**WEEK 1**

**DESIGN PATTERNS AND PRINCIPLES**

**EXERCISE 1**: SINGLETON PATTERN

CODE:

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

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("Singleton confirmed: Both references point to the same instance.");

} else {

System.out.println("Singleton failed: Different instances exist.");

}

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**EXERCISE 2: FACTORY METHOD PATTERN**

INTERFACE :

package FactoryMethodPatternExample;

public interface Document {

void open();

}

WORD DOCUMENT CLASS

package FactoryMethodPatternExample;

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

PDF DOCUMENT CLASS

package FactoryMethodPatternExample;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

EXCEL DOCUMENT CLASS

package FactoryMethodPatternExample;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

ABSTRACT FACTORY CLASS

package FactoryMethodPatternExample;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

WORDFACTORY

package FactoryMethodPatternExample;

public class WordFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

PDF FACTORY

package FactoryMethodPatternExample;

public class PdfFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

EXCEL FACTORY

package FactoryMethodPatternExample;

public class ExcelFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

FACTORY PATTERN

package FactoryMethodPatternExample;

public class TestFactoryPattern {

public static void main(String[] args) {

// Create Word document

DocumentFactory wordFactory = new WordFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

// Create PDF document

DocumentFactory pdfFactory = new PdfFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

// Create Excel document

DocumentFactory excelFactory = new ExcelFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**DATA STRUCTURES AND ALGORITHMS**

**EXERCISE 2: E-COMMERCE PLATFORM SEARCH FUNCTION**:

PRODUCT CLASS

package com.ecommerce.search;

public class Product implements Comparable<Product> {

private String productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

// Getters

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

@Override

public int compareTo(Product other) {

return this.productId.compareTo(other.productId);

}

@Override

public String toString() {

return "Product [ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";

}

}

SEARCH IMPLEMENTATION CLASS:

package com.ecommerce.search;

import java.util.Arrays;

public class SearchImplementation {

private Product[] productsArray; // For linear search

private Product[] sortedProductsArray; // For binary search

public SearchImplementation(Product[] products) {

this.productsArray = Arrays.copyOf(products, products.length);

this.sortedProductsArray = Arrays.copyOf(products, products.length);

Arrays.sort(sortedProductsArray); // Sort for binary search

}

// Linear search by product ID

public Product linearSearchById(String productId) {

for (Product product : productsArray) {

if (product.getProductId().equals(productId)) {

return product;

}

}

return null;

}

// Binary search by product ID

public Product binarySearchById(String productId) {

int left = 0;

int right = sortedProductsArray.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int comparison = sortedProductsArray[mid].getProductId().compareTo(productId);

if (comparison == 0) {

return sortedProductsArray[mid];

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

// Linear search by product name

public Product linearSearchByName(String productName) {

for (Product product : productsArray) {

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

return product;

}

}

return null;

}

// Display all products (for verification)

public void displayProducts() {

System.out.println("All Products:");

for (Product product : productsArray) {

System.out.println(product);

}

}

// Display sorted products (for verification)

public void displaySortedProducts() {

System.out.println("Sorted Products (by ID):");

for (Product product : sortedProductsArray) {

System.out.println(product);

}

}

}

MAIN CLASS:

package com.ecommerce.search;

public class Main {

public static void main(String[] args) {

// Create sample products

Product[] products = {

new Product("P100", "Smartphone", "Electronics"),

new Product("P200", "Laptop", "Electronics"),

new Product("P300", "Desk Chair", "Furniture"),

new Product("P400", "Coffee Maker", "Appliances"),

new Product("P500", "Bluetooth Speaker", "Electronics")

};

SearchImplementation search = new SearchImplementation(products);

// Display products

search.displayProducts();

System.out.println();

search.displaySortedProducts();

System.out.println();

// Test searches

String searchId = "P300";

String searchName = "Laptop";

// Linear search by ID

System.out.println("Linear Search by ID (" + searchId + "):");

Product resultLinearId = search.linearSearchById(searchId);

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

// Binary search by ID

System.out.println("\nBinary Search by ID (" + searchId + "):");

Product resultBinaryId = search.binarySearchById(searchId);

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

// Linear search by name

System.out.println("\nLinear Search by Name (" + searchName + "):");

Product resultLinearName = search.linearSearchByName(searchName);

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

// Search for non-existent product

String nonExistentId = "P999";

System.out.println("\nSearch for non-existent ID (" + nonExistentId + "):");

Product resultNotFound = search.binarySearchById(nonExistentId);

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

}

}

OUTPUT:

A screenshot of a computer program

AI-generated content may be incorrect.

**EXERCISE 7: FINANCIAL FORECASTING**

FINANCIAL FORECAST CLASS:

public class FinancialForecast {

// Recursive method to calculate future value

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

if (years == 0) {

return presentValue;

}

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

}

public static void main(String[] args) {

double pv = 10000; // Present Value

double growthRate = 0.05; // 5% annual growth

int period = 5; // Forecasting for 5 years

double fv = futureValue(pv, growthRate, period);

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

}

}

A screen shot of a computer program

AI-generated content may be incorrect.