**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.

**Solution-**

**1. Create a Project in eclipse**

**2. Defining Singleton Class**

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

**private** **static** Logger *obj*; // private instance

**private** Logger() { // private constructor

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

}

**public** **static** Logger getObj() { // public method to get instance

**if**(*obj* == **null**) {

*obj* = **new** Logger();

}

**return** *obj*;

}

**public** **void** log(String msg) {

System.***out***.println(msg);

}

}

**3. Implementation and Testing**

**public** **class** SingletonTest {

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

Logger l1 = Logger.*getObj*();

l1.log("First Log");

Logger l2 = Logger.*getObj*();

l2.log("Second Log");

**if**(l1 == l2) { // verify both logger instances are same

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

}

**else** {

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

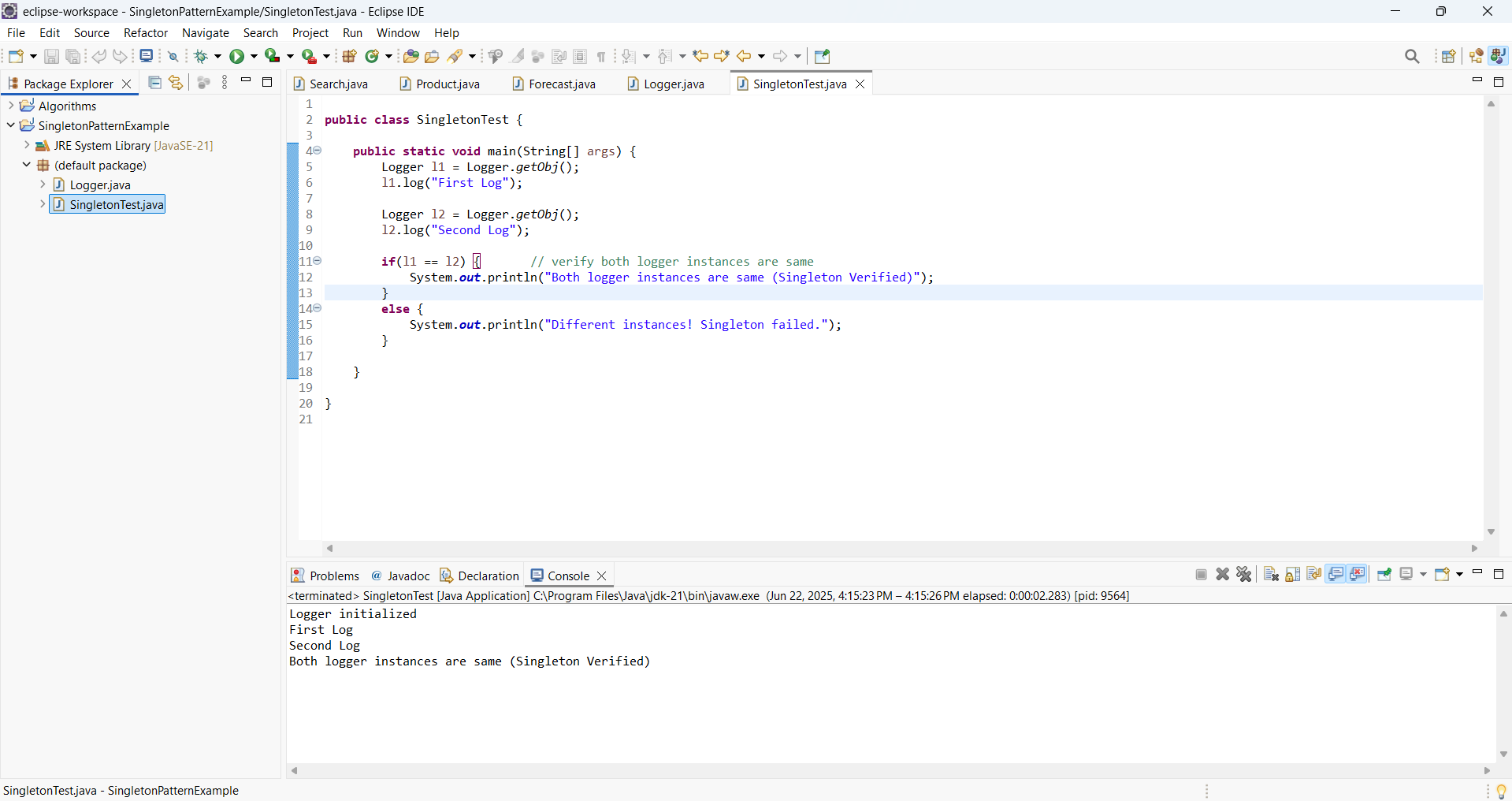
}

}

}

Here only one instance of Logger class will be created in the whole application.

