**Week-1**

**Design Patterns, Data Structures & Algorithms**

**(Engineering Concepts)**

1. **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.

**Logged.java:-**

package singleton;

public class Logged {

private static Logged *singleInstance*;

private Logged() {

System.*out*.println("Logged instance created.");

}

public static Logged getInstance() {

if (*singleInstance* == null) {

*singleInstance* = new Logged();

}

return *singleInstance*;

}

public void log(String message) {

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

}

}

**Main.java :-**

package singleton;

public class Main {

public static void main(String[] args) {

Logged logged1 = Logged.*getInstance*();

Logged logged2 = Logged.*getInstance*();

logged1.log("First message.");

logged2.log("Second message.");

if (logged1 == logged2) {

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

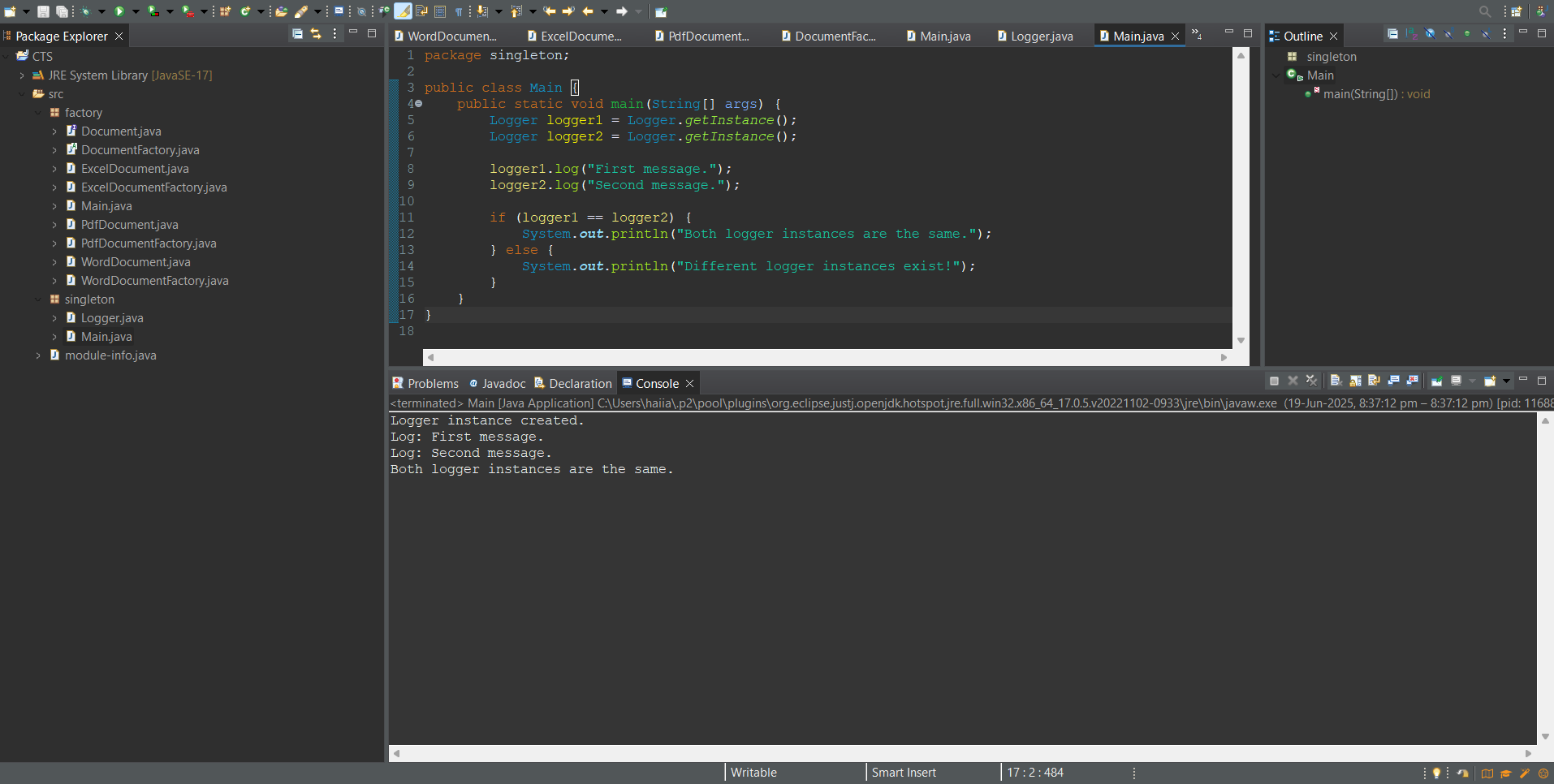
} else {

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

}

}

}

**Output :- **

1. **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.

