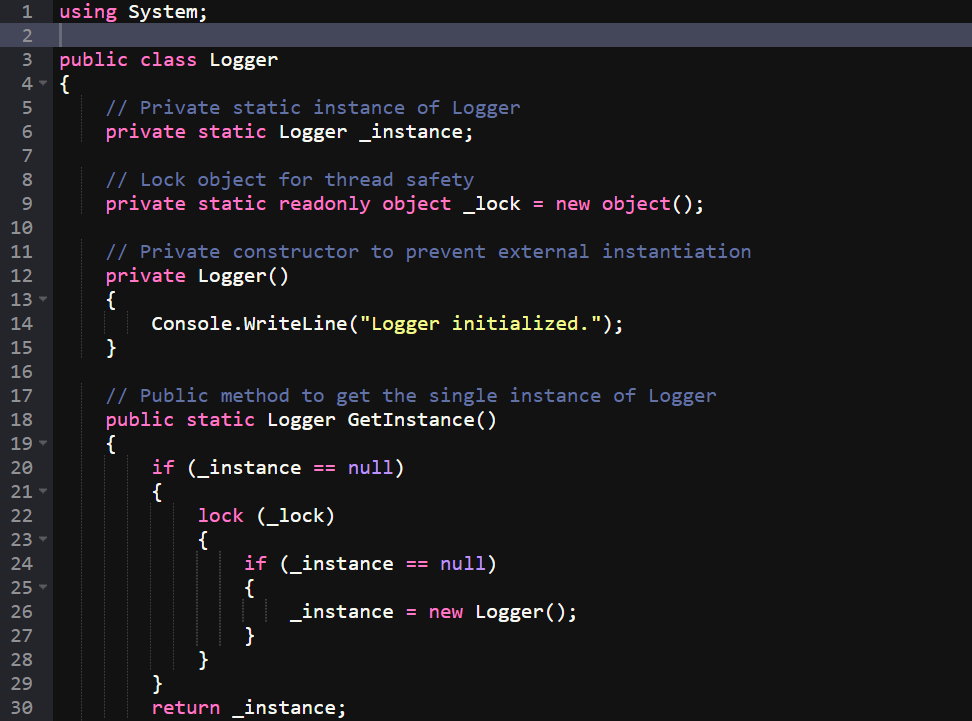
**Name:** Manu Rajesh Naik

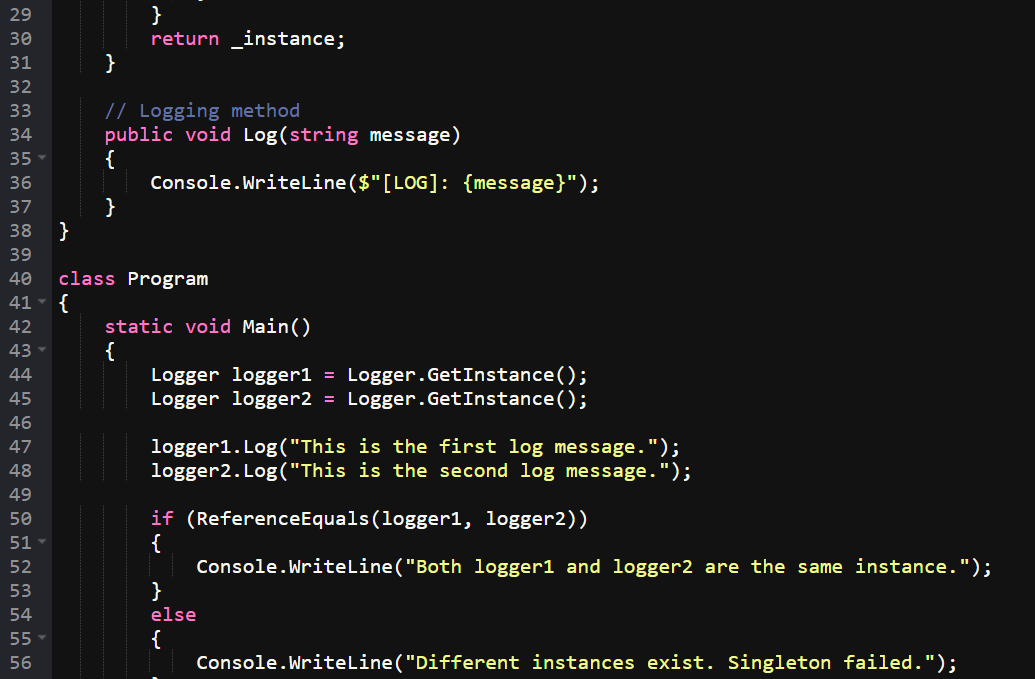
**Superset ID:** 6365342

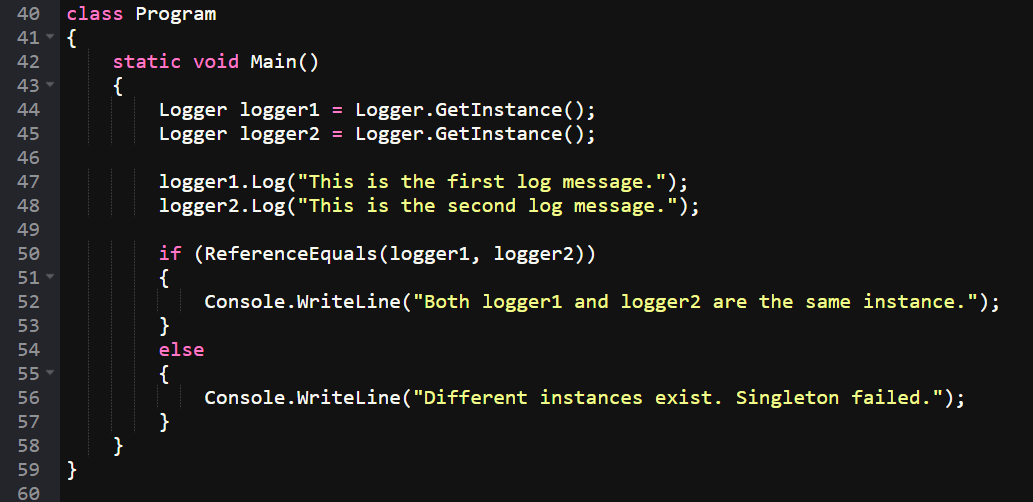
***Design Principal and Patterns***

**Exercise 1: Implementing the Singleton Pattern**

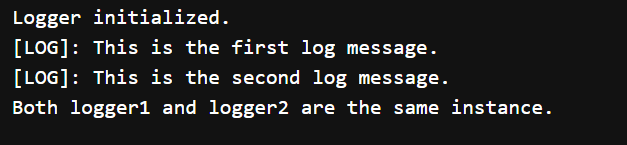
Input:





