# **Assignment: Exercise 1 – Singleton Pattern Example: Logger Utility**

## **1. Creating a New Java Project**

I started by creating a new Java project in my IDE and named it **SingletonPatternExample**. This project is for implementing and testing how the **Singleton Design Pattern** works in a real scenario. I decided to do it with a simple Logger class that just logs messages to the console.

## **2. Defining a Singleton Class**

Then I created a class called Logger. To implement Singleton, I made a static variable to store the single object. The constructor is made private so that no one can create a new object from outside.

public class Logger {

private static Logger singleInstance;

private Logger() {

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

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

## **3. Implementing the Singleton Pattern**

In the above code, I used a condition inside getInstance() method. If the singleInstance is null, then a new Logger object is created. If not, it just returns the existing one. So only one object will ever be created and reused every time. This is the whole idea behind the **Singleton Pattern**.

## **4. Testing the Singleton Implementation**

To check if everything is working as expected, I created another class called Main. In that, I tried getting two instances and logged different messages using them. Then I compared if both instances are pointing to the same object.

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first log");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second log");

if (logger1 == logger2) {

System.out.println("Both are the same instance");

} else {

System.out.println("Different instances - Singleton failed");

}

}

}

When I ran the code, it printed “Logger Initialized” only once even though I used getInstance() two times. This proved that only one object of the Logger class was created and both logger1 and logger2 were using the same instance.

## **6. Output**

When I ran the code, this is what got printed on the console:

Logger Initialized

Log: This is the first log

Log: This is the second log

Both are the same instance