**Design Patterns and Principles – Exercise-1**

**Implementing the Singleton Pattern**

**Main.java**

public class Main {

public static void main(String[] args) {

// Lazy Instantiation

Logger objLog1 = Logger.getInstance();

Logger objLog2 = Logger.getInstance();

if (objLog1 == objLog2)

System.out.println("Only one instance exists in Lazy Instantiation");

else

System.out.println("Multiple instances exist in Lazy Instantiation");

// Synchronized Instantiation with multithreading

Thread t1 = new Thread(() -> {

Logger objSync1 = Logger.getInstanceSync();

});

Thread t2 = new Thread(() -> {

Logger objSync2 = Logger.getInstanceSync();

});

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

Logger objDouble1 = Logger.getInstanceDouble();

Logger objDouble2 = Logger.getInstanceDouble();

if (objDouble1 == objDouble2)

System.out.println("Same Instance in Double Checked Locking");

else

System.out.println("Different Instances in Double Checked Locking");

}

}

**Logger.java**

class Logger {

private static Logger objLog;

private static Logger objSync;

private static volatile Logger objDouble;

private Logger() {

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

}

public static Logger getInstance() {

if (objLog == null) {

objLog = new Logger();

}

return objLog;

}

public static synchronized Logger getInstanceSync() {

if (objSync == null) {

objSync = new Logger();

System.out.println("This is the Synchronous Logger");

}

return objSync;

}

public static Logger getInstanceDouble() {

if (objDouble == null) {

synchronized (Logger.class) {

if (objDouble == null) {

objDouble = new Logger();

}

}

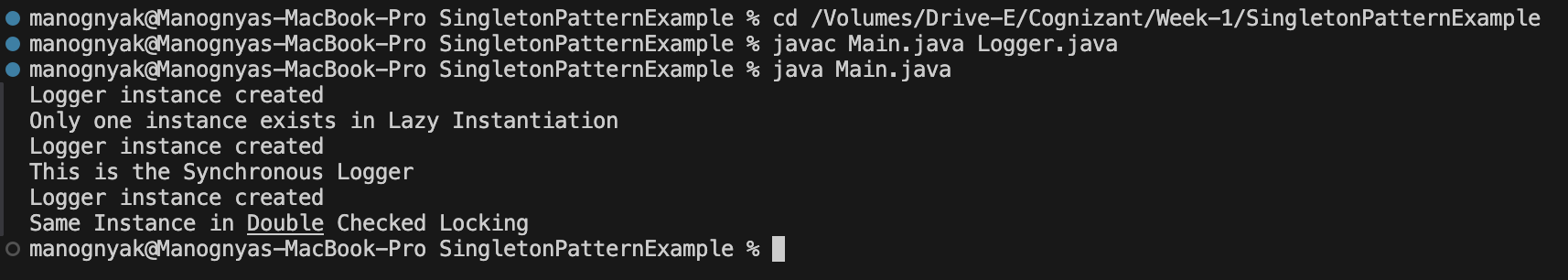
}

return objDouble;

}

}

**Output**

****