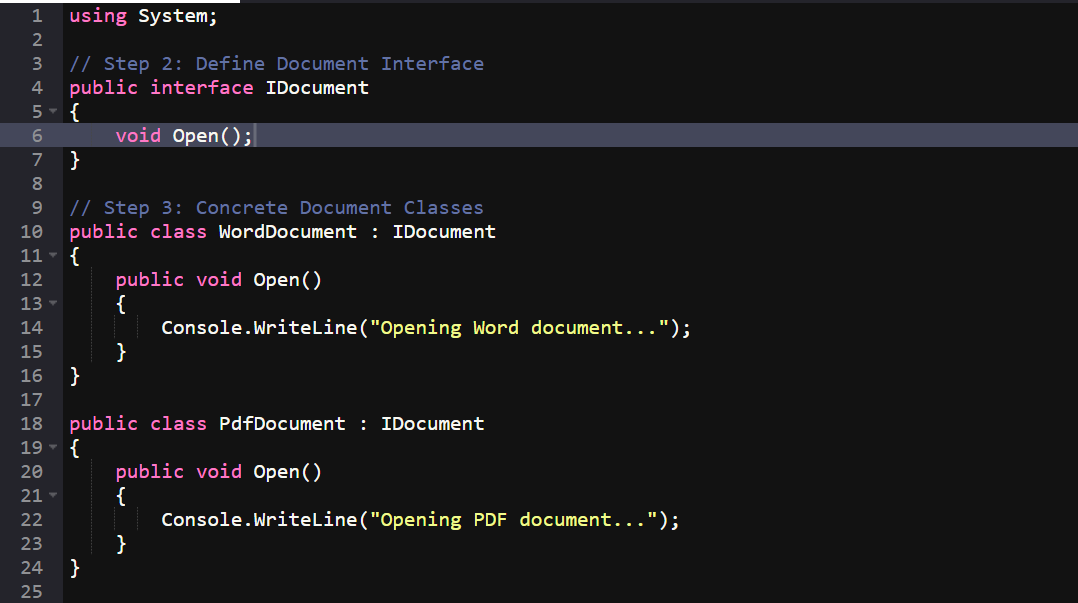


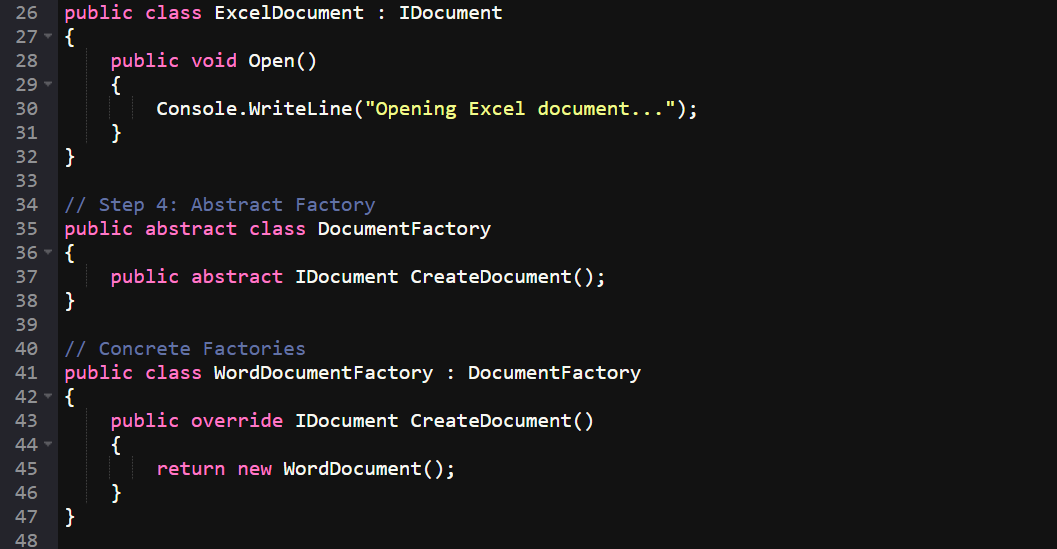
Output:

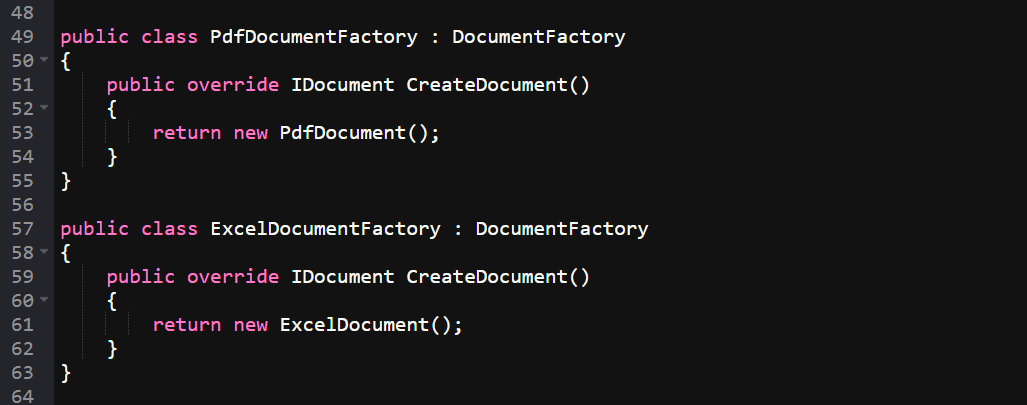


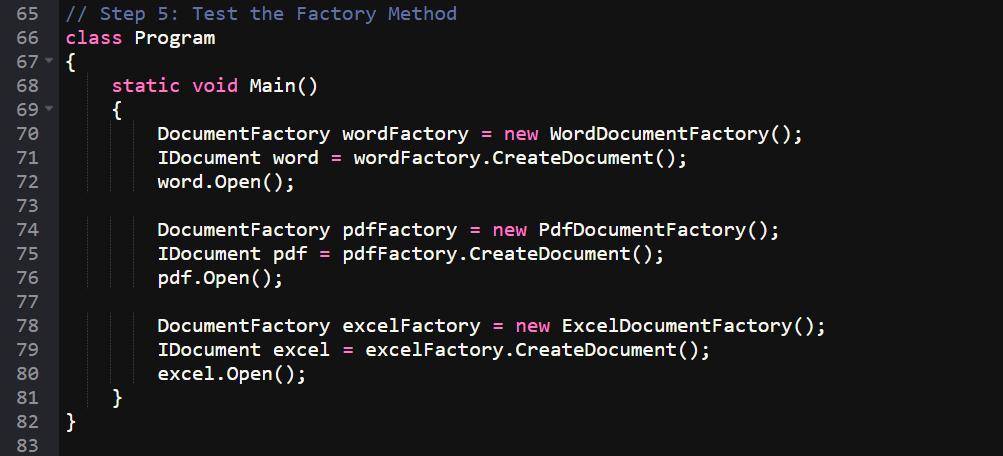
**Exercise 2: Implementing the Factory Method Pattern**

Input:

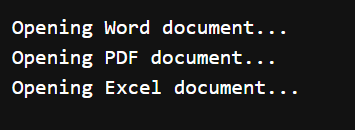








Output:



***Algorithms & Data Structures***

**Exercise 2: E-commerce Platform Search Function**

**Big O Notation**

Big O notation is a way to describe the time or space complexity of an algorithm as a function of input size n. It tells you how the performance (execution time or memory usage) of an algorithm scales as the input size increases.

For example:

O(1) → Constant time (independent of input size)

O(n) → Linear time (time grows proportionally with input size)

O(log n) → Logarithmic time (efficient, cuts input in half each step)

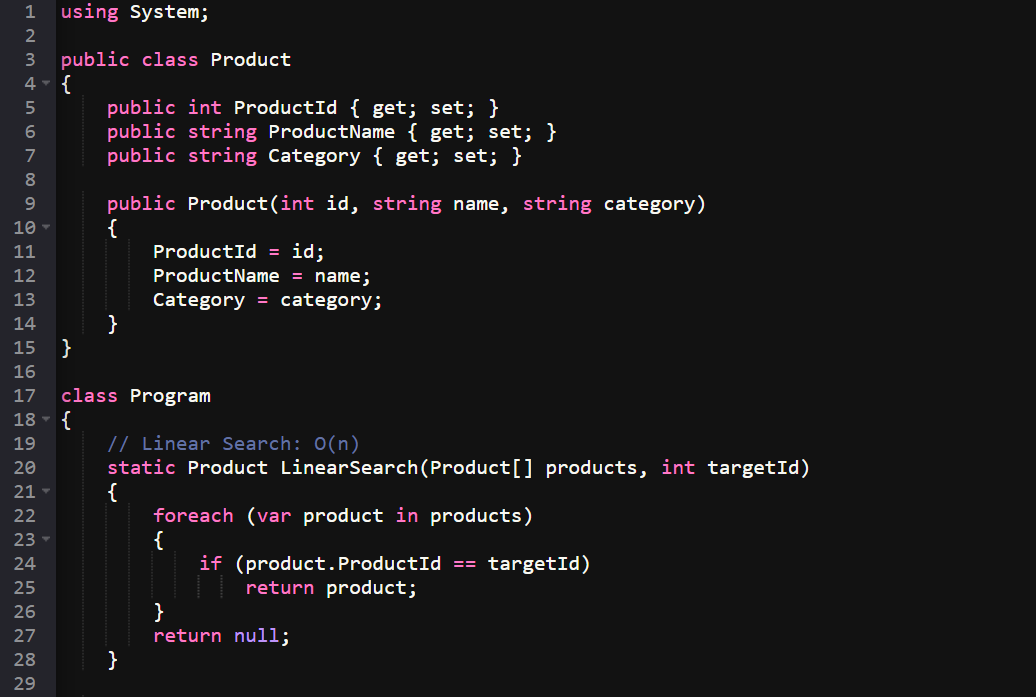
O(n^2) → Quadratic time (bad for large inputs)

