**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.

1. **Define a Singleton Class:**

* Create a class named Logger that has a private static instance of itself.

1. **Ensure the constructor of Logger is private.**

* Provide a public static method to get the instance of the Logger class.

1. **Implement the Singleton Pattern:**

* Write code to ensure that the Logger class follows the Singleton design pattern.

1. **Test the Singleton Implementation:**

* Create a test class to verify that only one instance of Logger is created and used across the application.

**Step 1:** Create a New Java Project:

Create a new Java project named SingletonPatternExample

**Step 2:** Define a Singleton Class:

public class Logger

{

private static Logger instance;

private Logger()

{ }

}

**Step 3:** Implement the Singleton Pattern:

public static Logger getInstance()

{

if (instance == null)

{

synchronized (Logger.class)

{

if (instance == null)

{

instance = new Logger();

}

}

}

return instance;

}

public void log(String message)

{

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

}

}

**Step 4:** Test the Singleton Implementation:

public class SingletonPatternTest

{

public static void main(String[] args)

{

Logger logger1 = Logger.getInstance();

logger1.log(" hi, this is first message");

Logger logger2 = Logger.getInstance();

logger2.log(" hi, this is second message");

if (logger1 == logger2)

{

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

}else{

System.out.println("Logger instances are different.");

}

}

}

**SAMPLE OUTPUT:**

