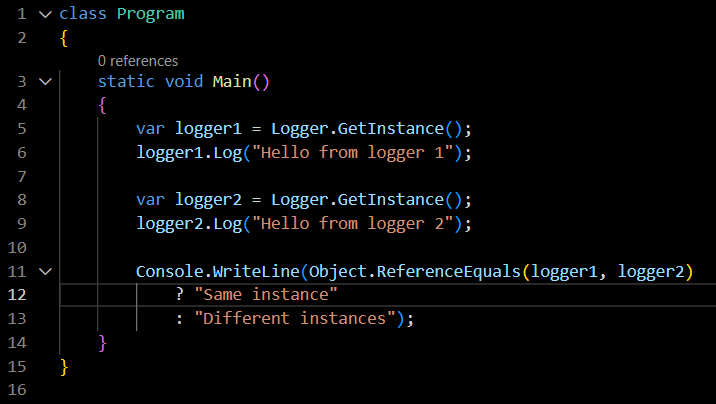
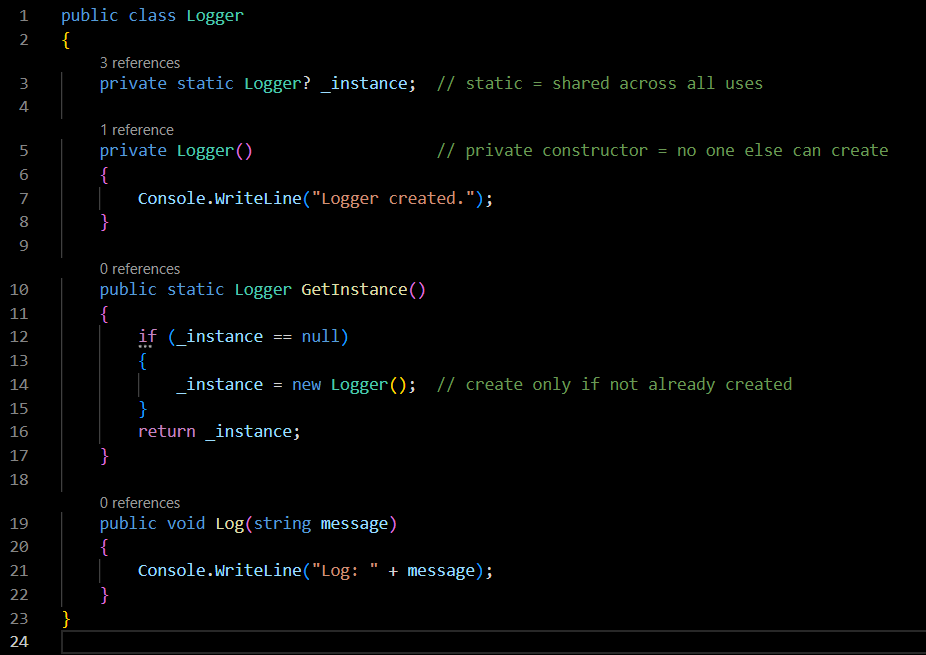
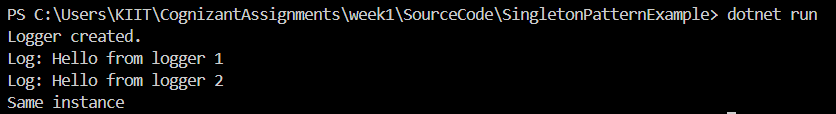
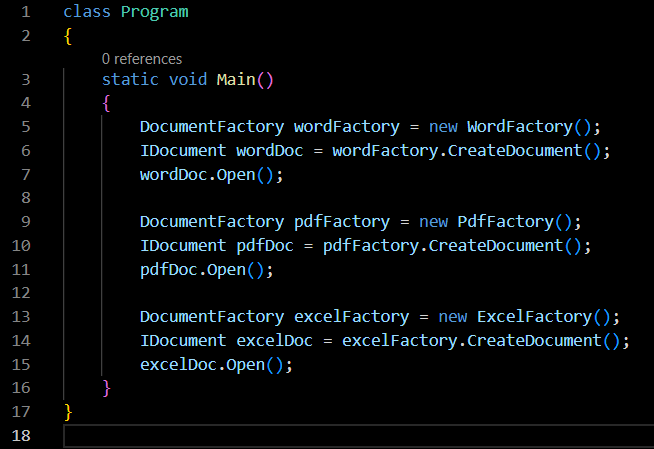
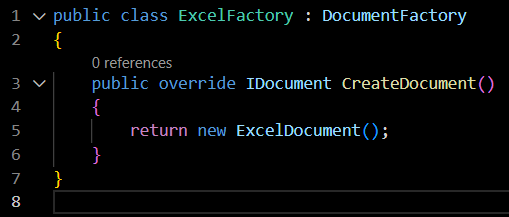
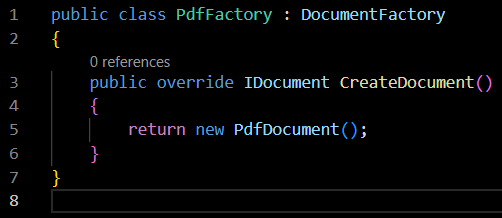
