// 1. Understand Recursive Algorithms:

// Recursion is a technique where a function calls itself with a modified input to solve a problem.

// It simplifies problems by breaking them into smaller, similar subproblems, such as calculating future values

// based on past growth rates iteratively. Base cases prevent infinite recursion, making it suitable for

// problems like financial forecasting where each step depends on the previous one.

import java.util.Scanner;

public class FinancialForecast {

    public static double calculateFutureValue(double initialValue, double[] growthRates, int year) {

        if (year <= 0) return initialValue;

        return calculateFutureValue(initialValue \* (1 + growthRates[year - 1]), growthRates, year - 1);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        boolean running = true;

        while (running) {

            System.out.print("Enter initial value (e.g., 1000.0): ");

            double initialValue;

            try {

                initialValue = Double.parseDouble(scanner.nextLine().trim());

                if (initialValue < 0) {

                    System.out.println("Initial value cannot be negative.");

                    continue;

                }

            } catch (NumberFormatException e) {

                System.out.println("Invalid input. Please enter a valid number.");

                continue;

            }

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

            int years;

            try {

                years = Integer.parseInt(scanner.nextLine().trim());

                if (years <= 0) {

                    System.out.println("Number of years must be positive.");

                    continue;

                }

            } catch (NumberFormatException e) {

                System.out.println("Invalid input. Please enter a valid integer.");

                continue;

            }

            double[] growthRates = new double[years];

            for (int i = 0; i < years; i++) {

                System.out.print("Enter growth rate for year " + (i + 1) + " (e.g., 0.05 for 5%): ");

                try {

                    growthRates[i] = Double.parseDouble(scanner.nextLine().trim());

                } catch (NumberFormatException e) {

                    System.out.println("Invalid growth rate. Please enter a valid number.");

                    i--;

                }

            }

            double futureValue = calculateFutureValue(initialValue, growthRates, years);

            System.out.println("Future Value after " + years + " years: Rs." + String.format("%.2f", futureValue));

            System.out.print("Do you want to perform another forecast? (yes/no): ");

            String response = scanner.nextLine().trim().toLowerCase();

            if (!response.equals("yes")) {

                running = false;

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

            }

        }

        scanner.close();

    }

}

// Analysis (unchanged):

// Time Complexity:

// - The recursive algorithm has O(n) time complexity, where n is the number of years,

//   as it makes n recursive calls, each performing a constant-time multiplication and addition.

// Optimization:

// - To avoid excessive computation, the recursive solution can be optimized using dynamic programming or memoization

//   to store intermediate results, reducing redundant calculations if the same subproblems are solved multiple times.

// - Alternatively, an iterative approach using a loop can eliminate recursion overhead, achieving the same O(n)

//   complexity with better space efficiency (O(1) vs. O(n) stack space for recursion).

OUTPUT:

A screen shot of a computer

AI-generated content may be incorrect.