**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

public class SingletonPatternExample{

    static class Logger{

        private static Logger instance;

        private Logger(){

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

        }

        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();

        Logger logger2 = Logger.getinstance();

        logger1.log("The First Message");

        logger2.log("The Second Message");

        if(logger1==logger2){

            System.out.println("The pattern works!!");

        }

        else{

            System.out.println("The pattern does not work.");

        }

    }

}



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

class FactoryMethodPatternExample{

    // Document interface

    interface Document{

        void open();

    }

    // Document classes

    static class WordDocument implements Document{

        public void open(){

            System.out.println("This is a Word Document");

        }

    }

    static class PdfDocument implements Document{

        public void open(){

            System.out.println("This is a Pdf Document");

        }

    }

    static class ExcelDocument implements Document{

        public void open(){

            System.out.println("This is an Excel Document");

        }

    }

    // Abstract Document class

    static abstract class DocumentFactory{

        public abstract Document createDocument();

    }

    //Concrete factory classes for abstract Document class

    static class WordDocumentFactory extends DocumentFactory{

        public Document createDocument(){

            return new WordDocument();

        }

    }

    static class PdfDocumentFactory extends DocumentFactory{

        public Document createDocument(){

            return new PdfDocument();

        }

    }

    static class ExcelDocumentFactory extends DocumentFactory{

        public Document createDocument(){

            return new ExcelDocument();

        }

    }

    // Testing method

    public static void main(String []args){

        DocumentFactory word = new WordDocumentFactory();

        Document w = word.createDocument();

        w.open();

        DocumentFactory pdf = new PdfDocumentFactory();

        Document p = pdf.createDocument();

        p.open();

        DocumentFactory excel = new ExcelDocumentFactory();

        Document e = excel.createDocument();

        e.open();

    }

}



Algorithms and Data Structures

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

**import java.util.Arrays;**

**import java.util.Scanner;**

**class Product implements Comparable<Product> {**

**int productId;**

**String productName;**

**String category;**

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

**this.productId = productId;**

**this.productName = productName;**

**this.category = category;**

**}**

**// For binary search to sort and compare**

**public int compareTo(Product other) {**

**return this.productName.compareToIgnoreCase(other.productName);**

**}**

**public String toString() {**

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

**}**

**}**

**public class ECommerceSearch {**

**// Linear Search**

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

**for (Product p : products) {**

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

**return p;**

**}**

**}**

**return null;**

**}**

**// Binary Search (assumes sorted array)**

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

**int left = 0, right = products.length - 1;**

**while (left <= right) {**

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

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

**if (cmp == 0)**

**return products[mid];**

**else if (cmp < 0)**

**left = mid + 1;**

**else**

**right = mid - 1;**

**}**

**return null;**

**}**

**public static void main(String[] args) {**

**Product[] productList = {**

**new Product(101, "Shoes", "Footwear"),**

**new Product(102, "T-Shirt", "Clothing"),**

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

**new Product(104, "Watch", "Accessories"),**

**new Product(105, "Laptop", "Electronics")**

**};**

**Scanner sc = new Scanner(System.in);**

**System.out.println("Enter product name to search:");**

**String searchName = sc.nextLine();**

**// Linear Search**

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

**Product linearResult = linearSearch(productList, searchName);**

**if (linearResult != null)**

**System.out.println(linearResult);**

**else**

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

**// Sort for binary search**

**Arrays.sort(productList);**

**// Binary Search**

**System.out.println("Binary Search Result:");**

**Product binaryResult = binarySearch(productList, searchName);**

**if (binaryResult != null)**

**System.out.println(binaryResult);**

**else**

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

**sc.close();**

**}**

**}**

A screenshot of a computer

AI-generated content may be incorrect.

Explain Big O notation and how it helps in analyzing algorithms.

1. It tells us how fast or how slow an algorithm is when a data grows.
2. It ignores small constants and focusses on worst-case growth scenario.

Describe the best, average, and worst-case scenarios for search operations.

1. In case of linear search: Best Case: O(1)- found early, Average Case: O(n/2) and Worst Case: O(n)- at the end
2. In case of binary search: Best Case: O(1)- middle hit, Average Case: O(log n) and Worst Case: O(log n)

Compare the time complexity of linear and binary search algorithms.

1. Linear Search - O(n)
2. Binary Search – O(log n)

Discuss which algorithm is more suitable for your platform and why.

The algorithm which is most suitable for my platform is Binary Search. This is because Binary Search is significantly faster, especially as product data grows, also a time complexity of log n ensures quick responses.

**Exercise 7: Financial Forecasting**

**import java.util.Scanner;**

**class FinancialForecast{**

**static double calculateFutureValue(double presentValue, double annualRate, int years) {**

**if (years == 0) {**

**return presentValue;**

**}**

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

**}**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("Financial Forecasting Tool");**

**System.out.print("Enter current investment amount: ");**

**double presentValue = sc.nextDouble();**

**System.out.print("Enter expected annual growth rate (e.g., 0.05 for 5%): ");**

**double growthRate = sc.nextDouble();**

**System.out.print("Enter number of years to forecast: ");**

**int years = sc.nextInt();**

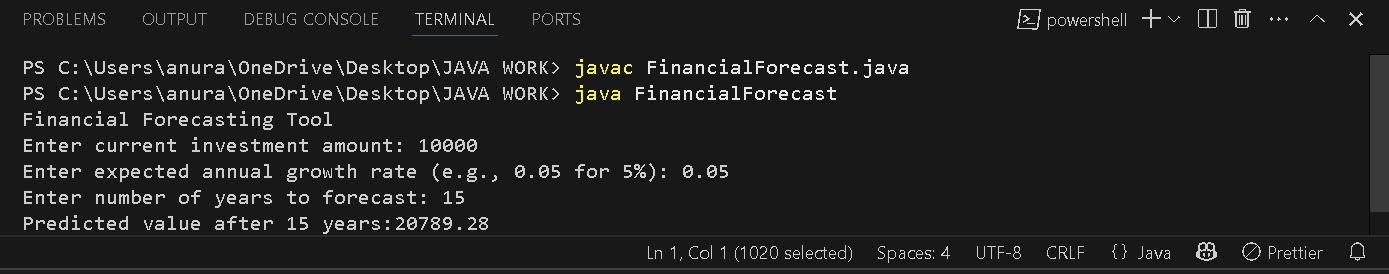
**double future Value = calculateFutureValue(presentValue, growthRate, years);**

**System.out.printf("Predicted value after %d years:%.2f\n", years, futureValue);**

**sc.close();**

**}**

**}**



Explain the concept of recursion and how it can simplify certain problems.

Recursion is when a method calls itself to solve a smaller version of the original problem. It can simplify certain problems by:

1. The problem can be broken into sub-problems.
2. Since, the same logic repeats in a structured way, so we can solve it in a single statement which is bound to that specific logic and pattern.

Discuss the time complexity of your recursive algorithm.

The time complexity of this recursive algorithm is O(n) where n=years.

Explain how to optimize the recursive solution to avoid excessive computation.

1. **Memoization**
   * Store results of previous recursive calls to avoid repeating calculations.
2. **Use Iteration**
   * Convert recursion to a loop to avoid function call overhead and stack overflow.
3. **Use Formula (Best for forecasting)**
   * Apply the formula: Future Value = Present Value × (1 + rate)^years
   * Fastest and most efficient (O(1) time).