**1. Understanding Asymptotic Notation**

Asymptotic notation is a mathematical tool used to describe the efficiency of algorithms, especially as the input size becomes very large. Instead of measuring the exact time or memory an algorithm takes, we use this notation to compare algorithms based on how they grow with input size (n).

**Big O Notation**

Big O (written as O(n), O(log n), etc.) is the most commonly used asymptotic notation. It describes the worst-case performance of an algorithm — meaning, it shows the maximum number of steps the **algorithm could take in the worst possible scenario.**

**Why It Matters:**

* Helps compare different algorithms regardless of the computer or programming language.
* Useful in predicting how the algorithm will perform as data increases.
* Focuses on scalability rather than actual speed.

**Example:**

A simple loop that runs n times has a time complexity of O(n) — this means the time grows linearly as input grows.

Best, Average, and Worst-Case Scenarios (Search Operations)

When analyzing algorithms, it's important to consider different scenarios:

**1. Best Case:**

The scenario where the algorithm performs the least amount of work.  
For example, in a linear search, if the item is found at the first position, the search completes in O(1) time.

**2. Average Case:**

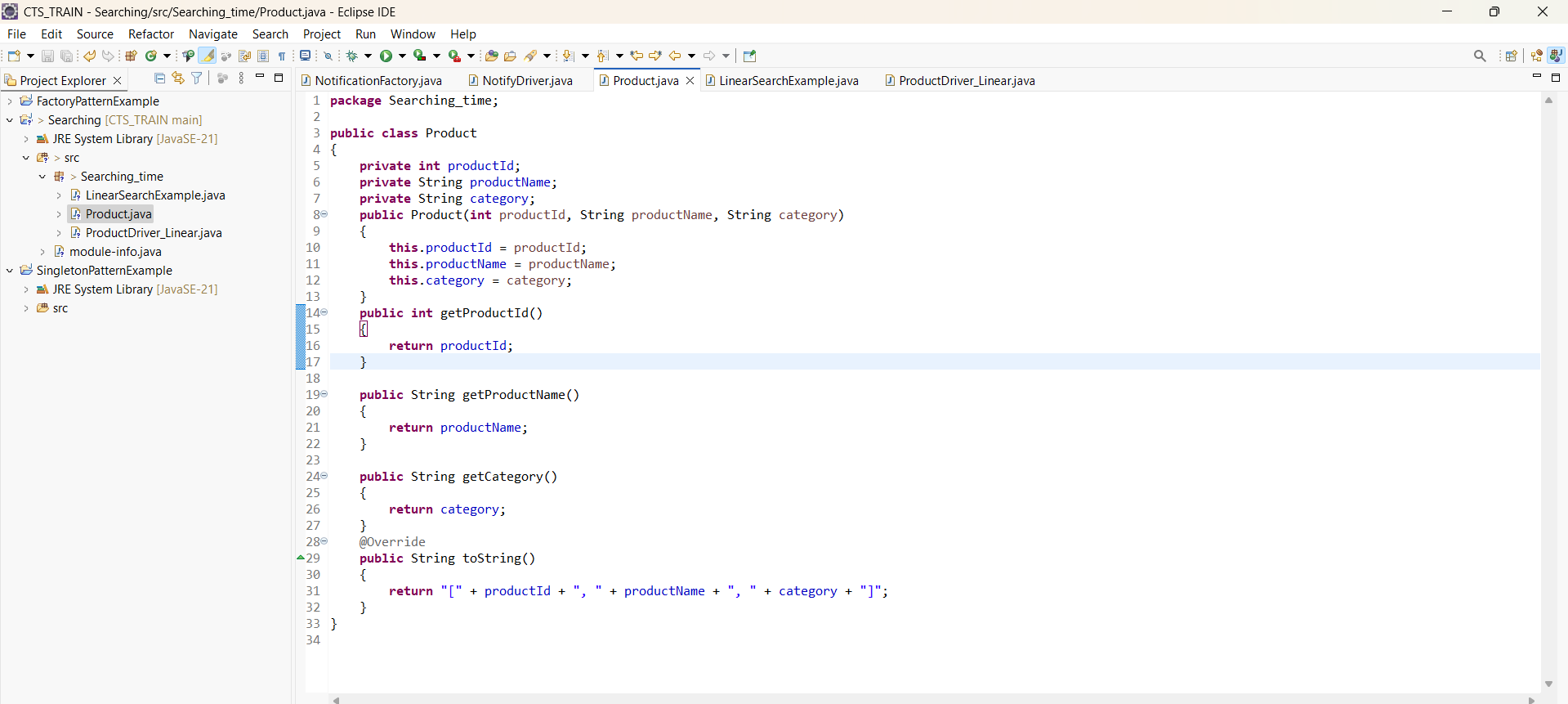
The expected performance assuming the data is random and the item is somewhere in the middle.  
In linear search, this is around O(n/2) which simplifies to O(n).

