**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

package singleton;

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);

}

}

**Main.java**

package singleton;

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Starting application...");

Logger logger2 = Logger.getInstance();

logger2.log("Logging second message.");

if (logger1 == logger2) {

System.out.println("Both logger1 and logger2 are the same instance.");

} else {

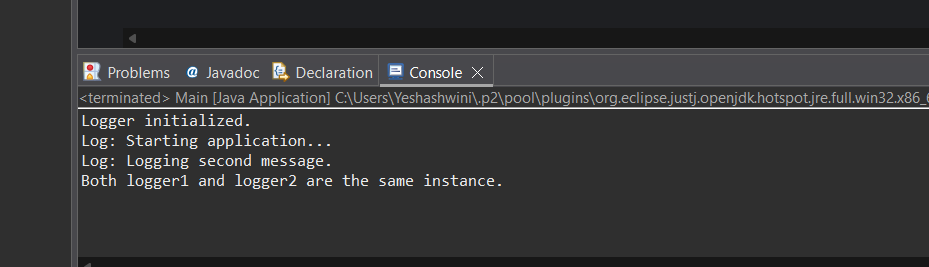
System.out.println("Different instances detected.");

}

}

}

**Output:**



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

**Document.java**

package factorymethod;

public interface Document {

void open();

}

**WordDocument.java**

package factorymethod;

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocument.java**

package factorymethod;

public class PdfDocument implements Document {

public void open() {

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

}

}

**ExcelDocument.java**

package factorymethod;

public class ExcelDocument implements Document {

public void open() {

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

}

}

**DocumentFactory.java**

package factorymethod;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package factorymethod;

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package factorymethod;

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package factorymethod;

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**

package factorymethod;

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();

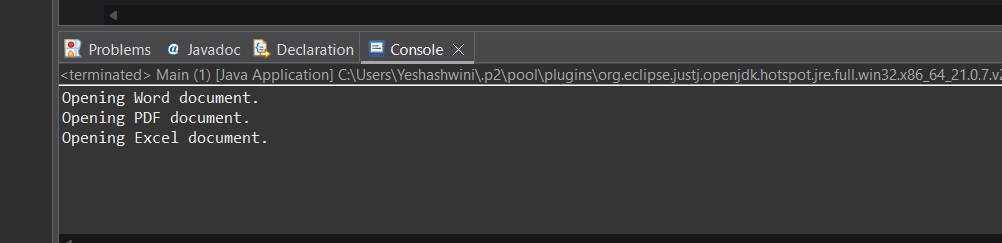
Document excel = excelFactory.createDocument();

excel.open();

}

}

**Output:**

****

**Algorithms\_Data Structures**

**Exercise 2: E-commerce Platform Search Function**

**Product.java**

package ecommerce;

public class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**SearchUtil.java**

package ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtil {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

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

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

else high = mid - 1;

}

return null;

}

public static void sortByName(Product[] products) {

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

}

}

**Main.java**

package ecommerce;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "T-shirt", "Clothing"),

new Product(103, "Coffee Mug", "Kitchen"),

new Product(104, "Headphones", "Electronics"),

new Product(105, "Notebook", "Stationery")

};

String searchTerm = "Headphones";

Product result1 = SearchUtil.linearSearch(products, searchTerm);

System.out.println("Linear Search Result:");

System.out.println(result1 != null ? result1 : "Product not found");

SearchUtil.sortByName(products);

Product result2 = SearchUtil.binarySearch(products, searchTerm);

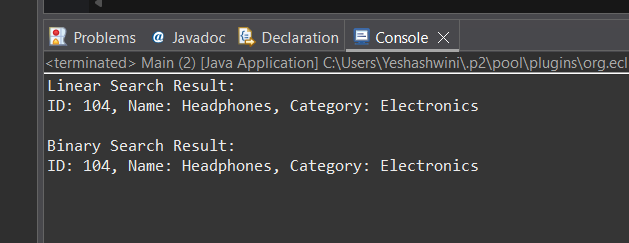
System.out.println("\nBinary Search Result:");

System.out.println(result2 != null ? result2 : "Product not found");

}

}

**Output:**

****

**Exercise 7: Financial Forecasting**

**FinancialForecast.java**

package finance;

public class FinancialForecast {

public static double futureValue(double presentValue, double rate, int years) {

if (years == 0) return presentValue;

return futureValue(presentValue, rate, years - 1) \* (1 + rate);

}

public static double futureValueMemo(double presentValue, double rate, int years, double[] memo) {

if (years == 0) return presentValue;

if (memo[years] != 0) return memo[years];

memo[years] = futureValueMemo(presentValue, rate, years - 1, memo) \* (1 + rate);

return memo[years];

}

public static double futureValueIterative(double presentValue, double rate, int years) {

double result = presentValue;

for (int i = 0; i < years; i++) {

result \*= (1 + rate);

}

return result;

}

public static void main(String[] args) {

double presentValue = 10000;

double rate = 0.05;

int years = 5;

double fvRec = futureValue(presentValue, rate, years);

System.out.printf("Future Value (Recursive): %.2f%n", fvRec);

double[] memo = new double[years + 1];

double fvMemo = futureValueMemo(presentValue, rate, years, memo);

System.out.printf("Future Value (Memoized): %.2f%n", fvMemo);

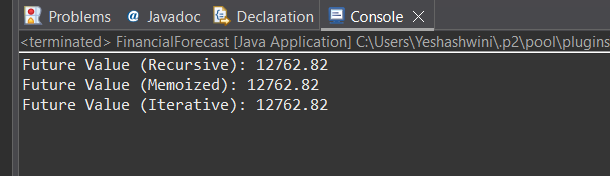
double fvIter = futureValueIterative(presentValue, rate, years);

System.out.printf("Future Value (Iterative): %.2f%n", fvIter);

}

}

**Output:**

****