**WEEK 1**

1. **DESIGN PRINCIPLES AND PATTERNS**
2. **IMPLEMENTING THE SINGLETON PATTERN**

**INPUT:**

class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger(); // Lazy initialization

}

return instance;

}

public void log(String message) {

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

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First message");

Logger logger2 = Logger.getInstance();

logger2.log("Second message");

if (logger1 == logger2) {

System.out.println("Same Logger instance is used.");

} else {

System.out.println("Different Logger instances exist!");

}

}

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**2.IMPLEMENTING THE FACTORY METHOD PATTERN**

**INPUT:**

interface Notification {

interface Document {

void open();

void close();

}

class WordDocument implements Document {

public void open() {

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

}

public void close() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

public void close() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

public void close() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

public void openAndCloseDocument() {

Document doc = createDocument();

doc.open();

doc.close();

}

}

class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory factory;

Document document;

factory = new WordDocumentFactory();

document = factory.createDocument();

document.open();

document.close();

System.out.println();

factory = new PdfDocumentFactory();

document = factory.createDocument();

document.open();

document.close();

System.out.println();

factory = new ExcelDocumentFactory();

document = factory.createDocument();

document.open();

document.close();

}

}

**OUTPUT:**

**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

**B.DATASTRUCTURES AND ALGORITHMS**

**1. E-COMMERCE PLATFORM SEARCH FUNCTION**

**INPUT:**

import java.util.\*;

class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

public String toString() {

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

}

}

public class Main {

// 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 (Array must be sorted by productName)

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

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

else if (cmp < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "T-shirt", "Clothing"),

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

};

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

Product result1 = linearSearch(products, "Mouse");

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

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

System.out.println("\n🔍 Binary Search for 'Mouse':");

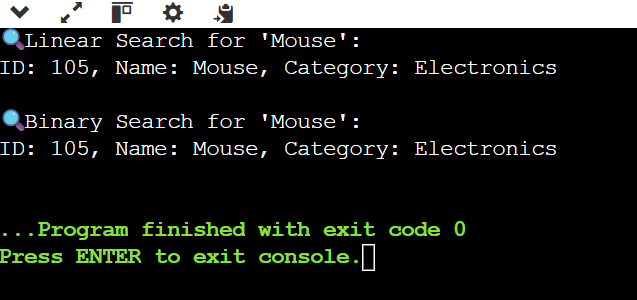
Product result2 = binarySearch(products, "Mouse");

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

}

}

**OUTPUT:**

****

**2.FINANCIAL FORECASTING**

**INPUT:**

public class Main {

// Recursive method to calculate future value

public static double futureValueRecursive(double initial, double rate, int years) {

if (years == 0) {

return initial;

} else {

return futureValueRecursive(initial, rate, years - 1) \* (1 + rate);

}

}

// Optimized version using memoization

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

if (years == 0) return initial;

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

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

return memo[years];

}

public static void main(String[] args) {

double initial = 1000.0; // Starting investment

double rate = 0.05; // 5% annual growth

int years = 10;

System.out.println("📈 Recursive Forecast:");

double result1 = futureValueRecursive(initial, rate, years);

System.out.printf("Future value after %d years: $%.2f\n", years, result1);

System.out.println("\n⚡ Optimized Forecast with Memoization:");

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

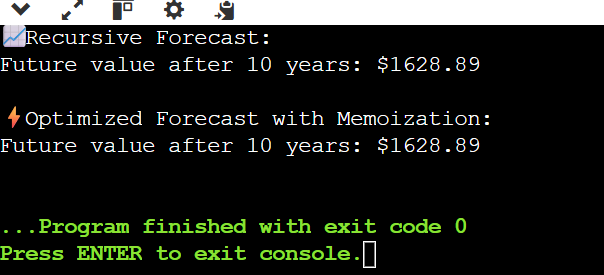
double result2 = futureValueMemo(initial, rate, years, memo);

System.out.printf("Future value after %d years: $%.2f\n", years, result2);

}

}

**OUTPUT:**

****