**DIGITAL NURTURE 4.0 JAVA FSE**

**WEEK\_1 HANDS\_ON**

**SUPERSET ID : 6391195**

**Name : Monish B**

**Design principles & Patterns**

**Exercise\_1 : Implementing the Singleton Pattern**

**Logger.java**

package Week\_1.Ex\_1\_Singleton;

public class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger initialized.");

}

public static synchronized Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("LOG: " + message);

}

}

**Main.java**

package Week\_1.Ex\_1\_Singleton;

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("Application started.");

logger2.log("User logged in.");

if (logger1 == logger2) {

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

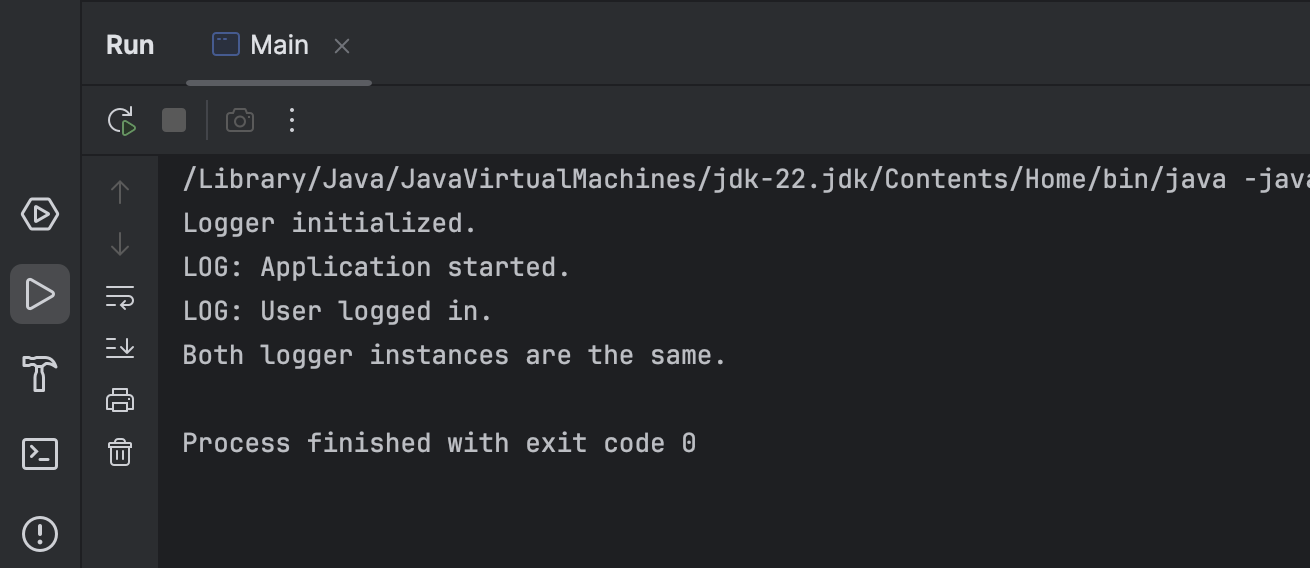
} else {

System.out.println("Different logger instances exist.");

}

}

}

**OUTPUT**

**Design principles & Patterns**

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

**Document.java**

package Week\_1.Ex\_2\_FactoryMethod;

public interface Document {

void open();

}

**WordDocument.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class WordDocument implements Document {

public void open() {

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

}

}

**PdfDocment.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class PdfDocument implements Document {

public void open() {

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

}

}

**ExcelDocument.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class ExcelDocument implements Document {

public void open() {

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

}

}

**DocumentFactory.java**

package Week\_1.Ex\_2\_FactoryMethod;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package Week\_1.Ex\_2\_FactoryMethod;

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**

package Week\_1.Ex\_2\_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();

