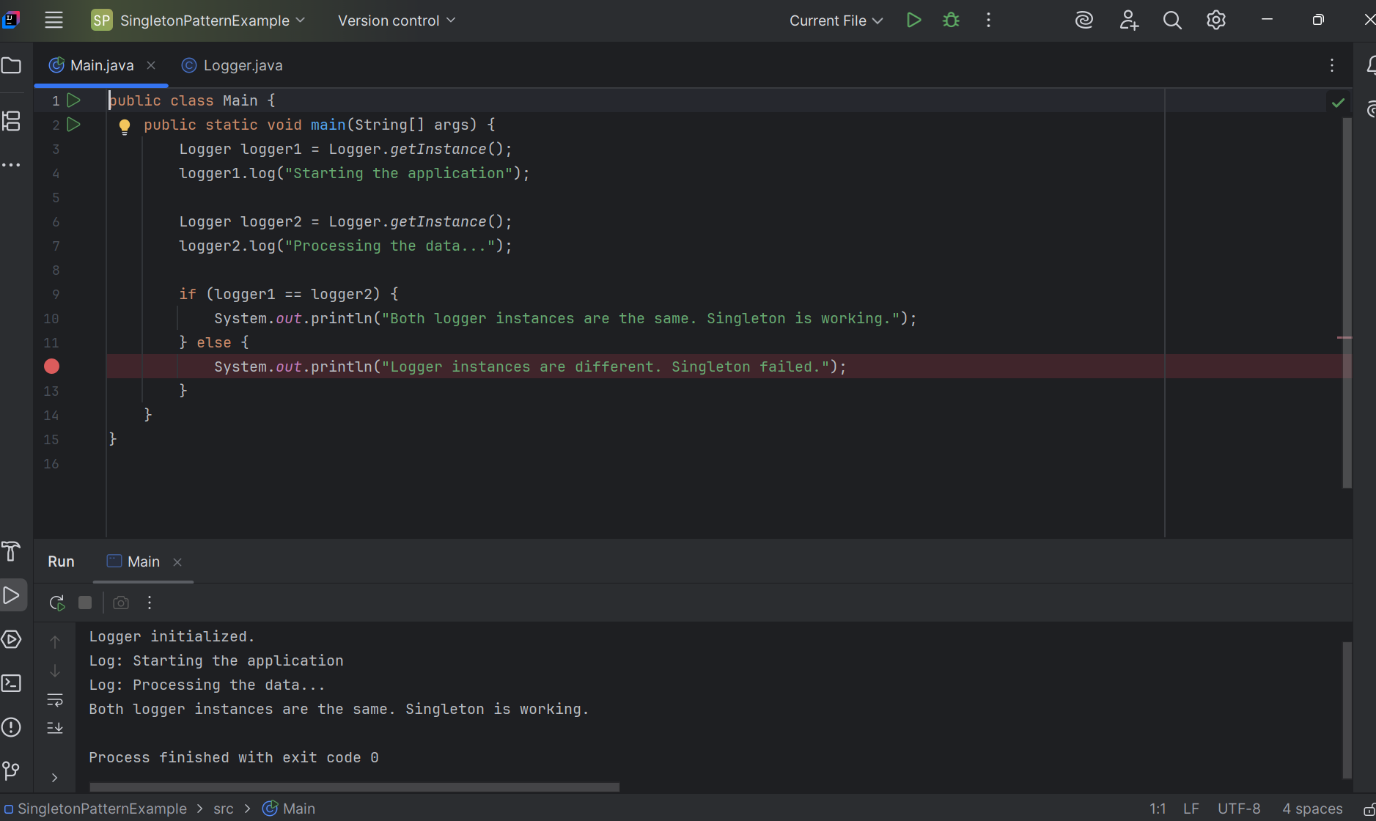
