**DESIGN PRINCIPLES AND PATTERNS**

**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.

**IMPLEMENTATION:**

**Logger class:**

package com.example.singleton;

public class Logger {

private static volatile Logger *instance*;

private Logger() {

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

}

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);

}

}

**SingletonTest class:**

public class SingletonTest {

public static void main(String[] args) {

Logger first = Logger.*getInstance*();

Logger second = Logger.*getInstance*();

first.log("Application started");

second.log("Still using the same logger!");

if (first == second) {

System.*out*.println("✅ Only one Logger instance exists.");

} else {

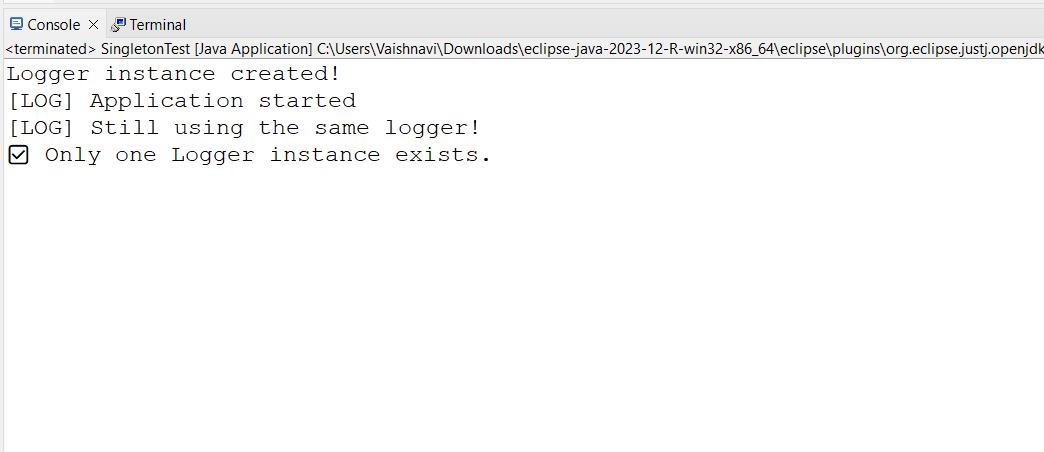
System.*out*.println("❌ Different Logger instances were created!");

}

}

}

**OUTPUT :**



**SHORT NOTE:**

The double-checked locking ensures lazy, thread-safe instantiation; the identity check in SingletonTest confirms a single instance.

**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.

**IMPLEMENTATION:**

**1.Document.java**

package com.example.factory;

public interface Document {

void open();

}

**2.WordDocument.java**

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**3.PdfDocument.java**

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**4. ExcelDocument.java**

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**5.** **DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**6.** **WordFactory.java**

public class WordFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**7.** **PdfFactory.java**

public class PdfFactory extends DocumentFactory{

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**8. ExcelFactory.java**

public class ExcelFactory extends DocumentFactory{

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**9. FactoryTest.java**

public class FactoryTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

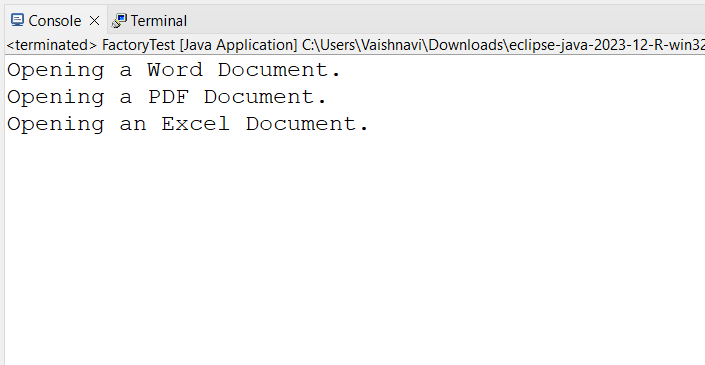
DocumentFactory excelFactory = new ExcelFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT:**

**SHORT NOTE:** Each factory creates a different type of document using the same createDocument() method interface. This follows the Factory Method pattern