**Cognizant Digital Nurture 4.0**

***WEEK-1 Module 2 - Data Structures and Algorithms***

**Exercise 1:**

**Inventory Management System** **(Mandatory)**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**>> Explain why data structures and algorithms are essential in handling large inventories.**

By using data structures and algorithms in handling large inventories, the operations like searching, adding, deleting, or updating the data would be easier and quicker. They can handle large number of datasets and ensure scalability and high performance.

**>> Discuss the types of data structures suitable for this problem.**

I can think of two type of data structures which will be suitable for this scenario:

1. **ArrayList:**

It can used to maintain the order of insertion of product but it is slower when doing operations like searching or deleting specific information.

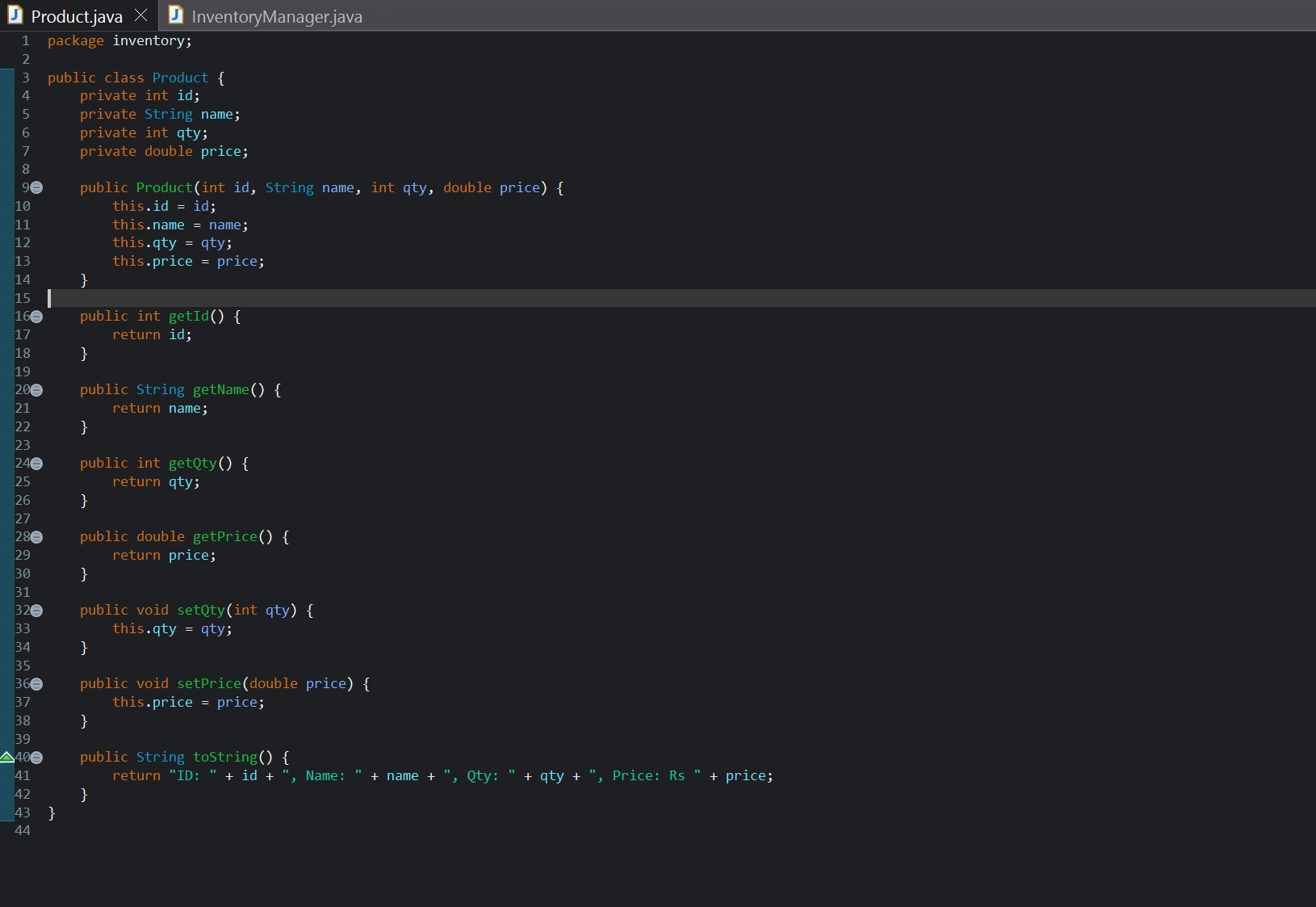
1. **HashMap:**

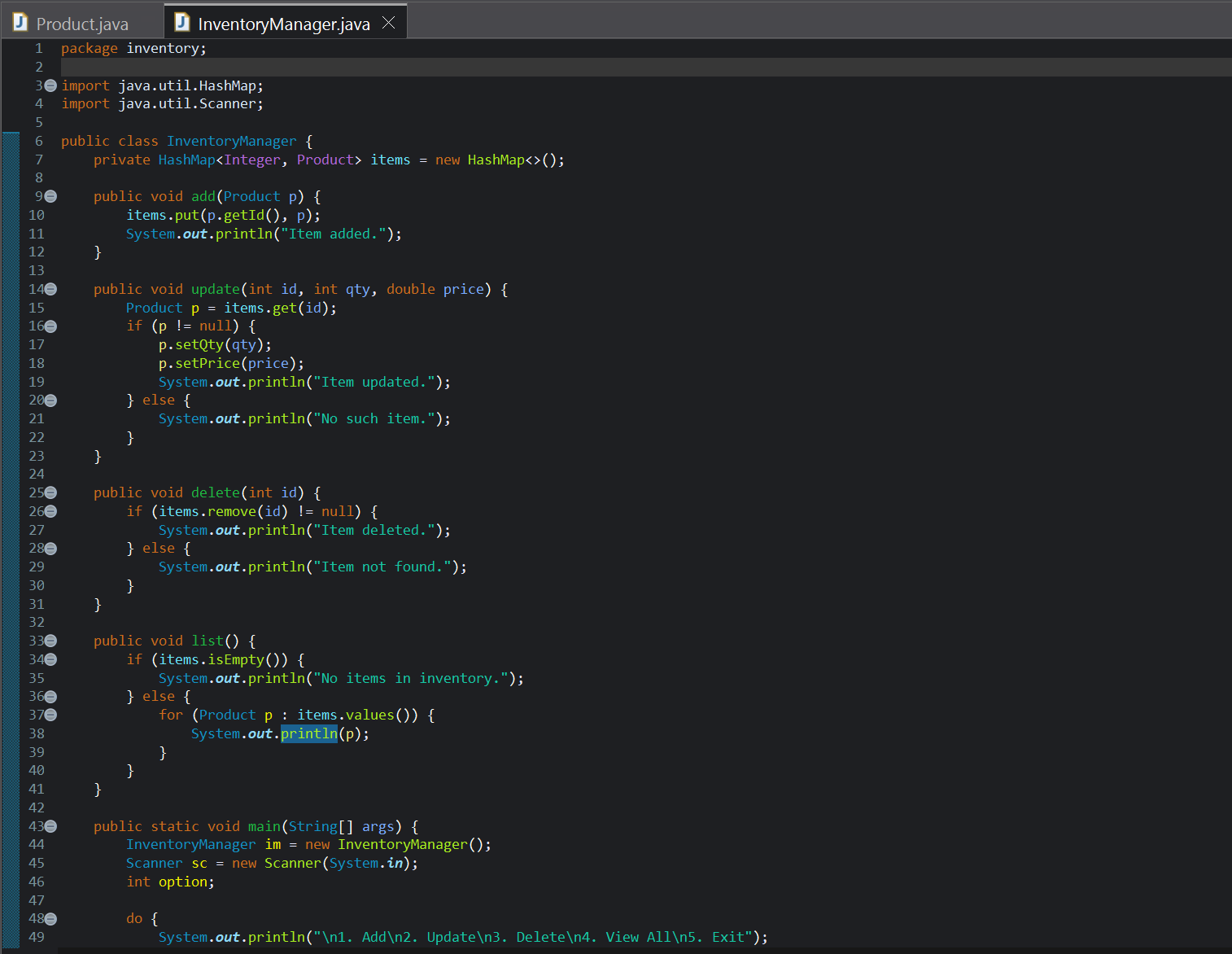
It is more effective when searching for an element using a unique key (like productId). It is most ideal for operations like lookup, update, and delete.

