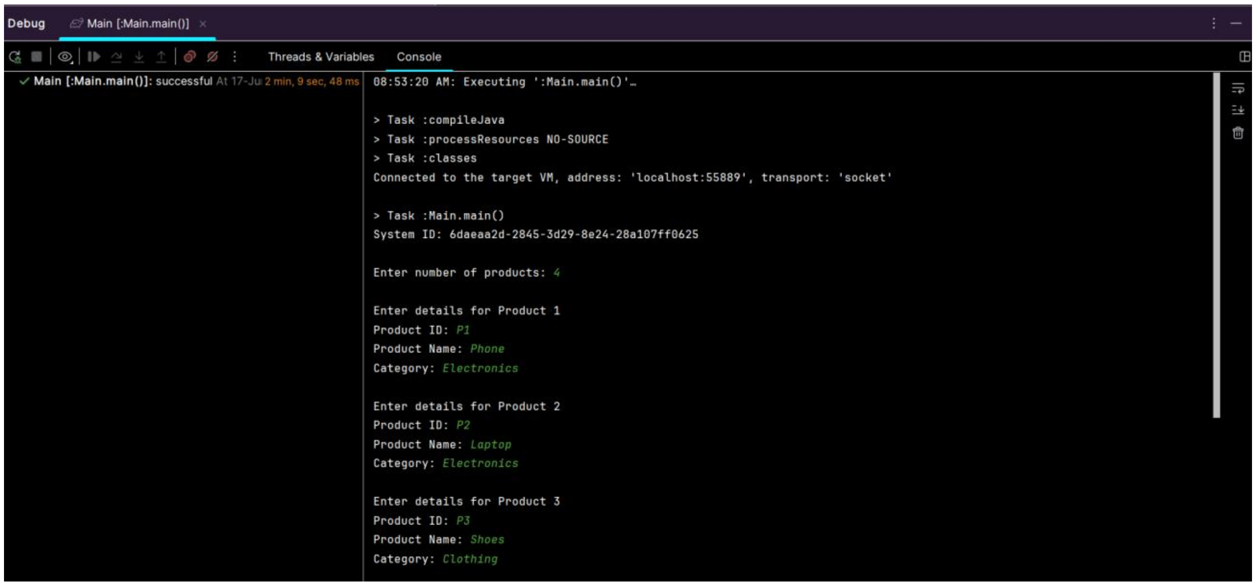
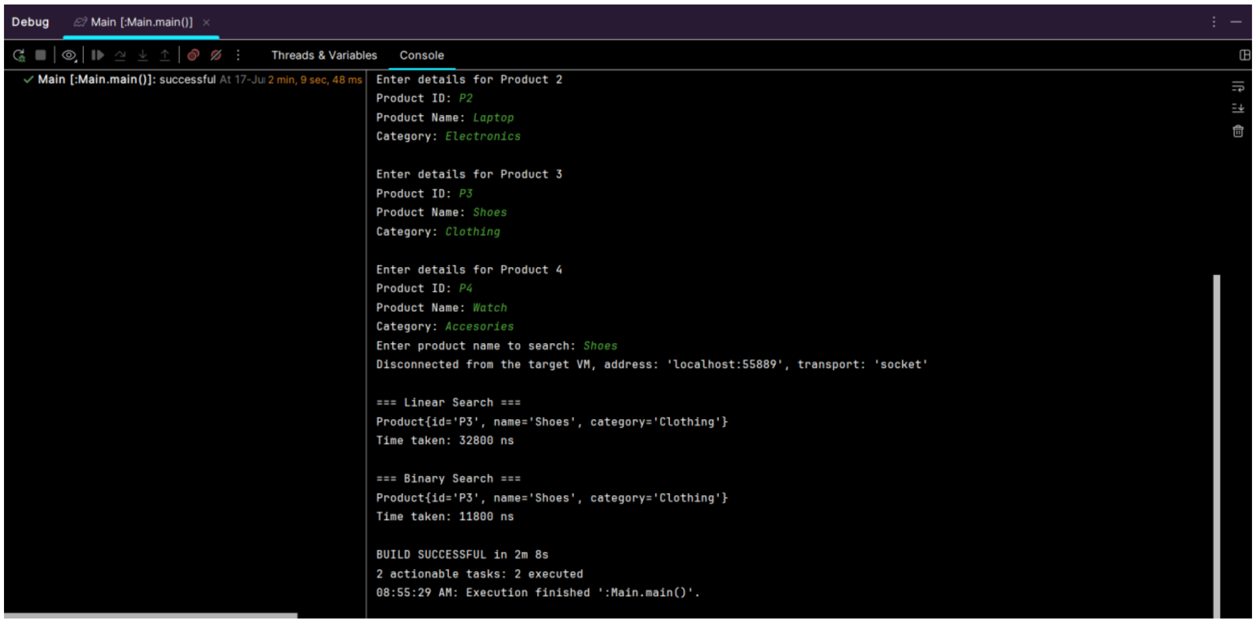
File by Sayan Saha (sayansaha00876@gmail.com)

