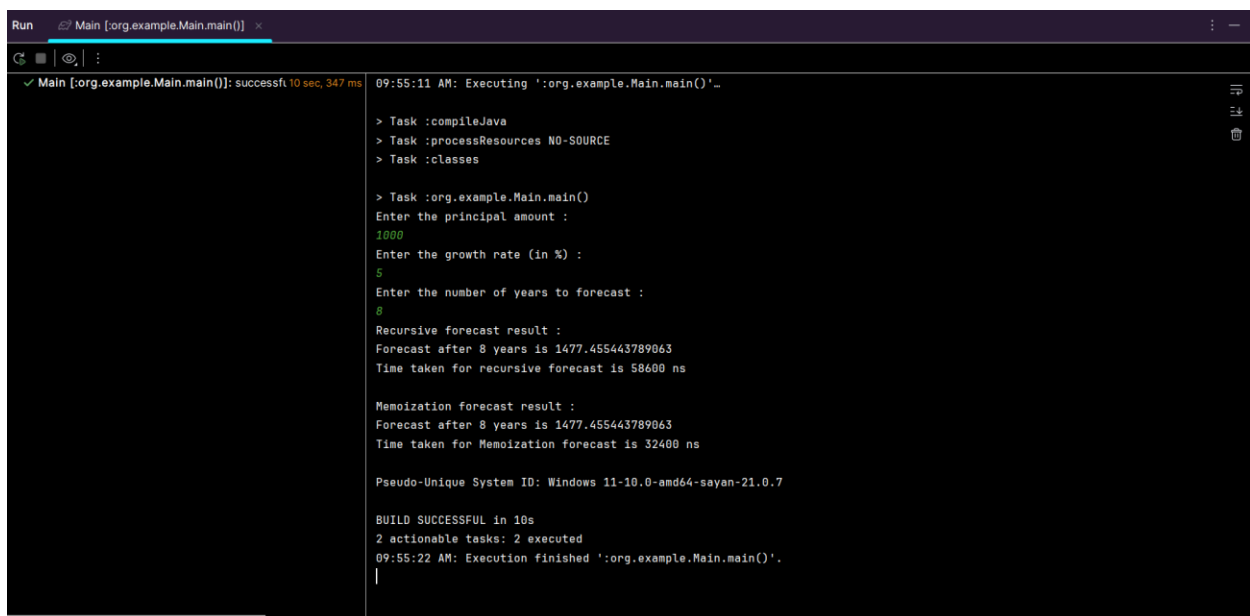


File by Sayan Saha ([sayansaha00876@gmail.com](mailto:sayansaha00876@gmail.com))

## Exercise 7: Financial Forecasting:

### Output Screenshots :-



```
Run Main [org.example.Main.main()] x
Main [org.example.Main.main()]: successft 10 sec, 347 ms
09:55:11 AM: Executing 'org.example.Main.main()'
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :org.example.Main.main()
Enter the principal amount :
1000
Enter the growth rate (in %) :
5
Enter the number of years to forecast :
8
Recursive forecast result :
Forecast after 8 years is 1477.455443789063
Time taken for recursive forecast is 58600 ns
Memoization forecast result :
Forecast after 8 years is 1477.455443789063
Time taken for Memoization forecast is 32400 ns
Pseudo-Unique System ID: Windows 11-10.0-amd64-sayan-21.0.7
BUILD SUCCESSFUL in 10s
2 actionable tasks: 2 executed
09:55:22 AM: Execution finished 'org.example.Main.main()'.
```

## 1. Understand Recursive Algorithms

### What is recursion ?

Recursion is a technique used in programming where a function calls itself to solve it efficiently by breaking it down into smaller subproblems.

### Why is recursion useful in forecasting ?

In forecasting, the future values depend on the past values. Recursion thus help in forecasting future data on past growth patterns.

## 2. Setup

We define a method to forecast future values based on past data. The essential inputs are:

- Initial amount
- Growth rate (as a percentage)
- Number of years to forecast

## 3. Implementation of Recursive Algorithm

The entire code implementation files are attached in the folder.

I have used this recursive formula :

$$\text{futureValue}(\text{year}) = \text{futureValue}(\text{year} - 1) * (1 + \text{growthRate})$$

## 4. Analysis

### Time Complexity (Unoptimized Recursive Version):

- **O(n)**: One recursive call per year, but no repeated subproblems.
- Still inefficient in practice due to repeated function calls and stack overhead.

### Time Complexity (Memoized Version):

- **O(n)**: Same number of logical operations, but faster due to cached results.
- Reduces redundant computation significantly.

### How to Optimize:

- We used **memoization**: Stored already-computed year values in a map or array.
- We used **iteration** (loop) instead of recursion for large datasets to avoid stack overflow.
- We can use **dynamic programming** when dealing with multiple interdependent variables.