**3. Worst Case:**

The scenario where the algorithm performs the most amount of work.  
In linear search, if the item is not found or is at the last position, it requires scanning all elements, so time complexity is O(n).

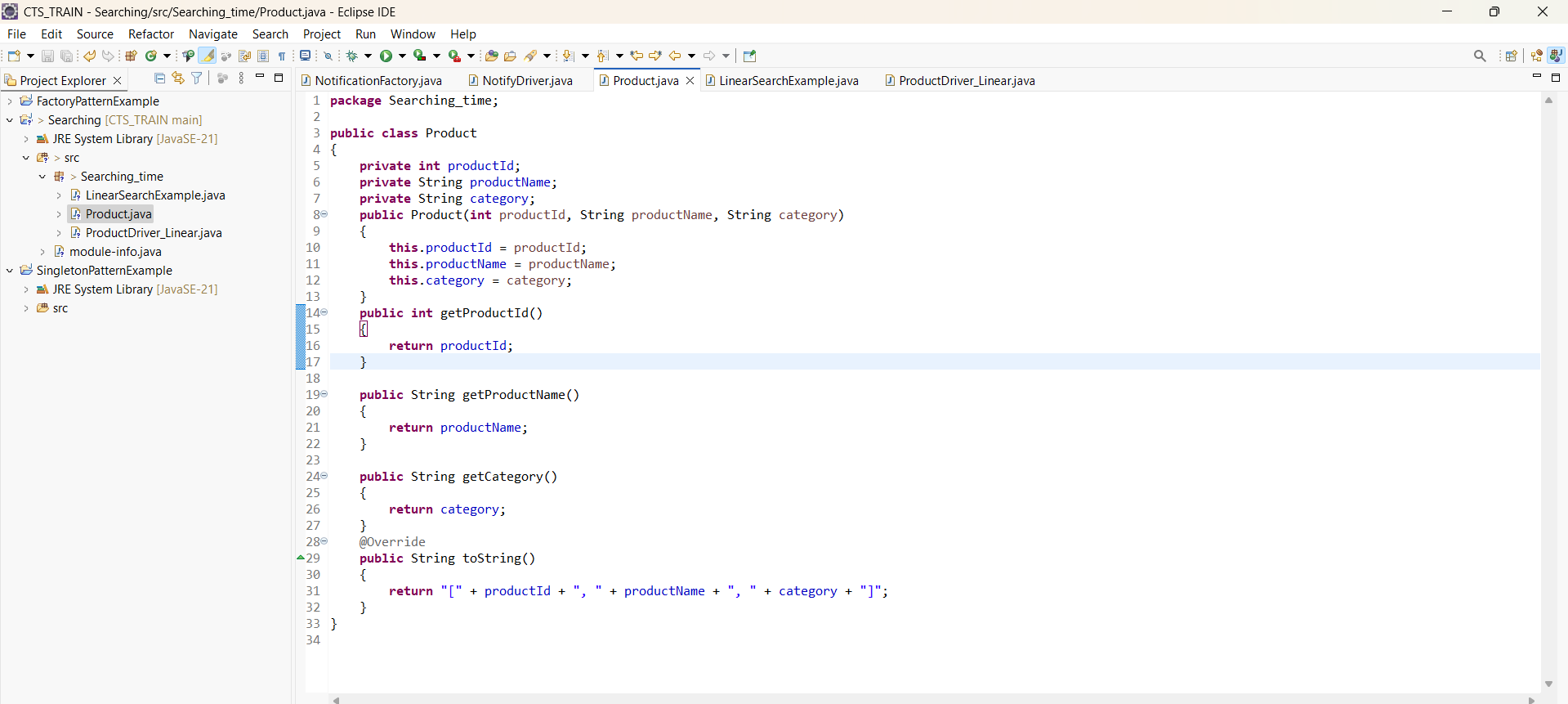
2**. Setup:**

**Product.java created !**

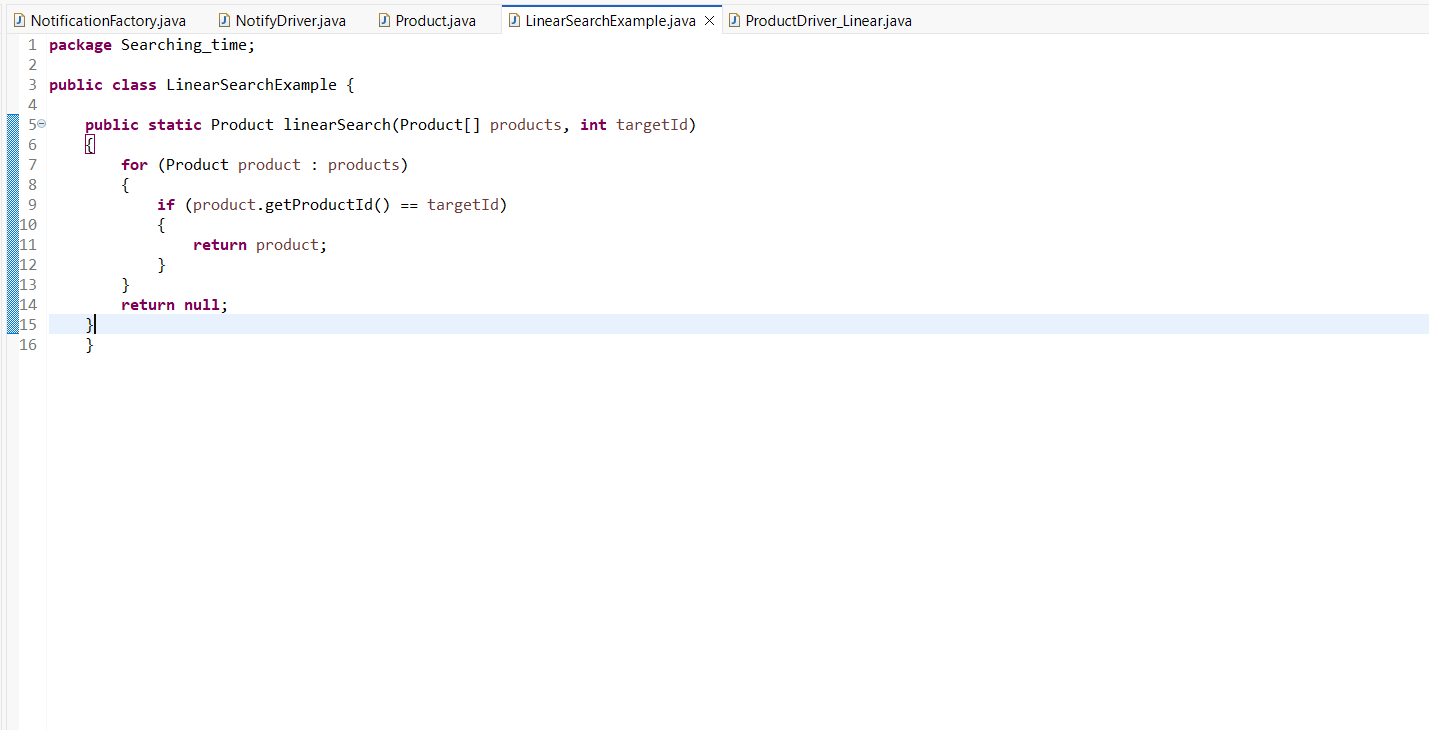


