**Exercise 1**: Implementing the Singleton class

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.

**Logger.java:**

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);

    }

}

**LoggerTest.java:**

public class LoggerTest {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("Application started.");

        logger2.log("Logging from another class.");

        if (logger1 == logger2) {

            System.out.println("Same Logger instance used (Singleton works).");

        } else {

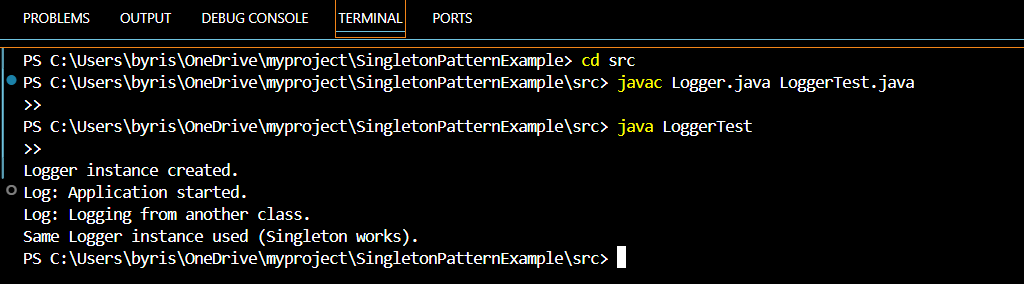
            System.out.println("Different instances created (Singleton failed).");

        }

    }

}

**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.

**Document.java**

public interface Document {

    void open();

}

**WordDocument.java**

public class WordDocument implements Document {

    public void open() {

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

    }

}

**PdfDocument.java**

public class PdfDocument implements Document {

    public void open() {

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

    }

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

    public void open() {

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

    }

}

**DocumentFactory.java**

public abstract class DocumentFactory {

    public abstract Document createDocument();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

**DocumentFactoryTest.java**

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();

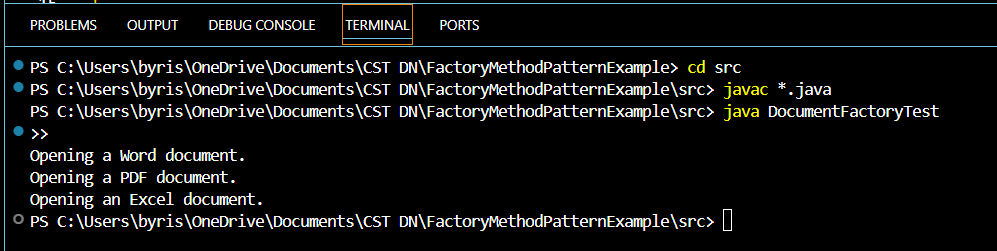
        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

**Output:**



**Explaination**

**How It Follows the Pattern:**

* The **abstraction of object creation** is achieved using the DocumentFactory class.
* The **responsibility of creating the specific object** is moved to subclasses of the factory.
* This allows the client code to **use a factory** without knowing the exact class being instantiated.