**Design Patterns and Principles**

**EXERSICE-1 : IMPLEMENTING 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.

**SOURCE CODE:**

public class Logger {

    private static volatile Logger instance;

    private Logger() {

        // Initialization code here

        System.out.println("Logger initialized.");

    }

    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("[LOG] " + message);

    }

}

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("Starting the application...");

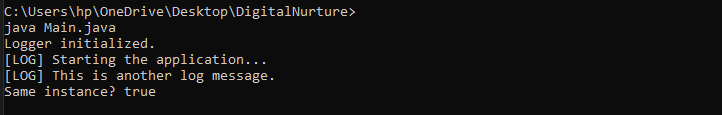
        logger2.log("This is another log message.");

        System.out.println("Same instance? " + (logger1 == logger2)); // true

    }

}

**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.

**SOURCE CODE :**

**WordDocument.java**

package com.factory.document;

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocument.java**

package com.factory.document;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java**

package com.factory.document;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java**

package com.factory.document;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package com.factory.document;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package com.factory.document;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package com.factory.document;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**DocumentFactoryTest.java**

package com.factory.document;

public class DocumentFactoryTest {

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();

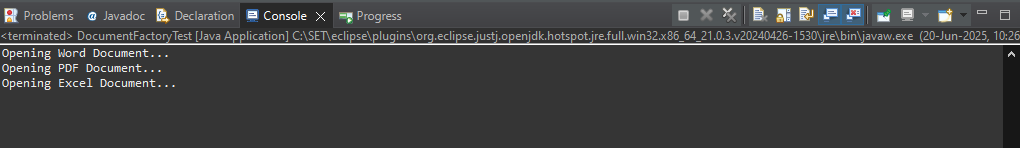
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT :**

****