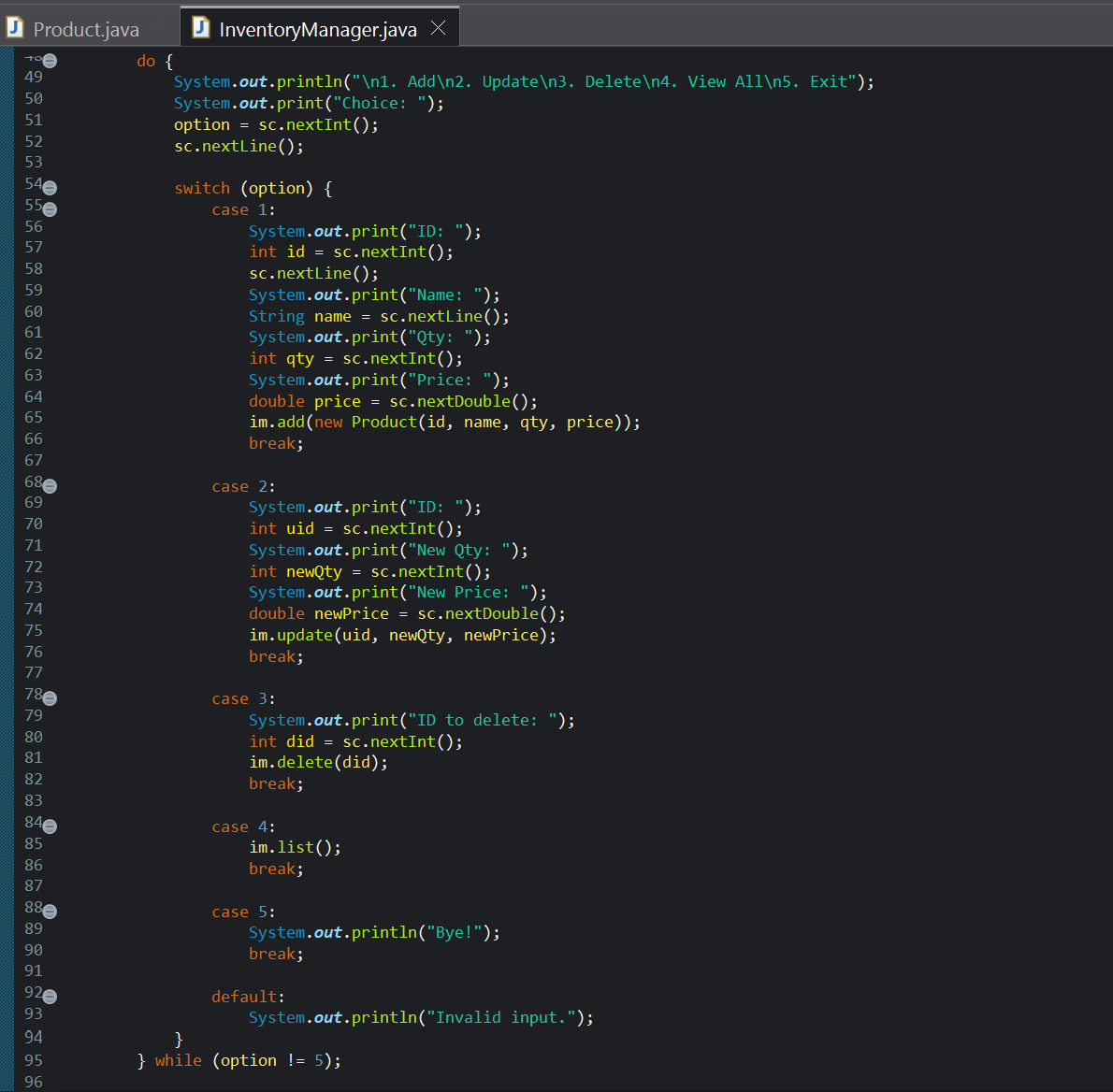
I created a class named Product with attributes like productId, productName, quantity, and price.

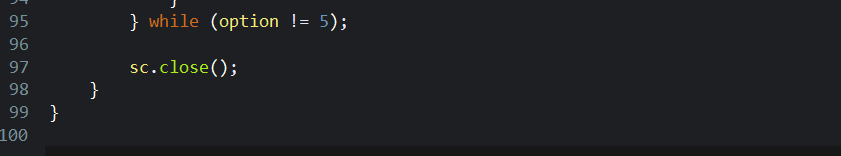
For this problem I will be using HashMap<Integer, Product> where productId will be the key.

I Implemented methods to add, update, and delete products from the inventory in class called

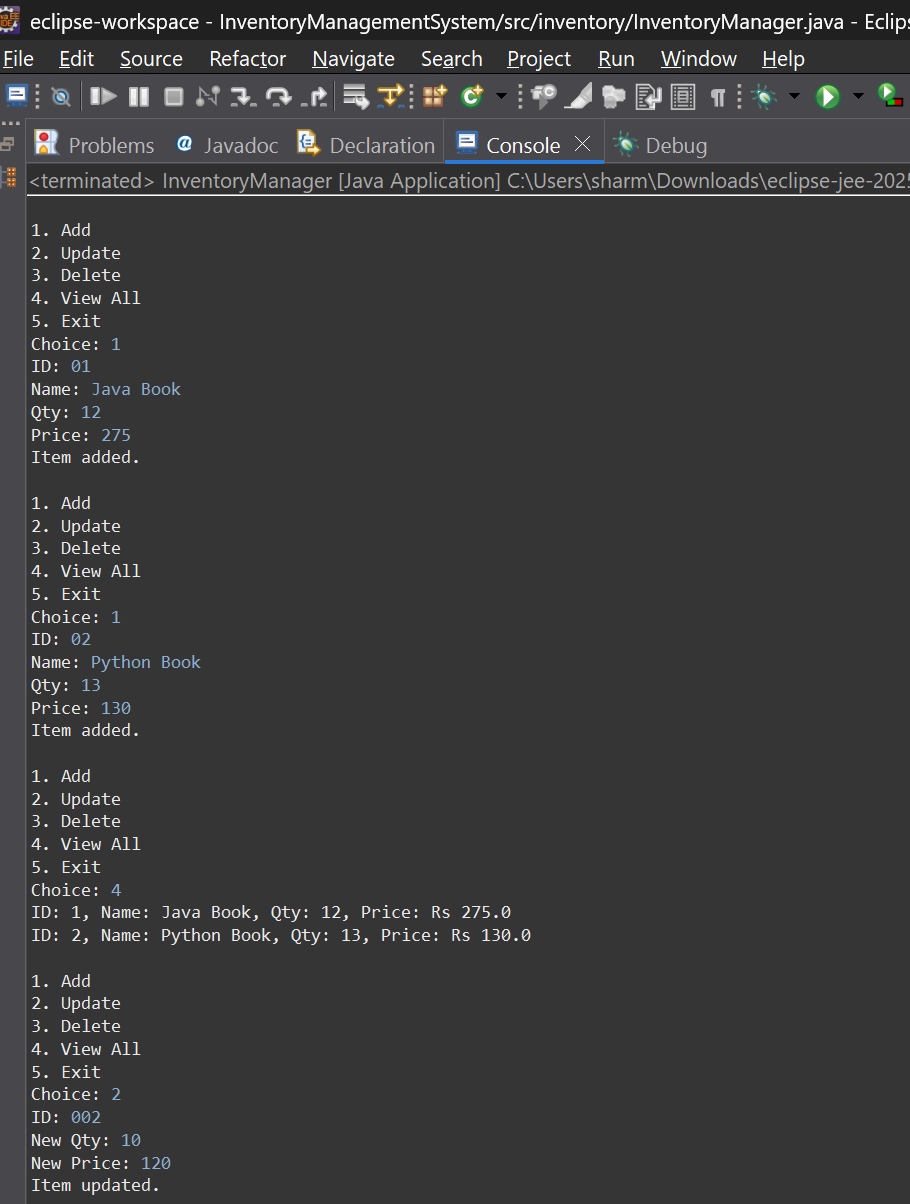


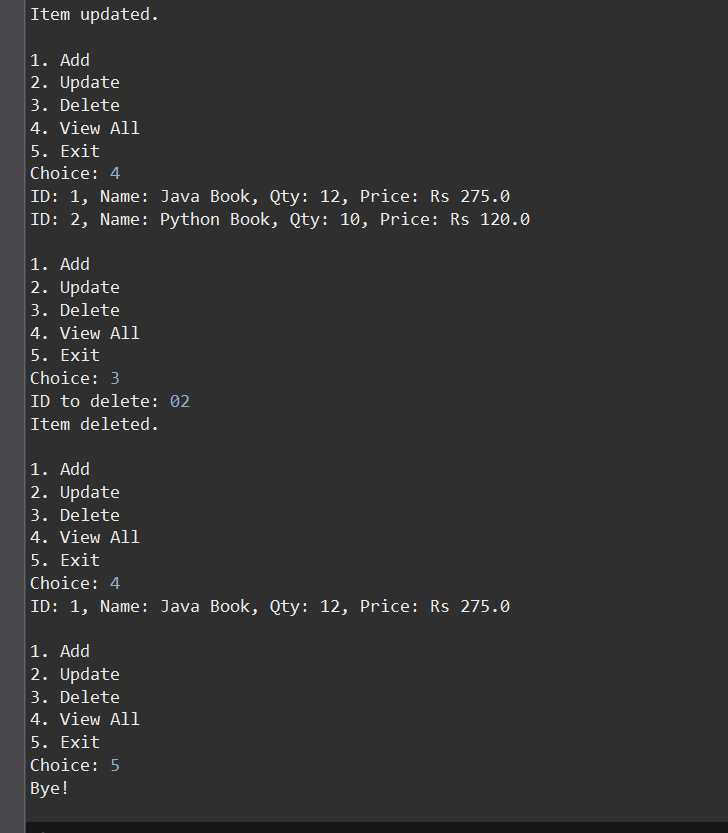






Output:





**Analysis of Time Complexity**

| **Operation** | **Data Structure** | **Time Complexity** | **Explanation** |
| --- | --- | --- | --- |
| Add Product | HashMap | O(1) average | Direct key-based insertion. |
| Update Product | HashMap | O(1) average | Direct access by productId. |
| Delete Product | HashMap | O(1) average | Direct removal by key. |
| Search Product | HashMap | O(1) average | Fast key lookup. |

**Discussion on optimization:**

* I need to avoid over writing to ensure unique productid
* If required, I will use additional Map<String, List<Product>> to search by name

**-------------------------------------------------------------------------------------------------------------**

**Exercise 2:**

**E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**>> What is Asymptotic Notation?**

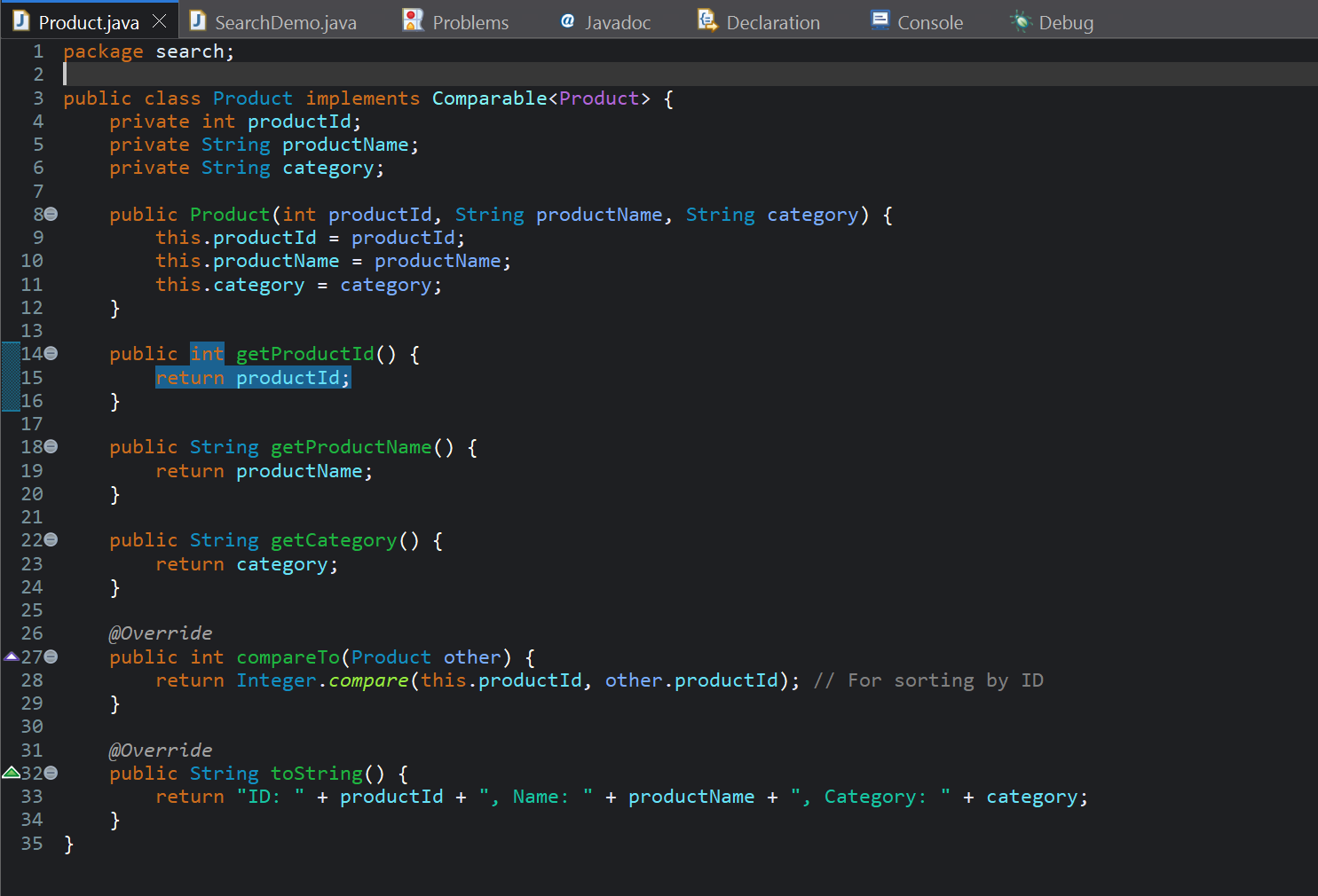
In programming, we use Big O notation to describe time complexity or space complexity of an algorithm in accordance to the input (n) taken. We can know how efficient an algorithm is when compared to others using this notation. In big O notation we ignore any existing contant and focus on worst cases. Other Notation include Big Omega (Ω), and Big Theta (Θ).

**>> The best, average, and worst-case scenarios for search operations:**

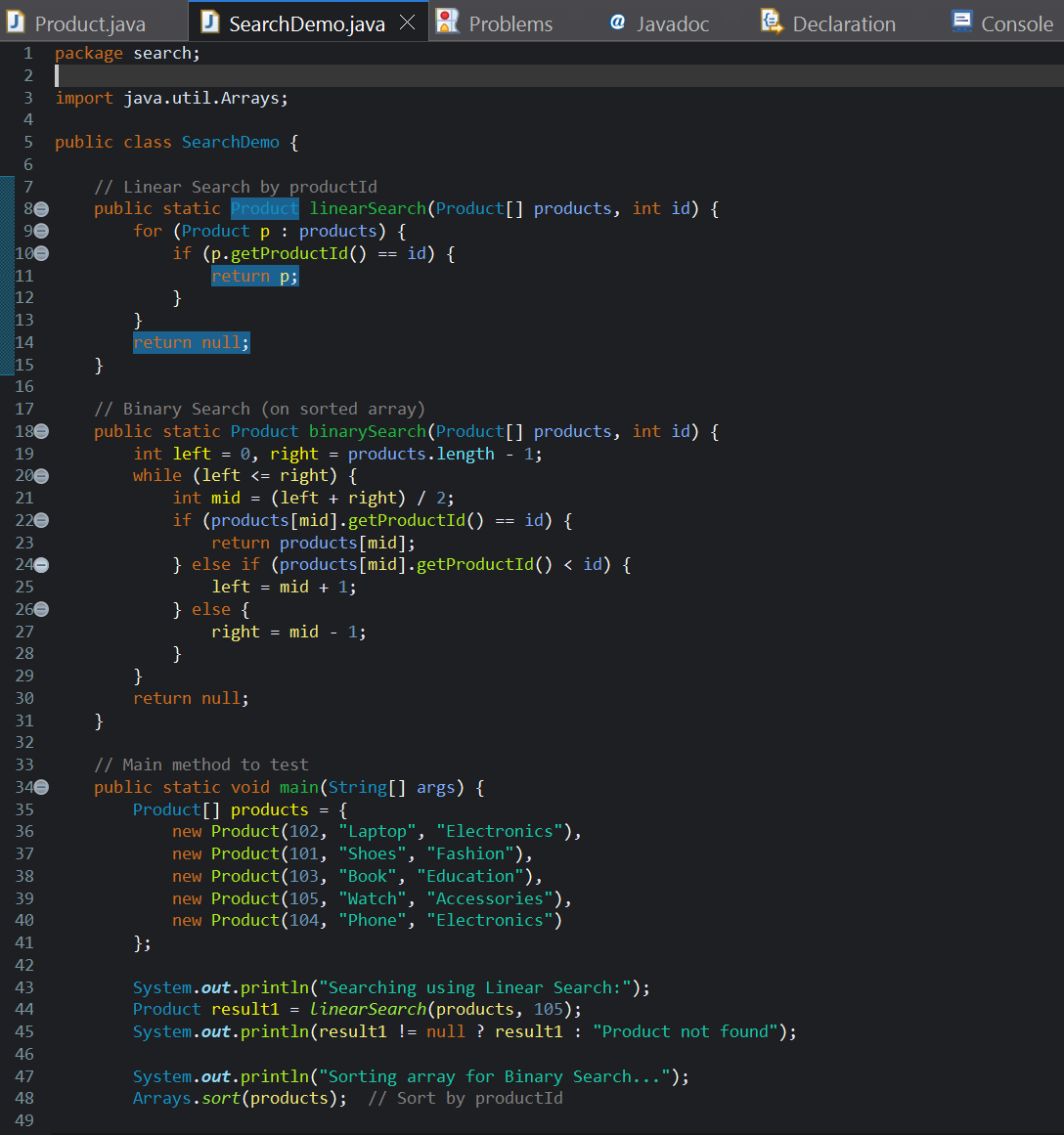
| **Case** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Best** | O(1) – first element | O(1) – middle is match |
| **Average** | O(n/2) → O(n) | O(log n) |
| **Worst** | O(n) | O(log n) |

