**Module 1:**

**Exercise 1**:**Implementing the Singleton Pattern**

**Step 1: Create a New Java Project**

Create a new Java project named **SingletonPatternExample** in your preferred IDE (like IntelliJ IDEA, Eclipse, etc.).

**Step 2: Define a Singleton Class**

Create a class named **Logger** that implements the Singleton design pattern.

**Step 3: Implement the Singleton Pattern**

The implementation above ensures that the **Logger** class follows the Singleton design pattern by:

* Having a private constructor to prevent instantiation from outside the class.
* Providing a static method **getInstance()** that returns the single instance of the class.

**Step 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:**

public class Logger {

// Private static instance of the class

private static Logger instance;

// Private constructor to prevent instantiation

private Logger() {

if (instance != null) {

throw new RuntimeException("Use getInstance() method to get the single instance of this class.");

}

}

// Public static method to get the instance

public static Logger getInstance() {

if (instance == null) {

synchronized (Logger.class) {

if (instance == null) {

instance = new Logger();

}

}

}

return instance;

}

// Sample logging method

public void log(String message) {

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

}

// Main method to test the Logger

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("This is a log message.");

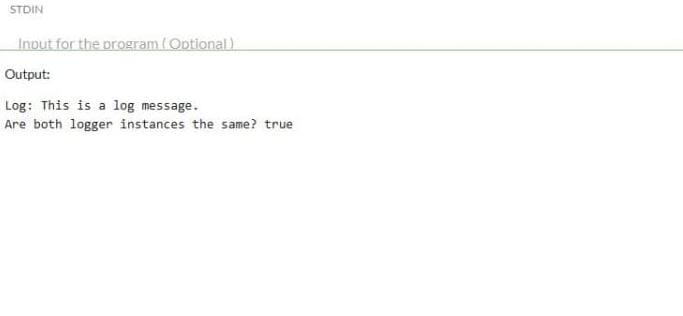
// Check if both references point to the same object

System.out.println("Are both logger instances the same? " + (logger1 == logger2));

    }

}

**Output:**



**Exercise 2**: **Implementing the Factory Method Pattern**

**Step 1: Create a New Java Project**

1. Open your IDE (like IntelliJ IDEA, Eclipse, or NetBeans).
2. Create a new Java project named **FactoryMethodPatternExample**.

**Step 2: Define Document Classes**

Create an interface for the document types.

**Step 3: Create Concrete Document Classes**

Implement concrete classes for each document type.

**WordDocument.java**

package com.example.factory; // Package declaration

...

**PdfDocument.java**

package com.example.factory; // Package declaration

...

**ExcelDocument.java**

package com.example.factory; // Package declaration

...

**Step 4: Implement the Factory Method**

Create an abstract class **DocumentFactory** and concrete factory classes for each document type.

**DocumentFactory.java**

package com.example.factory; // Package declaration

**WordDocumentFactory.java**

package com.example.factory; // Package declaration

**PdfDocumentFactory.java**

package com.example.factory; // Package declaration

**ExcelDocumentFactory.java**

package com.example.factory; // Package declaration

**Step 5: Test the Factory Method Implementation**

Create a test class to demonstrate the creation of different document types using the factory method.

**FactoryMethodTest.java**

package com.example.factory; // Package declaration

Summary:

* You have successfully implemented the Factory Method Pattern to create different types of documents.
* The **DocumentFactory** class serves as the abstract factory, while the concrete factories create specific document types.
* The test class demonstrates the creation and usage of different document types.

**Code:**

interface Document {

void open();

void save();

void close();

}

class WordDocument implements Document {

public void open() {

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

}

public void save() {

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

}

public void close() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

public void save() {

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

}

public void close() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

public void save() {

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

}

public void close() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

// ✅ Main class must match file name (Main.java)

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document wordDoc = wordFactory.createDocument();

Document pdfDoc = pdfFactory.createDocument();

Document excelDoc = excelFactory.createDocument();

System.out.println("Testing Word Document:");

wordDoc.open();

wordDoc.save();

wordDoc.close();

System.out.println("\nTesting PDF Document:");

pdfDoc.open();

pdfDoc.save();

pdfDoc.close();

System.out.println("\nTesting Excel Document:");

excelDoc.open();

excelDoc.save();

excelDoc.close();

    }

}

Output:

