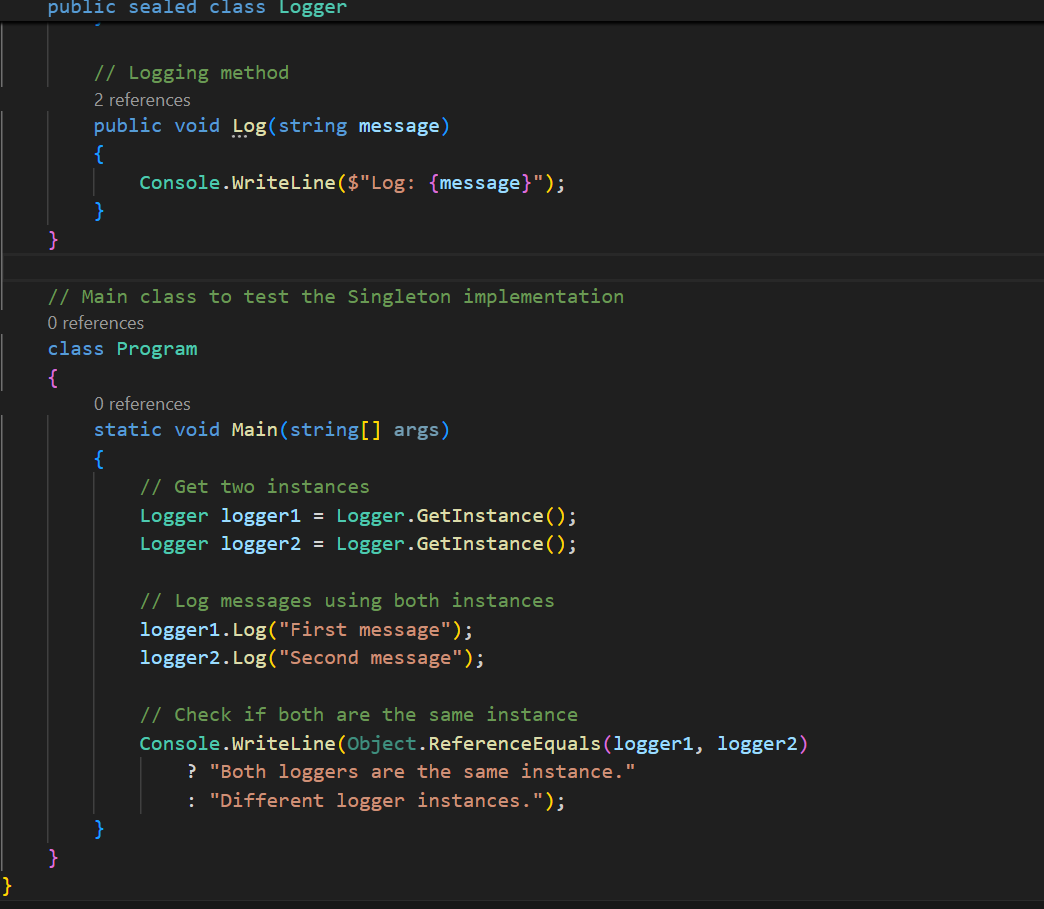
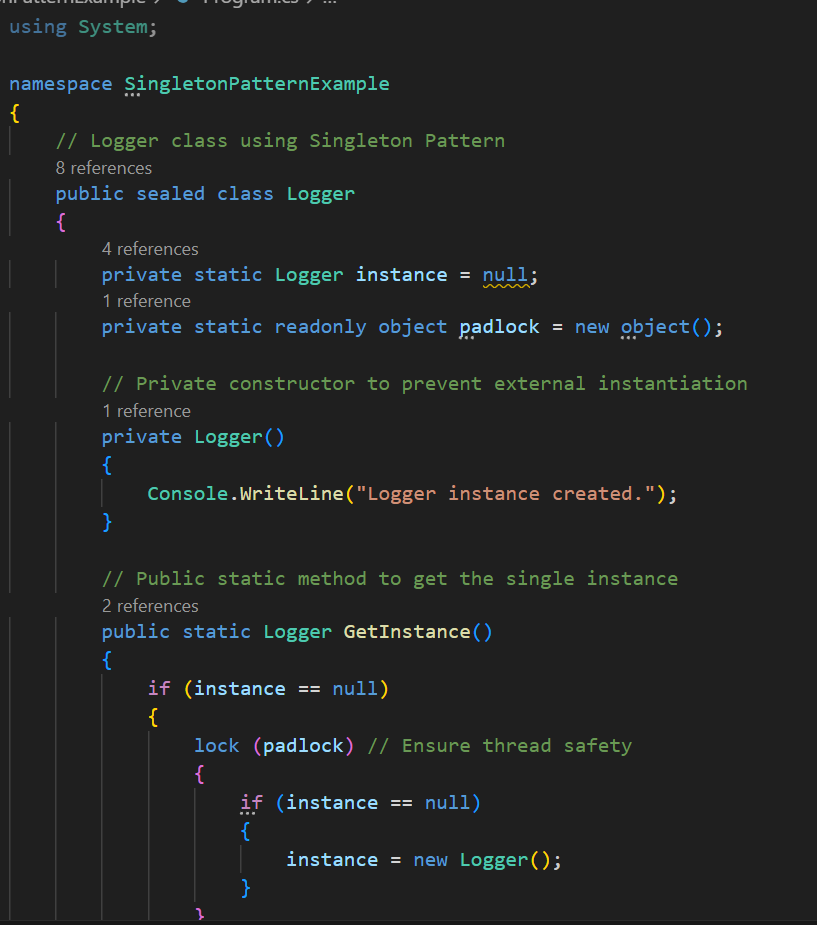
**WEEK-1 HANDS-ON QUESTION**

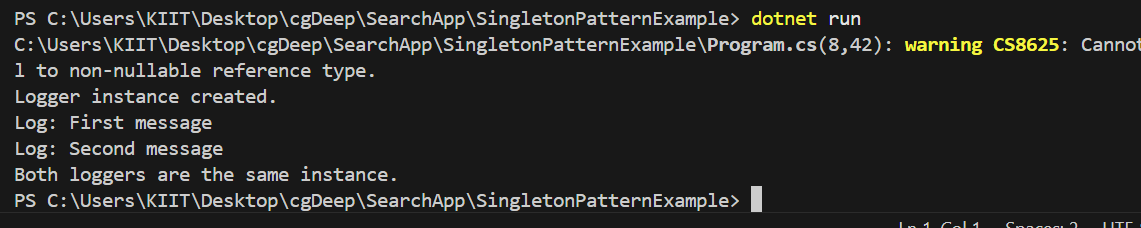
***DESIGN PRINCIPLES AND PATTERNS***

**Exercise 1**: Implementing the Singleton Pattern

**Ans**

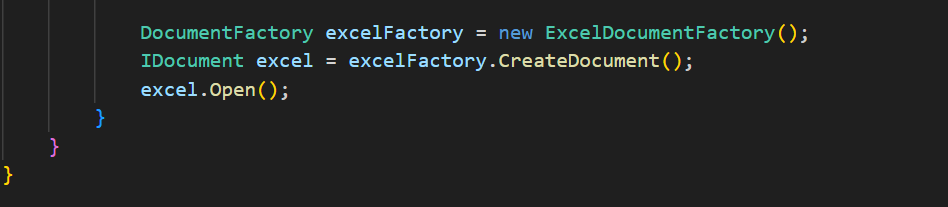
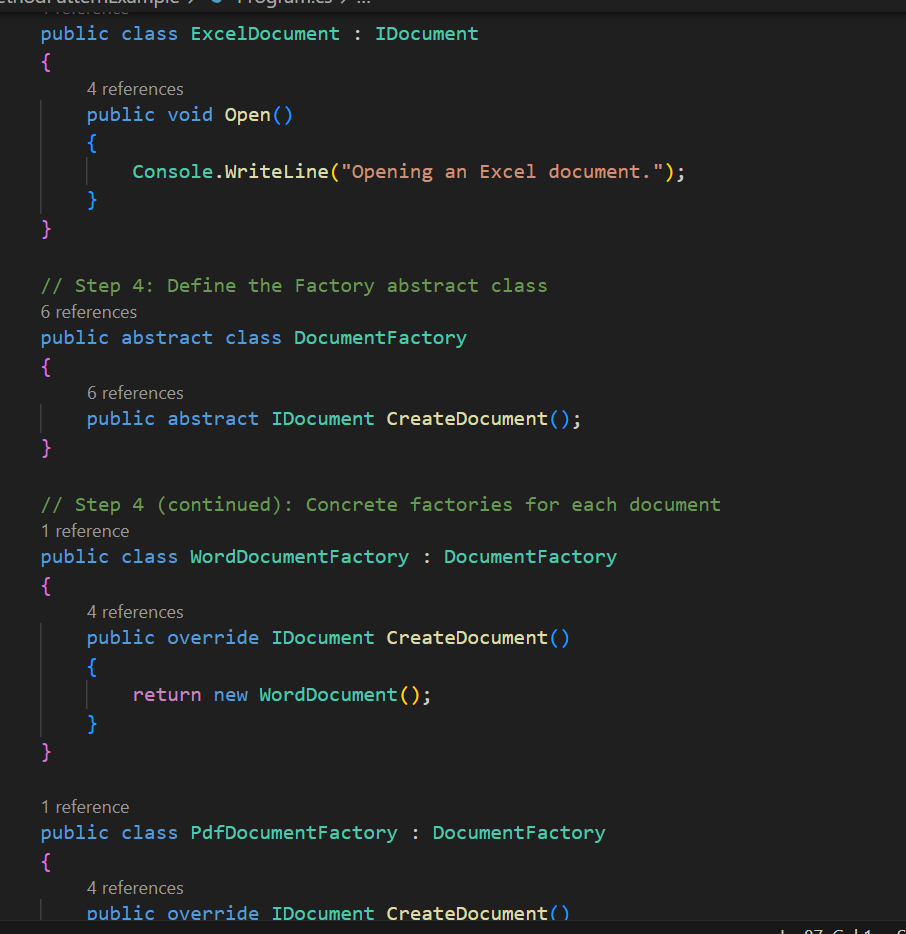
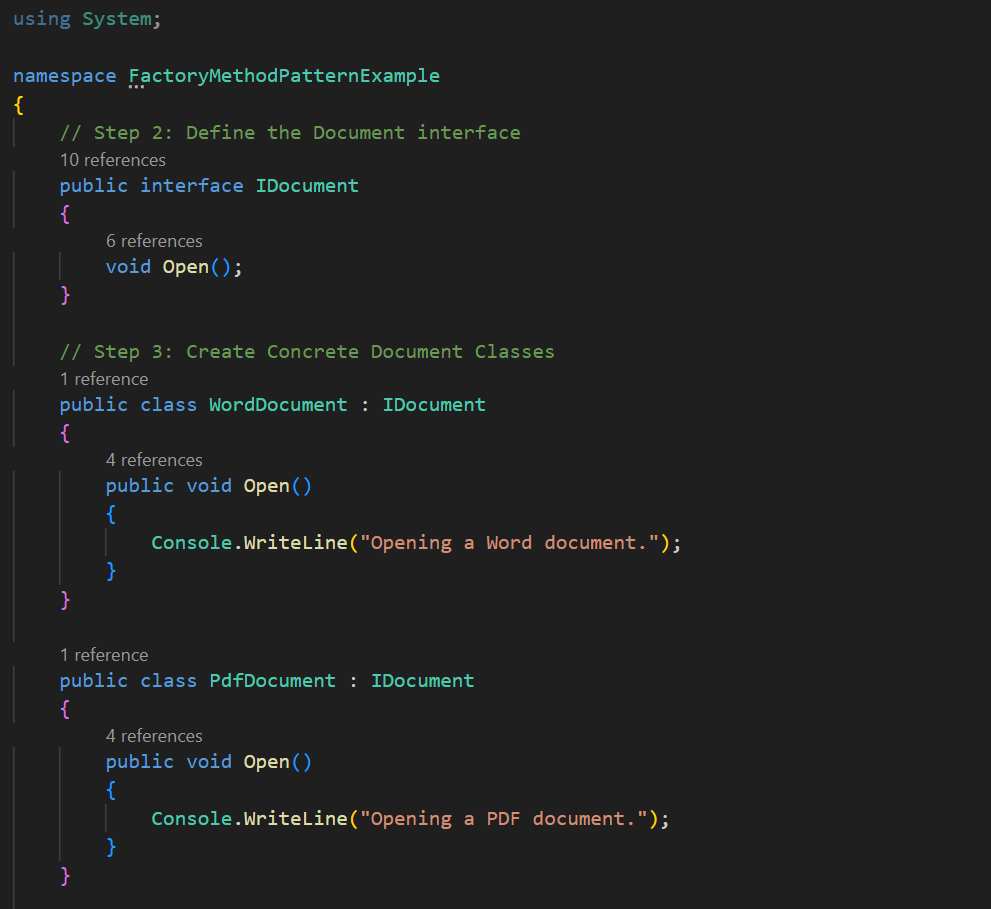


