**What is Recursion?**

**Recursion** is a technique where a method **calls itself** to solve smaller versions of the same problem. It continues until a **base case** (stopping condition) is met.

**How Recursion Simplifies Problems**

* Recursion is useful for problems that can be **broken down into smaller, similar subproblems**.
* It often leads to **shorter and clearer code**, especially for problems involving repetition or mathematical formulas (e.g., factorial, Fibonacci, compound growth).

**Setup:**

Formula:

FutureValue(years) = currentValue \* (1 + growthRate) ^ years

**Implementaion:**

**Code:**

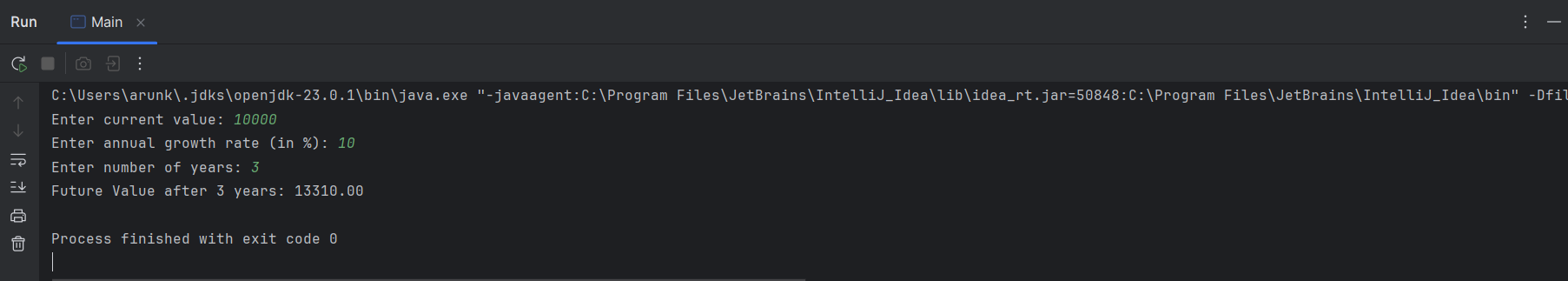
**FinancialForecast.java**

public class FinancialForecast {  
 public static double futureValue(double currentValue, double growthRate, int years) {  
 if (years == 0) {  
 return currentValue;  
 }  
 return *futureValue*(currentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
}

**Main.java**

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter current value: ");  
 double currentValue = scanner.nextDouble();  
  
 System.*out*.print("Enter annual growth rate (in %): ");  
 double ratePercent = scanner.nextDouble();  
 double growthRate = ratePercent / 100;  
  
 System.*out*.print("Enter number of years: ");  
 int years = scanner.nextInt();  
  
 double futureValue = FinancialForecast.*futureValue*(currentValue, growthRate, years);  
  
 System.*out*.printf("Future Value after %d years: %.2f%n", years, futureValue);  
  
 scanner.close();  
 }  
}

**Output:**

****

**Time Complexity:**

Time Complexity of the Recursive Algorithm:

The recursive method to calculate future value makes one recursive call for each year until the base case is reached (year = 0). Therefore:

Time Complexity: O(n)

(n is the number of years; the function is called n times)

Space Complexity: O(n)

(Each recursive call adds a new frame to the call stack, so for n years, there are n stack frames)

This approach works well for small inputs but may lead to stack overflow if n is very large.

**Optimization:**

**Use Iteration Instead of Recursion:**

The most effective and straightforward optimization is to replace the recursive function with a loop. This avoids the overhead of recursive calls and uses constant memory.

**Memoization:**

Memoization means storing the results of previous computations so they don’t have to be recalculated.

Note: For this specific problem, memoization isn't needed because each step only depends on the previous step (no repeated subproblems). But for problems like Fibonacci numbers, it is useful.