger {

    private static Logger instance;

    // Private constructor to prevent instantiation from outside

    private Logger() {

        // Initialization code if needed

    }

    // Public method to get the instance of Logger class

    public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

        System.out.println("[LOG] " + message);

    }

}

public class LoggerTest {

    public static void main(String[] args) {

        // Get the instance of Logger

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        // Verify that logger1 and logger2 point to the same instance

        System.out.println("Logger instance equality: " + (logger1 == logger2));

        // Log messages using the logger

        logger1.log("This is a log message.");

        logger2.log("Another log message.");

    }

}