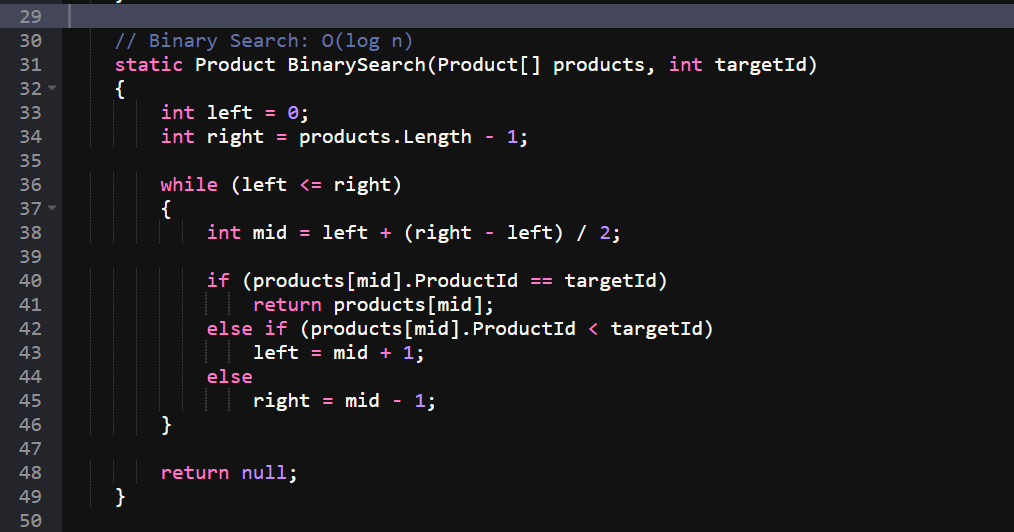
We use Big O to compare algorithms, predict performance issues, and make decisions about which algorithm is more efficient in practice.

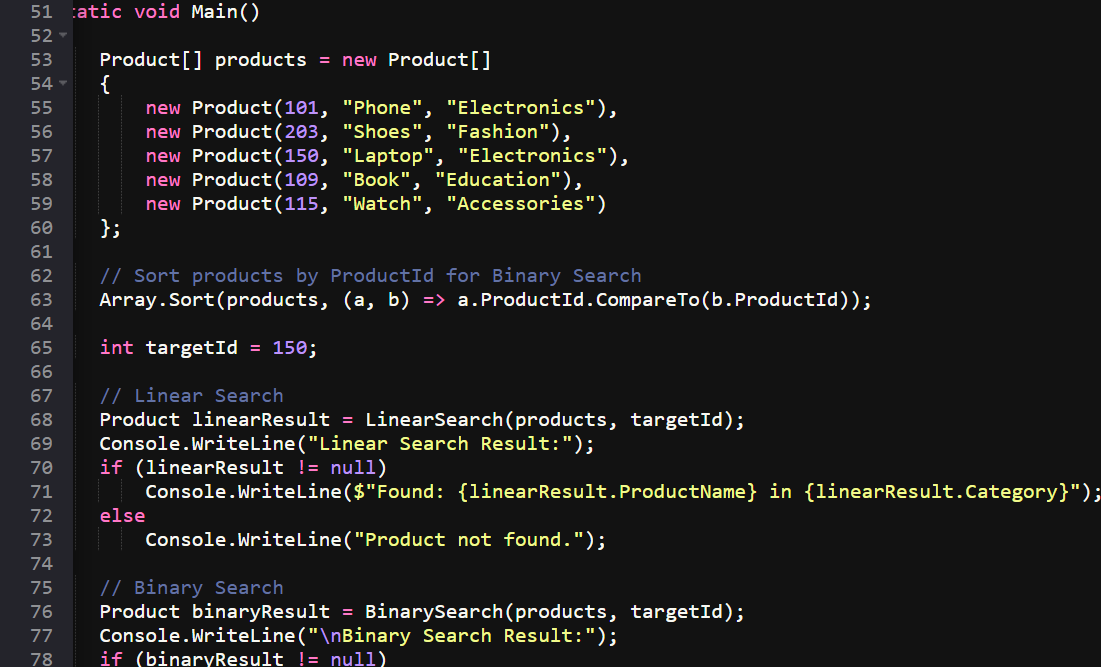
1. Best Case The ideal scenario — the element is found immediately. Example (Linear Search): O(1) if the element is the first item.
2. Average Case The expected performance across many random inputs. Example (Linear Search): O(n/2) → simplified to O(n).

