**Design principles and patterns**

**HandsOn 1 - Singleton Implementation:**

public 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();

        logger1.log("This is the first log message.");

        logger logger2=logger.getInstance();

        logger2.log("This is the second log message.");

        if (logger1==logger2) {

            System.out.println("Both logger1 and logger2 are the same instance.");

        } else {

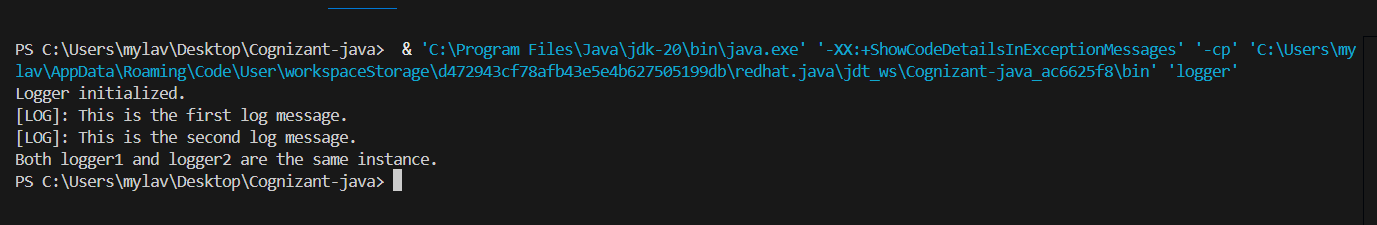
            System.out.println("Different instances were created!");

        }

    }

}

**Output:**



**HandOn 2: Implementing factory pattern**

package factorymethodexample;

interface Document {

    void open();

}

class WordDocument implements Document {

    public void open() {

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

    }

}

class PdfDocument implements Document {

    public void open() {

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

    }

}

class ExcelDocument implements Document {

    public void open() {

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

    }

}

abstract class DocumentFactory {

    public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

public class FactoryMethodPatternExample {

    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();

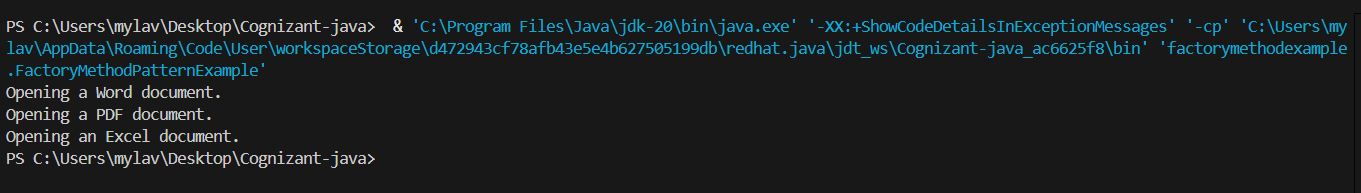
        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

**Output:**



**DSA**

**HandsOn 1: Ecommerce Searching:**

package DSA;

import java.util.Arrays;

import java.util.Scanner;

public class EcommerceSearch {

    // Linear Search

    public static int linearSearch(String[] products, String target) {

        for (int i = 0; i < products.length; i++) {

            if (products[i].equalsIgnoreCase(target)) {

                return i;

            }

        }

        return -1;

    }

    // Binary Search

    public static int binarySearch(String[] products, String target) {

        Arrays.sort(products, String.CASE\_INSENSITIVE\_ORDER);

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

        while (low <= high) {

            int mid = (low + high) / 2;

            int compare = products[mid].compareToIgnoreCase(target);

            if (compare == 0) return mid;

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

            else high = mid - 1;

        }

        return -1;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String[] products = {"Backpack","Desk Chair","Laptop","Notebook","Running Shoes","Shampoo","Smartphone","Table Lamp","Water Bottle","Wireless Mouse"};

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

        String target = scanner.nextLine();

        // Linear Search

        long startLinear = System.nanoTime();

        int indexLinear = linearSearch(products, target);

        long endLinear = System.nanoTime();

        if (indexLinear != -1)

            System.out.println("Linear Search → Found at index " + indexLinear);

        else

            System.out.println("Linear Search → Not found.");

        System.out.println("Time taken: " + (endLinear - startLinear) + " ns\n");

        // Binary Search

        long startBinary = System.nanoTime();

        int indexBinary = binarySearch(products, target);

        long endBinary = System.nanoTime();

        if (indexBinary != -1)

            System.out.println("Binary Search → Found in sorted array at index " + indexBinary);

        else

            System.out.println("Binary Search → Not found.");

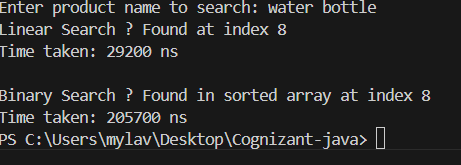
        System.out.println("Time taken: " + (endBinary - startBinary) + " ns");

        scanner.close();

    }

}

**Output:**



**HandsOn 2: Financial Forecasting**

package DSA;

import java.util.Scanner;

public class FinancialForecast {

    // Recursive method to calculate future value

    public static double predictFutureValue(double currentValue, double rate, int years) {

        // Base case: if 0 years left, return current value

        if (years == 0) {

            return currentValue;

        }

        // Recursive case: grow one year at a time

        return predictFutureValue(currentValue \* (1 + rate), rate, years - 1);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Input

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

        double initialValue = scanner.nextDouble();

        System.out.print("Enter annual growth rate (in %): ");

        double growthRate = scanner.nextDouble() / 100.0;

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

        int years = scanner.nextInt();

        // Forecasting

        double futureValue = predictFutureValue(initialValue, growthRate, years);

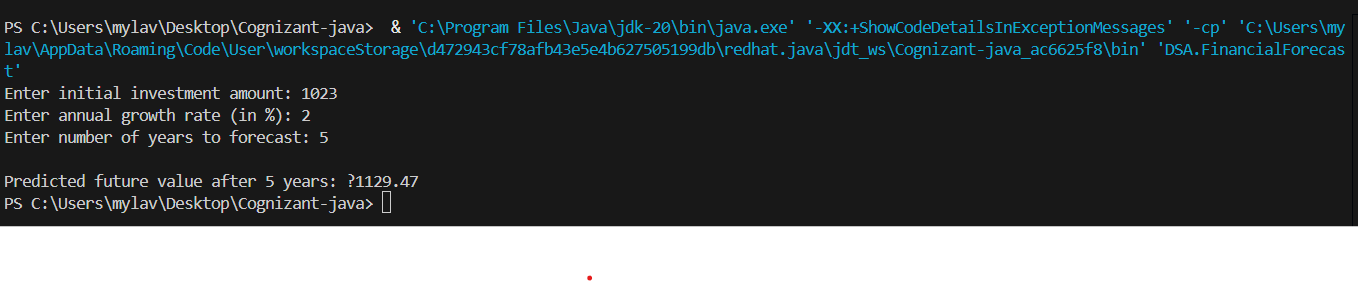
        System.out.printf("\nPredicted future value after %d years: ₹%.2f\n", years, futureValue);

        scanner.close();

    }

}

**Output:**

****