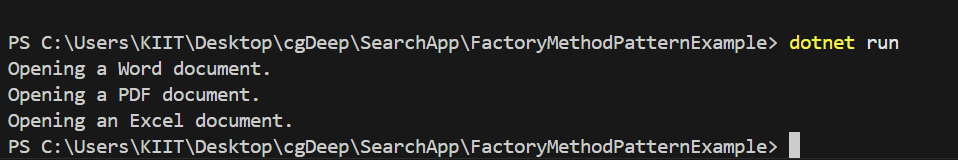
**Output**



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

**Ans**





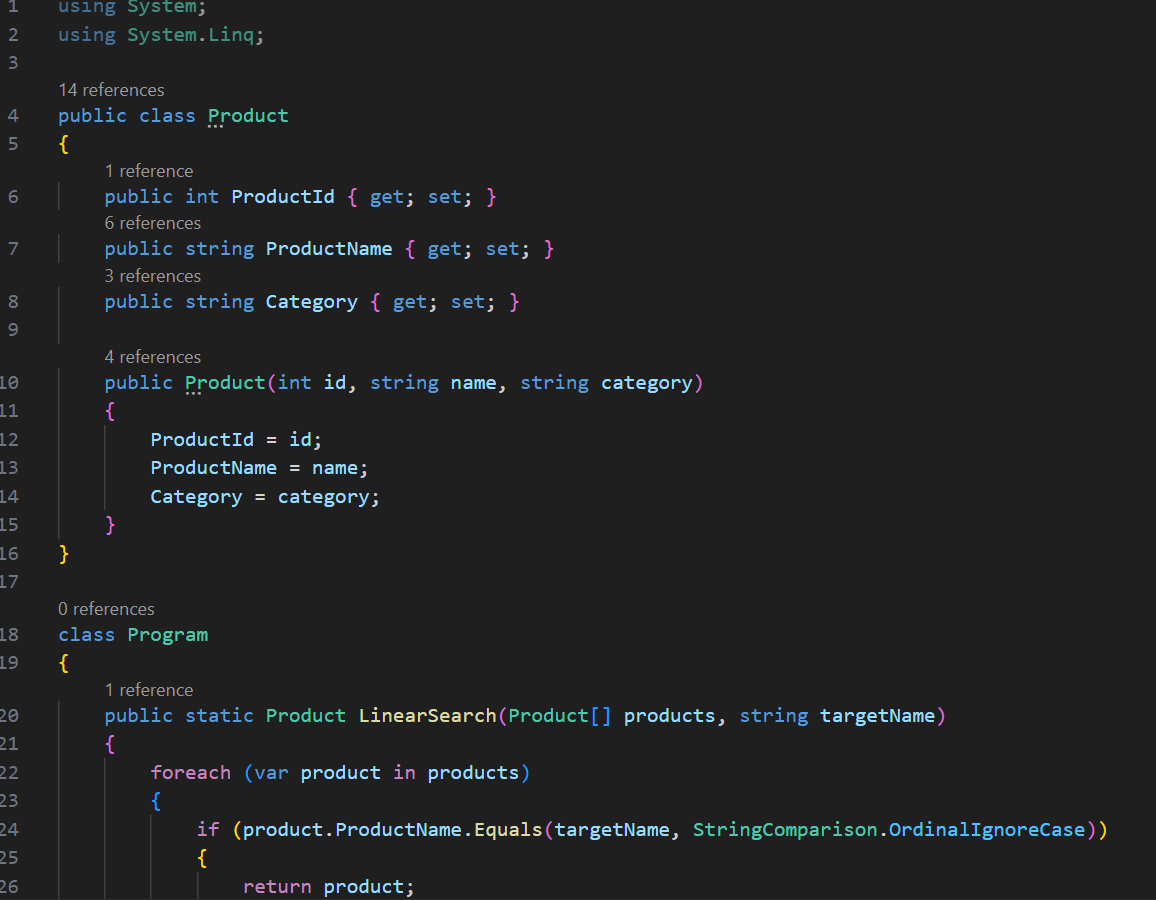
**DSA**

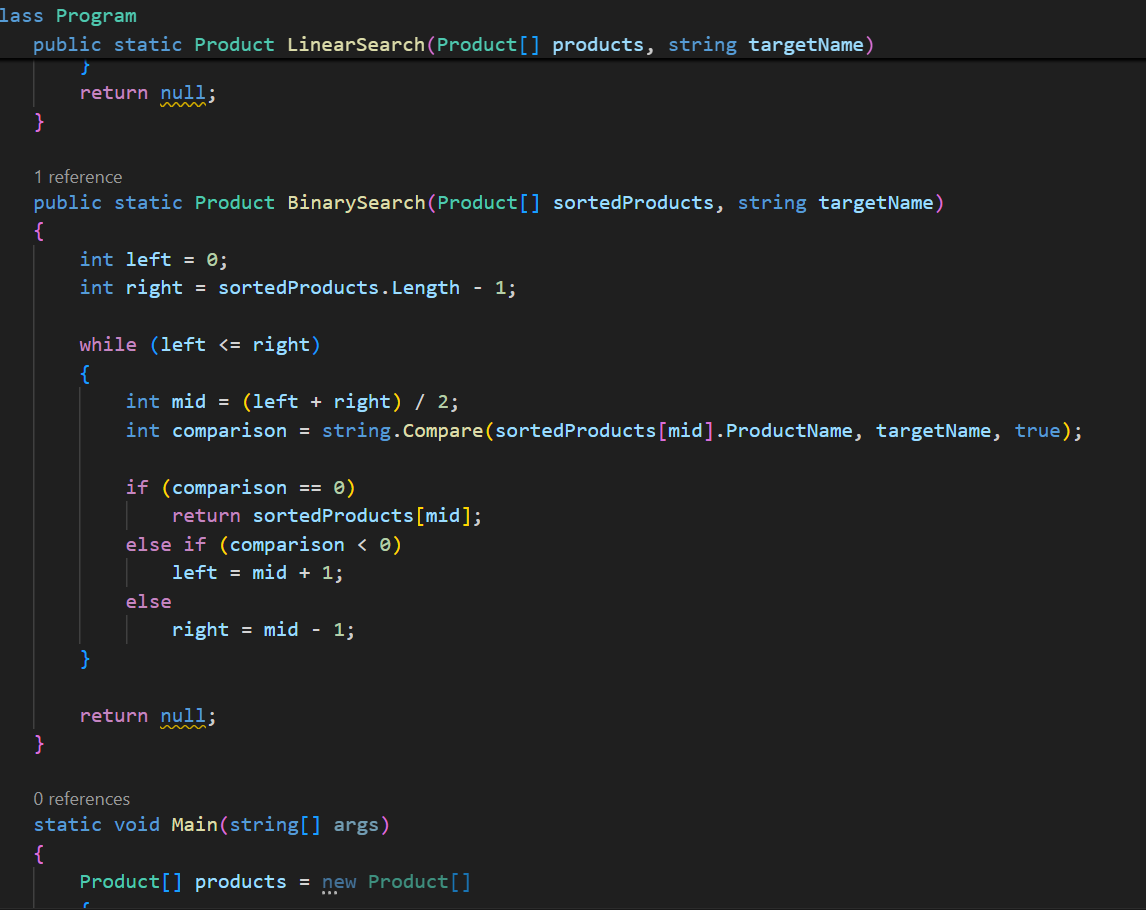
**Exercise 2:** E-commerce Platform Search Function

**Ans:**

Big O notation describes the **upper bound** of an algorithm’s running time as input size increases.

