**Week 1 – Design Patterns and Principles SupersetId-6431499**

**Name : Balaji V**

**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:**

**Logger.java:**

package SingletonPatternExample;

public class Logger{

private static volatile Logger instance = null;

private Logger(){

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

}

public static Logger getInstance(){

if(instance==null){

synchronized (Logger.class){

if(instance==null){

instance=new Logger();

}

}

}

return instance;

}

public void log(String message){

System.out.println("[LOG] " + message);

}

}

**LoggerTest.java:**

package SingletonPatternExample;

public class LoggerTest{

public static void main(String[] args){

Logger logger1=Logger.getInstance();

Logger logger2=Logger.getInstance();

System.out.println("\n6431499--Balaji V--CTS\n");

logger1.log("Application started.");

logger2.log("Performing an operation.");

if(logger1==logger2){

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

}

else{

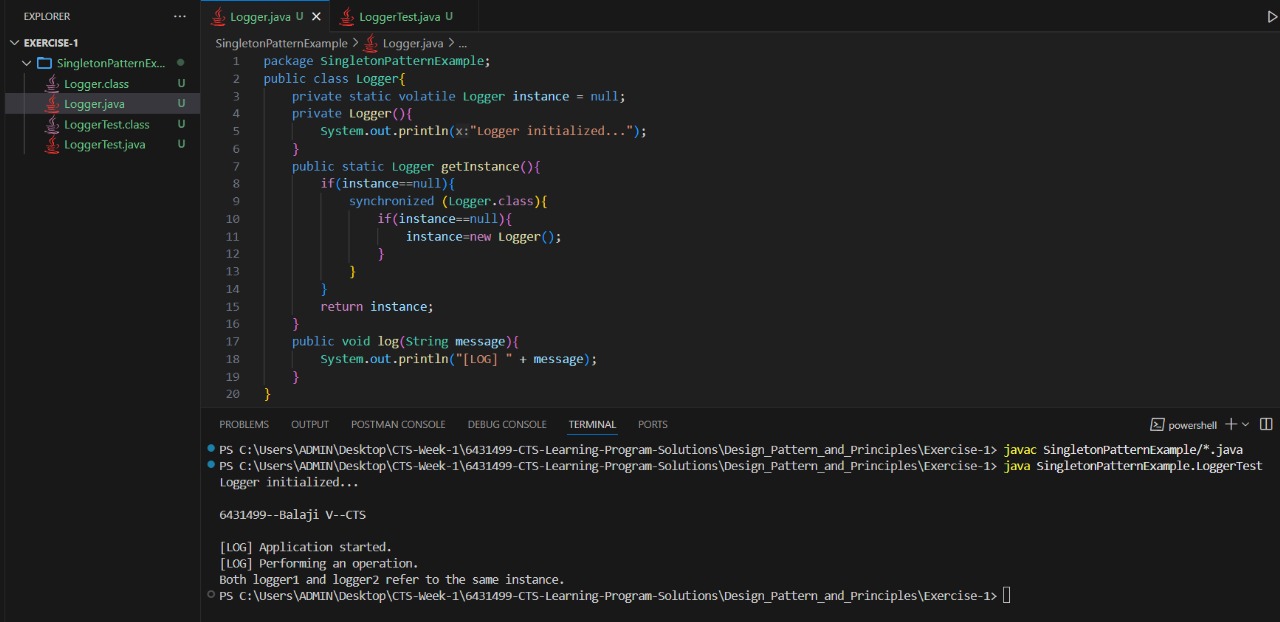
System.out.println("Different instances found!");

}

}

}

**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.

**Code:**

**Document.java**

package FactoryMethodPatternExample;

public interface Document {

void open();

}

**WordDocument.java**

package FactoryMethodPatternExample;

public class WordDocument implements Document{

@Override

public void open(){

System.out.println("\n6431499--Balaji V--CTS\n");

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

}

}

**PdfDocument.java**

package FactoryMethodPatternExample;

public class PdfDocument implements Document{

@Override

public void open() {

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

}

}

**ExcelDocument.java**

package FactoryMethodPatternExample;

public class ExcelDocument implements Document{

@Override

public void open(){

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

}

}

**DocumentFactory.java**

package FactoryMethodPatternExample;

public abstract class DocumentFactory{

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package FactoryMethodPatternExample;

public class WordDocumentFactory extends DocumentFactory{

@Override

public Document createDocument(){

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package FactoryMethodPatternExample;

public class PdfDocumentFactory extends DocumentFactory{

@Override

public Document createDocument(){

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package FactoryMethodPatternExample;

public class ExcelDocumentFactory extends DocumentFactory{

@Override

public Document createDocument(){

return new ExcelDocument();

}

}

**FactoryTest.java**

package FactoryMethodPatternExample;

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:**  
