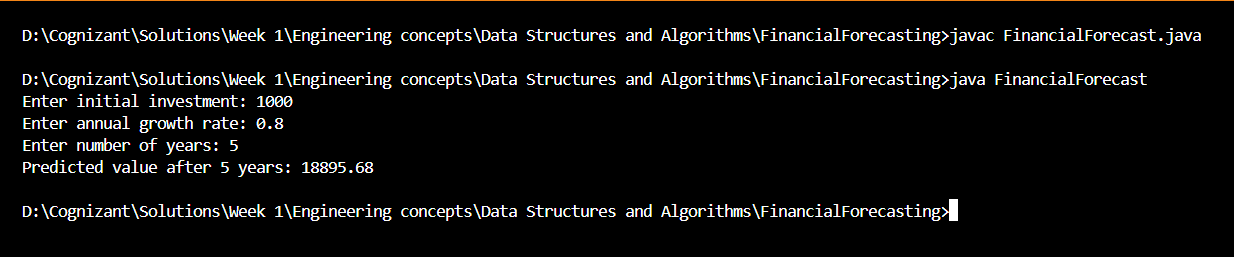
**Answers:**

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

Recursion is a programming technique where **a method calls itself directly or indirectly to solve a problem**. This technique is particularly useful for **problems that can be broken down into smaller, similar subproblems** like calculating the factorial of a number.

Recursion simplifies certain problems by breaking them down into smaller, self-similar subproblems, providing **concise solutions to complex problems that might be difficult to solve iteratively.**

* **Output of Code:**



* Discuss the time complexity of your recursive algorithm.

In my recursive financial forecasting algorithm, the future value is calculated based on the current value and a constant annual growth rate. The function calls itself recursively, reducing the number of years (n) by one in each call until it reaches zero.

The time complexity of this recursive approach is **O(n)**, where n is the number of years into the future for which the forecast is being calculated. This is because the algorithm performs one recursive call for each year, leading to a total of n calls before reaching the base case.

* Explain how to optimize the recursive solution to avoid excessive computation.

We can optimize a recursive solution by:

* **Using Tail Recursion:** Structure the recursive method so that the recursive call is the last operation. While Java does not optimize tail recursion by default, it helps keep the logic clean and easier to convert if needed.
* **Avoiding Recomputations:** Ensure that calculations inside the recursion are not repeated unnecessarily. Pass computed values as parameters to the next recursive call instead of recalculating them.
* **Limiting Recursion Depth:** Set clear and efficient base cases to stop recursion as early as possible, avoiding deep call stacks.