**Exercise 2: Implementing the Factory Method Pattern**

**Document.java**

package mypackage;

public interface Document {

void open();

}

**WordDocument.java**

package mypackage;

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocument.java**

package mypackage;

public class PdfDocument implements Document {

public void open() {

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

}

}

**ExcelDocument.java**

package mypackage;

public class ExcelDocument implements Document {

public void open() {

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

}

}

**DocumentFactory.java**

package mypackage;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package mypackage;

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package mypackage;

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package mypackage;

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**TestFactoryMethod.java**

package mypackage;

import java.util.Scanner;

public class TestFactoryMethod {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Select Document Type:");

System.out.println("1. Word Document");

System.out.println("2. PDF Document");

System.out.println("3. Excel Document");

System.out.print("Enter your choice: ");

int choice = sc.nextInt();

DocumentFactory factory = null;

switch (choice) {

case 1:

factory = new WordDocumentFactory();

break;

case 2:

factory = new PdfDocumentFactory();

break;

case 3:

factory = new ExcelDocumentFactory();

break;

default:

System.out.println("Invalid choice");

System.exit(0);

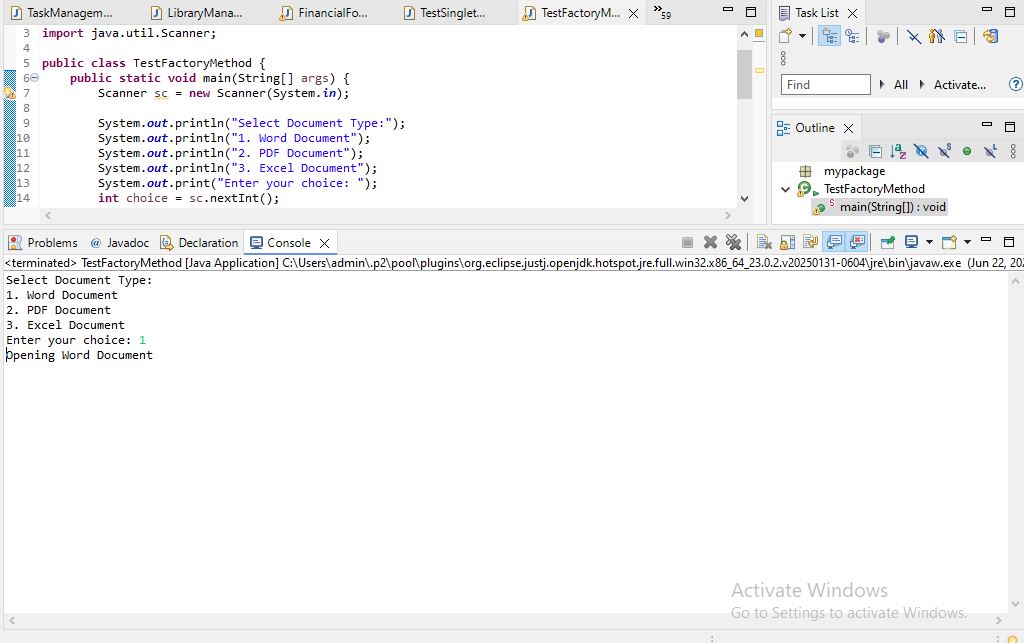
}

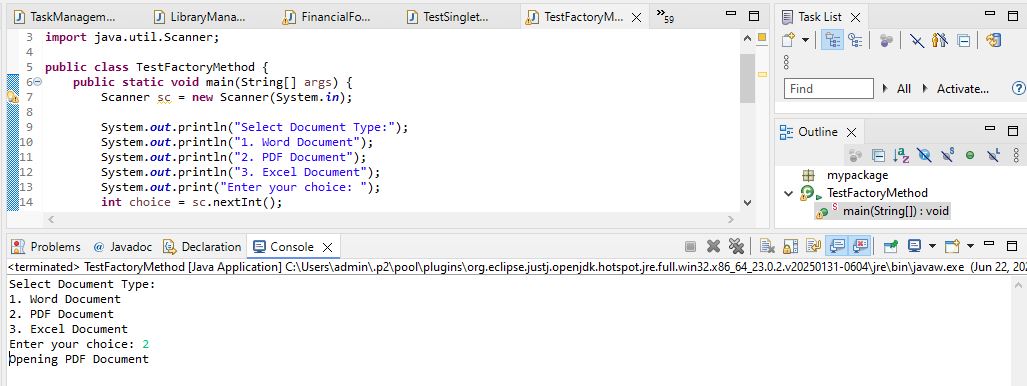
Document doc = factory.createDocument();

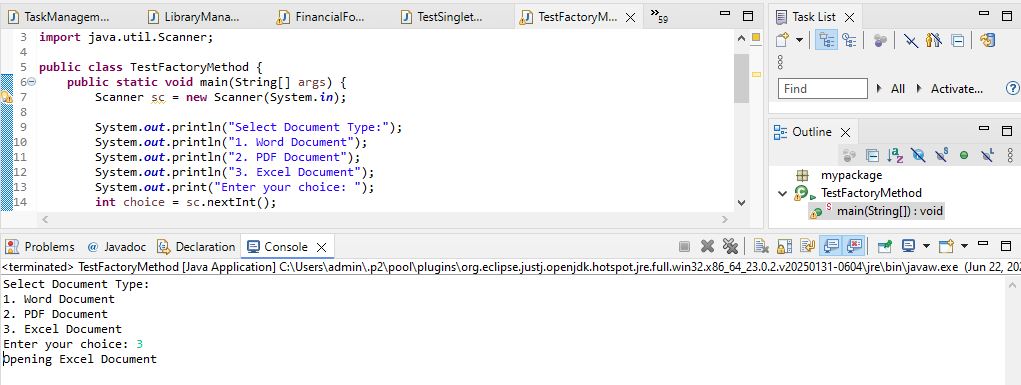
doc.open();

}

}







We are developing a document management system that creates different types of documents (Word, PDF, Excel). To achieve loose coupling and scalability, we use the Factory Method Pattern.

**1. Create Java Project:**

* A new Java project named FactoryMethodPatternExample is created.

**2. Define Document Interface:**

* A common interface Document is created with a method open() that all document types implement.

**3. Create Concrete Document Classes:**

* WordDocument, PdfDocument, and ExcelDocument classes implement the Document interface and provide their own version of the open() method.

**4. Implement Factory Method:**

* An abstract class DocumentFactory is created with an abstract method createDocument().
* Concrete factory classes WordDocumentFactory, PdfDocumentFactory, and ExcelDocumentFactory extend DocumentFactory and implement the createDocument() method to return the corresponding document object.

**5. Test Factory Method Implementation:**

* In the TestFactoryMethod class, the user selects the document type.
* Based on user choice, the corresponding factory is created.
* The factory creates the document object and calls open() method to simulate opening the document.

**Advantages of Factory Method Pattern:**

* Provides loose coupling between client code and object creation logic.
* Adding new document types requires creating new factory classes without changing existing code.
* Promotes open/closed principle: open for extension, closed for modification.

**Time Complexity:**

* The time complexity for object creation is O(1), as only one object is created at a time.

**Real-life Usage:**

* Document editors (Microsoft Office, Google Docs)
* File readers
* GUI toolkits