WEEK-1

**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**

SingletonPatternExample

**Logger.java**

package com.Singleton.Demo;

public class Logger

{

private static Logger *instance*;

private Logger() {

System.*out*.println("Logger Initialized");

}

// lazy initialization

public static Logger getInstance() {

if (*instance* == null) {

*instance* = new Logger();

}

return *instance*;

}

public void log(String message) {

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

}

}

**LoggerTest.java**

package com.Singleton.Demo;

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

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

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

if (logger1 == logger2) {

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

} else {

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

}

}

}

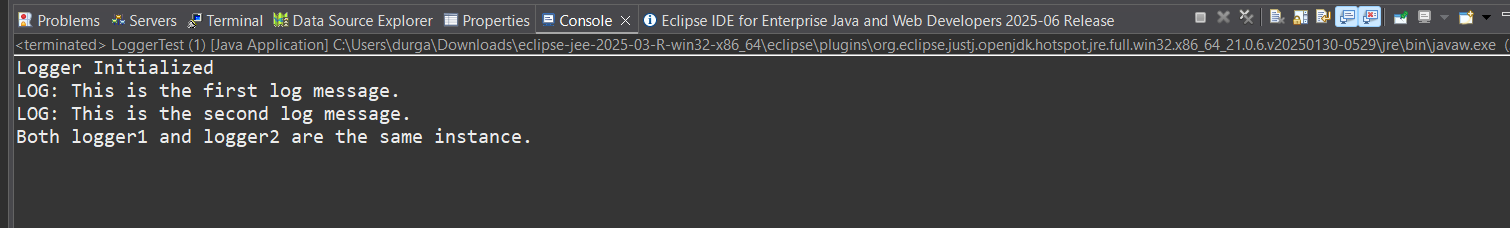
SingletonPatternExample

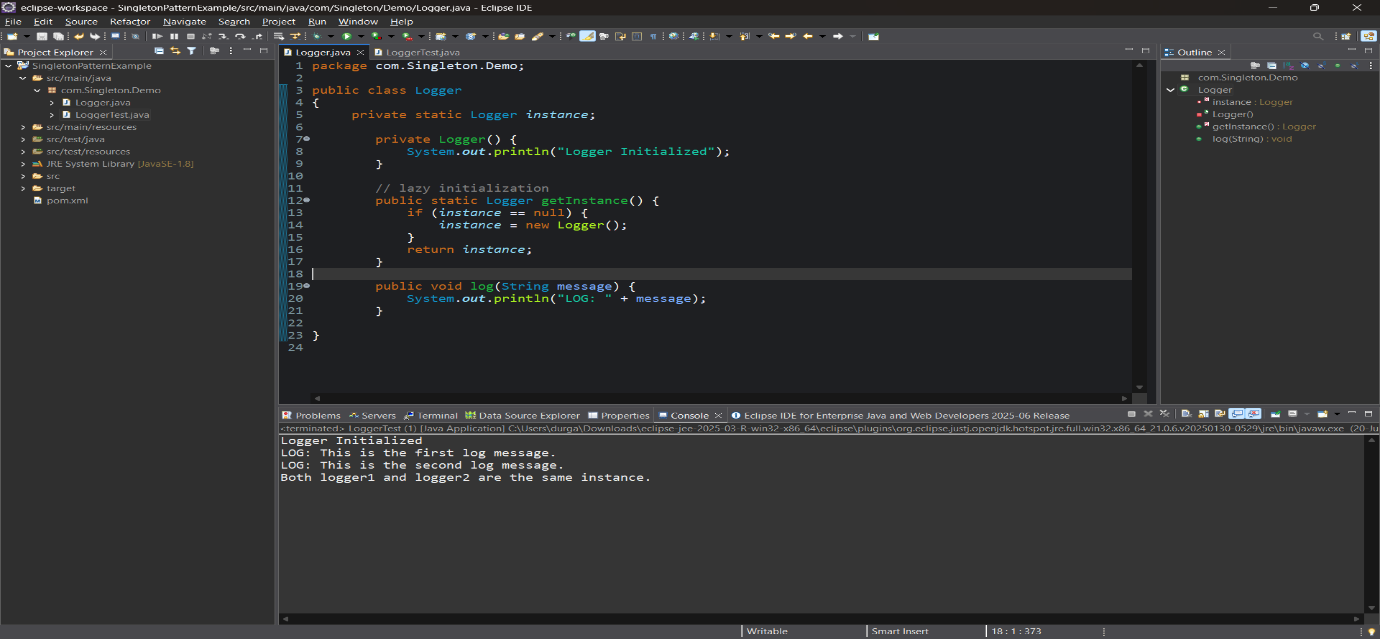
This experiment demonstrates the Singleton Design Pattern in Java.

