**WEEK 1 – Design Principles and Patterns MANDATORY EXERCISES- 6420952**

**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**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

ANSWER:

class Logger {

    private static Logger instance;

    private Logger() {}

    public static Logger getInstance() {

        if (instance == null) instance = new Logger();

        return instance;

    }

    public void log(String msg) {

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

    }

}

class SingleTon {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

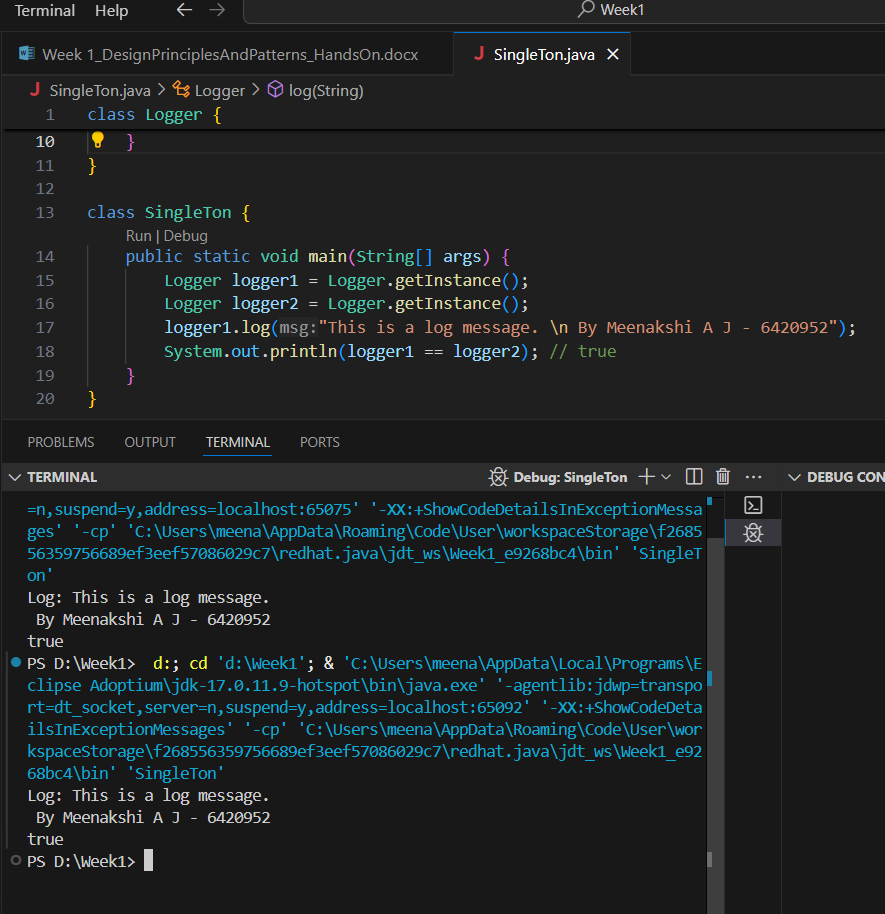
        Logger logger2 = Logger.getInstance();

        logger1.log("This is a log message. \n By Meenakshi A J - 6420952");

        System.out.println(logger1 == logger2); // true

    }

}



**Exercise 2: Implementing the Factory Method Pattern**

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class **DocumentFactory** with a method **createDocument()**.
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

interface Document {

void open();

}

class WordDocument implements Document {

public void open() { System.out.println("Opening Word document"); }

}

class PdfDocument implements Document {

public void open() { System.out.println("Opening PDF document"); }

}

class ExcelDocument implements Document {

public void open() { System.out.println("Opening Excel document"); }

}

abstract class DocumentFactory {

public abstract Document createDocument();}

class WordFactory extends DocumentFactory {

public Document createDocument() { return new WordDocument(); }

}

class PdfFactory extends DocumentFactory {

public Document createDocument() { return new PdfDocument(); }}

class ExcelFactory extends DocumentFactory {

public Document createDocument() { return new ExcelDocument(); }

}

class FactoryTest {

public static void main(String[] args) {

DocumentFactory factory = new PdfFactory();

Document doc = factory.createDocument();

doc.open();

System.out.println("\n By Meenakshi A J - 6420952\n"); }