**Doc.java:-**

package factory;

public interface Doc {

void open();

}

**WordDoc.java:-**

package factory;

public class WordDoc implements Doc {

public void open() {

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

}

}

**PdfDoc.java:-**

package factory;

public class PdfDoc implements Doc {

public void open() {

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

}

}

**ExcelDoc.java:-**

package factory;

public class ExcelDoc implements Doc {

public void open() {

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

}

}

**DocFactory.java:-**

package factory;

public abstract class DocFactory {

public abstract Document createDoc();

}

**WordDocFactory.java:-**

package factory;

public class WordDocFactory extends DocFactory {

public Document createDoc() {

return new WordDoc();

}

}

**PdfDocFactory.java:-**

package factory;

public class PdfDocFactory extends DocFactory {

public Document createDoc() {

return new PdfDoc();

}

}

**ExcelDocFactory.java:-**

package factory;

public class ExcelDocFactory extends DocFactory {

public Document createDoc() {

return new ExcelDoc();

}

}

**Main.java:-**

package factory;

public class Main {

public static void main(String[] args) {

DocFactory wordFactory = new WordDocFactory();

Doc wordDoc = wordFactory.createDoc();

wordDoc.open();

DocFactory pdfFactory = new PdfDocFactory();

Doc pdfDoc = pdfFactory.createDoc();

pdfDoc.open();

DocFactory excelFactory = new ExcelDocFactory();

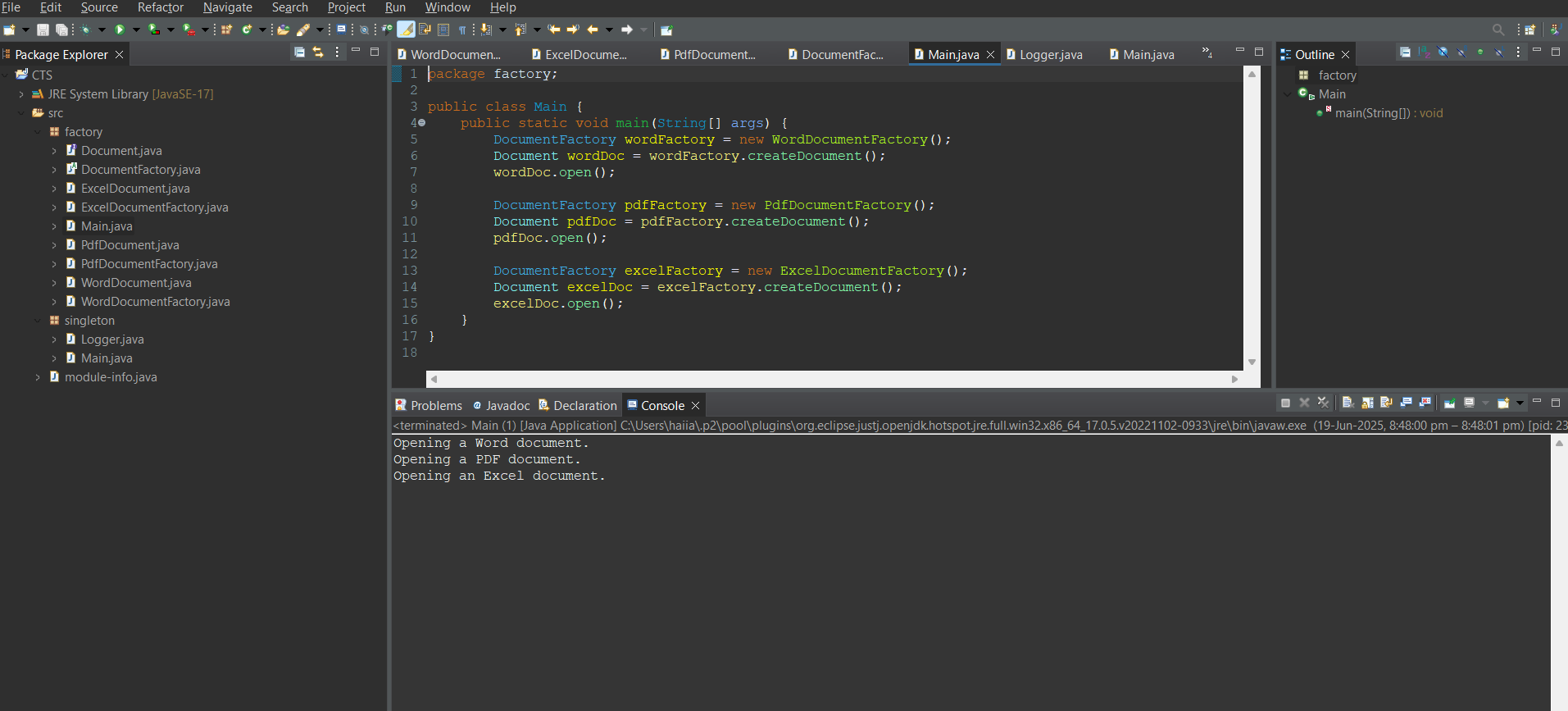
Doc excelDoc = excelFactory.createDoc();

