**Digital Nurture 4.0 Deep Skilling - Java FSE**  
**WEEK –1 Hands-on Exercises**  
**Module 1 - Design Patterns and Principles**

**1. 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.  
  
**Solution:**

**Singleton Pattern:**

The Singleton Pattern ensures that a class has only one instance throughout the application's lifecycle and provides a global point of access to that instance.

**Code:**

**Logger.java**  
package singletonPattern;

public 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);

}

}  
  
  
**SingletonTest.java**

package singletonPattern;

public class SingletonTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

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

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

if (logger1 == logger2) {

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

} else {

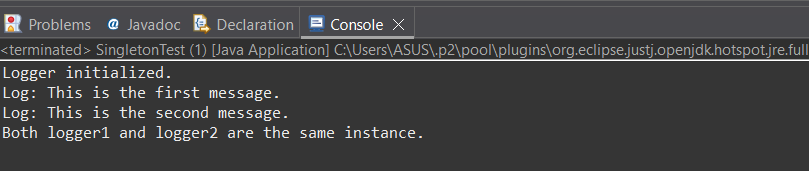
System.out.println("Different instances were created.");

}

}

}

**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. 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.

**Solution:**

**Factory Method Pattern**   
The Factory Method Pattern is a creational design pattern that provides an interface for creating objects but allows subclasses to alter the type of objects that will be created.

* Encapsulate object creation logic.
* Adhere to Open/Closed Principle (open for extension, closed for modification).
* Promote loose coupling between client code and specific classes.

**Code:**

**Document.java**

package factoryMethod;

public interface Document {

void open();

}  
  
**WordDocument.java**

package factoryMethod;

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocument.java**  
package factoryMethod;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java**

package factoryMethod;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java**  
package factoryMethod;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package factoryMethod;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**  
package factoryMethod;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package factoryMethod;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}  
  
**FactoryTest .java**

package factoryMethod;

public class FactoryTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

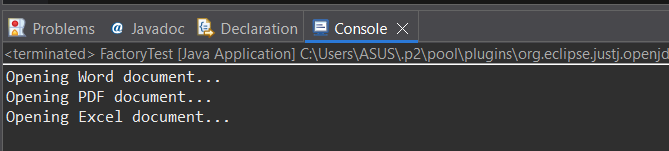
Document excel = excelFactory.createDocument();

excel.open();

}

}

**Output:**



**Explanation:**

1. Document Interface

It defines a method open().

All document types (Word, PDF, Excel) must use this method.

1. WordDocument Class

This class implements Document.

It gives its own message for open(), like "Opening Word document...".

1. DocumentFactory Abstract Class   
   It has an abstract method createDocument().

It doesn’t say how to create the document, it leaves that to the subclasses.

1. WordDocumentFactory Class   
   This is a subclass of DocumentFactory.

It knows how to create a WordDocument.

When createDocument() is called, it returns a new WordDocument.

1. FactoryTest (Main Method )  
   This is where the program runs.

It uses WordDocumentFactory to create a document.

It then calls open() on that document.

The client doesn’t need to know the actual class being used ,it just works.