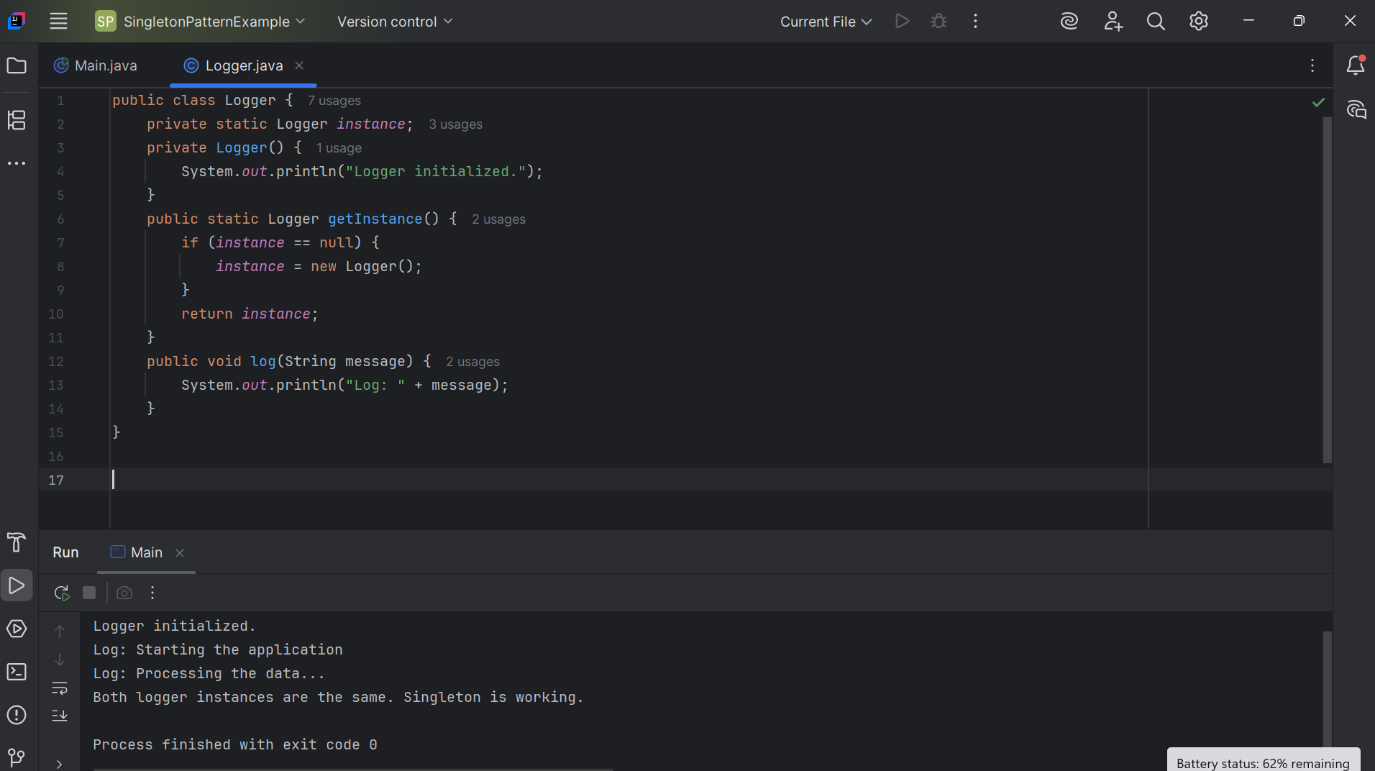
**Cognizant Java FSE Program**

