**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.

**Code**

**Logger.java**

package DesignPatternsAndPrinciples;

public class Logger {

private static Logger singleInstance;

private Logger() {

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

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

**LoggerTest.Java**

**package** DesignPatternsAndPrinciples;

**public** **class** LoggerTest {

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

Logger logger1 = Logger.*getInstance*();

logger1.log("First log message");

Logger logger2 = Logger.*getInstance*();

logger2.log("Second log message");

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

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

} **else** {

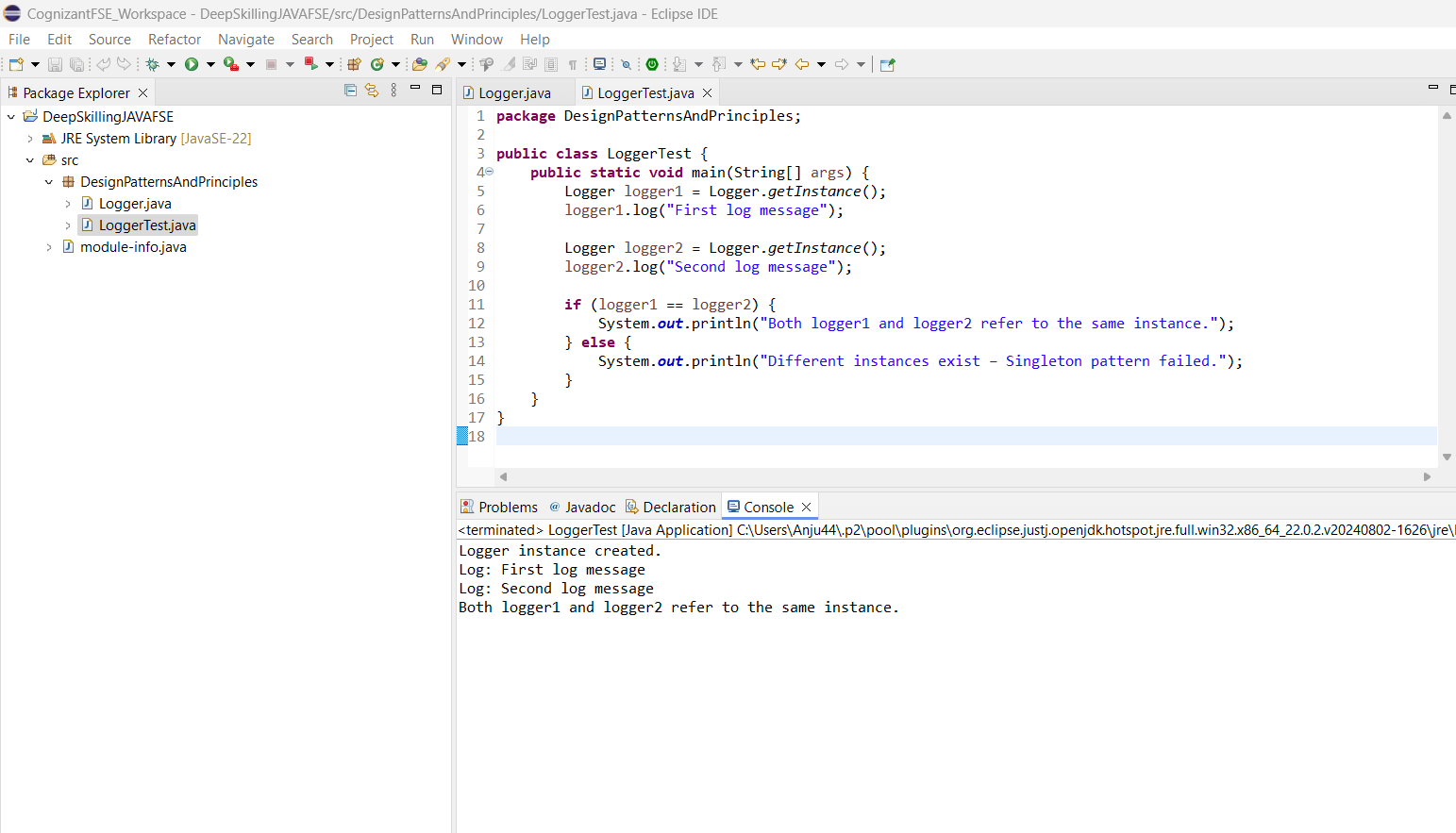
System.***out***.println("Different instances exist – Singleton pattern failed.");

}

}

}

**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.

**Code**

**FactoryMethodPattern Package**

package factorymethod;

public interface Document {

void open();

void close();

void save(String content);

