**WEEK1 - Design Patterns and Principles**

**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.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**CODE :**

**Looger.java**

**package** Singletonpatternexample;

**public** **class** Logger {

// Step 1: Private static instance of the same class

**private** **static** Logger *singleInstance*;

// Step 2: Private constructor to prevent instantiation

**private** Logger() {

System.***out***.println("Logger instance created.");

}

// Step 3: Public static method to get the single instance

**public** **static** Logger getInstance() {

**if** (*singleInstance* == **null**) {

*singleInstance* = **new** Logger(); // create only once

}

**return** *singleInstance*;

}

// Method to simulate logging

**public** **void** log(String message) {

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

}

}

**TestLogger.java**

**package** Singletonpatternexample;

**public** **class** TestLogger {

**public** **static** **void** main(String[] args) {

// Getting Logger instance the first time

Logger logger1 = Logger.*getInstance*();

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

// Getting Logger instance again

Logger logger2 = Logger.*getInstance*();

logger2.log("Continuing application...");

// Checking if both instances are same

**if** (logger1 == logger2) {

System.***out***.println("Both logger instances are the same. Singleton works!");

} **else** {

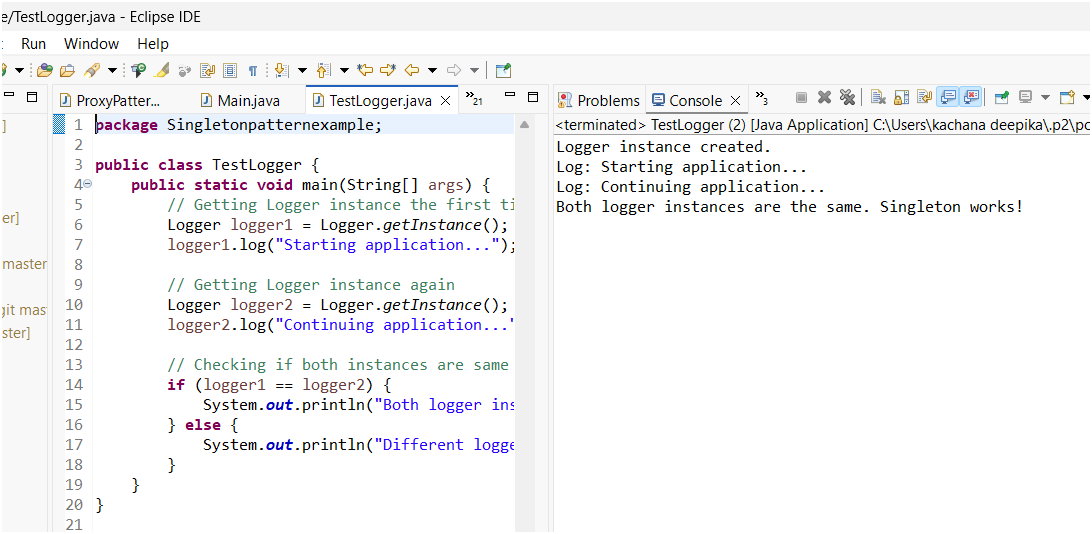
System.***out***.println("Different logger instances. Singleton failed!");

}

}

}

**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.

Steps:

1. **Create a New Java Project:**
   * Create a new Java project named FactoryMethodPatternExample.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as WordDocument, PdfDocument, and ExcelDocument.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class DocumentFactory with a method createDocument().
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the createDocument() method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

**Structure of Code:**

FactoryMethodPattern-

1. Document.java - interface

2.Document Classes

i. WordDocument.java - implements Document

ii. PdfDocument.java - implements Document

iii. ExcelDocument.java - implements Document

3. Factory Classes

i. DocumentFactory.java - abstract class

ii. WordDocumentFactory.java - extends DocumentFactory

iii. PdfDocumentFactory.java - extends DocumentFactory

iv. ExcelDocumentFactory.java - extends DocumentFactory

4. DocumentTest.java - main test class

**Code:**

**Document.java – Interface:**

**package** FactoryMethodPatternExample;

**public** **interface** Document

{

**void** open();

}

**Document Classes:**

**WordDocument.java:**

**package** FactoryMethodPatternExample;

**public** **class** WordDocument **implements** Document

{

**public** **void** open()

{

System.***out***.println("Opening a word Document");

}

}

**PdfDocument.java:**

**package** FactoryMethodPatternExample;

**public** **class** PdfDocument **implements** Document {

**public** **void** open()

{

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

}

}

**ExcelDocument.java:**

package FactoryMethodPatternExample;

public class ExcelDocument implements Document {

public void open() {

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

}

}

**Factory Classes:**

**DocumentFactory.java:**

**package** FactoryMethodPatternExample;

**public** **abstract** **class** DocumentFactory

{

**public** **abstract** Document createDocument();

}

**WordDocumentFactory.java:**

**package** FactoryMethodPatternExample;

**public** **class** WordDocumentFactory **extends** DocumentFactory

{

**public** Document createDocument()

{

**return** **new** WordDocument();

}

}

**PdfDocumentFactory.java:**

**package** FactoryMethodPatternExample;

**public** **class** PdfDocumentFactory **extends** DocumentFactory

{

**public** Document createDocument()

{

**return** **new** PdfDocument();

}

}

**ExcelDocumentFactory.java:**

**package** FactoryMethodPatternExample;

**public** **class** ExcelDocumentFactory **extends** DocumentFactory

{

**public** Document createDocument()

{

**return** **new** ExcelDocument();

}

}

**DocumentTest.java:(Main)**

**package** FactoryMethodPatternExample;

**public** **class** DocumentTest {

**public** **static** **void** main(String[] args) {

DocumentFactory wordFactory = **new** WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open(); // Output: Opening a Word Document.

DocumentFactory pdfFactory = **new** PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open(); // Output: Opening a PDF Document.

DocumentFactory excelFactory = **new** ExcelDocumentFactory();

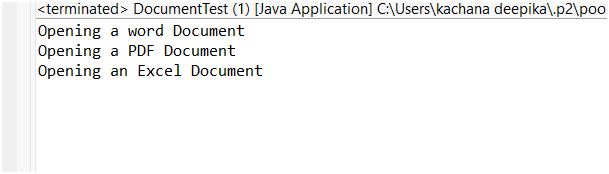
Document excelDoc = excelFactory.createDocument();

excelDoc.open(); // Output: Opening an Excel Document.

}

}

**OUTPUT:**

****