**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.

**Solution-**

**1. Create a new Java project in eclipse.**

**2. Define Document Classes**

Interface Class-

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

**void** open();

}

**3. Create Concrete Document Class**

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

@Override

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

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

}

}

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

@Override

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

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

}

}

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

@Override

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

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

}

}

**4. Implement the Factory Method**

1. Abstract Factory Class

public abstract class DocumentFactory {

public abstract Document createDocument();

}

2. Concrete Factory Class

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

@Override

**public** Document createDocument() {

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

}

}

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

@Override

**public** Document createDocument() {

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

}

}

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

@Override

**public** Document createDocument() {

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

}

}

**5. Test the factory method implementation**

public class FactoryTest {

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();

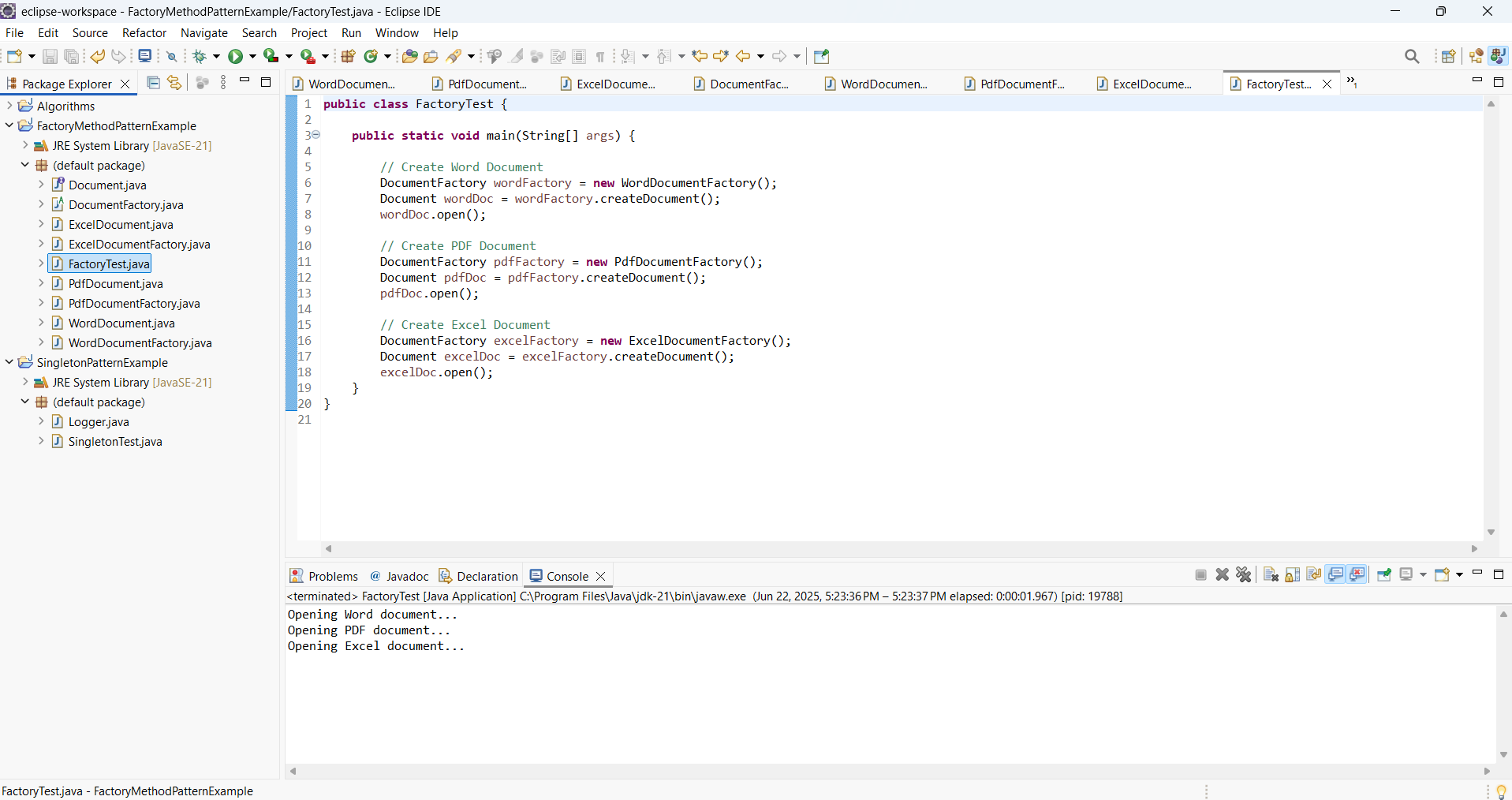
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output-**

****