}

package factorymethod;

public class WordDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

@Override

public void save(String content) {

System.out.println("Saving to Word document: " + content);

}

}

package factorymethod;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

@Override

public void save(String content) {

System.out.println("Saving to PDF document: " + content);

}

}

package factorymethod;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

@Override

public void save(String content) {

System.out.println("Saving to Excel document: " + content);

}

}

package factorymethod;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

package factorymethod;

public class WordFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

package factorymethod;

public class PdfFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

package factorymethod;

public class ExcelFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryMethodTest.Java**

package FactoryMethodPattern;

public class FactoryMethodTest {

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();

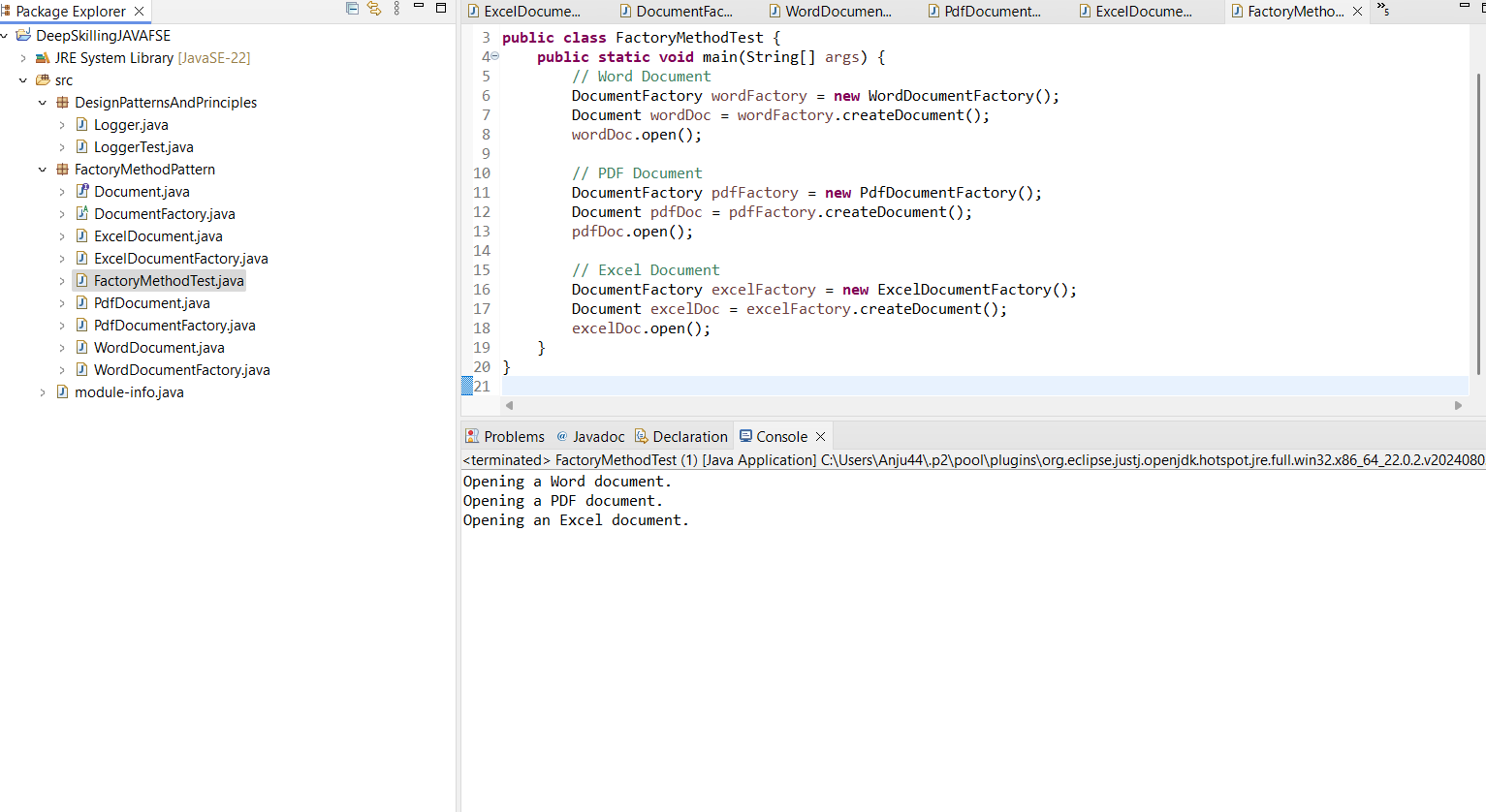
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output**

****