# Exercise 1: Implementing the Singleton Pattern

## Hands-On Question: 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.

Java Code: It’s written in Visual Studio Code

//SingletonPatternExample.java

public class SingletonPatternExample {

    static class Logger {

        private static Logger instance;

    private Logger() {

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

        }

        public static Logger getInstance() {

            if (instance == null) {

                instance = new Logger();

            }

            return instance;

        }

        public void log(String message) {

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

        }

    }

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("This is the first log message.");

        logger2.log("This is the second log message.");

        if (logger1 == logger2) {

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

        } else {

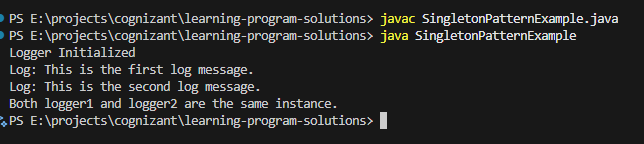
            System.out.println("Different logger instances exist.");

        }

    }

}

Output Screenshot:



(Next Exercise is on the next Page)

**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.

Java Code: It’s written in Visual Studio Code

// FactoryMethodPatternExample.java

// Step 1: Document Interface

interface Document {

    void open();

}

// Step 2: Concrete Document Classes

class WordDocument implements Document {

    public void open() {

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

    }

}

class PdfDocument implements Document {

    public void open() {

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

    }

}

class ExcelDocument implements Document {

    public void open() {

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

    }

}

// Step 3: Abstract Factory Class

abstract class DocumentFactory {

    public abstract Document createDocument();

}

// Step 4: Concrete Factory Classes

class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

// Step 5: Test Class

public class FactoryMethodPatternExample {

    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 Screenshot:

