**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.

**SOLUTION**

public class Main {

// Logger Singleton class

static class Logger {

private static Logger instance;

// Private constructor to prevent external instantiation

private Logger() {

System.out.println("Creating the Logger instance...");

}

// Public method to provide access to the single instance

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

// Logging method

public void log(String message) {

System.out.println("Logging: " + message);

}

}

// Main method to test the Singleton behavior

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message.");

Logger logger2 = Logger.getInstance();

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

// Verify that both logger1 and logger2 refer to the same instance

if (logger1 == logger2) {

System.out.println("Singleton Verified: Same instance.");

} else {

System.out.println("Singleton Failed: Different instances.");

}

}

}

**OUTPUT**

A screenshot of a computer

AI-generated content may be incorrect.