**Deep Skilling - Java FSE**  
**WEEK –1 Hands-on Exercises**  
**Module 1 - Design Patterns and Principles**

**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.

**Singleton Pattern:**

The Singleton Pattern is a design pattern that ensures a class has only one instance throughout the application and provides a global point of access to it.

**Code:**

**Logger.java:**

package singletonPattern;

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:**

package singletonPattern;

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message.");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message.");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (singleton verified).");

} else {

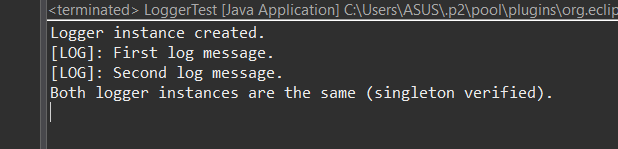
System.out.println("Different logger instances exist (singleton failed).");

}

}

}

**Output:**



**Explanation:**

1. private static Logger instance;

A static variable that will hold the one and only instance of the class.

1. private Logger()

Constructor is private so no other class can use new Logger().

Ensures control over object creation.

1. public static Logger getInstance()

If no instance exists, it creates one.

If already created, it returns the same one.

1. logger1 == logger2

This check proves that both references point to the same object in memory.

**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.

**Factory Method Pattern:**

The Factory Method Pattern is a creational design pattern that lets a class delegate the responsibility of object creation to subclasses or other classes.

Instead of using new directly, we use a factory method that returns an instance of a class.  
  
**Code:**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

public void open() {

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

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening 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();

}

}

**FactoryTest.java**

public class FactoryTest {

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();

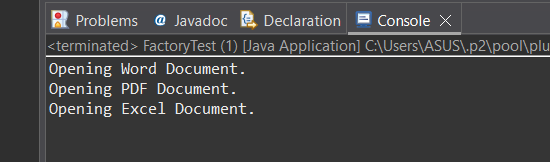
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**



**Explanation:**

1. Document Interface

Defines a common method open() for all document types.

All document classes implement this interface.

1. WordDocument, PdfDocument, ExcelDocument

These are concrete classes implementing the Document interface.

Each class defines its own behavior for open().

1. DocumentFactory (Abstract Class)

Declares the method createDocument().

This is the factory method that subclasses must implement.

1. WordDocumentFactory, PdfDocumentFactory, ExcelDocumentFactory

These are concrete factories that extend DocumentFactory.

Each one implements createDocument() to return its specific document type.

1. Test Class: FactoryTest

Instead of new WordDocument(), we use the factory.

This promotes loose coupling and flexibility.