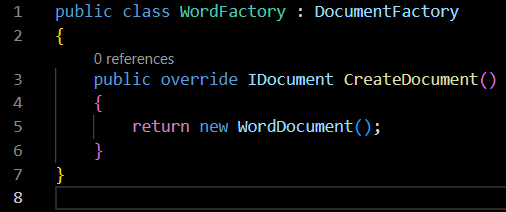
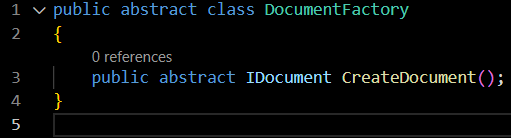
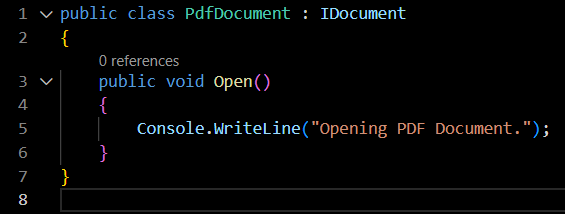
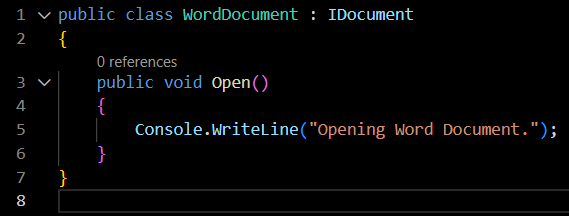
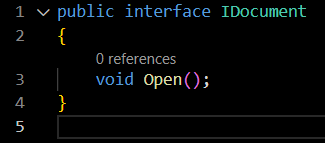
**Exercise : Implementing the Singleton Pattern**

