**COGNIZANT JAVA FSE PROGRAM**

**WEEK 1: DESIGN PRINCIPLES AND PATTERN**

**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.

**package** singletonPatternExample;

**public** **class** Logger {

**private** **static** **volatile** Logger *instance*;

**private** Logger() {

}

**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);

}

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

Logger logger1 = Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

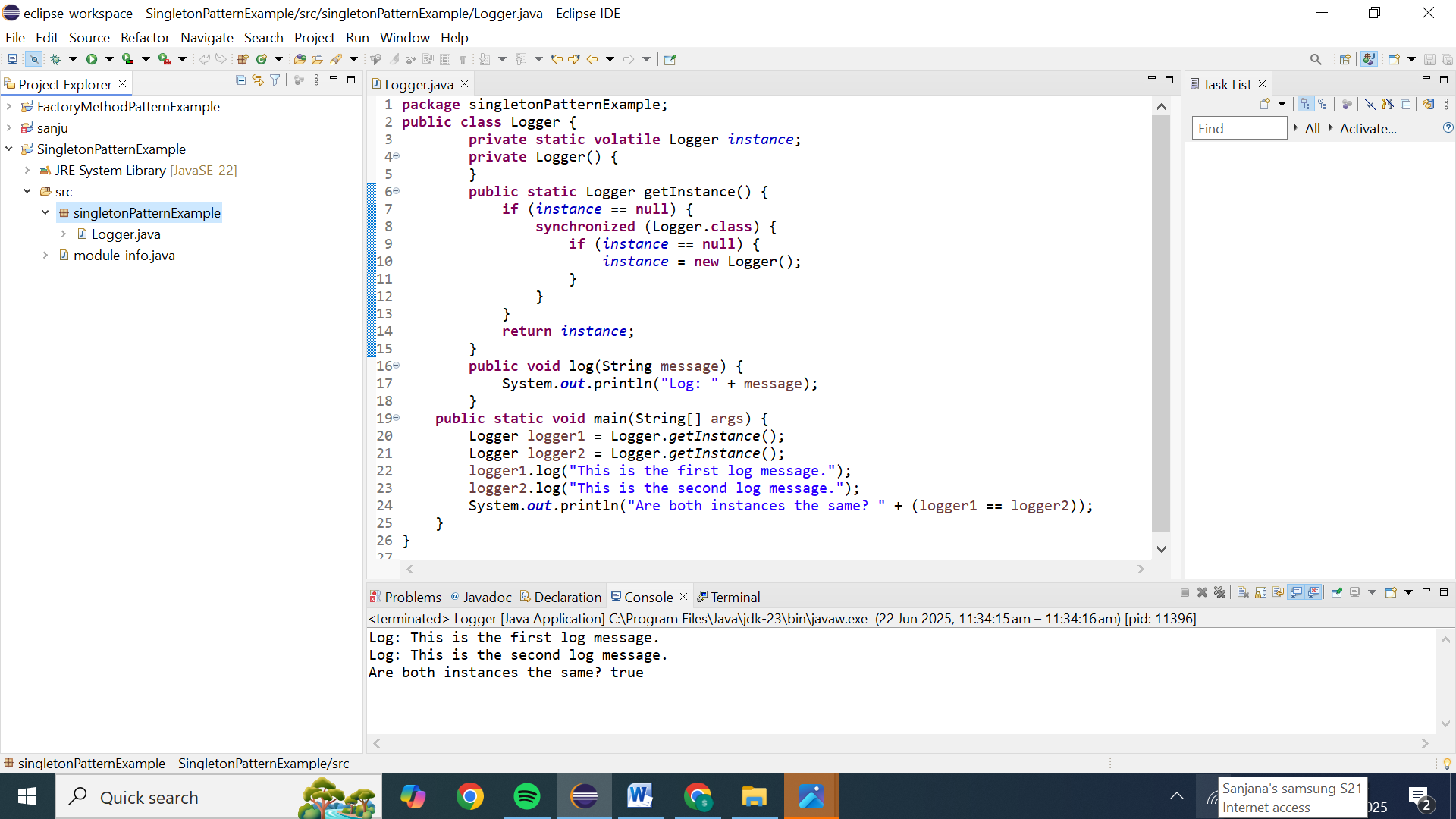
logger1.log("This is the first log message.");

