**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 :**

**File Structure :**FactoryMethodPatternExample/

└── src/

├── application/

│ └── Main.java # Main class in package

├── documents/

│ ├── Document.java

│ ├── WordDocument.java

│ ├── PdfDocument.java

│ └── ExcelDocument.java

└── factories/

├── DocumentFactory.java

├── WordDocumentFactory.java

├── PdfDocumentFactory.java

└── ExcelDocumentFactory.java

**// Document.java**

package documents;

public interface Document {

void open();

void save();

}

**// WordDocument.java**

package documents;

public class WordDocument implements Document {

@Override

public void open() {

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

}

@Override

public void save() {

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

}

}

**// PdfDocument.java**

package documents;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

@Override

public void save() {

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

}

}

**// ExcelDocument.java**

package documents;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

@Override

public void save() {

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

}

}

**// DocumentFactory.java (Abstract Creator)**

package factories;

import documents.Document;

public abstract class DocumentFactory {

// Factory method

public abstract Document createDocument();

public void newDocument() {

Document doc = createDocument();

doc.open();

}

}

**// WordDocumentFactory.java (Concrete Creator)**

package factories;

import documents.Document;

import documents.WordDocument;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**// PdfDocumentFactory.java (Concrete Creator)**

package factories;

import documents.Document;

import documents.PdfDocument;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**// ExcelDocumentFactory.java (Concrete Creator)**

package factories;

import documents.Document;

import documents.ExcelDocument;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**import documents.Document;

import factories.DocumentFactory;

import factories.ExcelDocumentFactory;

import factories.PdfDocumentFactory;

import factories.WordDocumentFactory;

public class Main {

public static void main(String[] args) {

// Create different document factories

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

// Use factories to create documents

Document wordDoc = wordFactory.createDocument();

Document pdfDoc = pdfFactory.createDocument();

Document excelDoc = excelFactory.createDocument();

// Test document operations

System.out.println("-- Word Document Operations --");

wordDoc.open();

wordDoc.save();

System.out.println("\n-- PDF Document Operations --");

pdfDoc.open();

pdfDoc.save();

System.out.println("\n-- Excel Document Operations --");

excelDoc.open();

excelDoc.save();

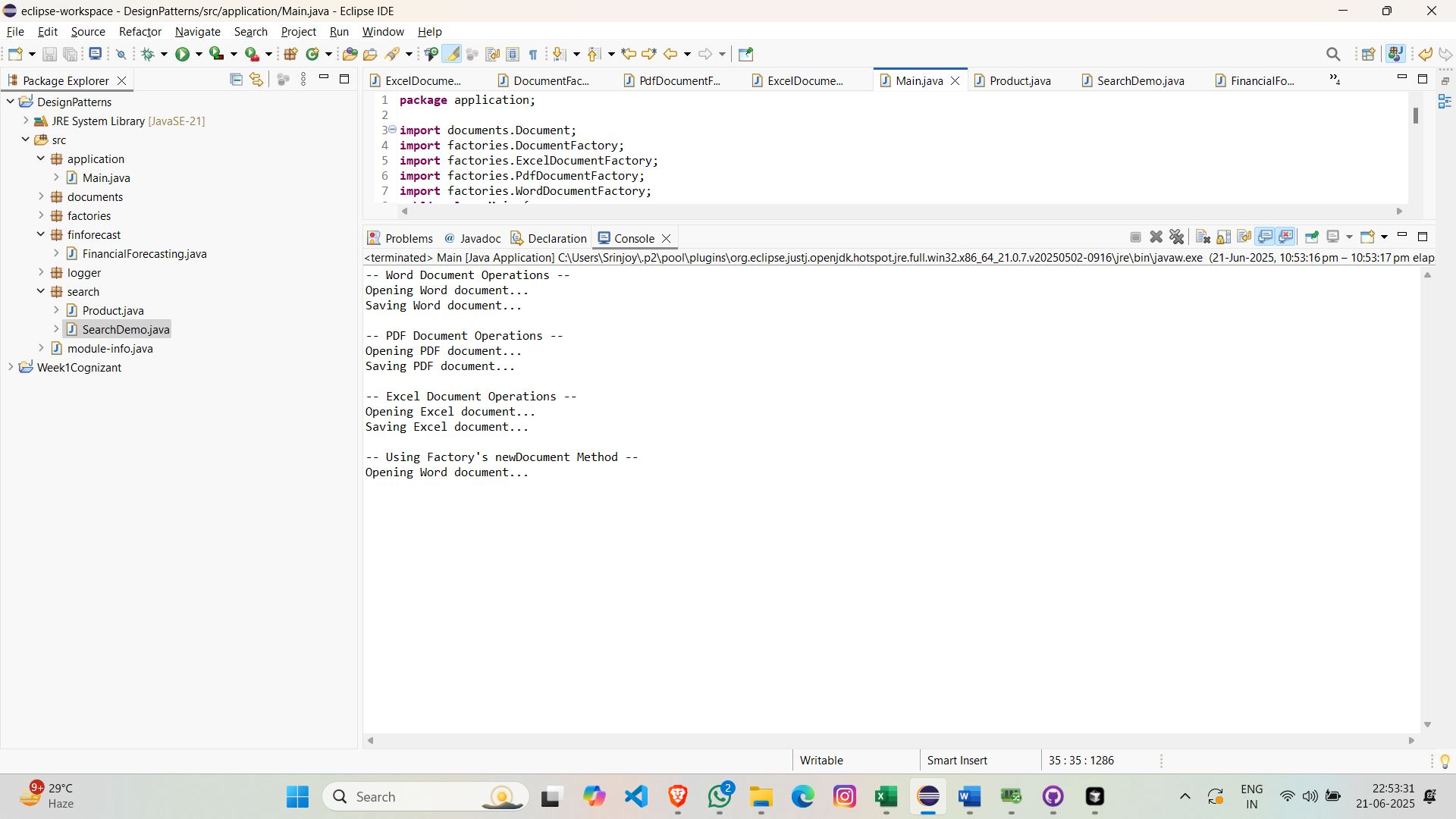
// Demonstrate polymorphic factory usage

System.out.println("\n-- Using Factory's newDocument Method --");

wordFactory.newDocument();

}

}

****

**Output :**

**Key Advantages of this Implementation:**

1. Loose Coupling: Client code interacts with interfaces, not concrete implementations
2. Single Responsibility: Each factory handles creation of one product type
3. Open/Closed Principle: New document types can be added by creating new factories without modifying existing code
4. Centralized Creation Logic: Document creation is encapsulated in dedicated factory classes
5. Polymorphic Behavior: Factories and products can be used interchangeably through their common interfaces