excelDoc.open();

}

}

**Output :-**

****

1. **E-commerce Platform Search Function:-**

**Scenario:** You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Product.java:-**

package E\_Com\_Search;

public class Product {

int pId;

String pName, cate;

public Product(int pId, String pName, String cate) {

this.pId = pId;

this.pName = pName;

this.cate = cate;}

public String toString() {

return "[" + pId + ", " + pName + ", " + cate + "]";

}

}

**SearchFunc.java:-**

package E\_ComSearch;

import java.util.\*;

public class SearchEngine {

public static Product linearSearch(Product[] products, String pName) {

for (Product product : products) {

if (product.productName.equalsIgnoreCase(productName)) {

return product;}}

return null;

}

public static Product binarySearch(Product[] products, String pName) {

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = products[mid].pName.compareToIgnoreCase(pName);

if (cmp == 0) return products[mid];

else if (cmp < 0) left = mid + 1;

else right = mid - 1;}

return null;}

public static void sortProducts(Product[] products) {

Arrays.*sort*(products, Comparator.*comparing*(p -> p.pName.toLowerCase()));

}

}

**Main.java:-**

package E\_Com\_Search;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter product name to search: ");

String input = sc.nextLine();

Product result1 = SearchFunc.*linearSearch*(products, inputProduct);

if (result1 != null) {

System.*out*.println("Product found using Linear Search: " + result1);

} else {

System.*out*.println("Product not found using Linear Search.");}

SearchFunc.*sortProducts*(products);

Product result2 = SearchFunc.*binarySearch*(products, inputProduct);

if (result2 != null) {

System.*out*.println("Product found using Binary Search: " + result2);

} else {

System.*out*.println("Product not found using Binary Search.");

}

sc.close();

}

**Output:-**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Financial Forecasting**

**Scenario:** You are developing a financial forecasting tool that predicts future values based on past data.

**Forecast.java:-**

package Financial\_Forecast;

import java.util.Scanner;

public class Forecast {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

double curVal = sc.nextDouble();

double growthRate = sc.nextDouble();

int yrs = sc.nextInt();

double futureVal = Forecast.*calculateFutureValue*(currentValue, growthRate, years);

System.*out*.printf("Predicted future value after %d years: %.2f\n", years, futureVal();

sc.close();

}

public static double calcuFutureVal(double curVal, double growthRate, int yrs) {

if (years == 0) {

return curVal;

}

return *calcuFutureVal*(curVal, growthRate, yrs - 1) \* (1 + growthRate);

}

}

**A computer screen shot of a program

AI-generated content may be incorrect.**