**Module 1 - Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Aim:**

To ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Implementation:** Here I used Double checking method.

**Code:**

public class SingletonPatternExample{

public static class Logger {

    private static Logger instance;

    private Logger(){

        System.out.println("Logger initialised succesfully");

    }

    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("Logging:" +message);

    }

           }

public static void main(String[] args) {

    Logger logger1 = Logger.getInstance();

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

    Logger logger2 = Logger.getInstance();

    logger2.log("This is the first log message");

    if(logger1==logger2){

        System.out.println("Both logger1 and logger2 are same");

    }

    else{

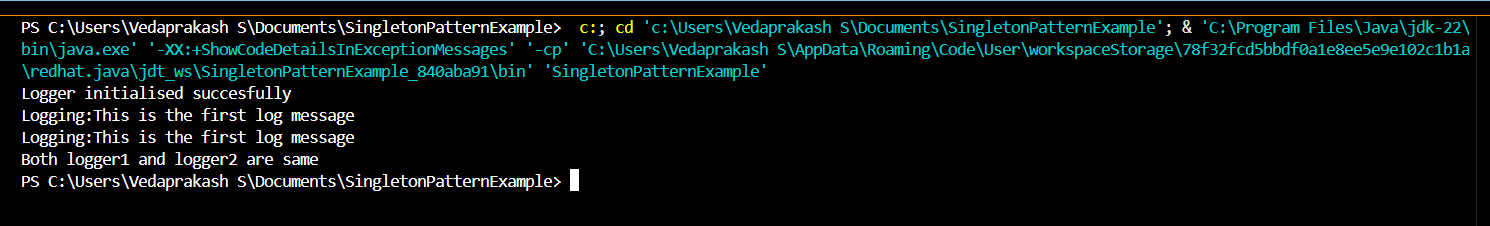
        System.out.println("Both logger1 and logger2 are not same");

    }

}

}

**Output:**

****