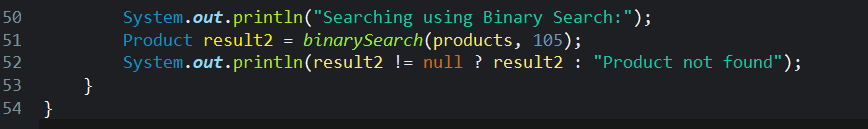
For this task I created two class: one is to hold product data - Product.java, other is to implement search algorithms and contain main method - SearchDemo.java

**Product.java:**

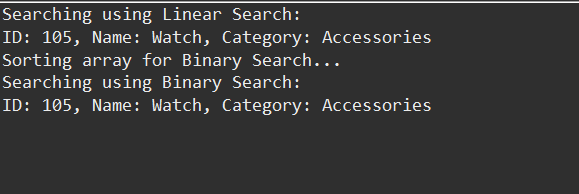


**SearchDemo.java:**





Output:



**Time Complexity Comparison**

| Algorithm | **Time Complexity** | **Sorted** |
| --- | --- | --- |
| Linear Search | O(n) | No |
| Binary Search | O(log n) | Yes |

We cannot make any assumption in a linear search, so we check every element. However, in binary search, we sort the data and each time we search we will split the rang in half.

**>> Which Algorithm is Better for E-commerce?**

When we are dealing with small data sets or any in case of any unsorted data input, linear search algorithm is acceptable.

However in real life scenario where thousands of products are present in a e-commerce platform, it is preferable to use binary search because:

* We can sort the products with unique IDs and Names
* The search operation is frequent and thus fast.

**Exercise 3**

**Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

Different types of sorting algorithms

1. **Bubble Sort:**

In this sorting algorithm, we will swap the elements repeatedly when  they’re in the wrong order. It has the time Complexity of O(n²). it is effective when the data set is Small or almost-sorted. It is used when simplicity is more important that performance.

1. **Insertion Sort:**

In this sorting algorithm, we will build the sorted list one item at a time. It has time complexity of O(n²). it is also effective only when the list size is small and the data set is almost sorted.

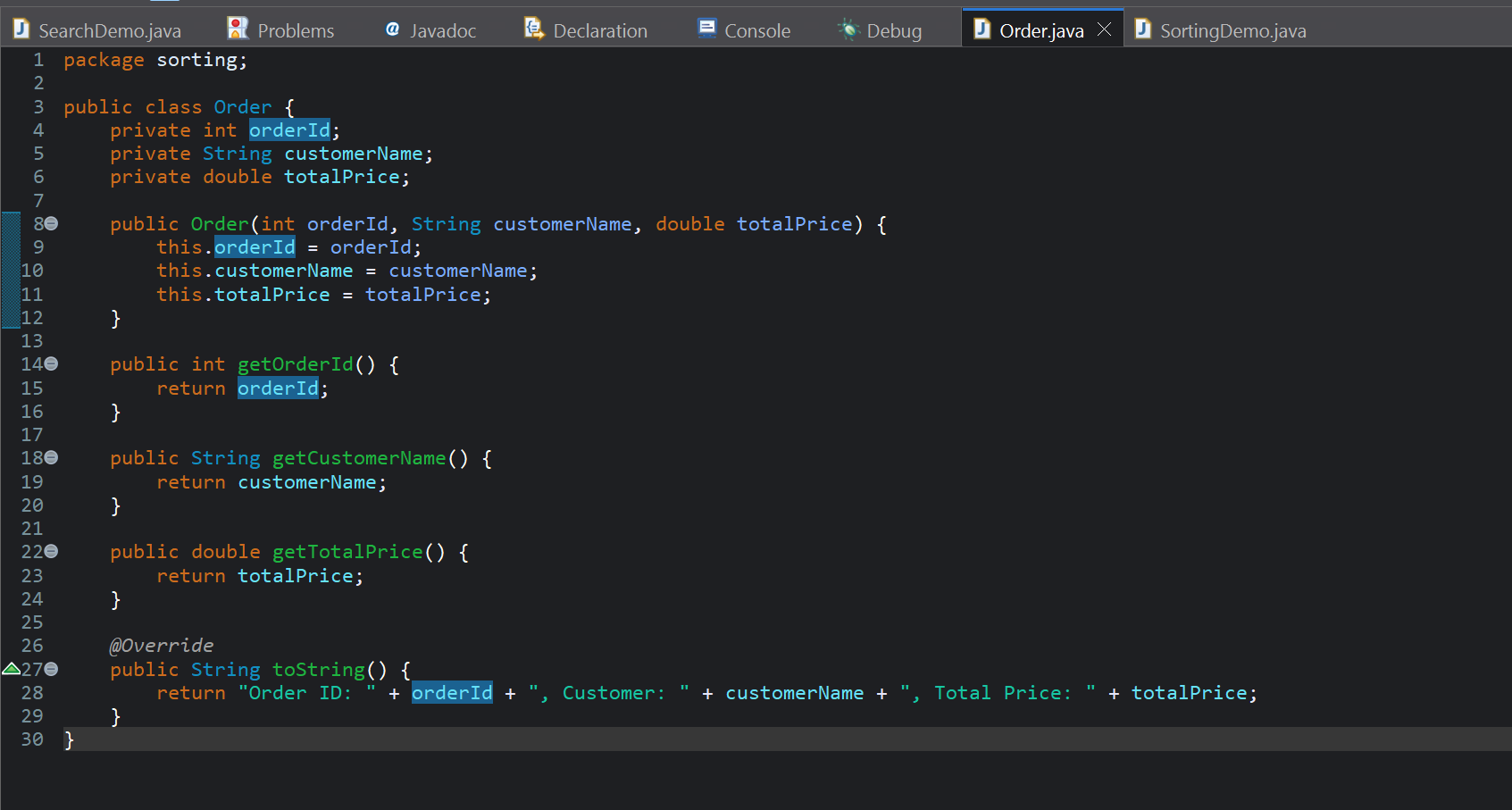
1. **Merge Sort:**

When the dataset is too large to fit into the memory, we can use merge sort. Merge sort is useful in such cases as it will split the array recursively, sorts them and merges them. It has a guaranteed time complexity of O(n log n).