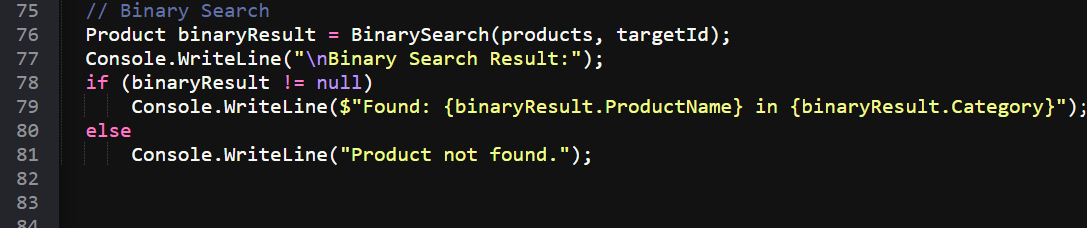
3. Worst Case The most time-consuming scenario — the element is last or not present. Example (Linear Search): O(n) — must check every element.

Input:

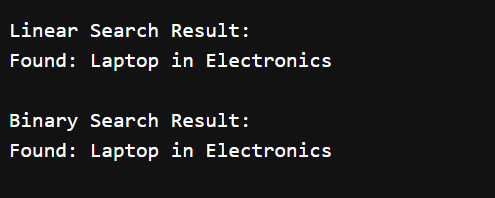








Output:



