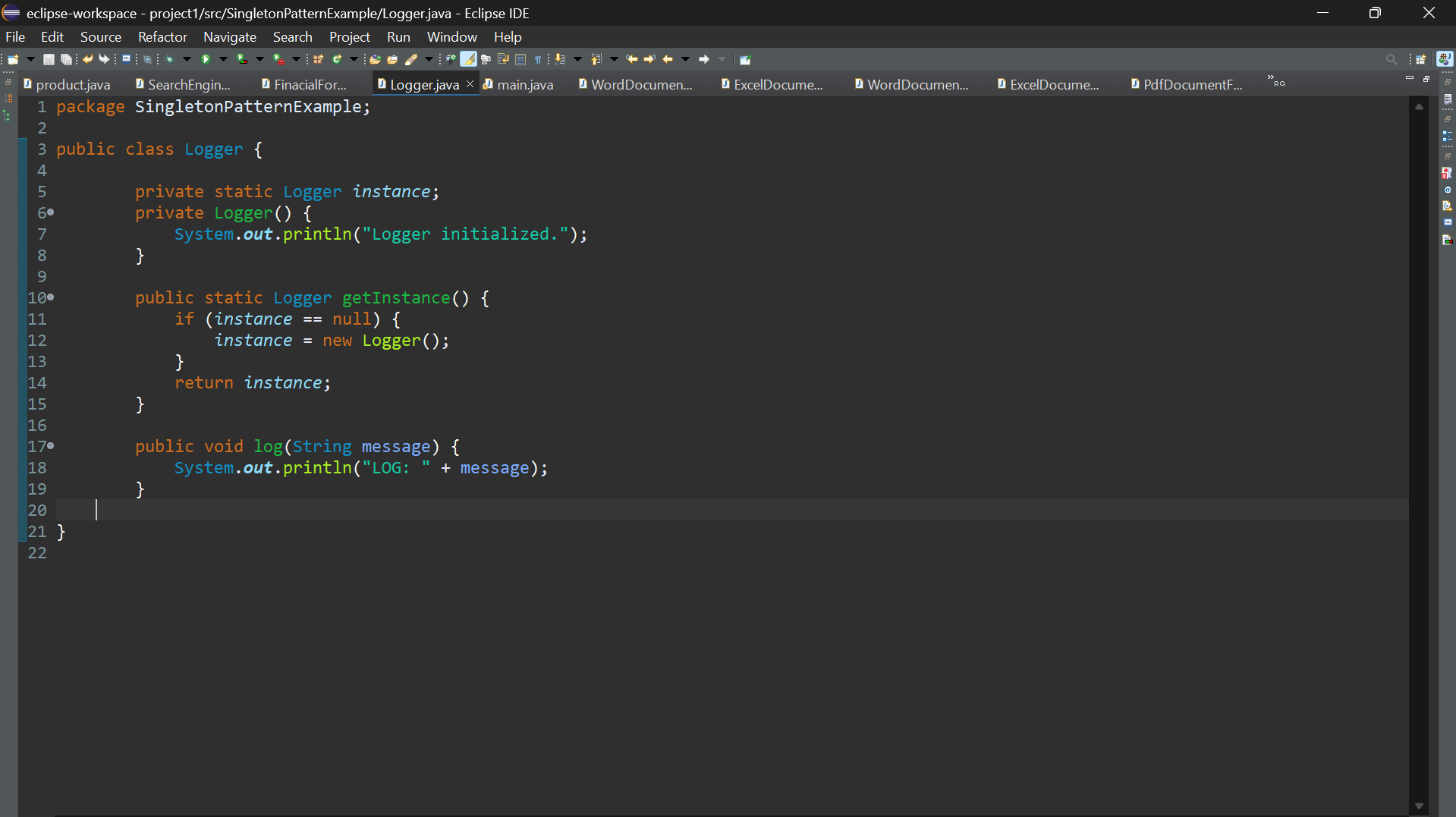
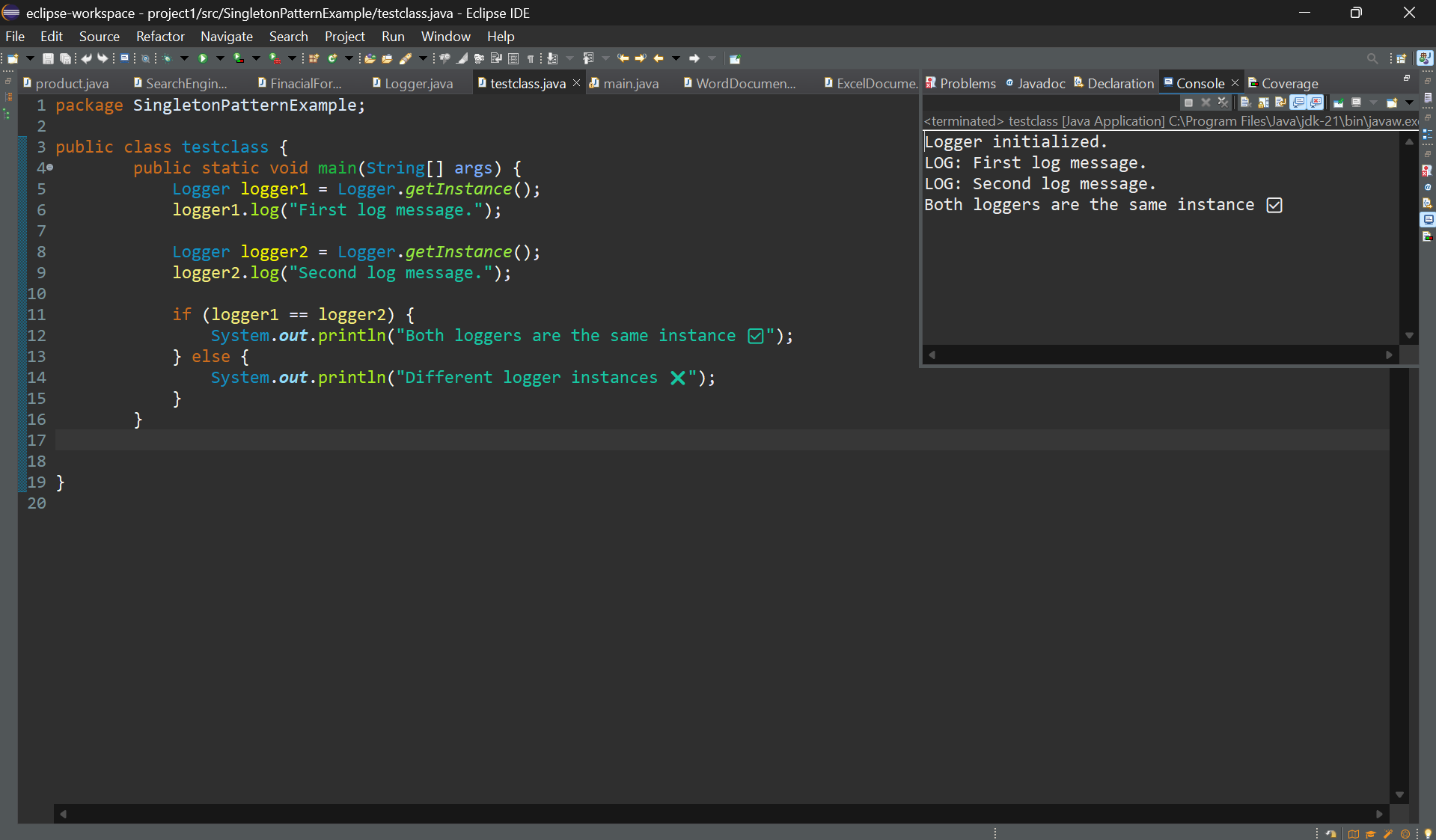
WEEK 1 MANDATORY HANDSON