3. **Implementation:**

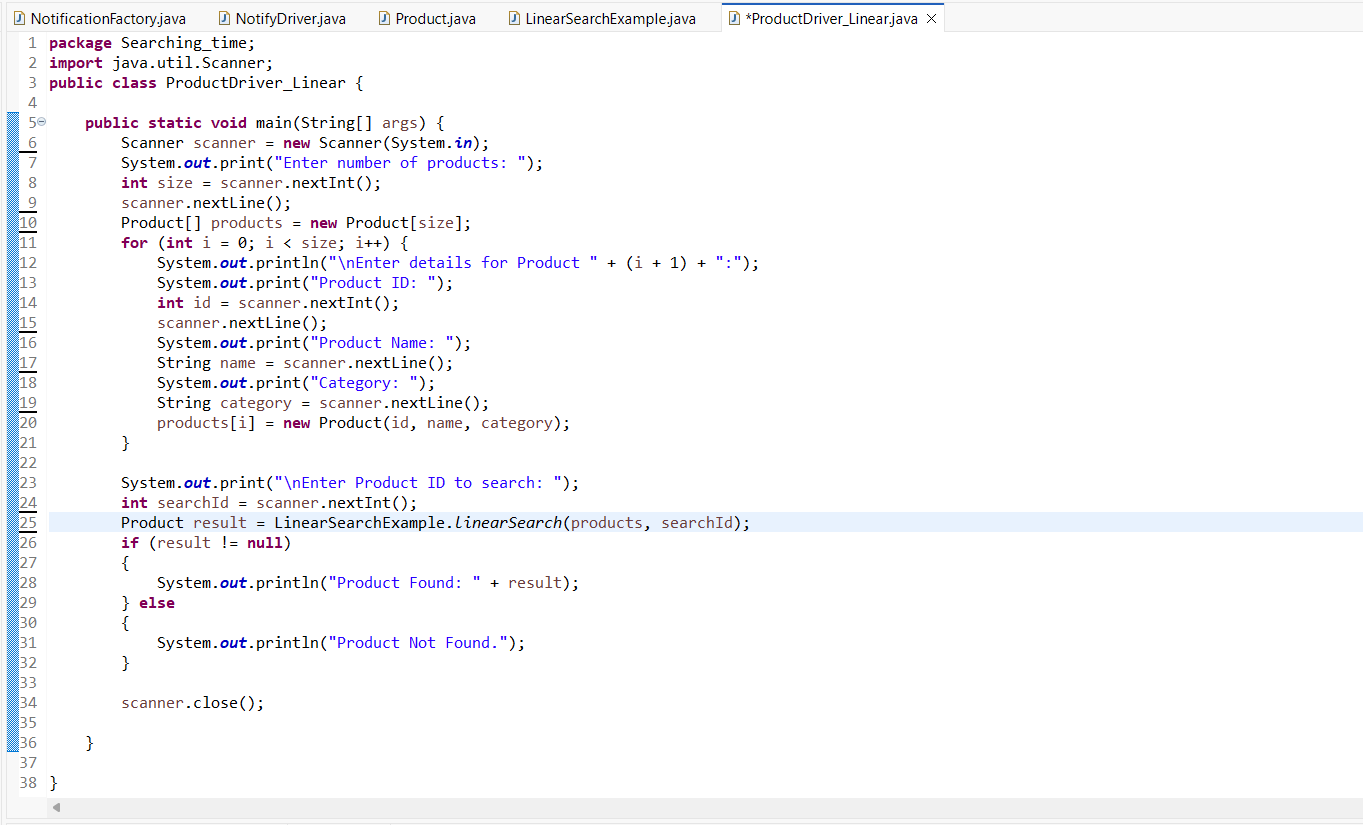
**Linear search:**

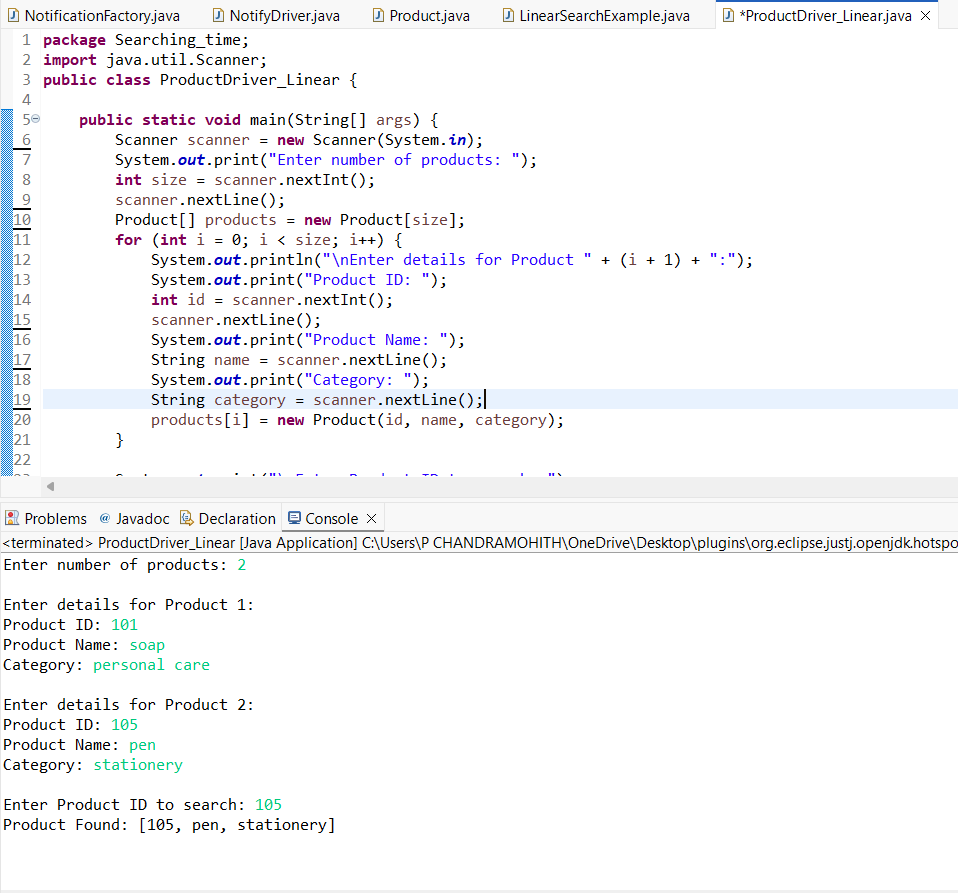
Product.java

LinearSearchExample.java



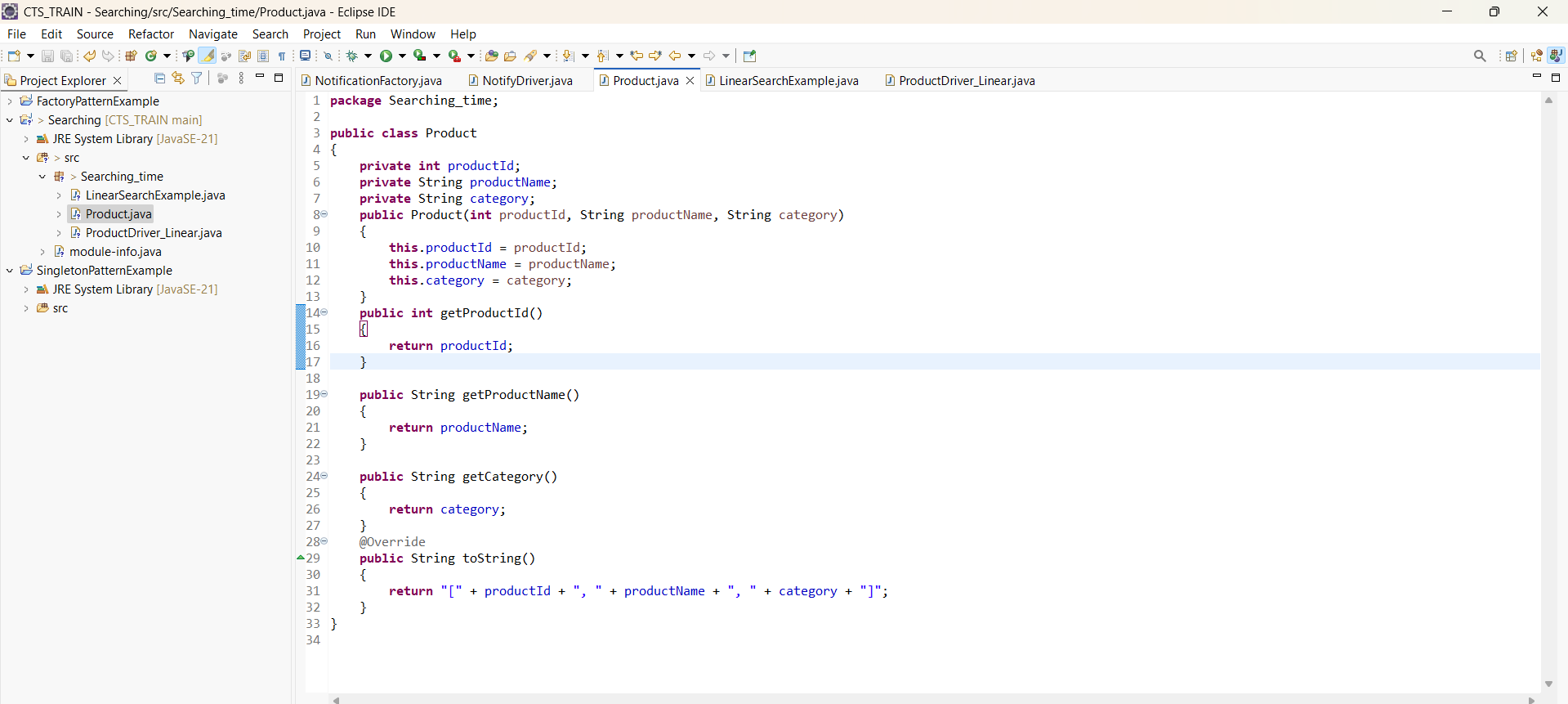
ProductDriver\_Linear.java



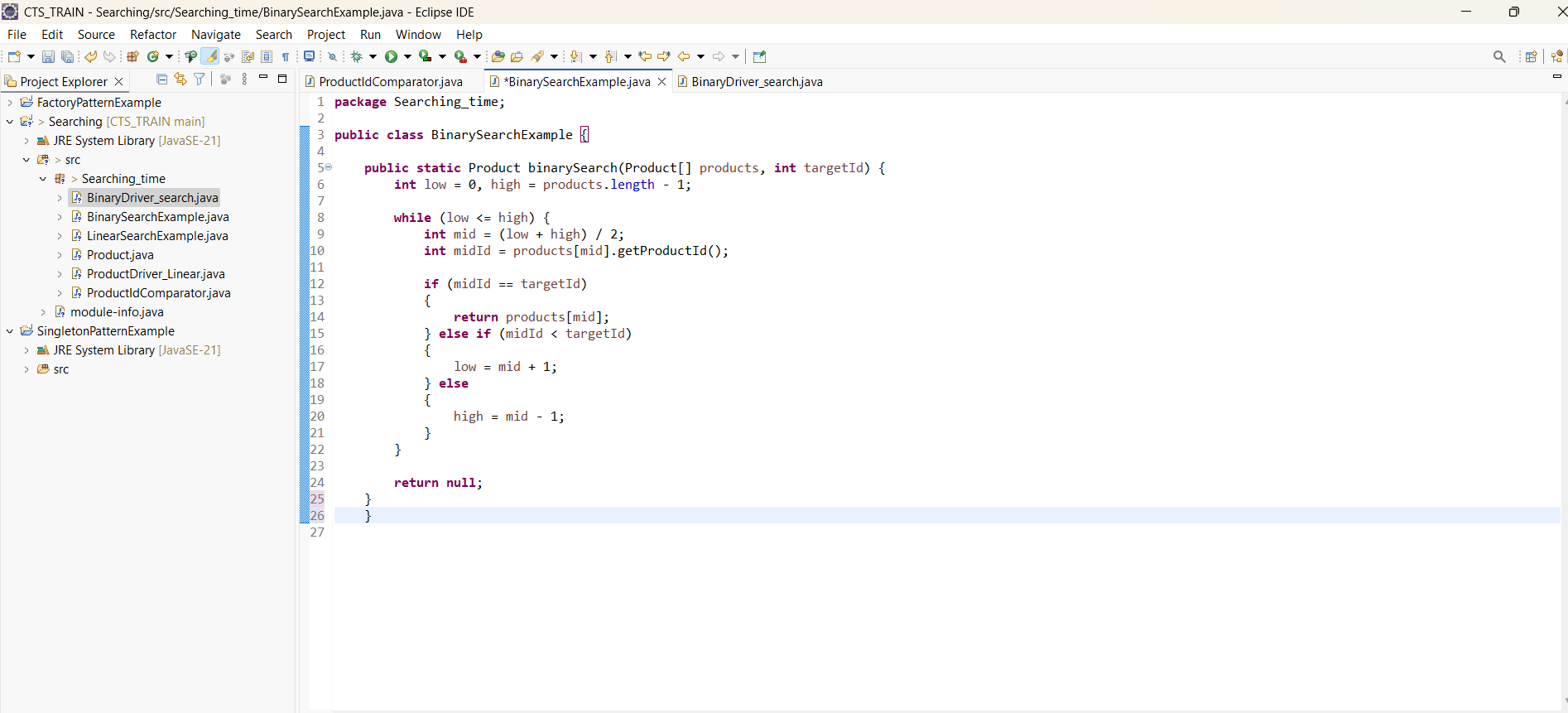
**OUTPUT:[LINEAR SEARCH]**

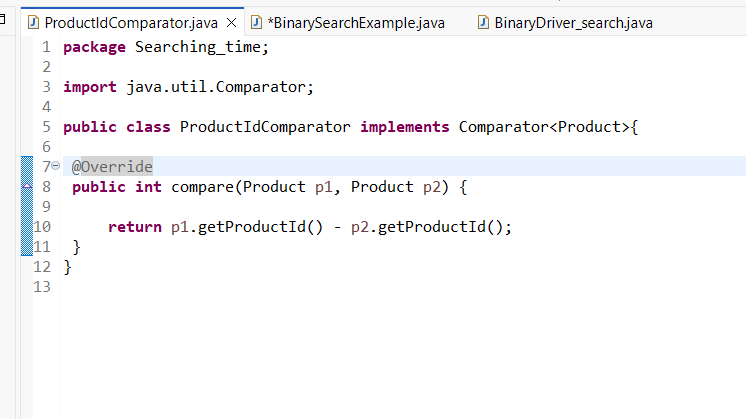
