**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:**
   1. Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   1. Create a class named Logger that has a private static instance of itself.
   2. Ensure the constructor of Logger is private.
   3. Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   1. Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   1. Create a test class to verify that only one instance of Logger is created and used across the application

**Code :**

* **Logger class**

**package** com.example.singleton;

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

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

**private** Logger()

{

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

}

**public** **static** Logger getInstance()

{

**if**(*instance*==**null**)

{

*instance*=**new** Logger();

}

**return** *instance*;

}

**public** **void** log(String message)

{

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

}

}

* **TestLogger class**

**package** com.example.singleton;

**public** **class** TestLogger {

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

{

Logger logger1=Logger.*getInstance*();

Logger logger2=Logger.*getInstance*();

logger1.log("First log message");

logger2.log("Second log message");

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

System.***out***.println("Both logger instances are the same (Singleton works)");

}

**else** {

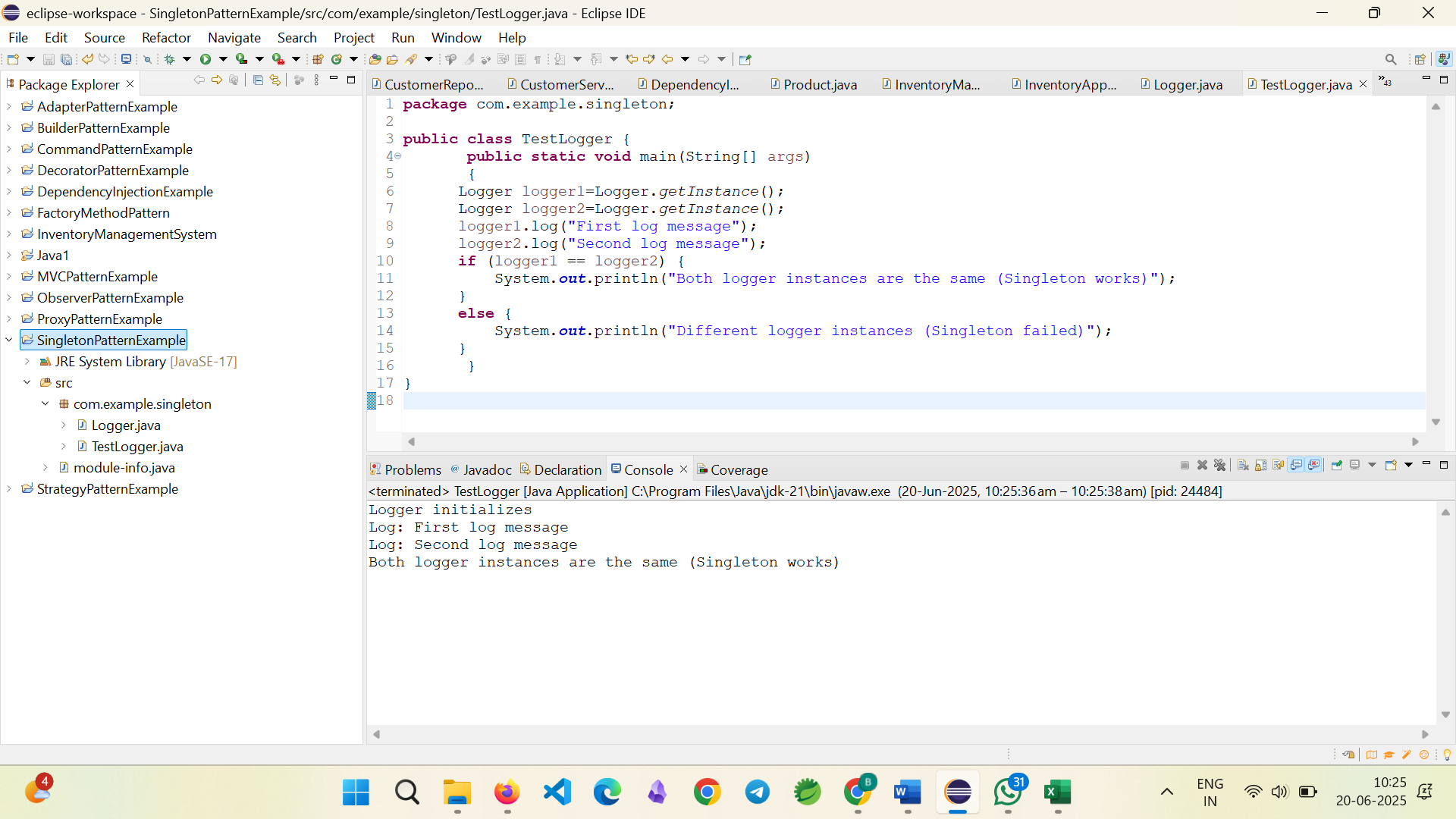
System.***out***.println("Different logger instances (Singleton failed)");

}

}

}

* **Output Screenshot:**



**----------------------------------------------------------------------------------------------------------**

**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:**
   1. Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   1. Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   1. Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   1. Create an abstract class **DocumentFactory** with a method **createDocument()**.
   2. Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   1. Create a test class to demonstrate the creation of different document types using the factory method.