SUPERSET ID:6410152

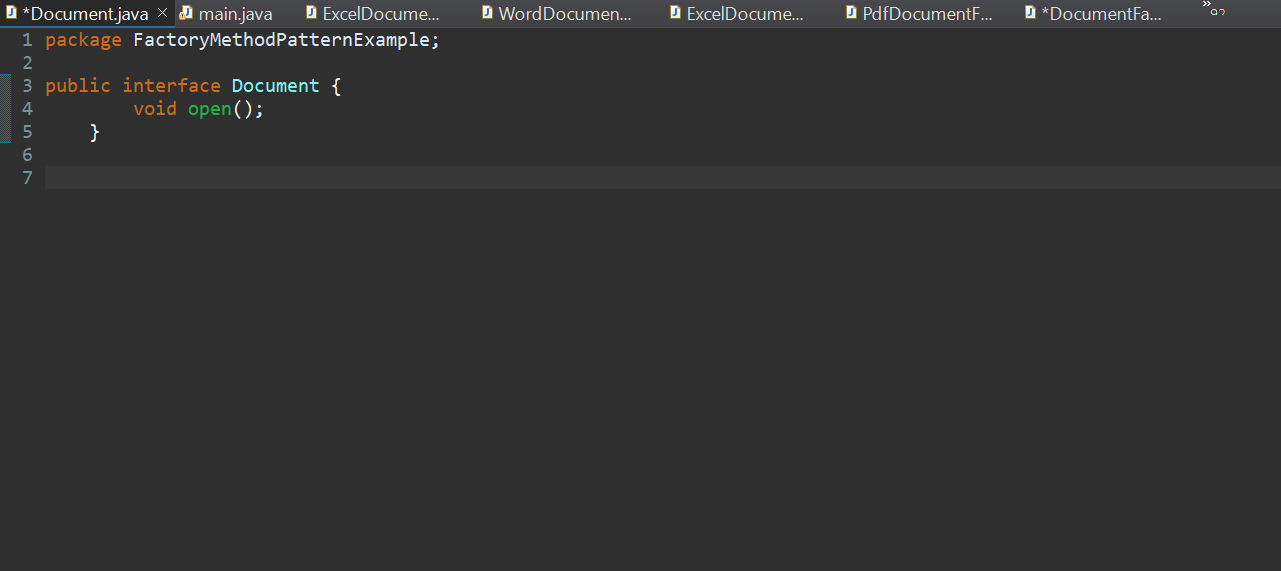
Design principles & Patterns:

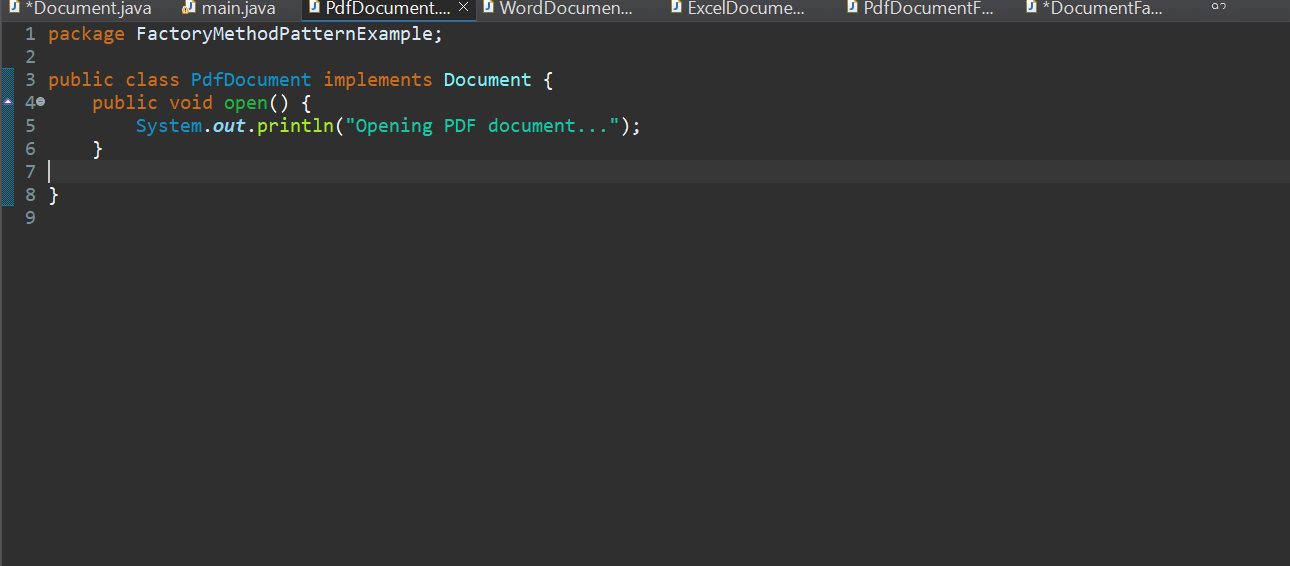
Exercise 1: Implementing the Singleton Pattern:

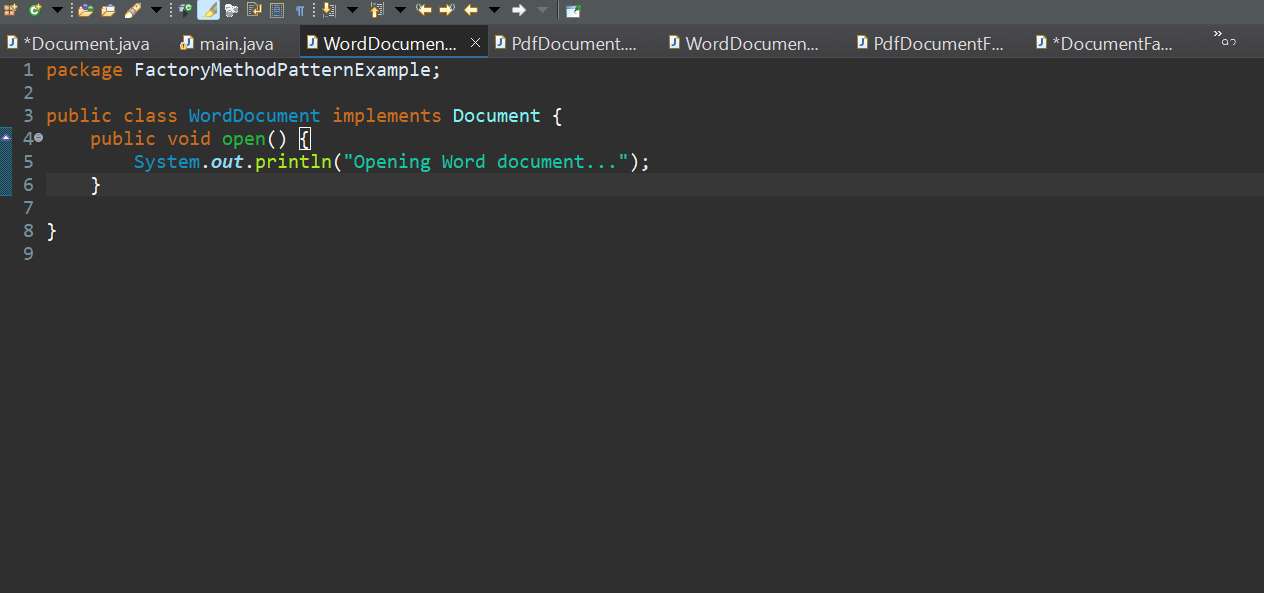


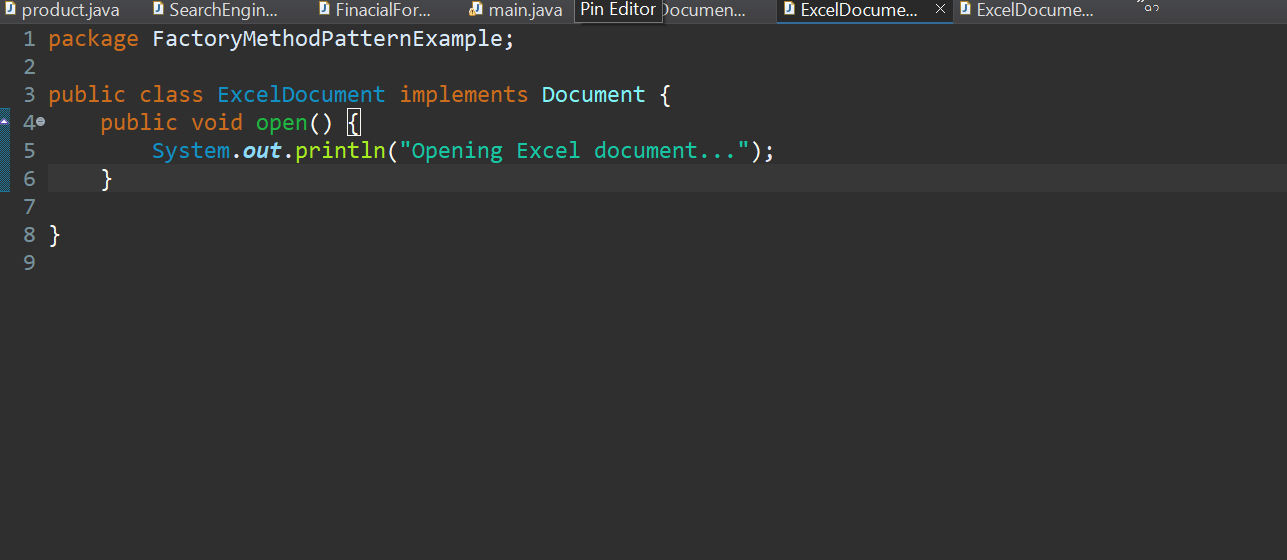


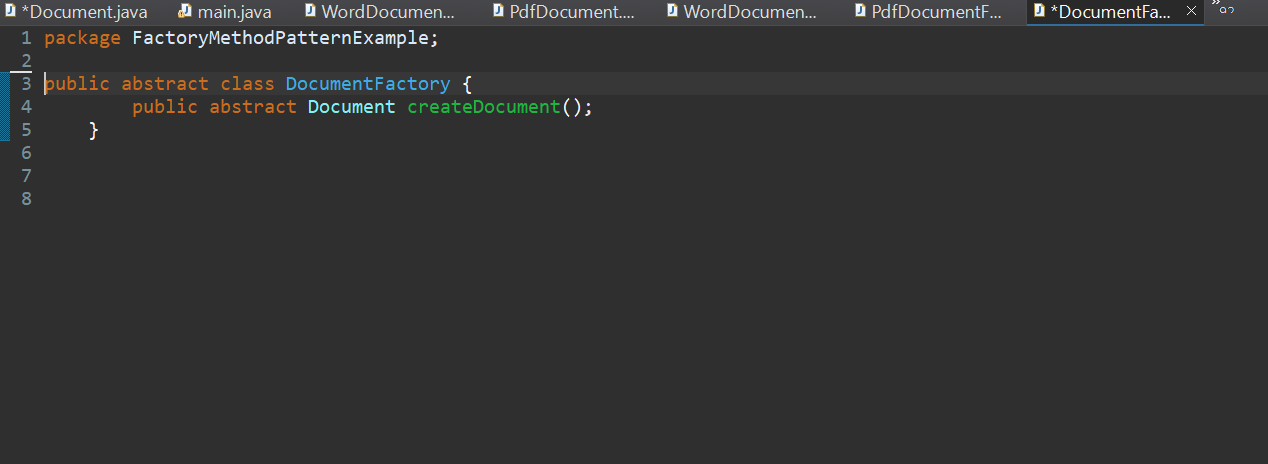
Exercise 2: Implementing the Factory Method Pattern:

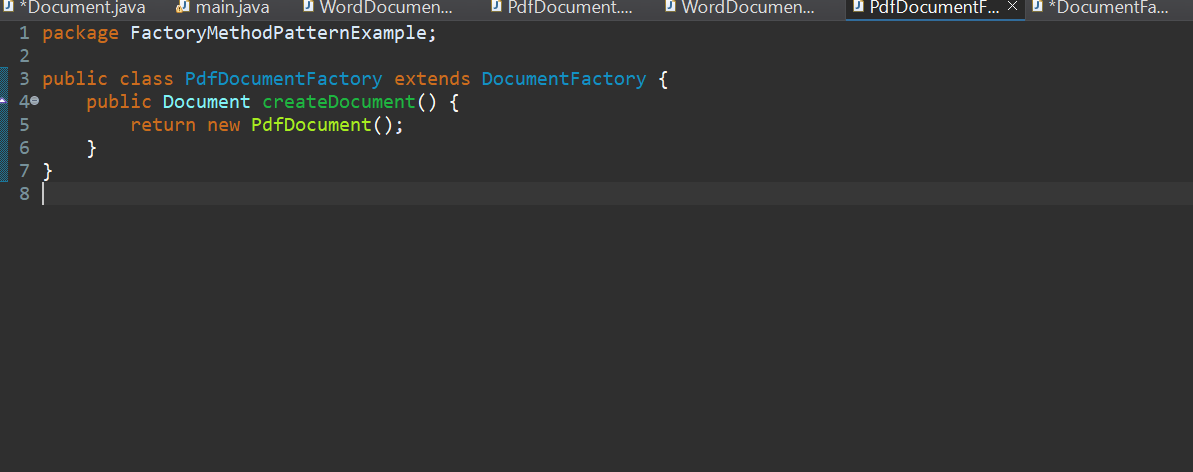


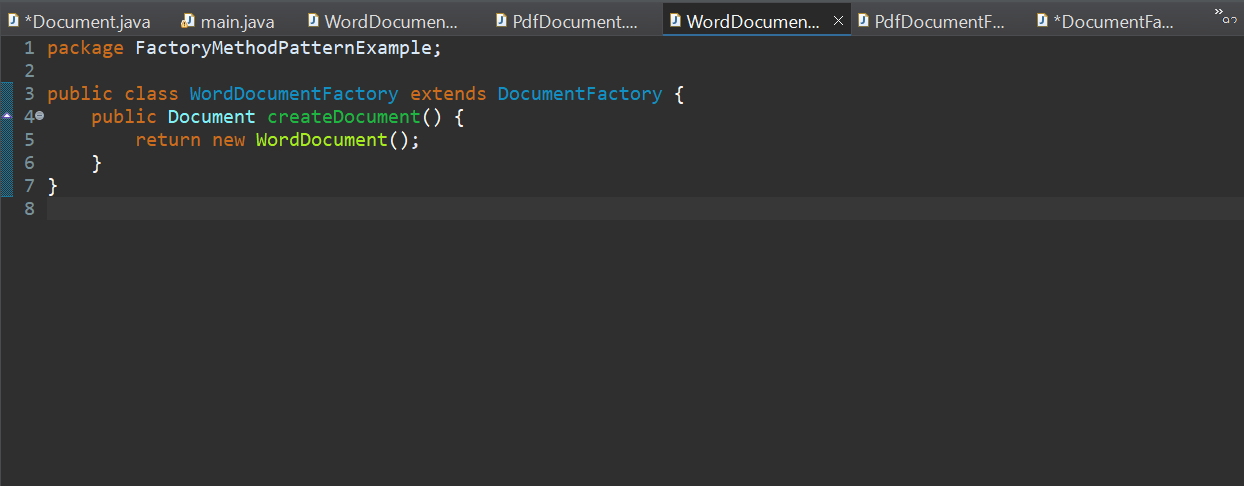


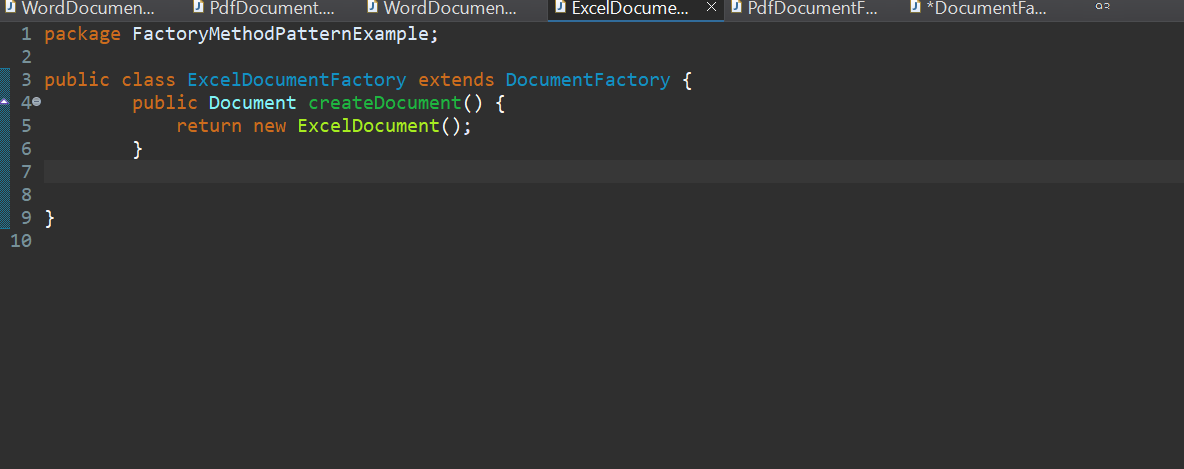


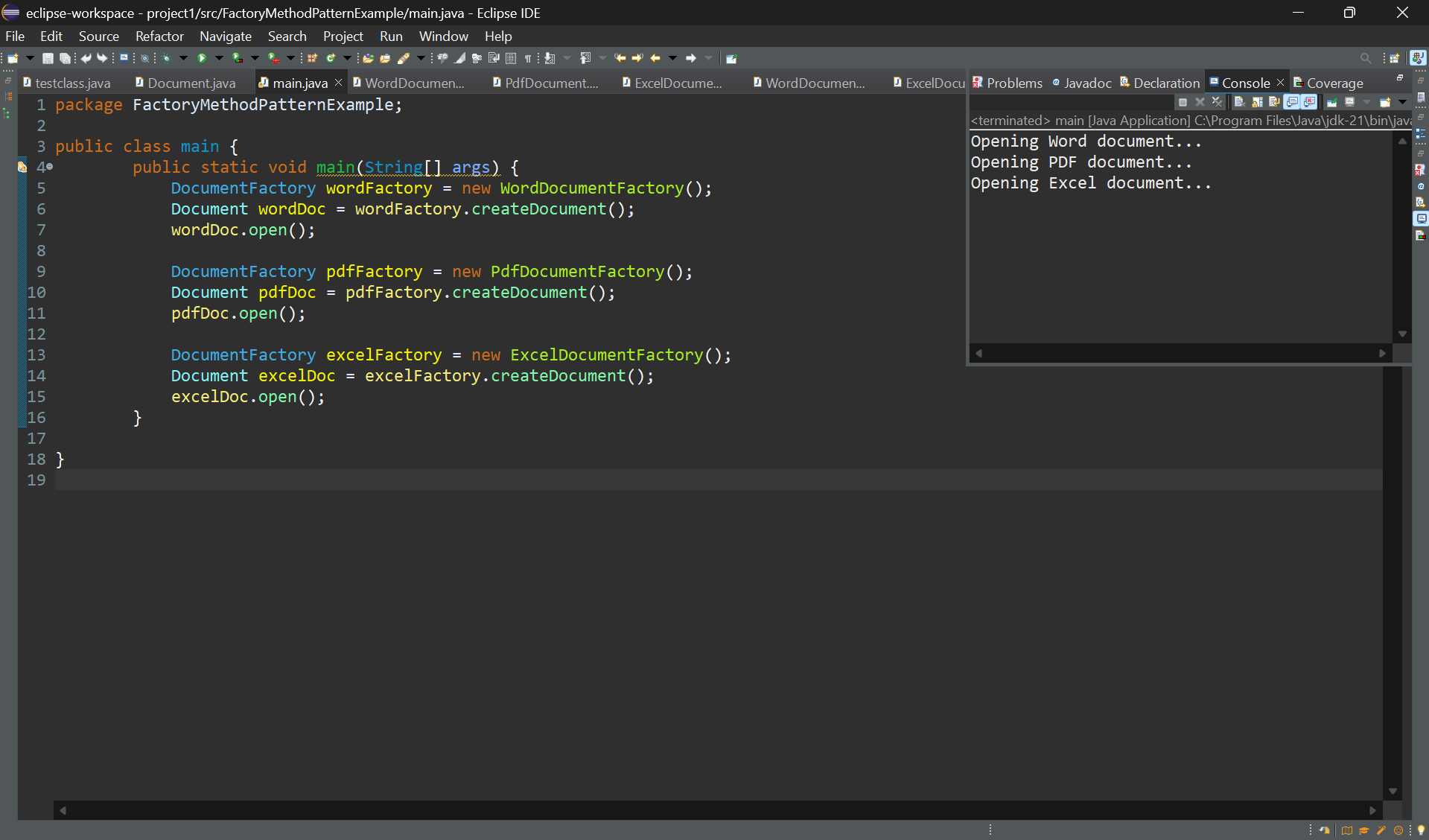