Exercise 2: E-commerce Platform Search Function:

Output Screenshots :-


Debug
Main [:Main.main()]
Threads & Variables
Console
Main [:Main.main()]: successful At 17-Ju 2 min, 9 sec, 48 ms

```
08:53:20 AM: Executing ':Main.main()'…

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
Connected to the target VM, address: 'localhost:55889', transport: 'socket'

> Task :Main.main()
System ID: 6daeaa2d-2845-3d29-8e24-28a107ff0625

Enter number of products: 4

Enter details for Product 1
Product ID: P1
Product Name: Phone
Category: Electronics

Enter details for Product 2
Product ID: P2
Product Name: Laptop
Category: Electronics

Enter details for Product 3
Product ID: P3
Product Name: Shoes
Category: Clothing
```


Debug
Main [:Main.main()]
Threads & Variables
Console
Main [:Main.main()]: successful At 17-Ju 2 min, 9 sec, 48 ms

```
Enter details for Product 2
Product ID: P2
Product Name: Laptop
Category: Electronics

Enter details for Product 3
Product ID: P3
Product Name: Shoes
Category: Clothing

Enter details for Product 4
Product ID: P4
Product Name: Watch
Category: Accesories
Enter product name to search: Shoes
Disconnected from the target VM, address: 'localhost:55889', transport: 'socket'

=== Linear Search ===
Product{id='P3', name='Shoes', category='Clothing'}
Time taken: 32800 ns

=== Binary Search ===
Product{id='P3', name='Shoes', category='Clothing'}
Time taken: 11800 ns

BUILD SUCCESSFUL in 2m 8s
2 actionable tasks: 2 executed
08:55:29 AM: Execution finished ':Main.main()'.
```

1. Understanding Asymptotic Notations :-

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

Big O notation is a mathematical concept used in computer science to describe the performance or complexity of an algorithm as a function of the input size. It gives us a way to classify algorithms according to how their running time or space requirements grow as the input size increases. Big O focuses on the worst-case scenario**,** helping developers understand the upper limit of an algorithm's behavior.

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

Best Case: This scenario occurs when the algorithm completes its task in the shortest possible time. For example, in a linear search, if the target is the first element**,** it completes in O(1) time.

Average Case: This considers the performance over many possible inputs. In a linear search, the average case assumes the target is somewhere in the middle, leading to a time complexity of O(n/2)**,** which simplifies to O(n).

Worst Case: This is the most time-consuming case, where the algorithm performs the maximum number of operations. For a linear search, this happens if the target is **l**ast or not present at all**,** resulting in O(n) time. For binary search, the worst case is when it has to reduce the search space down to a single element, still maintaining O(log n) complexity.

2. Setup

We defined a class Product with the following attributes for search oprations :

- String productId
- String productName
- String category

3. Implementation Approach

The entire coding implementation is given in the folder.

Linear Search

- Traverse each element in the array one by one.
- Compare target with the product name (or other fields).
- Return the product if matched.

Binary Search

- Requires the array to be sorted (e.g., by productName).
- Repeatedly divide the search interval in half:
    - If the target equals the middle element, return it.
    - If the target is less, search the left half.
    - Else, search the right half.

4. Analysis of Linear and Binary Search Algorithms

When comparing linear search and binary search for an e-commerce platform, it's essential to evaluate their efficiency in terms of time complexity and suitability for different use cases. Linear search is the simplest algorithm, scanning each product in the array one by one until a match is found. Its best-case performance occurs when the desired product is at the beginning of the list, resulting in O(1) time. However, in the average and worst-case scenarios, such as when the product is at the end or not present, the time complexity becomes O(n), making it inefficient for large inventories.

In contrast, binary search is a significantly more efficient algorithm with a time complexity of O(log n) in the best, average, and worst-case scenarios. This efficiency stems from its divide-and-conquer approach—repeatedly halving the search space. However, binary search requires the array to be sorted beforehand, which introduces additional complexity and overhead, especially if the product list is frequently updated.

From a practical standpoint, linear search is more appropriate for small or unsorted datasets or situations where search operations are infrequent. On the other hand, binary search is ideal for large, static datasets where search speed is a priority and the product list does not change frequently

In our case, we found that binary search is ideal as it is taking less time as shown in the output screenshot image.