1. **Quick Sort:**

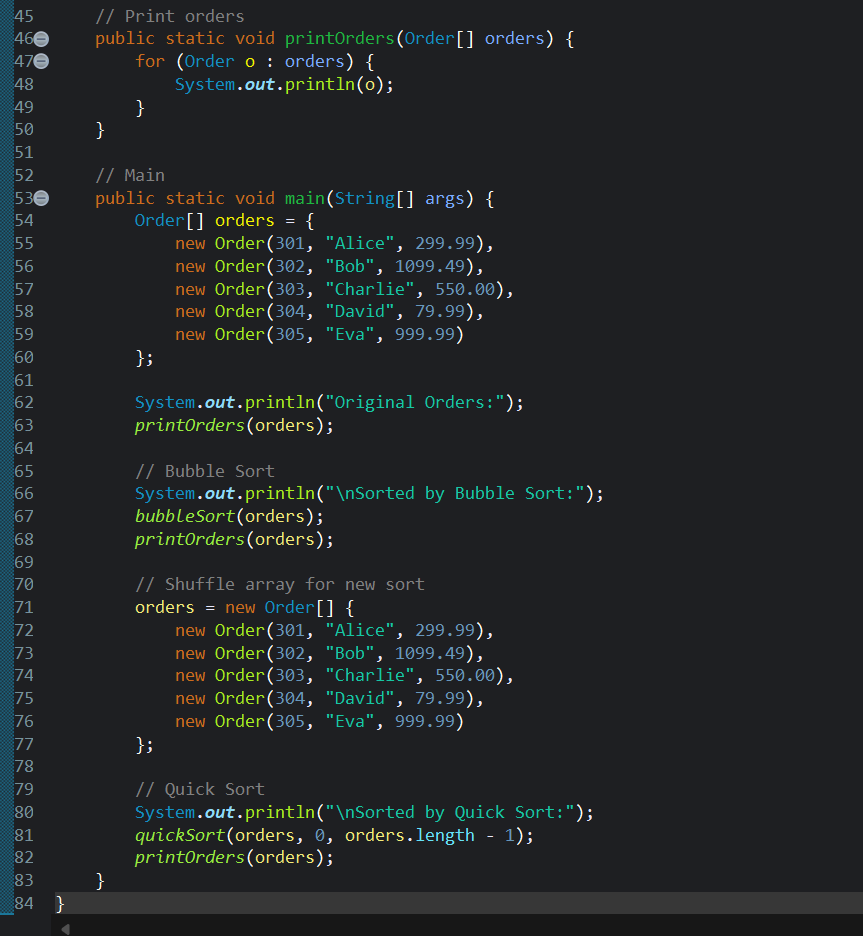
Quick sort is most popular among all these sorting algorithm as it has high performance on average. It is used when the whole sorting data can fit in the computer memory. It shows perfect locality of data reference and hence it can work better in scenarios where the data is required to accessed in sequence. It has time complexity of O(n log n) in best and average cases, in worst cases it has TC of O(n2) .

For this exercise, we have two class

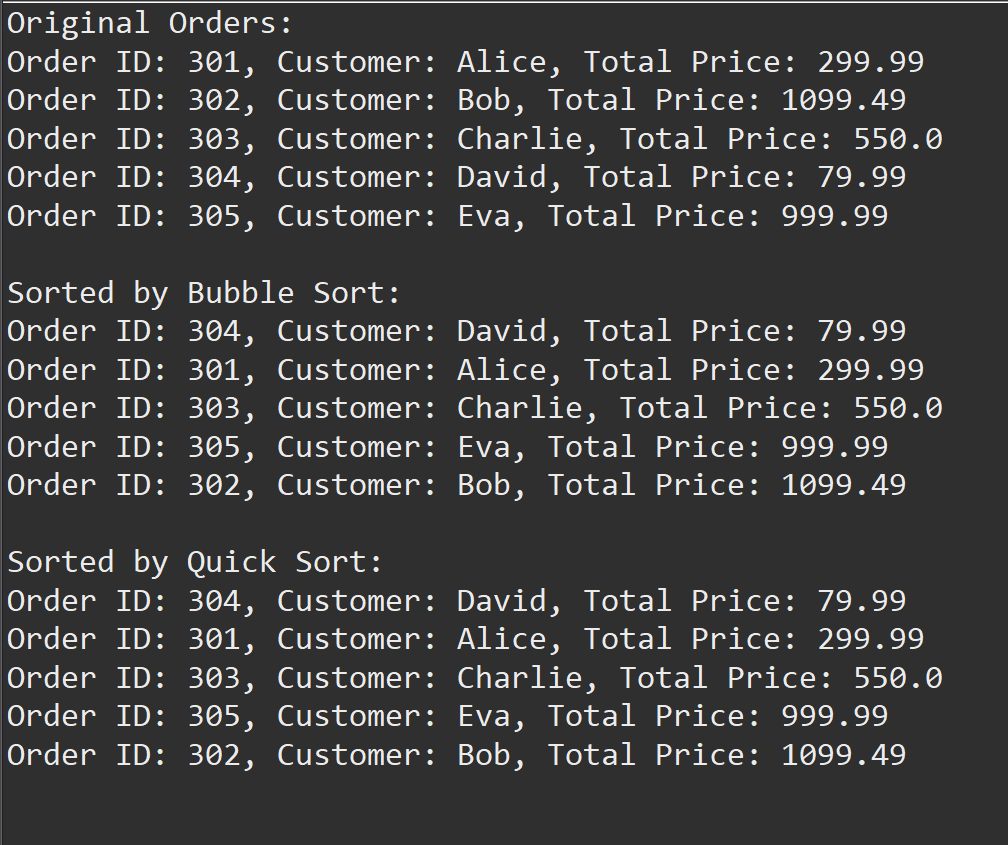
Order.java – to store the data  
  


SortingDemo to implement sorting of elements and test the logic.





Output:



**Time Complexity Comparison**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Bubble Sort** | O(n) | O(n²) | O(n²) |
| **Quick Sort** | O(n log n) | O(n log n) | O(n²) (rare) |

**Why Quick Sort is generally preferred over Bubble Sort?**

Quick Sort is a divide and conquer algorithm which makes it fast and effective when compared to bubble sort. Bubble sort is inefficient in case of large datasets. In addition, Quick sort uses less memory when compared to merge sort.

