## **WEEK 1 Mandatory Hands On**

## Algorithms\_Data Structures

## Question:

## Exercise 7: Financial Forecasting

## Scenario:

## You are developing a financial forecasting tool that predicts future values based on past data.

## Steps:

## 1. Understand Recursive Algorithms:

## o Explain the concept of recursion and how it can simplify certain problems.

## 2. Setup:

## o Create a method to calculate the future value using a recursive approach.

## 3. Implementation:

## o Implement a recursive algorithm to predict future values based on past growth rates.

## 4. Analysis:

## o Discuss the time complexity of your recursive algorithm.

## o Explain how to optimize the recursive solution to avoid excessive computation.

## Solution:

**Recursion** is a programming technique where a **function calls itself** directly or indirectly to solve a **smaller subproblem** of the original problem.

The idea is to **break a big problem into smaller, similar subproblems**, solve them recursively, and then combine the results.

Two Key Components of Recursive Functions:

1. Base Case
   * The condition under which the recursion stops.
   * Prevents infinite recursion.
2. Recursive Case
   * The part where the function **calls itself** with a smaller input.

Recursion is especially useful for problems that exhibit:

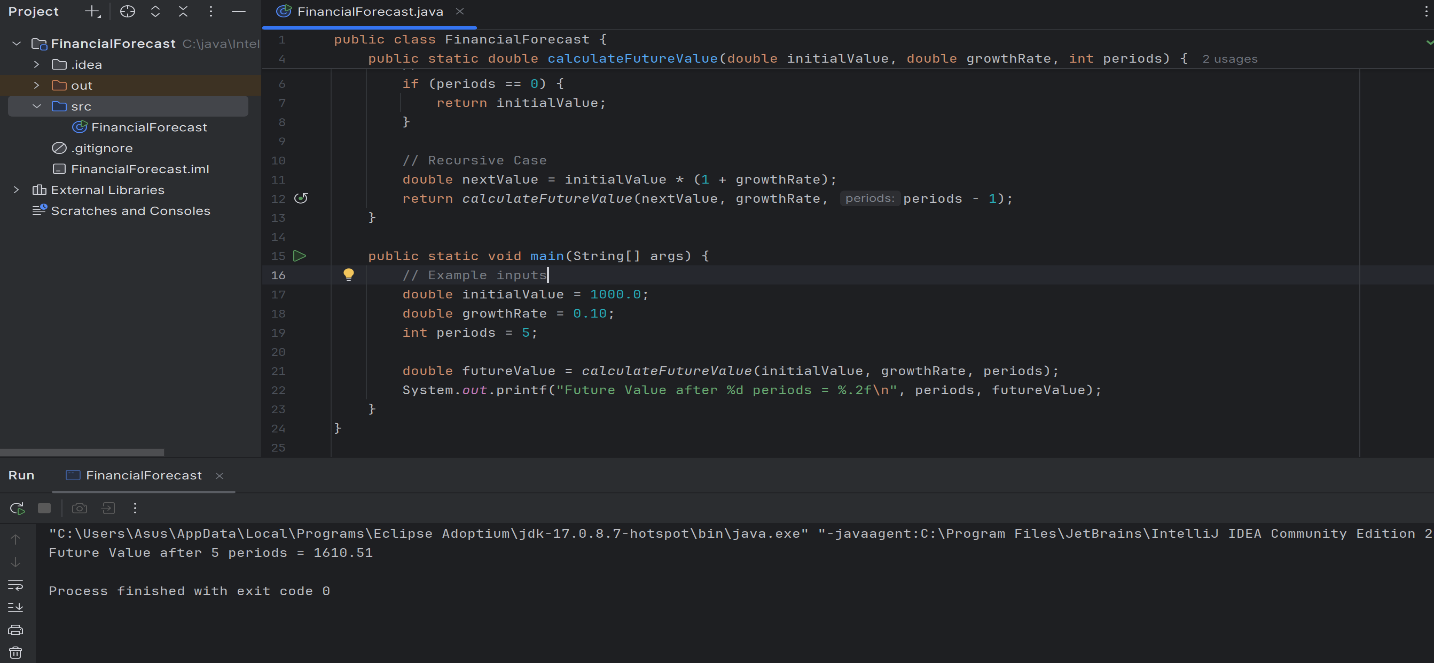
* **Self-similarity** (problem can be broken down into smaller versions of itself)
* **Divide-and-conquer** strategies
* Naturally hierarchical or tree-like structures

It allows us to **write elegant, short, and clear code** for problems that would be complicated using loops or iteration.

CODE: FinancialForecast.java

**public class FinancialForecast {  
  
 // Recursive method to calculate future value  
 public static double calculateFutureValue(double initialValue, double growthRate, int periods) {  
 // Base Case  
 if (periods == 0) {  
 return initialValue;  
 }  
  
 // Recursive Case  
 double nextValue = initialValue \* (1 + growthRate);  
 return *calculateFutureValue*(nextValue, growthRate, periods - 1);  
 }  
  
 public static void main(String[] args) {  
 // Example inputs  
 double initialValue = 1000.0;  
 double growthRate = 0.10;  
 int periods = 5;  
  
 double futureValue = *calculateFutureValue*(initialValue, growthRate, periods);  
 System.*out*.printf("Future Value after %d periods = %.2f\n", periods, futureValue);  
 }  
}**

OUTPUT:



**Time Complexity Analysis of the above algorithm**

Observations

* At each step, we make one recursive call.
* The number of recursive calls is equal to the number of periods n.
* Each call performs one multiplication and one function call.

## **Time Complexity:** T(n)=T(n−1)+O(1)

## This recurrence simplifies to: T(n)=O(n)

**Final Time Complexity: O(n)** — **Linear Time**

Each of the n periods contributes **one recursive step**, and the amount of work in each step is constant.

**To optimize a recursive solution and avoid excessive computation we can:**

1. **Use Memoization**: Store results of subproblems so they don't get recomputed.
2. **Convert to Iteration**: Use loops instead of recursion to save stack space.
3. **Use Tail Recursion**: If supported by the language, it reduces stack usage (not optimized in Java).
4. **Use Math/Direct Formulas**: When possible, replace recursion with a mathematical formula for efficiency.