SUPERSET ID:6410152

Data structure and Algorithm:

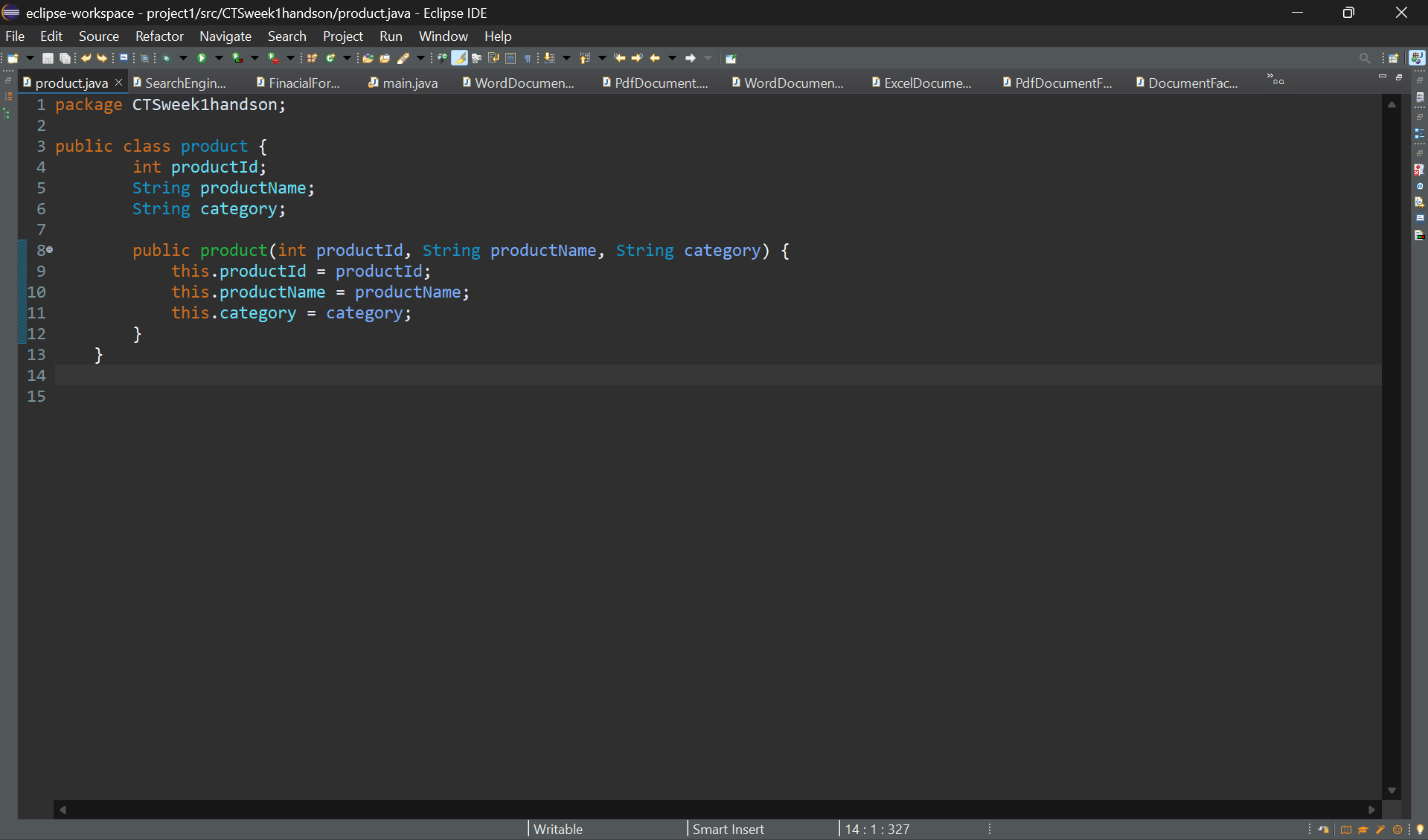
Exercise 2: E-commerce Platform Search Function:

