WEEK 1

Mandatory Hands-On

**SOLID PRINCIPLES AND DESIGN PATTERNS:**

**EX:1 Implementing the Singleton Pattern**

The reason behind my Logger module to be Singleton is to,

* Maintain a single log file.
* Avoid multiple instances writing simultaneously.
* Maintain a Global access point.

**Singleton Logger class (Definition and Implementation):**

public class Logger {

**private static Logger instance;**

**private Logger()** {

System.out.println("Logger instance created.");

}

**public static Logger getInstance()** {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

* Created a class Logger That has a private static instance of itself.
* Ensured that the constructor of the Logger is private.
* A public static method to get the instance of the Logger class is provided.

**Testing of the Logger class using Main class:**

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("Application started.");

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

if (logger1 == logger2) {

System.out.println("Both logger1 and logger2 refer to the same instance.");

} else {

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

}

}

}

* The Logger class is instantiated.
* Sample log() method is also used and both the instances are checked if they refer the same instance of logger class.

**TESTING OUTPUT:**