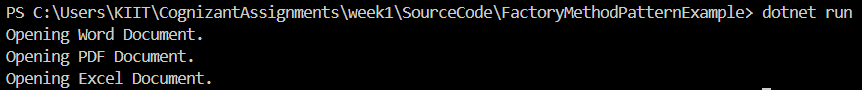
CODE :

OUTPUT :

**Exercise : Implementing the Factory Method Pattern**

CODE :



OUTPUT :

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

**Understanding Asymptotic Notation (Big O) :**

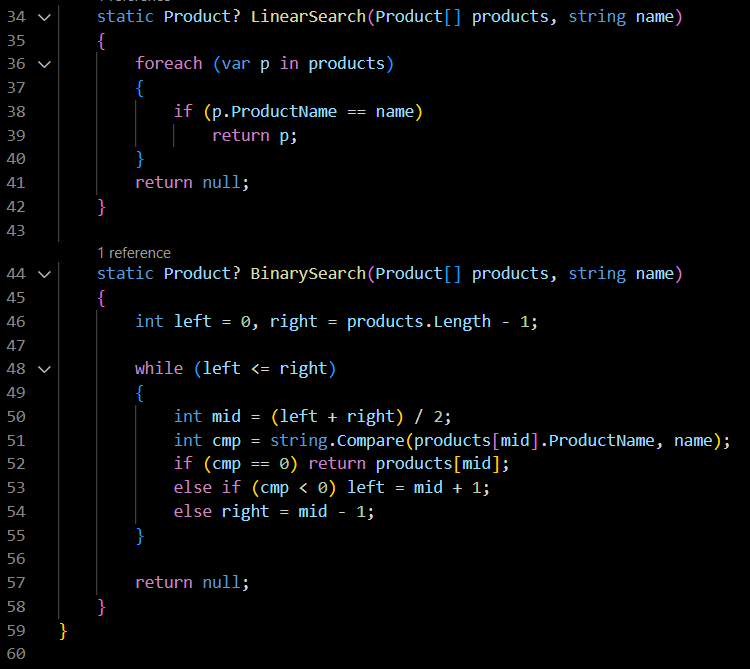
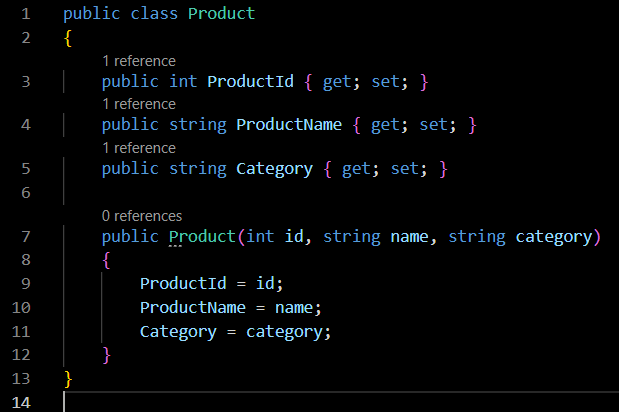
Big O notation describes how an algorithm’s runtime or space requirement grows as input size increases.

For Search Operations:

| **Case** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Best** | O(1) (first item) | O(1) (middle item) |
| **Average** | O(n/2) = O(n) | O(log n) |
| **Worst** | O(n) | O(log n) |

Binary search is much faster than linear search — but it only works on sorted data.

**Implementation :**

CODE :

OUTPUT :

**Analysis :**

Time Complexities:

Linear Search: O(n)

Binary Search: O(log n) — but only works if data is sorted.

Best Choice:

Use binary search for speed. We can store product data sorted by name or ID

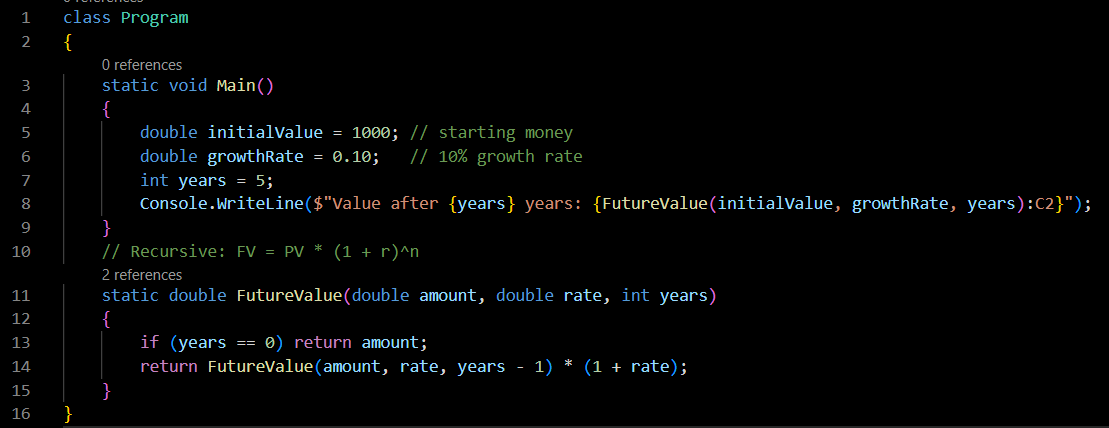
**Exercise : Financial Forecasting**

**Understanding Recursion :**

Recursion is when a method calls itself to break a problem into smaller parts.

Example Use Case : predicting next value based on previous ones.

**Implementation :**

CODE :

OUTPUT :Screenshot 2025-06-22 005203

**Analysis :**

Time Complexity:

| **Method** | **Time Complexity** |
| --- | --- |
| Basic Recursion | O(n) |
| Memoized | O(n) (but faster in practice due to caching) |

In this example, recursion is OK, but for large n, we should either use iteration if possible (loop) or use memoization to cache previous results.