**Binary Search:**

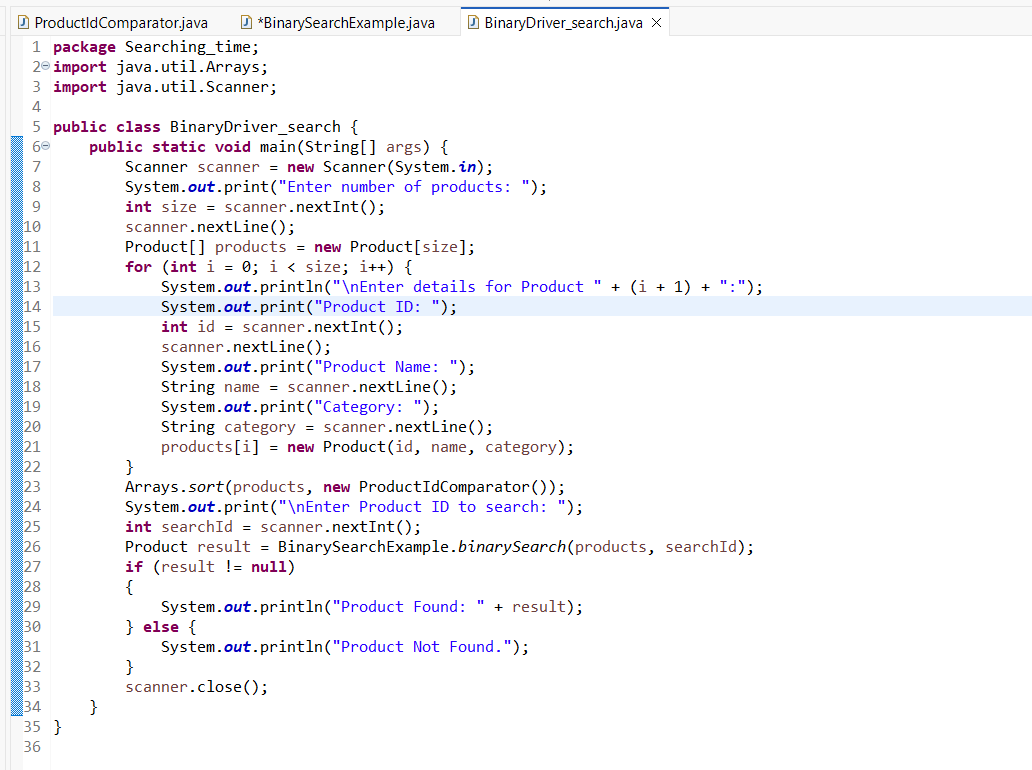
**Product.java**



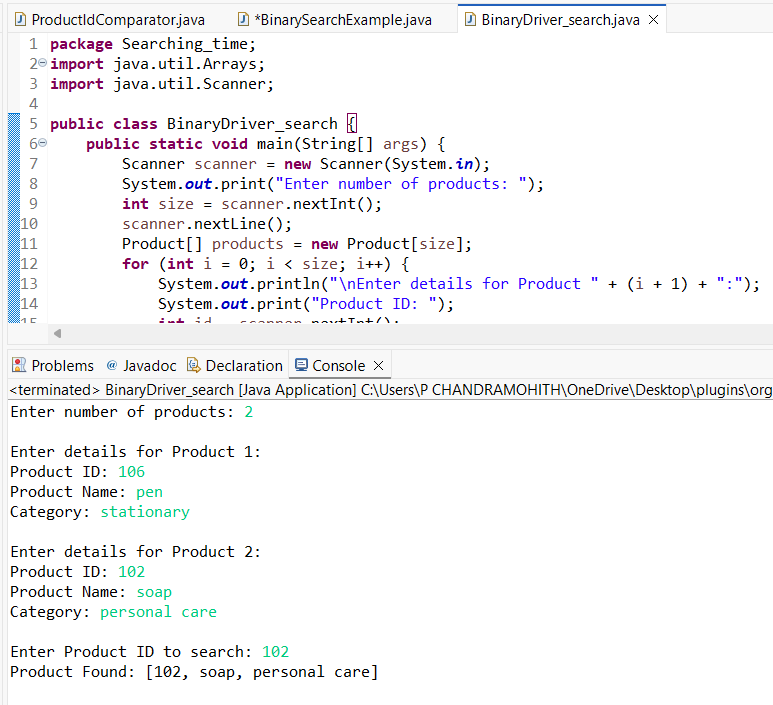
**BinarySearchExample.java**

****

**ProductIdComparator.java**

**BinaryDriver\_search.java**

**OUTPUT:[binary]**

****

**4.Analysis:**

**Time Complexity Comparison**

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | **O(1)** | **O(n)** | **O(n)** |
| **Binary Search** | **O(1)** | **O(log n)** | **O(log n)** |

**Linear Search – Simple but Slow for Large Data**

* How it works: It checks each item in the list one by one until it finds the match (or not).
* Best case: If the item is at the very beginning → super fast (O(1)).
* Worst case: If the item is at the end or not there → checks everything → O(n) time.

**When to use:**

* + The list is small
  + The list is unsorted
  + You want to avoid sorting overhead

**Binary Search – Fast but Needs Sorted Data**

* How it works: It divides the list in half each time, reducing the search space quickly.
* Always faster than linear search on large datasets, because it’s logarithmic in time.
* But: You must sort the data first (which takes time too).

**When to use:**

* + You already have sorted data
  + You're working with large datasets
  + You want fast searching repeatedly

**Which One Is Better for Your Platform?**

**It depends on the context:**

* If you're working on a student-level project, or your dataset is small or dynamic, Linear Search is easier and doesn’t need extra sorting.
* If you're building an app that handles thousands of records, and searching happens often (like in product catalogs, logs, etc.), then Binary Search is the better choice — but make sure your data is sorted before using it.