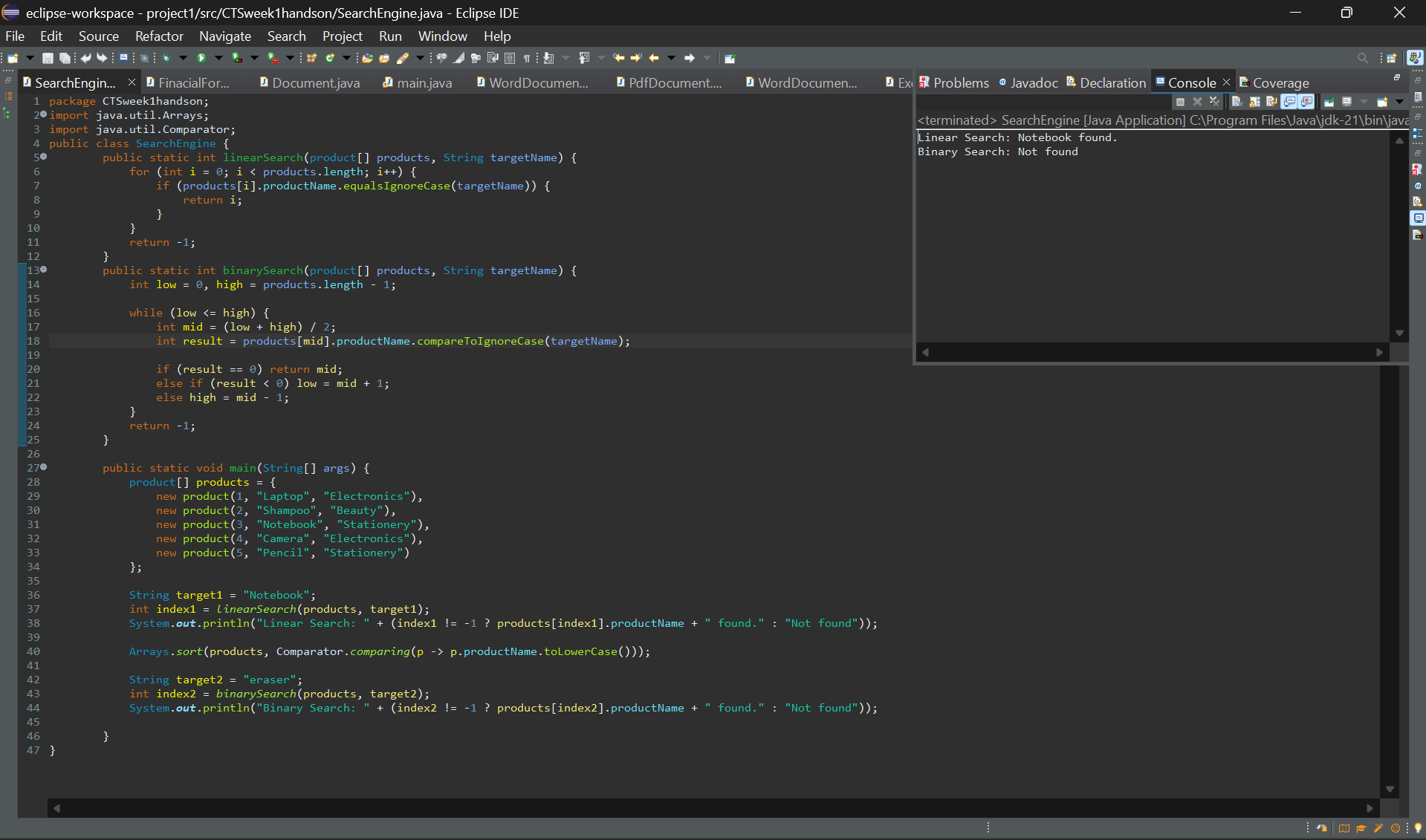
* Explain Big O notation and how it helps in analyzing algorithms.