**Exercise 4**

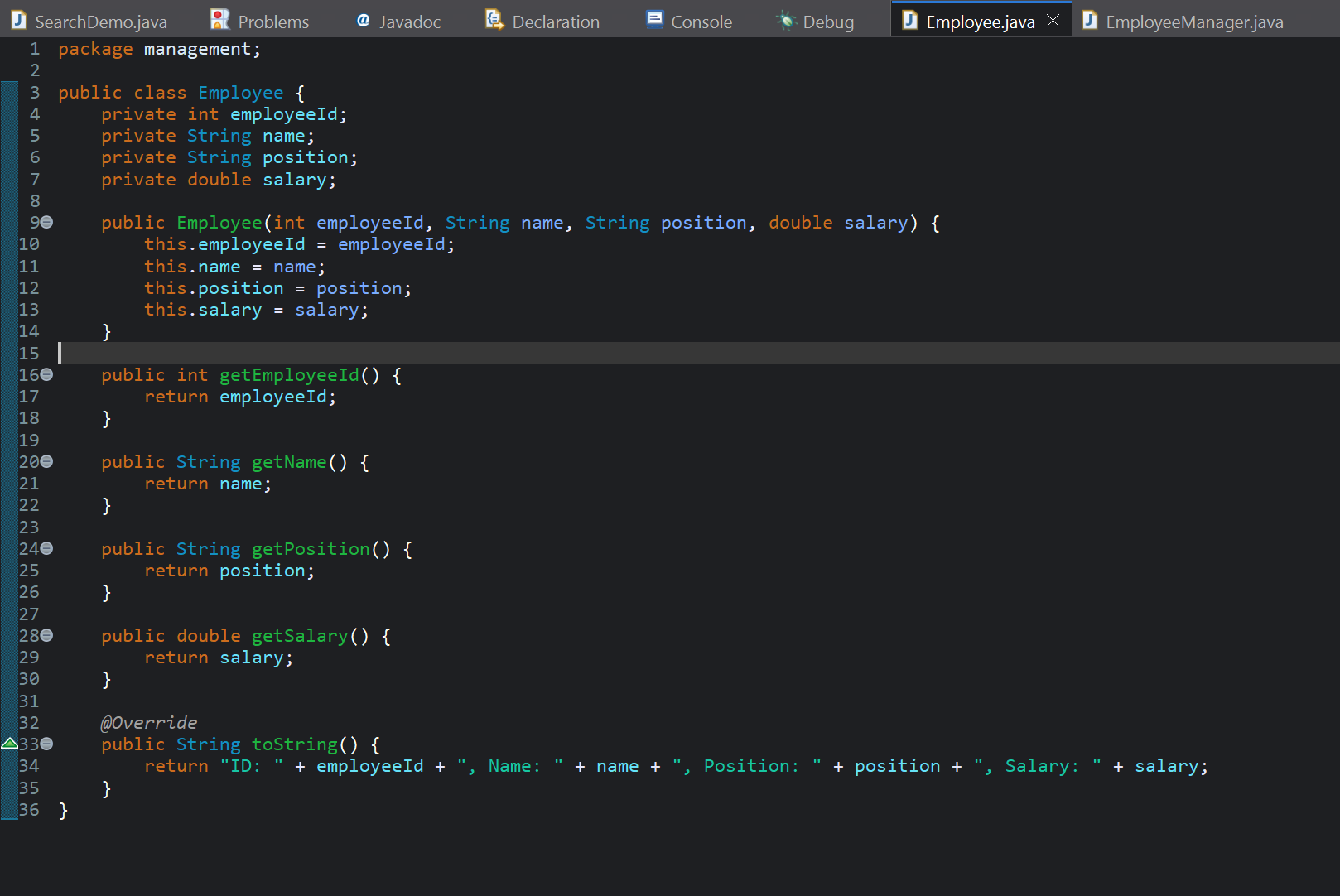
**Employee Management System**

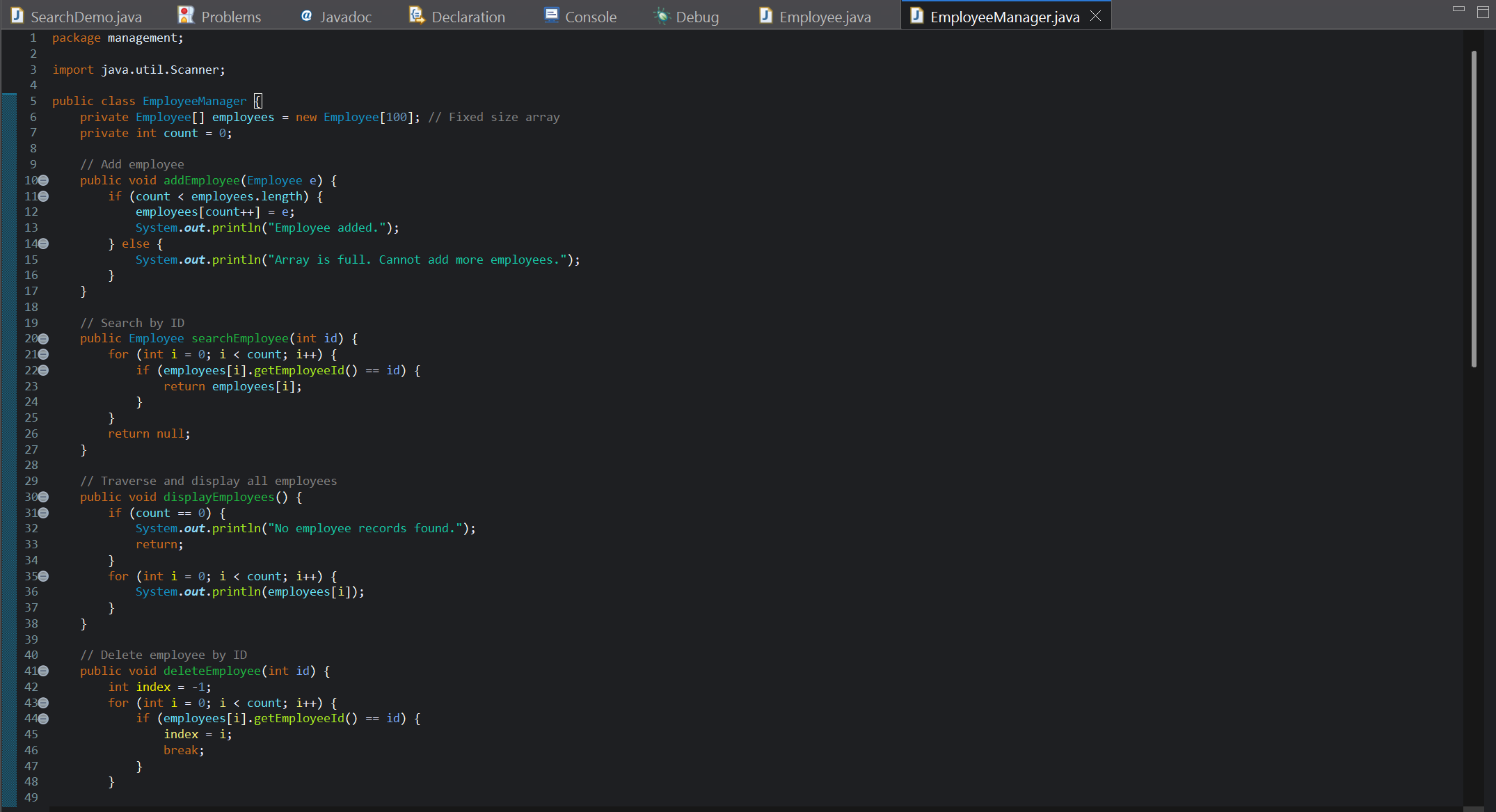
**Scenario:** You are developing an employee management system for a company. Efficiently managing employee records is crucial.

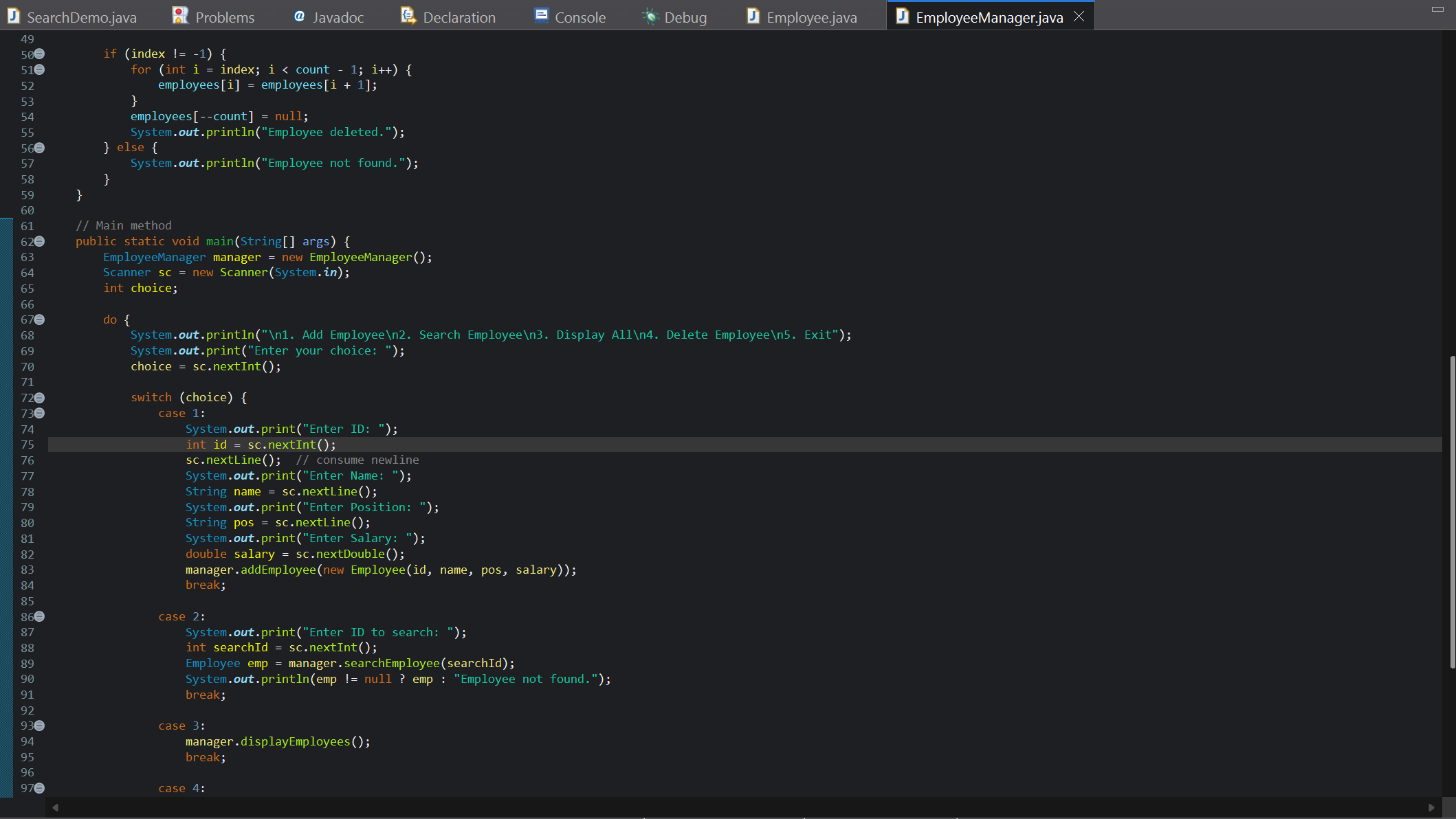
How Arrays Are Represented in Memory?