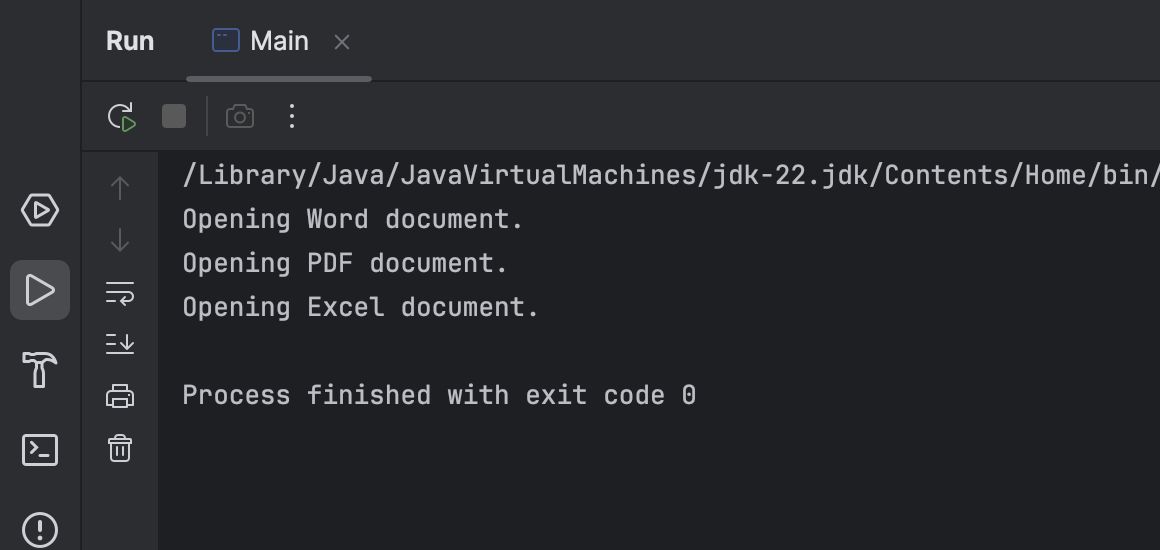
Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT**

****

**Data structures and Algorithms**

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

**Product.java**

package Week\_1.Ex\_3\_Search;

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 "[" + productId + "] " + productName + " (" + category + ")";

}

}

**SearchService.java**

package Week\_1.Ex\_3\_Search;

import java.util.Arrays;

import java.util.Comparator;

public class SearchService {

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

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

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 Week\_1.Ex\_3\_Search;

public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shoes", "Fashion"),

new Product(103, "Mobile", "Electronics"),

new Product(104, "Book", "Education"),

new Product(105, "Headphones", "Electronics")

};

System.out.println("Linear Search for 'Book':");

Product result1 = SearchService.linearSearch(products, "Book");

System.out.println(result1 != null ? result1 : "Not Found");

SearchService.sortByName(products);

System.out.println("\nBinary Search for 'Mobile':");

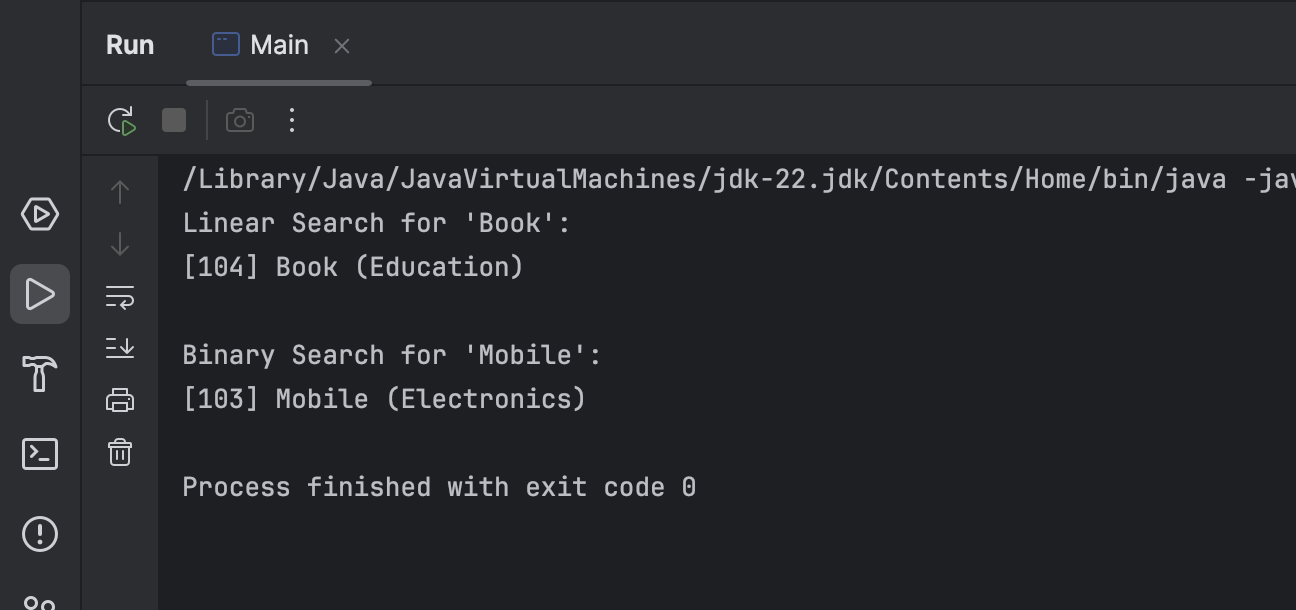
Product result2 = SearchService.binarySearch(products, "Mobile");

