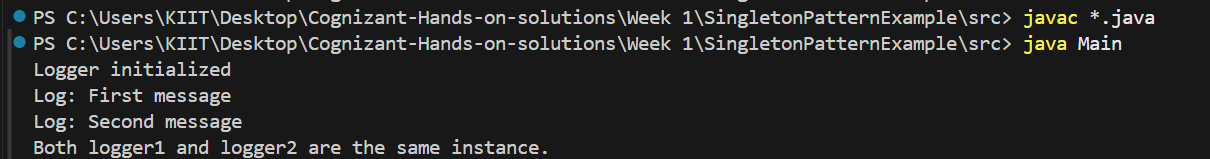
**Design Patterns and Principles:**

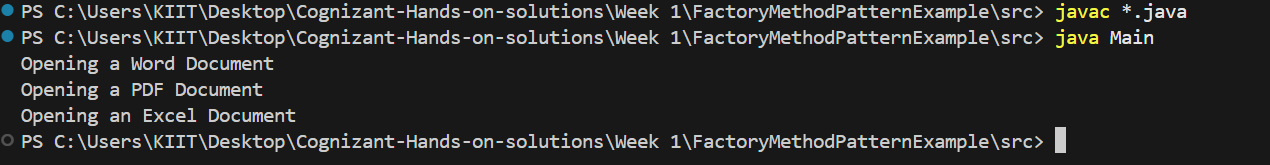
**Exercise 1: Implementing the Singleton Pattern**

Output:



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

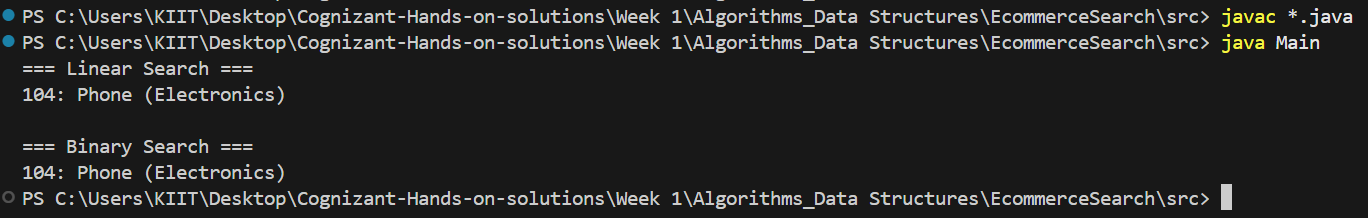
Output:



**Algorithms\_Data Structures:**

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

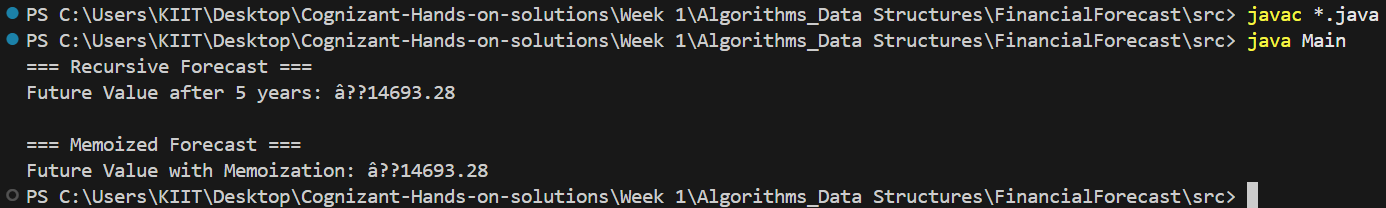
Output:



* **Linear Search** is simple, works on unsorted data, but gets slower as your product list grows. (**Time complexity:** O(n))
* **Binary Search** is much faster but needs **sorted data**, so use it when:
  + You have a large product catalog
  + Data can be sorted once and used repeatedly
  + (**Time complexity:** O(log n))

**Exercise 7: Financial Forecasting**

Output:



**Regular Recursive Version:**

* **Time Complexity:** O(n)
* **Space Complexity:** O(n) (due to function call stack)

Since we compute every year from n down to 0, we do n recursive calls

### Optimized Version (Memoization)

* **Time Complexity:** Still O(n), but **avoids redundant recalculations** (useful in more complex recurrence).
* **Space Complexity:** O(n) for the memo array + O(n) for call stack

In our case, memoization isn't a huge help because we're doing only one computation per year. But in complex cases (like multiple branches), memoization drastically cuts down computation.