**Mandatory Hands-on week 1**

**Algorithms Data Structure (CODE)**

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

import java.util.\*;

class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() { return productId; }

public String getProductName() { return productName; }

public String getCategory() { return category; }

@Override

public String toString() {

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

}

}

interface SearchStrategy {

Product search(Product[] products, String name);

}

class LinearSearch implements SearchStrategy {

public Product search(Product[] products, String name) {

for (Product product : products) {

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

return product;

}

}

return null;

}

}

class BinarySearch implements SearchStrategy {

public Product search(Product[] products, String name) {

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

while (low <= high) {

int mid = (low + high) / 2;

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

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

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

else high = mid - 1;

}

return null;

}

}

class ProductSorter {

public static void sortByName(Product[] products) {

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

}

}

class ProductDisplay {

public static void showResult(Product result) {

if (result != null) {

System.out.println("Found: " + result);

} else {

System.out.println("Product not found.");

}

}

}

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Footwear"),

new Product(3, "Watch", "Accessories"),

new Product(4, "Phone", "Electronics"),

new Product(5, "Bag", "Fashion")

};

String searchName = "Phone";

SearchStrategy linear = new LinearSearch();

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

ProductDisplay.showResult(linear.search(products, searchName));

ProductSorter.sortByName(products);

SearchStrategy binary = new BinarySearch();

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

ProductDisplay.showResult(binary.search(products, searchName));

}

}

**Exercise 7: Financial Forecasting**

import java.util.\*;

class FinancialData {

private final double initialValue;

private final double growthRate;

public FinancialData(double initialValue, double growthRate) {

this.initialValue = initialValue;

this.growthRate = growthRate;

}

public double getInitialValue() {

return initialValue;

}

public double getGrowthRate() {

return growthRate;

}

}

interface ForecastStrategy {

double forecast(int years);

}

class RecursiveForecaster implements ForecastStrategy {

private final FinancialData data;

public RecursiveForecaster(FinancialData data) {

this.data = data;

}

@Override

public double forecast(int years) {

if (years == 0) {

return data.getInitialValue();

}

return forecast(years - 1) \* (1 + data.getGrowthRate());

}

}

class MemoizedForecaster implements ForecastStrategy {

private final FinancialData data;

private final Map<Integer, Double> memo = new HashMap<>();

public MemoizedForecaster(FinancialData data) {

this.data = data;

}

@Override

public double forecast(int years) {

if (years == 0) return data.getInitialValue();

if (memo.containsKey(years)) return memo.get(years);

double result = forecast(years - 1) \* (1 + data.getGrowthRate());

memo.put(years, result);

return result;

}

}

class ForecastPrinter {

public static void print(double result, int years) {

System.out.printf("Projected value after %d years: %.2f\n", years, result);

}

}

public class Main {

public static void main(String[] args) {

double initial = 10000;

double growthRate = 0.07;

int years = 5;

FinancialData data = new FinancialData(initial, growthRate);

ForecastStrategy forecaster = new MemoizedForecaster(data);

double result = forecaster.forecast(years);

ForecastPrinter.print(result, years);

}

}

***DESIGN PATTERNS AND PRINCIPLE(CODE)***

**EXERCISE 1 : IMPLEMENTING THE SINGLETON PATTERN**

Creating singleton pattern class :

package SingletonPatternExample;

public class Logger {

private static Logger singleInstance;

private Logger() {

System.out.println("Logger instance created");

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

Test the singleton Implementation :

package SingletonPatternExample;

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

if (logger1 == logger2) {

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

} else {

System.out.println("Logger instances are different!");

}

}

}

**Exercise 2 :IMPLEMENTING THE FACTORY METHOD PATTERN**

Defining Document Classes

File: Document.java

public interface Document {

void open();

}

Creating Concrete Document Classes :

WordDocument.java

public class WordDocument implements Document {

public void open() {

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

}

}

PdfDocument.java

public class PdfDocument implements Document {

public void open() {

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

}

}

ExcelDocument.java

public class ExcelDocument implements Document {

public void open() {

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

}

}

Step 4: Implementing the Factory Method

File: DocumentFactory.java

public abstract class DocumentFactory {

public abstract Document createDocument(); // factory method

}

File: WordDocumentFactory.java

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

File: PdfDocumentFactory.java

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

File: ExcelDocumentFactory.java

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

Step 5: Test the Factory Method Implementation

File: Main.java

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}