System.out.println(result2 != null ? result2 : "Not Found");

}

}

**OUTPUT**

****

**Data structures and Algorithms**

**Exercise 7: Financial Forecasting**

**FinancialForecaster.java**

package Week\_1.Ex\_7\_Financial;

public class FinancialForecaster {

public static double forecastValue(double initialValue, double growthRate, int years) {

if (years == 0) {

return initialValue;

} else {

return forecastValue(initialValue, growthRate, years - 1) \* (1 + growthRate);

}

}

public static double forecastValueMemo(double initialValue, double growthRate, int years, double[] memo) {

if (years == 0) return initialValue;

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

memo[years] = forecastValueMemo(initialValue, growthRate, years - 1, memo) \* (1 + growthRate);

return memo[years];

}

}

**Main.java**

package Week\_1.Ex\_7\_Financial;

public class Main {

public static void main(String[] args) {

double initial = 10000.0; // Initial investment

double rate = 0.05; // 5% annual growth

int years = 10;

double futureVal = FinancialForecaster.forecastValue(initial, rate, years);

System.out.printf("Future Value (recursive): %.2f\n", futureVal);

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

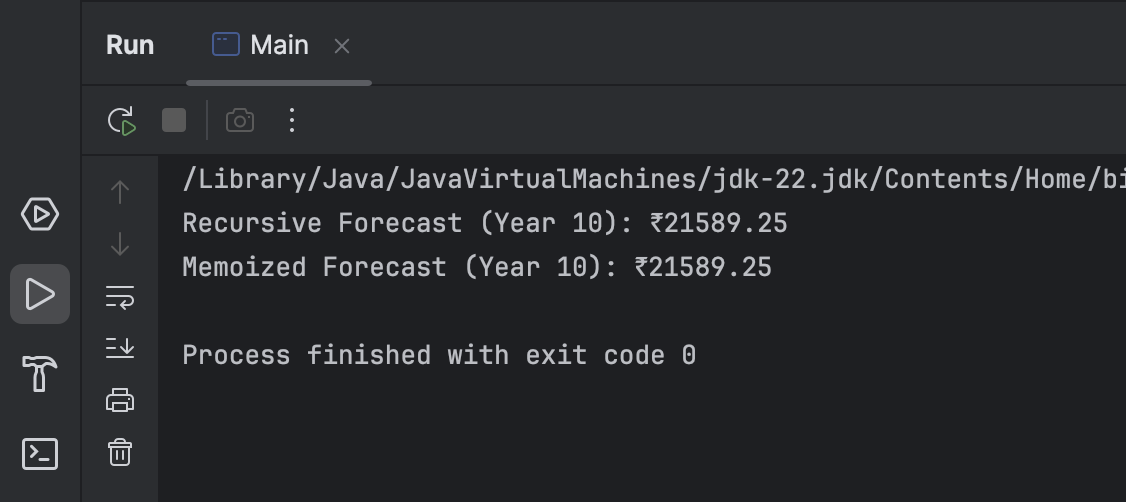
double futureMemo = FinancialForecaster.forecastValueMemo(initial, rate, years, memo);

System.out.printf("Future Value (memoized): %.2f\n", futureMemo);

}

}

**OUTPUT**

****