**Exercise 7: Financial Forecasting**

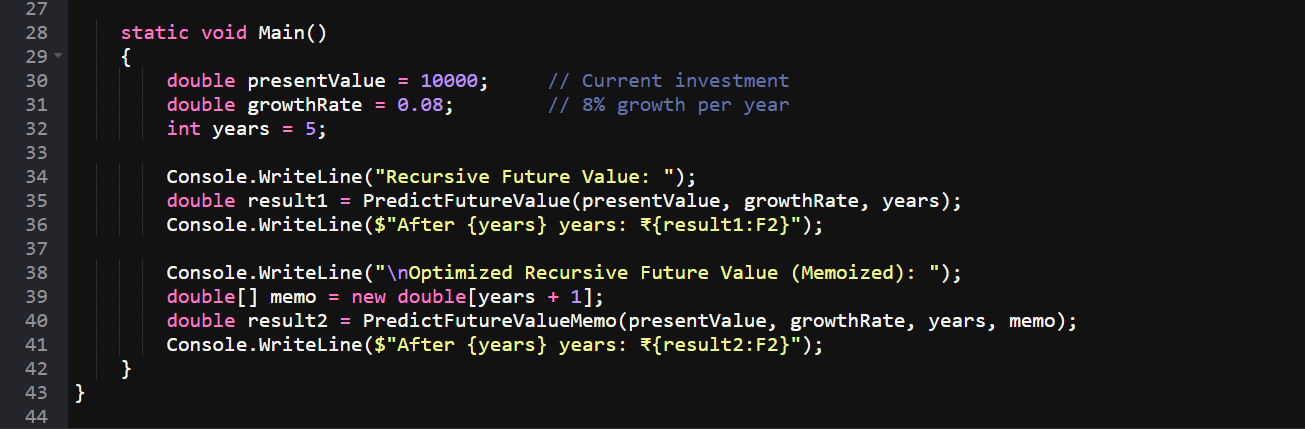
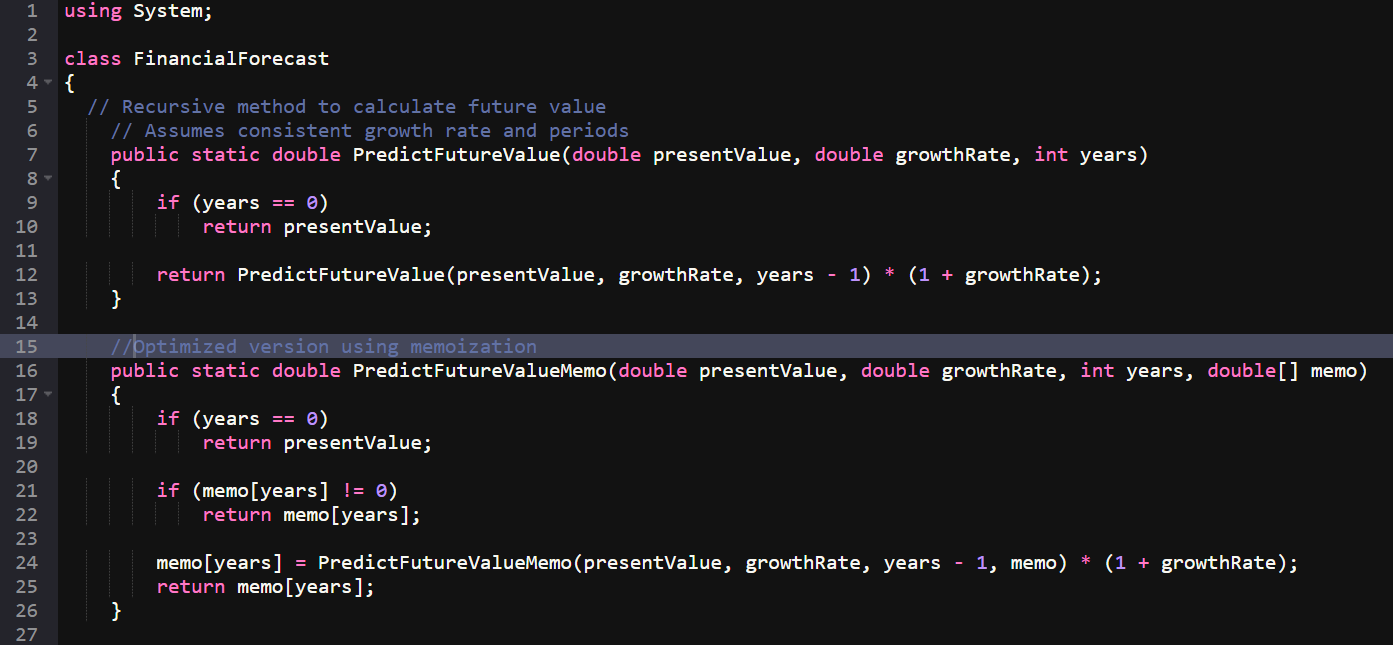
Recursion is a programming technique where a function calls itself to solve smaller instances of a problem. Instead of solving the problem all at once, recursion breaks it down into simpler sub-problems of the same type, and uses the solutions of those to build the solution to the original problem.

Recursion makes some problems easier to solve because it lets you:

1. Break Big Problems into Smaller Ones: Instead of solving a big problem all at once, recursion lets you solve one small piece at a time and build the answer step by step.
2. Follow Natural Structure: Some things, like trees or folders inside folders, naturally contain smaller versions of themselves. Recursion fits perfectly because it repeats the same process for each part.
3. Avoid Repeating Yourself: Instead of writing a lot of repeated code or using many loops, recursion lets the function "do the same thing again" with new data, which can keep the code shorter and cleaner.

1. Solve Problems Step by Step: It allows you to think about one step at a time. For example, if you know how to solve a smaller puzzle, recursion helps you solve the whole puzzle using that.

Input:



Output:

