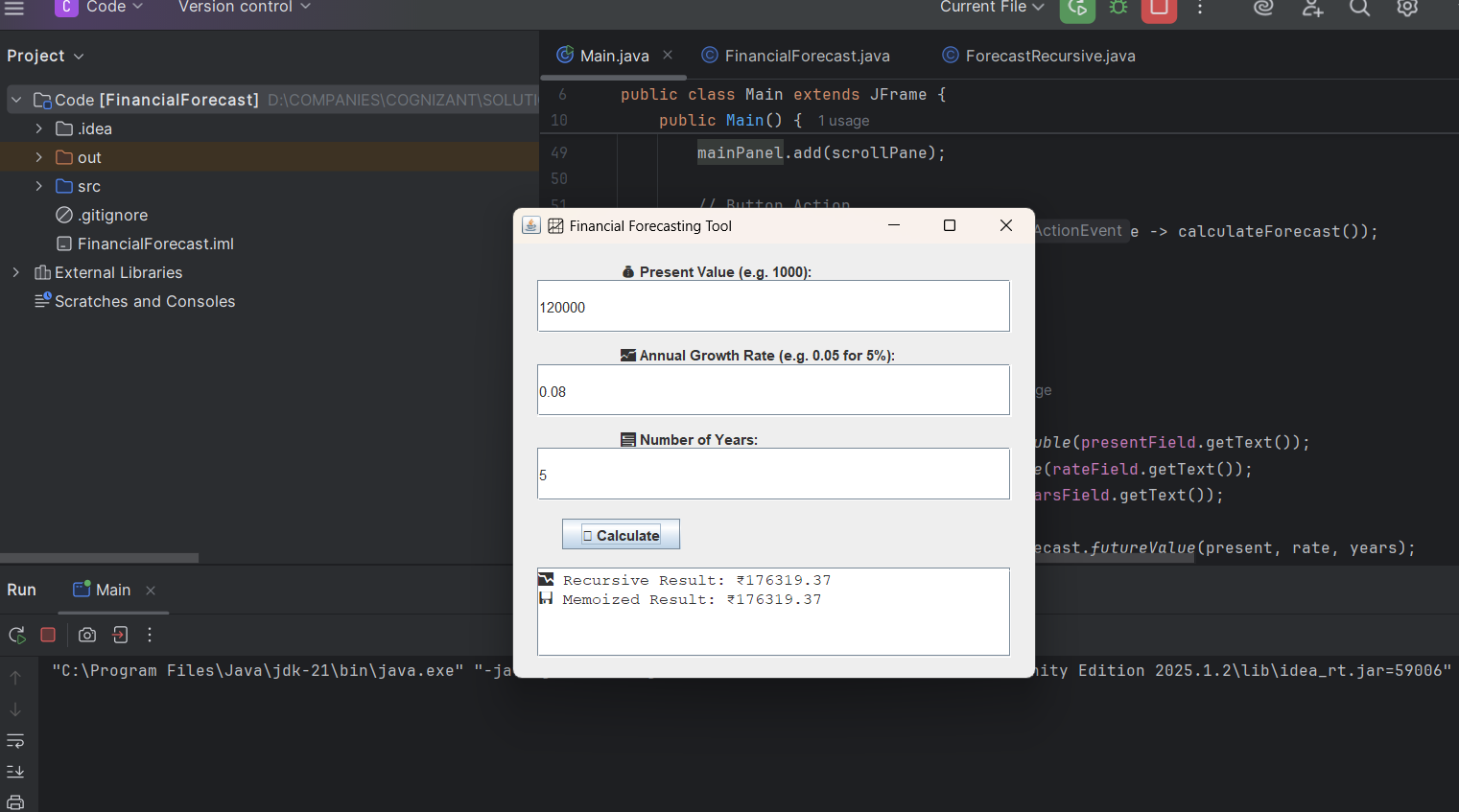
* **Exercise 7: Financial Forecasting**

Output :



**1. Understanding Recursive Algorithms**

Recursion is a programming concept where a function calls itself to solve smaller parts of a larger problem. It is useful when a problem can be broken down into simpler sub-problems of the same type. Recursion helps reduce code complexity and improves readability for certain problems like mathematical computations, tree structures, and forecasting models.

**2. Setup: Calculating Future Value Using Recursion**

In financial forecasting, recursion can be used to predict the future value of an investment based on a fixed growth rate. The idea is to start with a present value and apply the growth rate year by year. Each year’s value is calculated based on the previous year’s value. This process continues until we reach the target number of years.

**3. Implementation Concept**

The forecast starts with an initial amount called the present value. Each year, the value grows by a certain percentage known as the growth rate. Recursively, we apply the same calculation for each year until we reach the desired future year. This repetitive approach fits naturally with recursion, as each step depends on the result of the previous one.

**4. Analysis**

The recursive method is simple and easy to understand but can become inefficient for very large time periods due to repeated calculations. The time taken to compute increases with the number of years. To improve performance, the solution can be optimized using techniques like **memoization** (storing already calculated values) or using a **loop** instead of recursion for better efficiency.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Recursion** | **Memoization** |
| **Definition** | A function that calls itself to solve smaller subproblems | An optimization technique that stores results of previous function calls |
| **Repetition of Work** | May repeat calculations many times | Avoids repetition by caching results |
| **Speed** | Slower for large inputs | Faster due to reuse of stored results |
| **Memory Usage** | Uses call stack memory | Uses extra memory to store results |
| **Base Case Required** | Yes | Yes |
| **Time Complexity** | Often exponential (e.g., O(2ⁿ)) | Reduced to linear or polynomial time (e.g., O(n)) |
| **Use Case** | Simple problems, conceptual clarity | Performance-critical problems |
| **Example Use** | Fibonacci, factorial, tree traversals | Optimized Fibonacci, dynamic programming |