**Code**

**Document Interface:**

**package** com.example.factory;

**public** **interface** Document {

**void** open();

}

**WordDocument Class**

**package** com.example.factory;

**public** **class** WordDocument **implements** Document {

@Override

**public** **void** open()

{

System.***out***.println("Opening word Document");

}

}

* **PdfDocument Class:**

**package** com.example.factory;

**public** **class** PdfDocument **implements** Document{

@Override

**public** **void** open()

{

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

}

}

* **ExcelDocument Class:**

**package** com.example.factory;

**public** **class** ExcelDocument **implements** Document {

@Override

**public** **void** open()

{

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

}

}

* **Abstract Document Factory:**

**package** com.example.factory;

**public** **abstract** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

* **Word Document Factory class:**

**package** com.example.factory;

**public** **class** WordDocumentFactory **extends** DocumentFactory {

@Override

**public** Document createDocument()

{

**return** **new** WordDocument();

}

}

* **PDF Document Factory class:**

**package** com.example.factory;

**public** **class** PdfDocumentFactory **extends** DocumentFactory{

@Override

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

* **Excel Document Factory class:**

**package** com.example.factory;

**public** **class** ExcelDocumentFactory **extends** DocumentFactory{

@Override

**public** Document createDocument() {

**return** **new** ExcelDocument();

}

}

* **Factory Pattern test in main class:**

**package** com.example.factory;

**public** **class** Main {

**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();

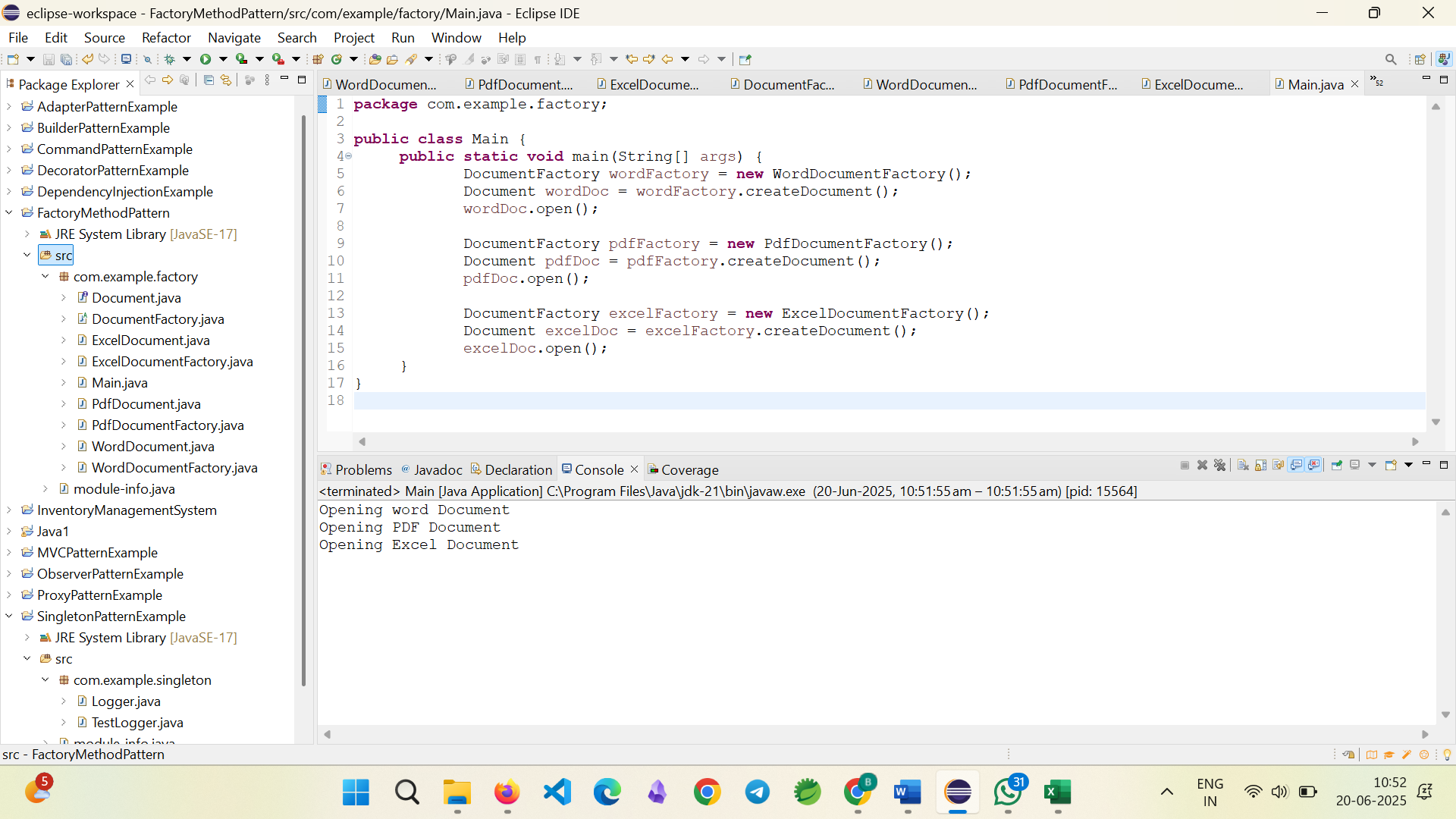
DocumentFactory excelFactory = **new** ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output Screenshot:**