**WEEK – 1 HANDS ON**

**SKILL – Design Pattern and Principles**

**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.**

**CODE:**

//Logger class

Package SingletonPatternExample;

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);

    }

}

//Main class

Package SingletonPatternExample;

public class TestLogger {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("First log message");

        logger2.log("Second log message");

        if (logger1 == logger2) {

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

        } else {

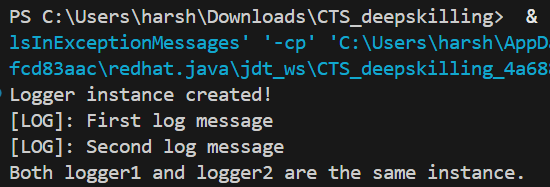
            System.out.println("Error: logger1 and logger2 are different instances.");

        }

    }

}

OUTPUT:



**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.

**CODE:**

**//interface Document**

package FactoryMethodPatternExample;

public interface Document {

    void open();

}

**//Abstract class Document Factory**

package FactoryMethodPatternExample;

public abstract class DocumentFactory {

    public abstract Document createDocument();

}

**// class Excel Document**

package FactoryMethodPatternExample;

public class ExcelDocument implements Document{

    @Override

    public void open() {

        System.out.println("Opening an Excel Document");

    }

}

**//class ExcelDocumentFactory**

package FactoryMethodPatternExample;

public class ExcelDocumentFactory extends DocumentFactory{

    public Document createDocument(){

        return new ExcelDocument();

    }

}

**// class PdfDocument**

package FactoryMethodPatternExample;

public class PdfDocument implements Document {

     @Override

        public void open() {

            System.out.println("Opening a PDF Document (.pdf)");

     }

}

**//class PdfDocumentFactory**

package FactoryMethodPatternExample;

public class PdfDocumentFactory extends DocumentFactory {

     @Override

        public Document createDocument() {

            return new PdfDocument();

        }

}

**//class TestFactoryMethod**

package FactoryMethodPatternExample;

public class TestFactoryMethod {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document wordDoc = wordFactory.createDocument();

        wordDoc.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdfDoc = pdfFactory.createDocument();

        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

**//class WordDocument**

package FactoryMethodPatternExample;

public class WordDocument implements Document{

      @Override

        public void open() {

            System.out.println("Opening a Word Document (.docx)");

        }

}

**//class WordDocumentFactory**

package FactoryMethodPatternExample;

public class WordDocumentFactory extends DocumentFactory{

    @Override

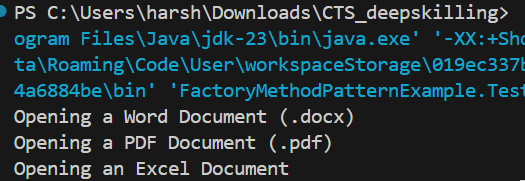
    public Document createDocument() {

    return new WordDocument();

}

}

**OUTPUT:**

****