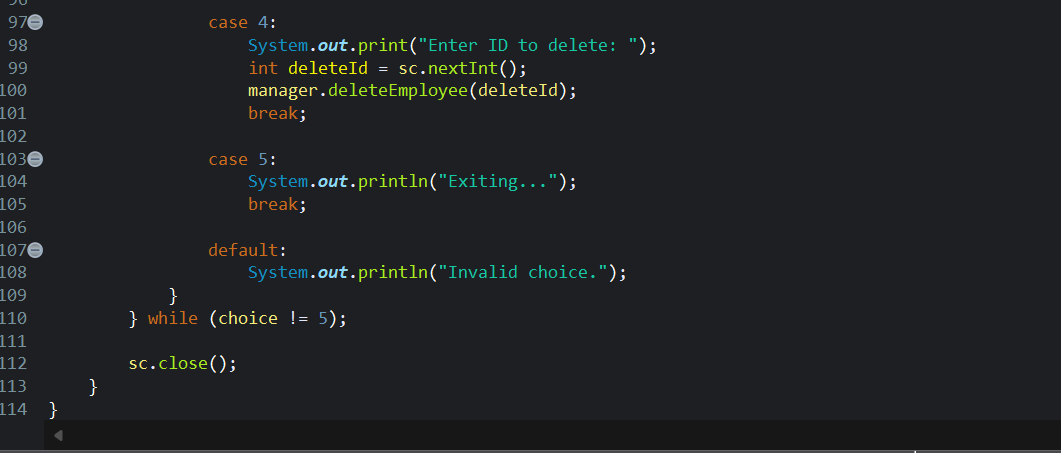
Array is a data structure which store the elements in a continuous block of memory which can accessed by index. In Java programming, the arrays can store element of same type and once they are declared, the size cannot be changed, it is fixed. Time complexity of accessing elements is O(1) in best cases (using indices). Array are easy to implement and understand. For fixed sized data, array are efficient in terms of memory and performance.

For this excerise I made two classes. One is Employee.java to store data and EmployeeManager.java for implementing logic and main() method.

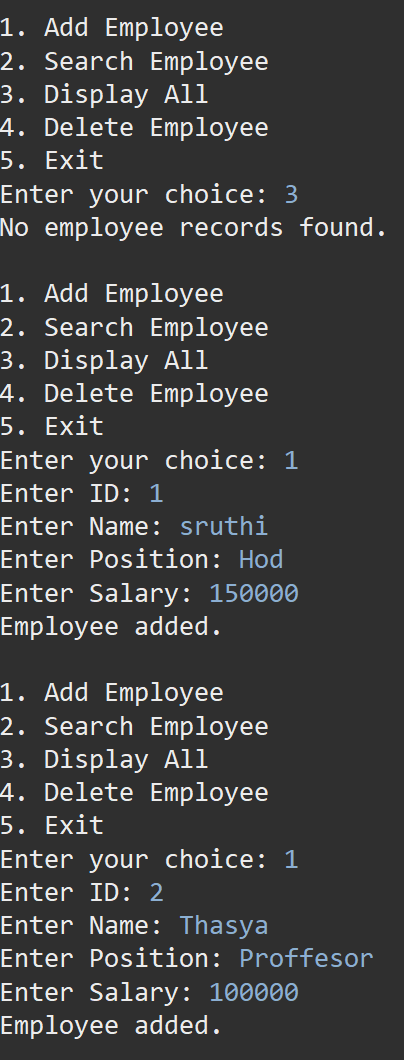


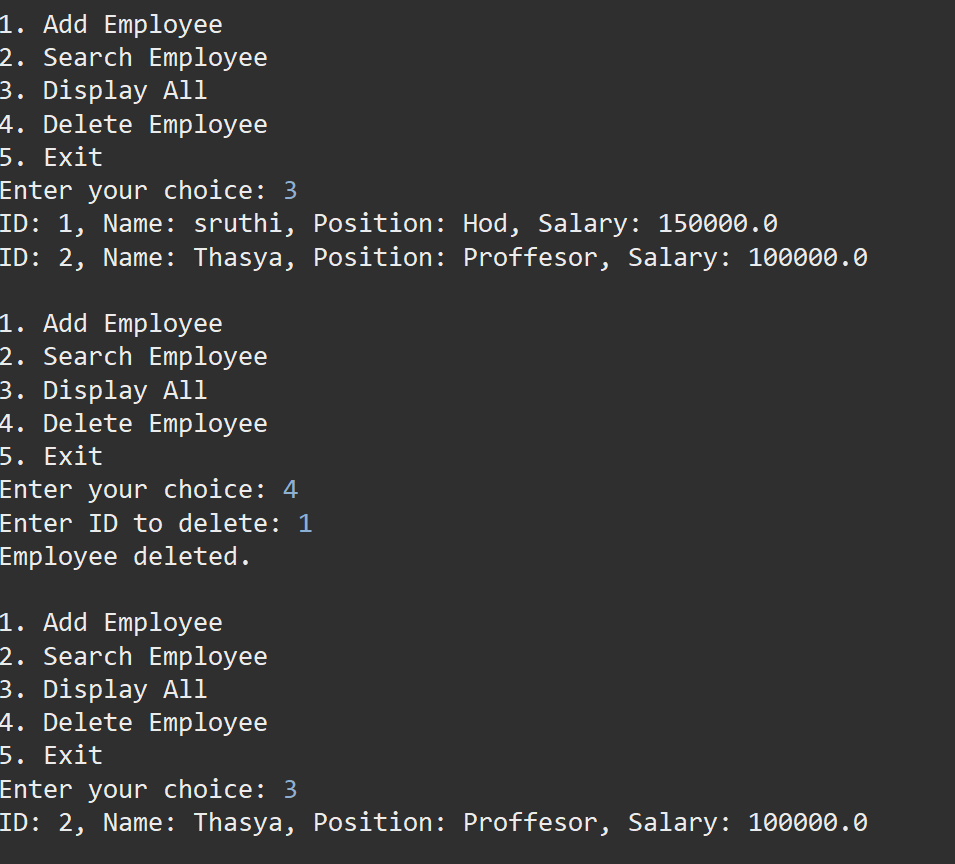


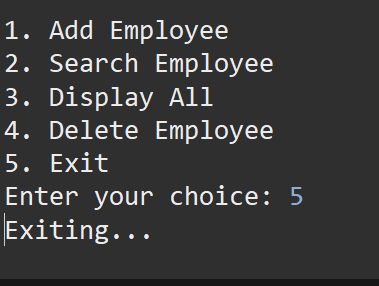




Output:







**Time complexity of different operations:**

| **Operation** | **Time Complexity** | **Reason** |
| --- | --- | --- |
| Add Employee | O(1) | Add at end of array using index. |
| Search Employee | O(n) | May need to check all elements. |
| Traverse Array | O(n) | Must visit each employee. |
| Delete Employee | O(n) | Search + shifting elements after deletion. |

**Limitation of Arrays:**

* Normal Arrays cannot dynamically grow, we need to use a special framework called “ArrayList”.
* During delete operations, we need shift elements manually.
* In absence of sorted lists, Arrays have slow search rate

**When to Use Arrays:**

* When we know the number of elements and they are fixed.
* When we need index-based access and the layout of memory is important.
* When we need to make simple applications or for any type of foundational learning.

**Exercise 5**

**Task Management System**

**Scenario**: You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**What is a Linked List?**

A data structure in which the elements (in form of nodes) are stored in non – continuous locations in system memory is called Linked List. In Linked lists, each node points to the next consecutive node (element).

There are two types of Linked List:

1. **Singly linked list:**

In this Linked list, each node contains data. Every node a pointer which points towards the next consecutive node.