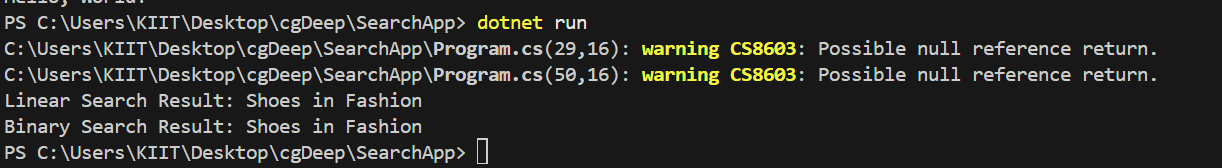
Best: The item is located in the first few comparisons, resulting in minimal search time;  
Average: The item is found somewhere in the middle, leading to moderate search time;  
Worst: The item is at the end or not present, requiring the maximum number of comparisons.







OUTPUT:



**For E-Commerce platformLinear Search** has a time complexity of **O(n)**, checking each element one by one. It works on **unsorted data** but becomes inefficient as the dataset grows.

**Binary Search** has a time complexity of **O(log n)** and works by repeatedly dividing a **sorted list** in half. It's much **faster for large datasets** but requires the data to be sorted.

**Exercise 7:** Financial Forecasting

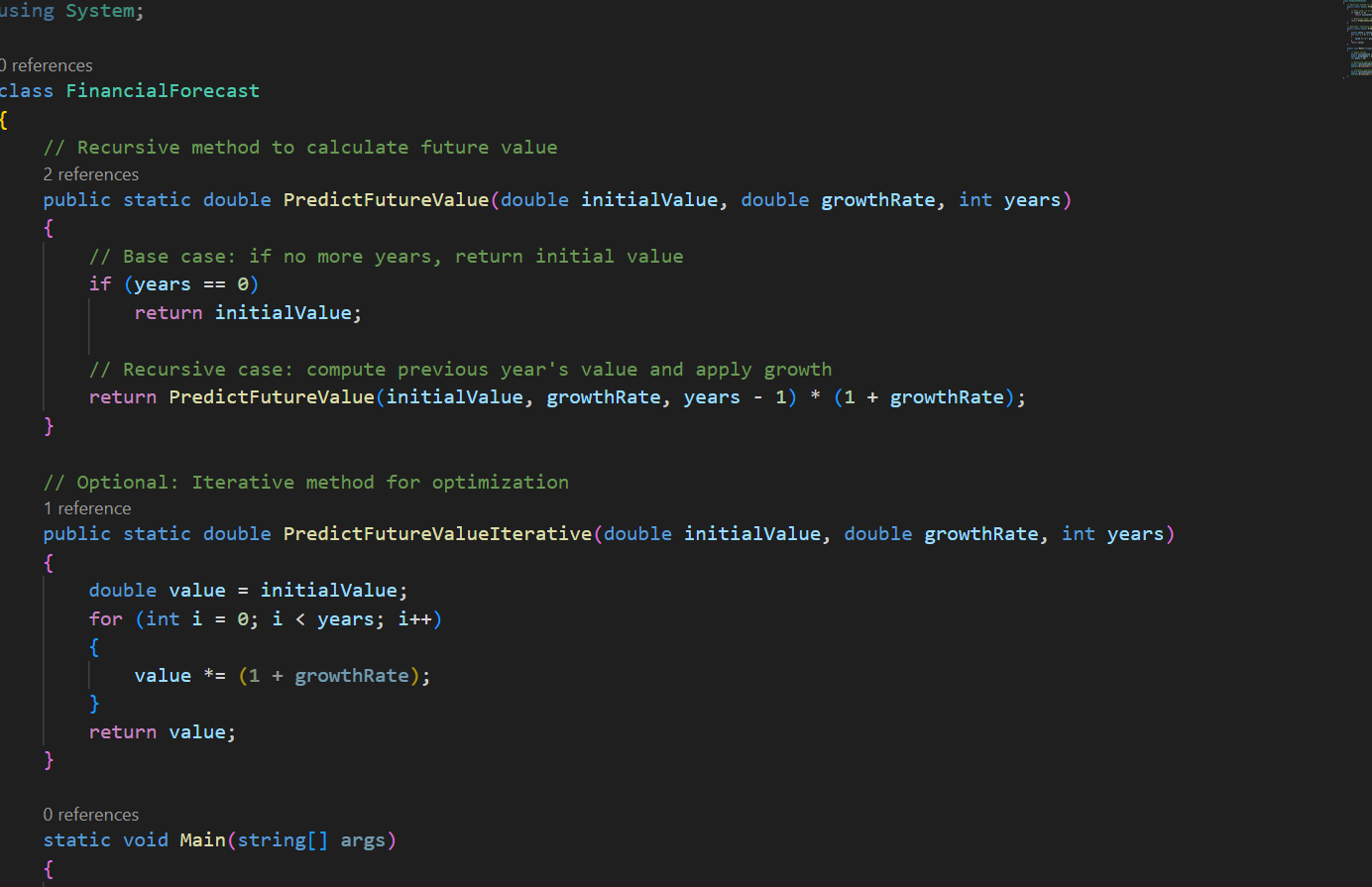
**Ans:**

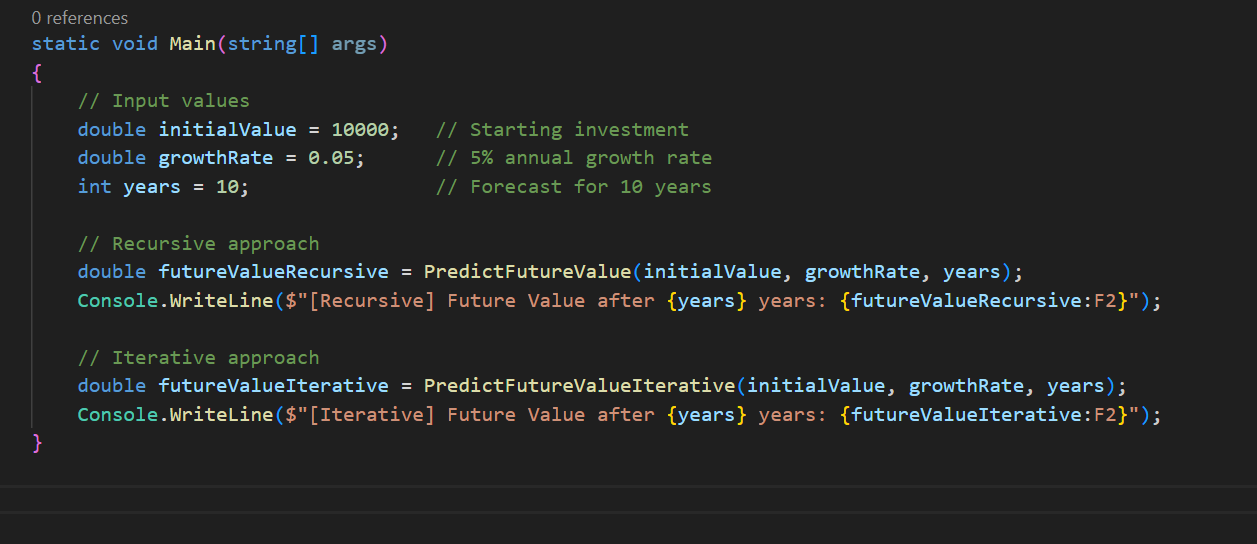
**Recursion** is a programming technique where a function calls itself to solve smaller instances of a problem until it reaches a base case.

**Example Analogy**: Calculating compound interest year after year can be thought of recursively:

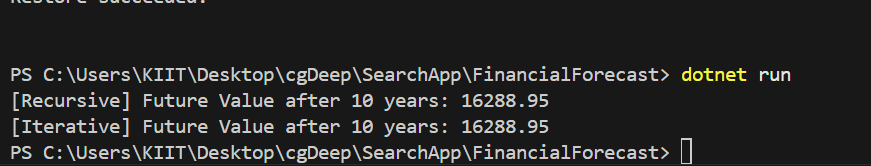
**FutureValue(n)** = **FutureValue(n - 1)** \* (1 + growth rate)

Recursion simplifies problems with repeated, self-similar computations — like calculating future value over time where each year's value builds on the previous one.





Output:



#### ****Time Complexity****

* The recursive approach has **O(n)** time complexity since it performs one computation per year.
* Each call reduces the problem by one (years - 1), so total calls = n.

**Optimization**

While this recursion is already linear and doesn't recompute overlapping subproblems, **you can optimize further** by:

* **Using Memoization** (not needed here unless growth rates vary over years).
* **Converting to Iterative Form** for efficiency.