**EXERCISE 7:**

**FINANCIAL FORECASTING**

**Recursion:**

Recursion is a process where a method calls itself directly or indirectly to solve a problem.

It can simplify problems by breaking them down into smaller, more manageable sub-problems.

A recursive function typically has a base case (stopping condition) and a recursive case (where the function calls itself).

**Optimizing the Recursive Solution:**

**Memoization:**

Memoization is a technique to optimize recursive algorithms by storing the results of expensive function calls and reusing them when the same inputs occur again.

In this implementation, a memo array is used to store the computed future values for each year. If a value has already been computed, it is retrieved from the array instead of recomputing it.

public class FinancialForecasting {  
  
 public static double calculateFutureValue(double currentValue, double growthRate, int years) {  
 if (years == 0) {  
 return currentValue;  
 }  
 return *calculateFutureValue*(currentValue \* (1 + growthRate), growthRate, years - 1);  
 }  
  
 public static double calculateFutureValueMemo(double currentValue, double growthRate, int years, double[] memo) {  
 if (years == 0) {  
 return currentValue;  
 }  
 if (memo[years] != -1) {  
 return memo[years];  
 }  
  
 memo[years] = *calculateFutureValueMemo*(currentValue \* (1 + growthRate), growthRate, years - 1, memo);  
 return memo[years];  
 }  
  
 public static void main(String[] args) {  
 double currentValue = 1000.0;  
 double growthRate = 0.05;  
 int years = 10;  
 double futureValue = *calculateFutureValue*(currentValue, growthRate, years);  
 System.*out*.println("Future Value (Recursion): " + futureValue);  
 double[] memo = new double[years + 1];  
 for (int i = 0; i <= years; i++) {  
 memo[i] = -1;  
 }  
 double futureValueMemo = *calculateFutureValueMemo*(currentValue, growthRate, years, memo);  
 System.*out*.println("Future Value (Memoization): " + futureValueMemo);  
 }  
}