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

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

}

}



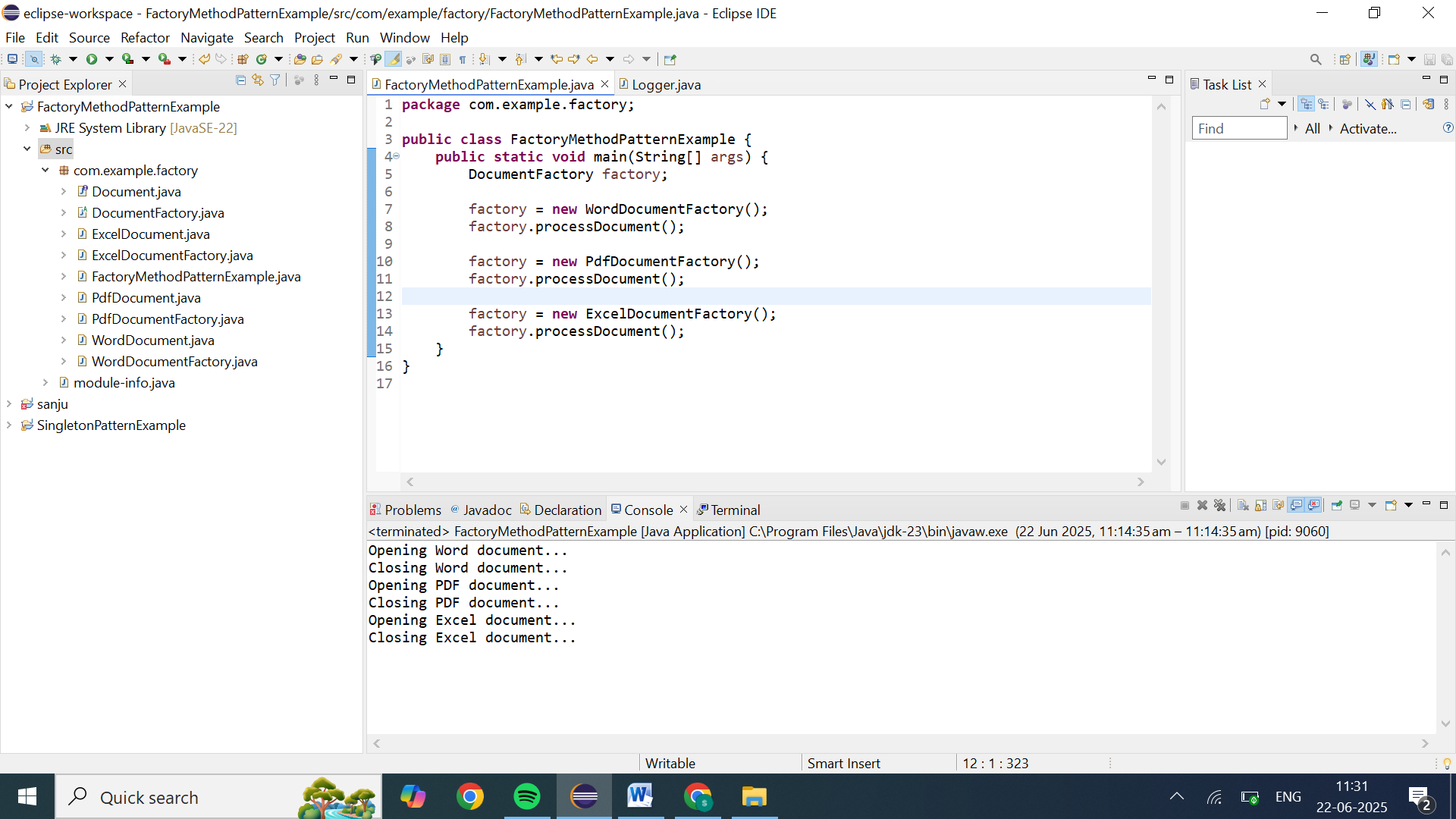
**2: Implementing the Factory Method Pattern**

**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.



**1. Split into separate files**

Create one .java file per **public** type, named exactly as the class/interface. For example:

FactoryMethodPatternExample/

src/

com/example/factory/

Document.java

WordDocument.java

PdfDocument.java

ExcelDocument.java

DocumentFactory.java

WordDocumentFactory.java

PdfDocumentFactory.java

ExcelDocumentFactory.java

FactoryMethodPatternExample.java

package com.example.factory;

**Document.java**

public interface Document {

void open();

void close();

}

**WordDocument.java**

package com.example.factory;

public class WordDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

}

**PdfDocument.java**

package com.example.factory;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

}

**ExcelDocument.java**

package com.example.factory;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

@Override

public void close() {

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

}

}

**DocumentFactory.java**

package com.example.factory;

public abstract class DocumentFactory {

public abstract Document createDocument();

public void processDocument() {

Document doc = createDocument();

doc.open();

// ... perform operations ...

doc.close();

}

}

**WordDocumentFactory.java**

package com.example.factory;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package com.example.factory;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package com.example.factory;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryMethodPatternExample.java**

**package** com.example.factory;

**public** **class** FactoryMethodPatternExample {

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

DocumentFactory factory;

factory = **new** WordDocumentFactory();

factory.processDocument();

factory = **new** PdfDocumentFactory();

factory.processDocument();

factory = **new** ExcelDocumentFactory();

factory.processDocument();

}

}