**Week 1 : Design principles & Patterns**

1. **Implementing the Singleton Pattern**





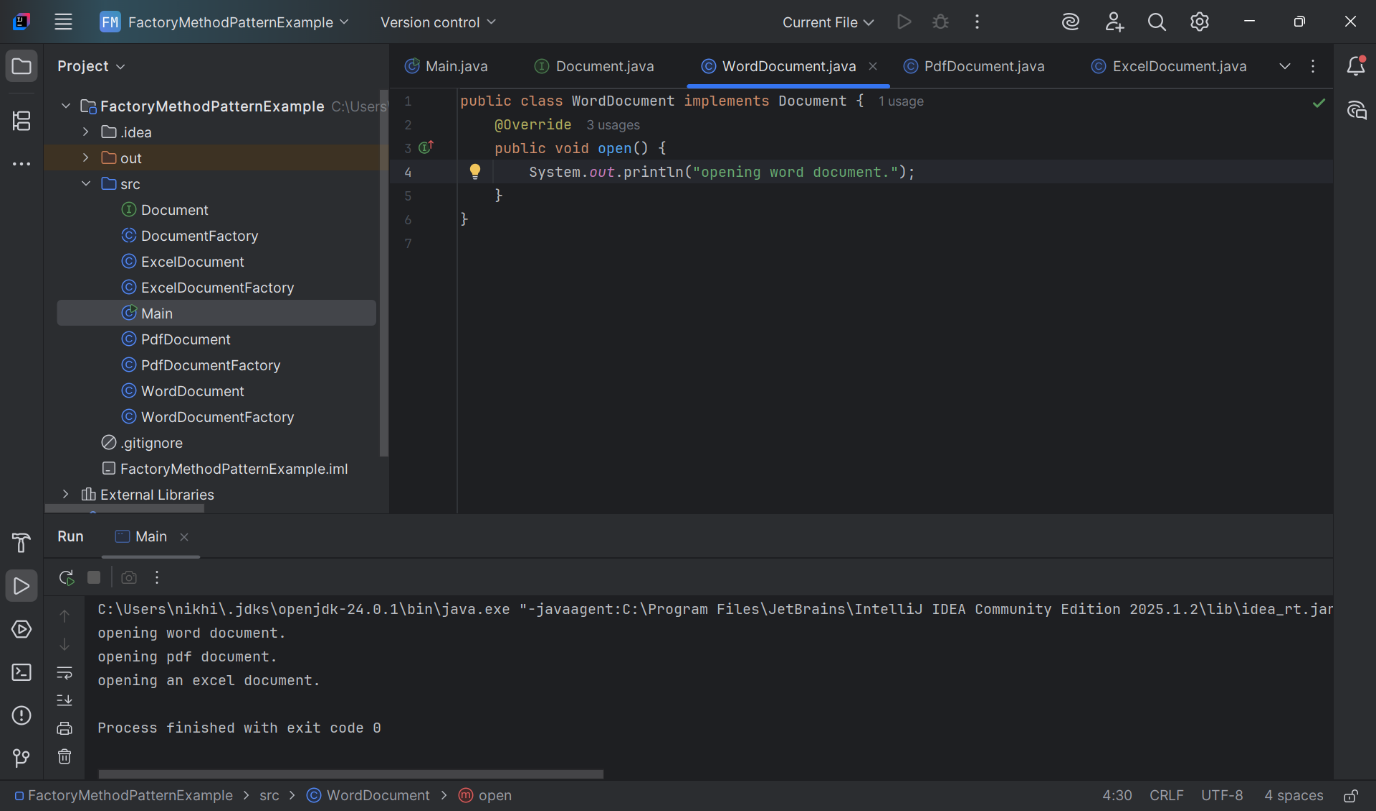
Code:

