**HANDS ON WEEK 1:**

**Module 1 - Design Principles and Pattern**

**Exercise 1 : Implementing the Singleton Pattern**

**CODE:**

public class Logger {

// Private static instance of the Logger class

private static Logger instance;

// Private constructor to prevent instantiation

private Logger() {

// Initialization code here

}

// Public static method to get the instance of Logger class

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

// Example method for logging

public void log(String message) {

System.out.println(message);

}

}

public class SingletonPatternTest {

public static void main(String[] args) {

// Get instance of Logger

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

// Checking if both instances are the same

System.out.println("Is logger1 == logger2 ? " + (logger1 == logger2)); // Should print true

// Test logging

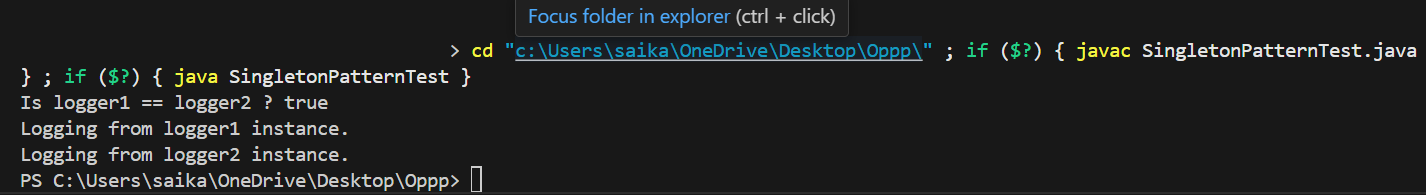
logger1.log("Logging from logger1 instance.");

logger2.log("Logging from logger2 instance.");

}

}

**OUTPUT:**

****

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

**CODE:**

// Document.java (Interface)

public interface Document {

void open();

}

// WordDocument.java

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

// PdfDocument.java

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

// ExcelDocument.java

public class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening 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();

}

}

// FactoryPatternTest.java

public class FactoryPatternTest {

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();

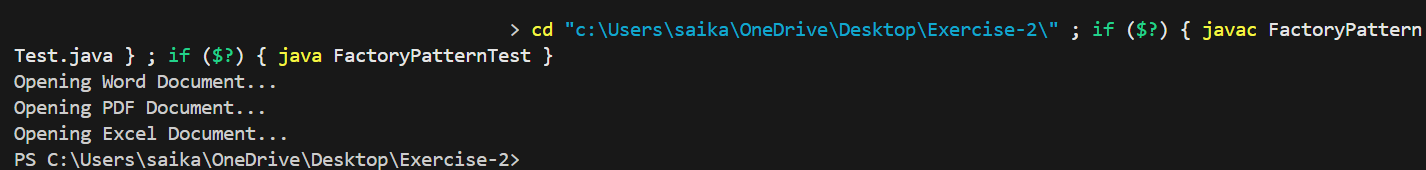
Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT:**



**Module 2 – Data Structures and Algorithms**

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

**CODE:**

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;

}

@Override

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

// Linear Search by product name

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

for (Product product : products) {

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

return product;

}

}

return null;

}

// Binary Search by product name (sorted by name)

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

int low = 0, 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;

}

// Sorting helper

public static Product[] getSortedProductsByName(Product[] products) {

Product[] sorted = Arrays.copyOf(products, products.length);

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

return sorted;

}

}

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "Table", "Furniture"),

new Product(105, "Watch", "Accessories")

};

// Linear Search

String searchTarget = "Phone";

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

Product foundLinear = ProductSearch.linearSearch(products, searchTarget);

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

// Binary Search

System.out.println("\n Binary Search:");

Product[] sorted = ProductSearch.getSortedProductsByName(products);

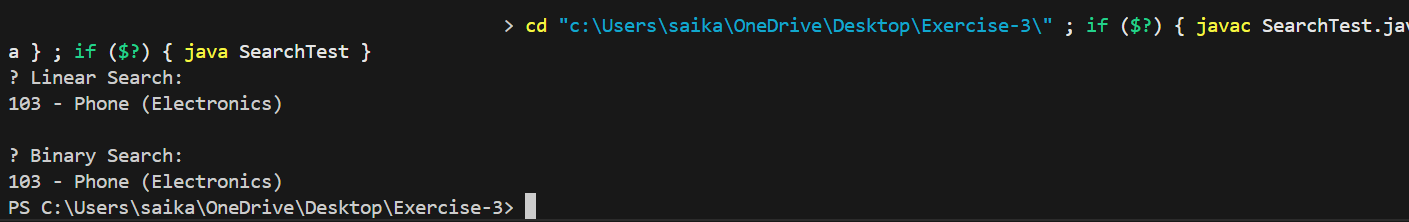
Product foundBinary = ProductSearch.binarySearch(sorted, searchTarget);

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

}

}

**OUTPUT:**

****

**Exercise 7: Financial Forecasting**

**CODE:**

public class FinancialForecast {

// Recursive method to calculate future value

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

if (years == 0) {

return presentValue;

}

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

}

}

public class ForecastTest {

public static void main(String[] args) {

double presentValue = 10000; // Initial investment

double annualRate = 0.08; // 8% growth per year

int years = 5;

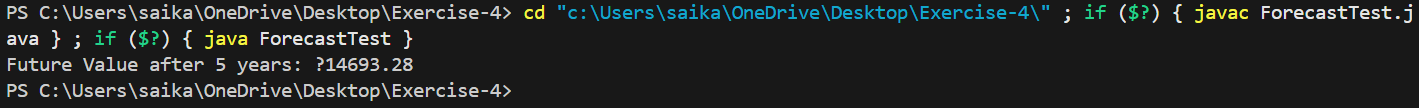
double futureValue = FinancialForecast.calculateFutureValue(presentValue, annualRate, years);

System.out.printf("Future Value after %d years: ₹%.2f%n", years, futureValue);

}

}

**OUTPUT:**

****