Big O notation describes the time or space complexity of an algorithm in terms of how it grows with input size. It helps in comparing algorithms and choosing the most efficient one, especially for large datasets.

* Describe the best, average, and worst-case scenarios for search operations.

In the best case, the item is found right away both linear and binary search take O(1) time. In the average case, linear search takes O(n) since it checks many items, while binary search takes O(log n) by cutting the list in half. In the worst case, the item isn’t found or is at the end, linear is still O(n), but binary stays efficient at O(log n) if the list is sorted.





* Compare the time complexity of linear and binary search algorithms:

The time complexity of linear search is O(n), as it checks each element one by one, which makes it slower for larger datasets. In contrast, binary search has a time complexity of O(log n) because it repeatedly divides the search space in half, making it much faster—but only if the data is sorted.

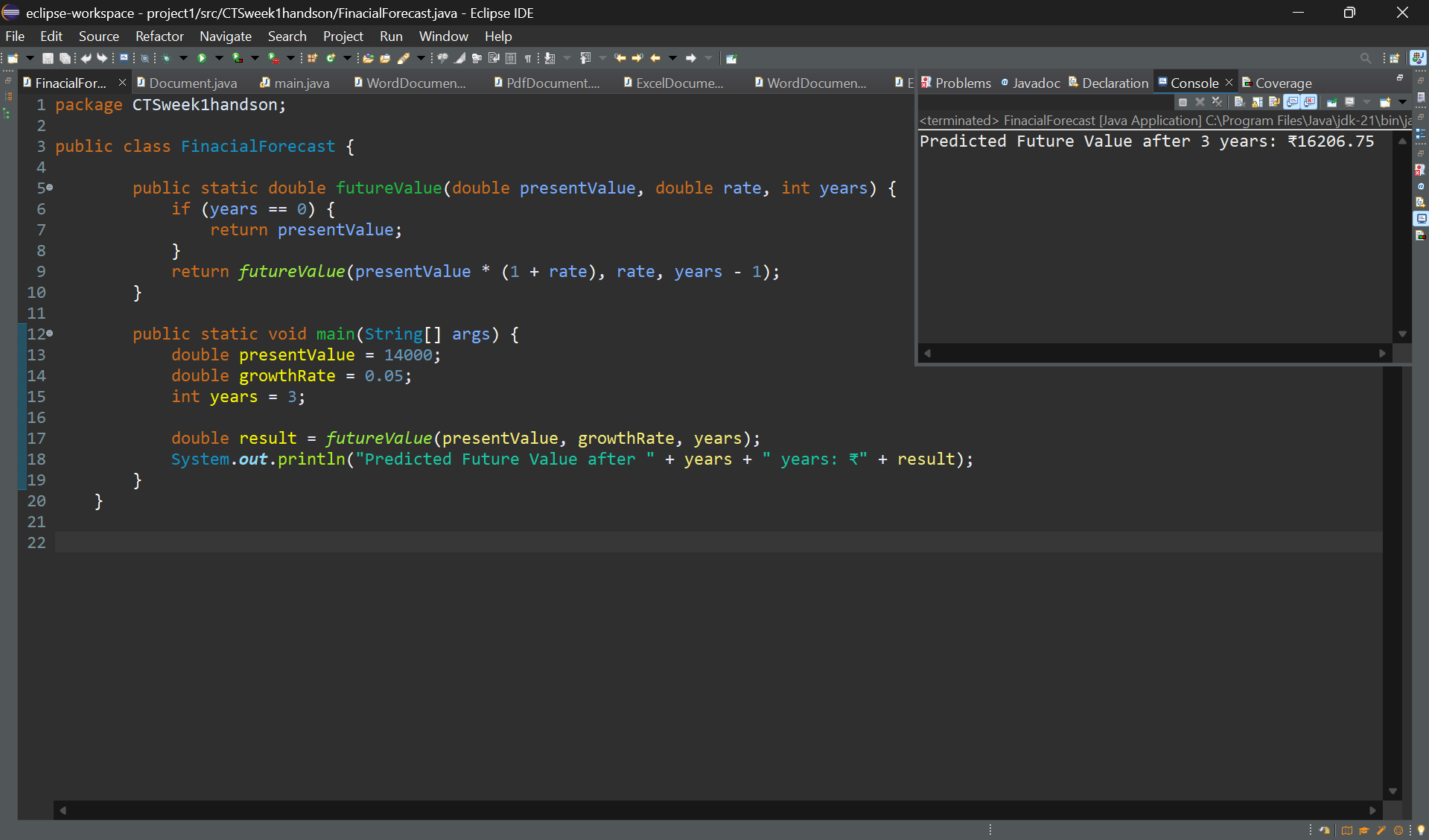
* Discuss which algorithm is more suitable for your platform and why?

For an e-commerce platform with a large and growing number of products, binary search is more suitable as it provides quicker search results and enhances user experience. However, if the data is small or unsorted, linear search can still be used effectively for simplicity.

Exercise 7: Financial Forecasting:

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

Recursion is when a method calls itself to solve smaller parts of a problem until it reaches a base case. It helps simplify complex problems like calculating future values, factorials, or Fibonacci numbers — where each step builds on the previous one.



* Discuss the time complexity of your recursive algorithm.

The recursive algorithm runs once per year, so it has a time complexity of O(n), where *n* is the number of years.