**Exercise 7: Financial Forecasting**

**Recursion**

Recursion is when a method calls itself to solve a problem.  
It simplifies complex problems like forecasting by breaking them down into base cases + recursive cases.

Solution 🡪

public class Forecaster {

// Recursive Forecast Method

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

if (years == 0) return presentValue;

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

}

// Optimized with Memoization

public static double forecastMemo(double presentValue, double rate, int years, double[] memo) {

if (years == 0) return presentValue;

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

return memo[years] = forecastMemo(presentValue, rate, years - 1, memo) \* (1 + rate);

}

}

public class Main {

public static void main(String[] args) {

double presentValue = 10000;

double rate = 0.08; // 8% growth

int years = 5;

System.out.println(" Recursive Forecast: " +

Forecaster.forecast(presentValue, rate, years));

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

System.out.println("Optimized Forecast with Memoization: " +

Forecaster.forecastMemo(presentValue, rate, years, memo));

}

}

**Time Complexity Analysis**

| **Version** | **Time Complexity** |
| --- | --- |
| Plain Recursion | O(n) |
| With Memoization | O(n) |
| Iterative (alt) | O(n) |

Recursive version recalculates each subproblem → can slow down.  
Memoization stores results → avoids recomputation.

**Optimization Idea**

To go further, use iterative DP or analytical approach (power formula). But recursion keeps your logic clean and extensible, especially for multi-scenario forecasting.

Output 🡪