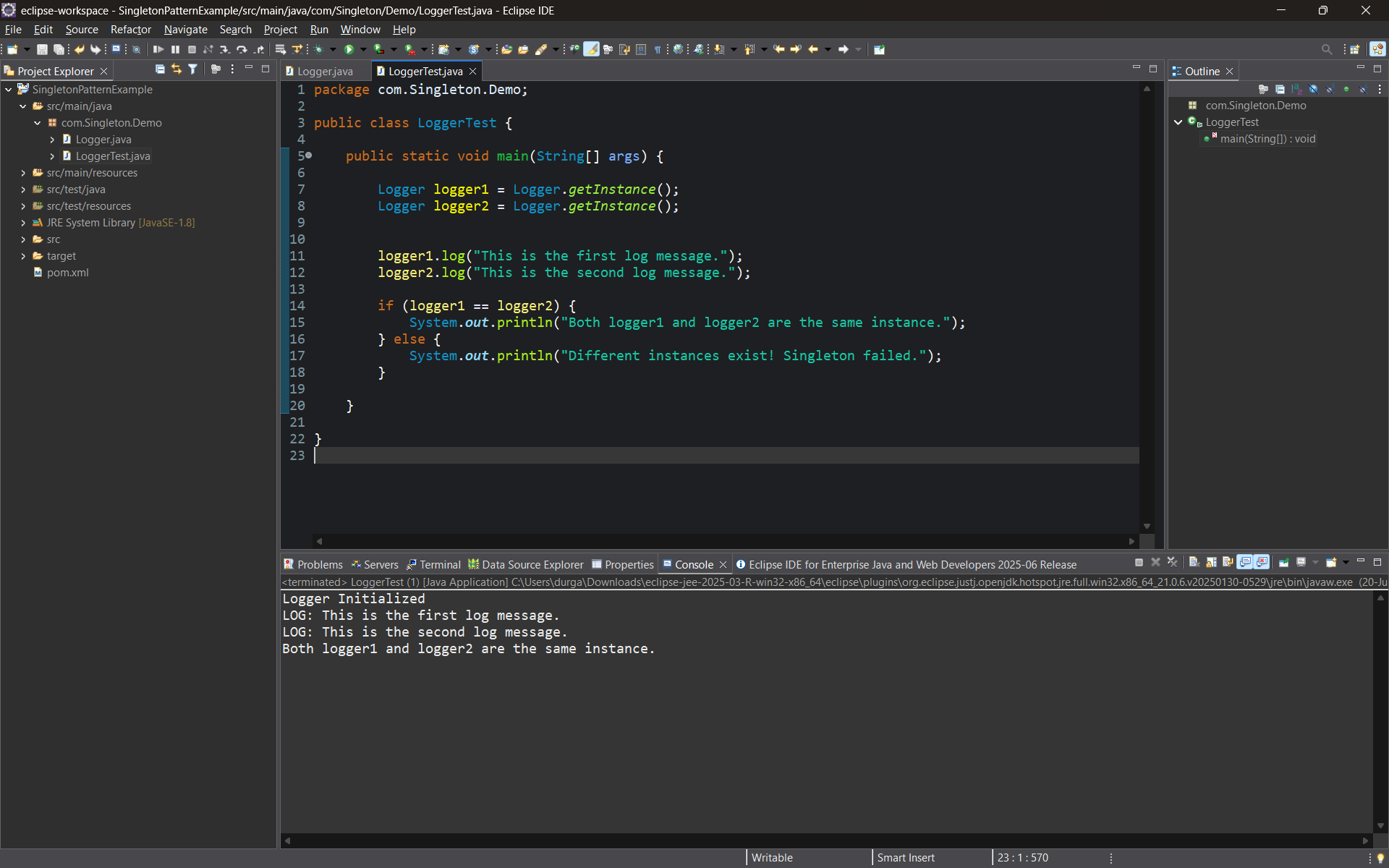
- Logger.java: Singleton Logger class.

- LoggerTest.java: Main class to test the Singleton behavior.

**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**

FactoryMethodPatternExample

**Document.java**

package com.Factory.Demo;

public interface Document {

void open();

}

//Concreate Document Classes

**WordDocument.java**

package com.Factory.Demo;

public class WordDocument implements Document {

public void open() {

System.*out*.println("Opening Word document->>>");

}

}

**PdfDocument.java**

package com.Factory.Demo;

public class PdfDocument implements Document {

public void open() {

System.*out*.println("Opening PDF document->>>");

}

}

**ExcelDocument.java**

package com.Factory.Demo;

public class ExcelDocument implements Document {

public void open() {

System.*out*.println("Opening Excel document->>>");

}

}

// Abstract class

**DocumentFactory.java**

package com.Factory.Demo;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

//Concrete factory class for Word

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

//Concrete factory class for PDF

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

//Concrete factory class for Excel

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

// Test method

**FactoryMethodTest.java**

package com.Factory.Demo;

public class FactoryMethodTest

{

public static void main(String[] args)

{

// Create Word document

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

// Create PDF document

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

// Create Excel document

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT**

