**Exercise 1: Implementing the Singleton Pattern - Documentation**

**Project Name:** SingletonPatternExample

**Objective:** To create a logging utility class in Java that ensures only one instance is created and used throughout the application's lifecycle using the Singleton Design Pattern.

**1. Introduction to Singleton Pattern:** The Singleton Pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to it. It is commonly used when exactly one object is needed to coordinate actions across a system.

**Use Case:** In this exercise, a logging utility must maintain a single shared instance so that logs from different parts of the system are managed consistently.

**2. Components of the Singleton Implementation:**

**Logger Class:**

* Private static instance of itself (eager initialization).
* Private constructor to prevent external instantiation.
* Public static method getInstance() to provide access to the instance.
* Method log(String message) to simulate logging.

public class Logger {

private static Logger instance = new Logger();

private Logger() {

System.out.println("Logger Initialized.");

}

public static Logger getInstance() {

return instance;

}

public void log(String message) {

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

}

}

**TestLogger Class:**

* Creates two references to the Logger instance.
* Verifies both references point to the same object.

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First message");

Logger logger2 = Logger.getInstance();

logger2.log("Second message");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton Verified).");

} else {

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

}

}

}

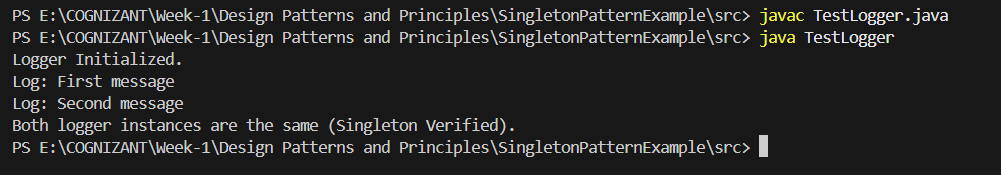
**3. Design Decisions:**

* **Private Constructor:** Restricts instantiation from other classes.
* **Static Instance:** Ensures a single shared instance.
* **Global Accessor Method:** getInstance() provides a way to get the shared instance.
* **Eager Initialization:** Chosen for simplicity and thread safety in a single-threaded context.

**4. Benefits of Singleton in This Scenario:**

* **Consistency:** One logger used throughout the app ensures all logs go to the same place.
* **Resource Efficiency:** Only one instance created, saving memory.
* **Global Access:** Logger can be accessed from anywhere in the app.

**5. Output of the Program:**



**6. Conclusion:** This exercise successfully demonstrates the use of the Singleton Pattern in a real-world use-case. By ensuring a single instance of the Logger class, we promote consistency, efficiency, and centralized control in logging operations.