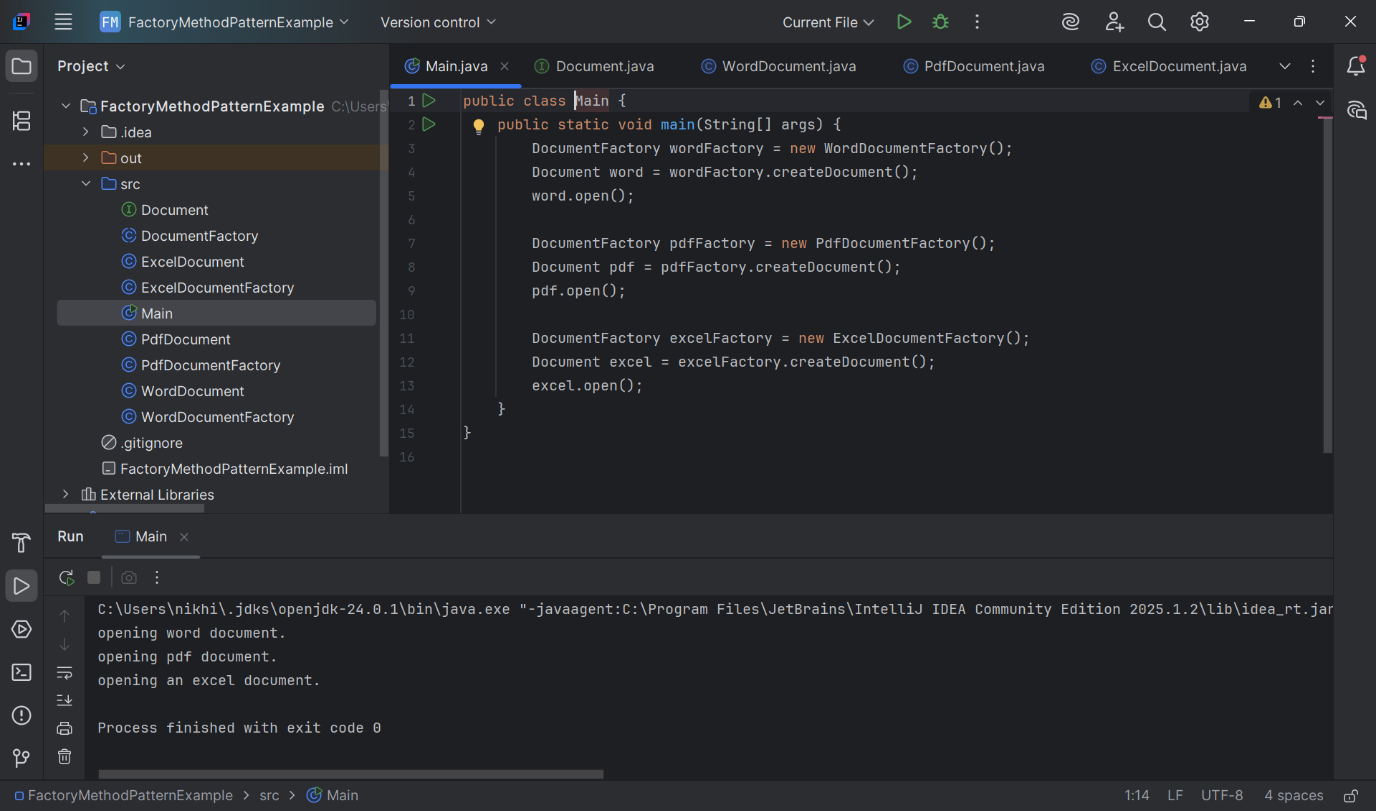
public class Logger {  
 private static Logger instance;  
 private Logger() {  
 System.out.println("Logger initialized.");  
 }  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
 public void log(String message) {  
 System.out.println("Log: " + message);  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 logger1.log("Starting the application");  
  
 Logger logger2 = Logger.getInstance();  
 logger2.log("Processing the data...");  
  
 if (logger1 == logger2) {  
 System.out.println("Both logger instances are the same. Singleton is working.");  
 } else {  
 System.out.println("Logger instances are different. Singleton failed.");  
 }  
 }  
}

After running the code, the output confirms that the Logger class is correctly implemented as a singleton. Only one instance is initialized, and all logging is done through that single instance.

**2. Implementing the Factory Method Pattern**





**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}

I implemented the Factory Method Design Pattern to build a flexible document management system. Instead of directly instantiating document objects, I created factories that produce documents like Word, PDF, and Excel.