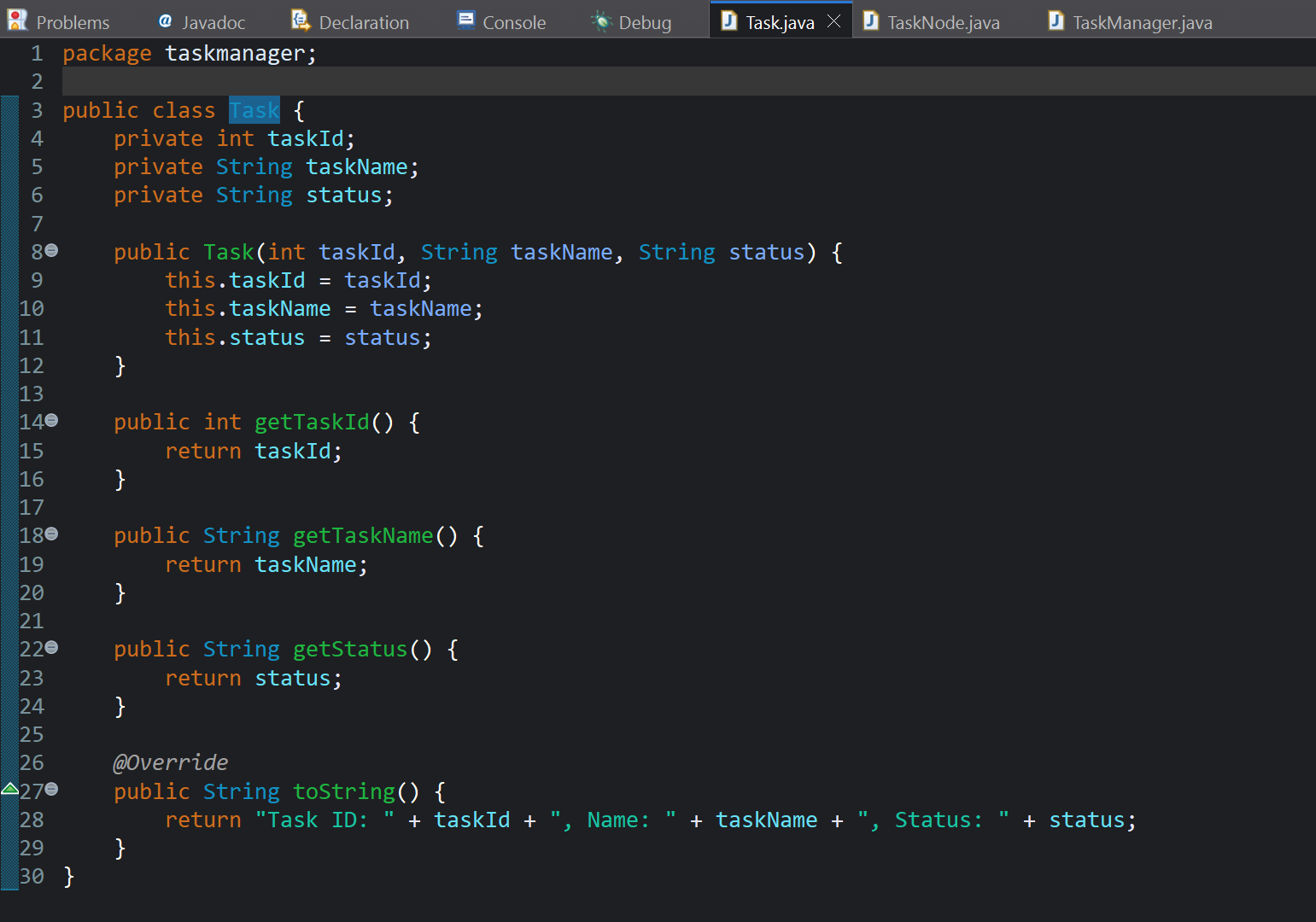
1. **Doubly Linked List:**

In the Doubly Linked List, a node along with data, contains two pointers. One of them point the previous node and the other points to the next node. This takes up more memory compared to singly linked list but it makes easier traversal.

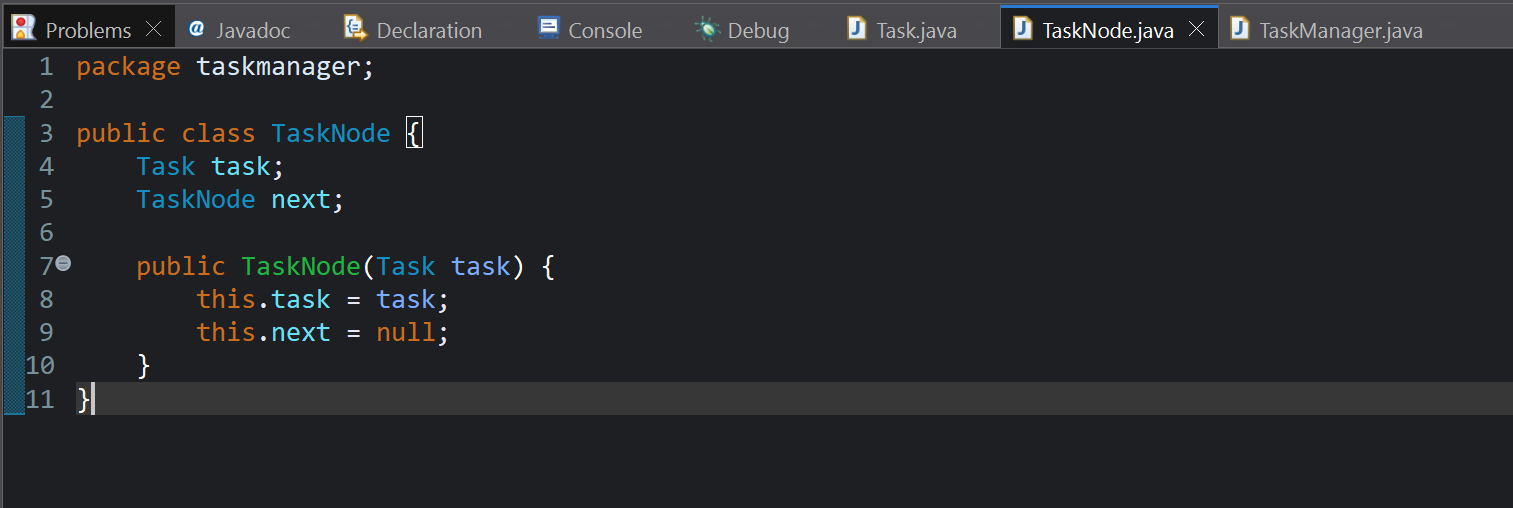
For this exercise, I am using singly linked list as it simple and efficient. I made three classes as following:

* 1. **Task.java**: to store data
  2. **TaskNode.java**: to make node required in the linked list
  3. **TaskManager.java**: it contains the main method and other required logic and testing.

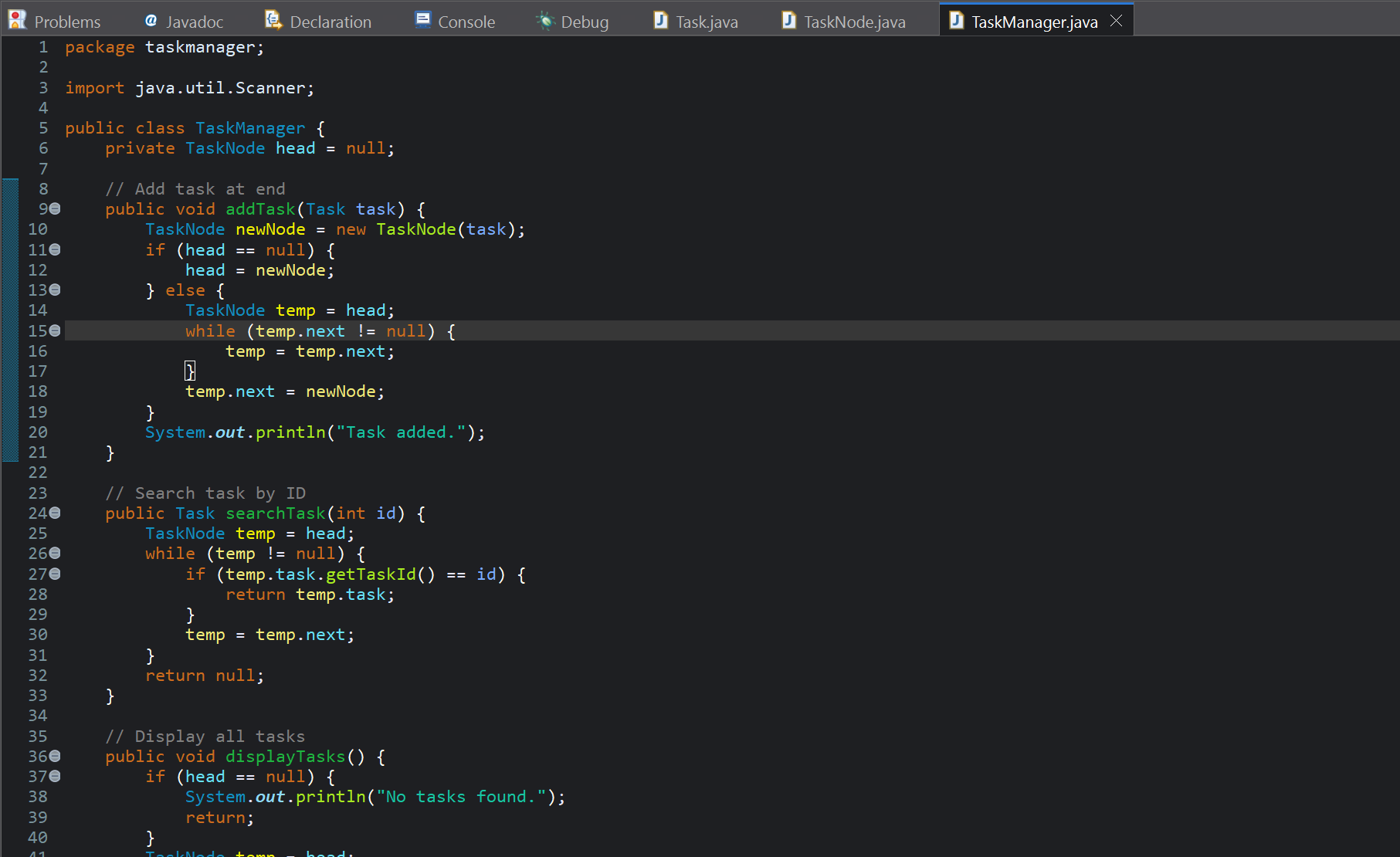
**Task.java**

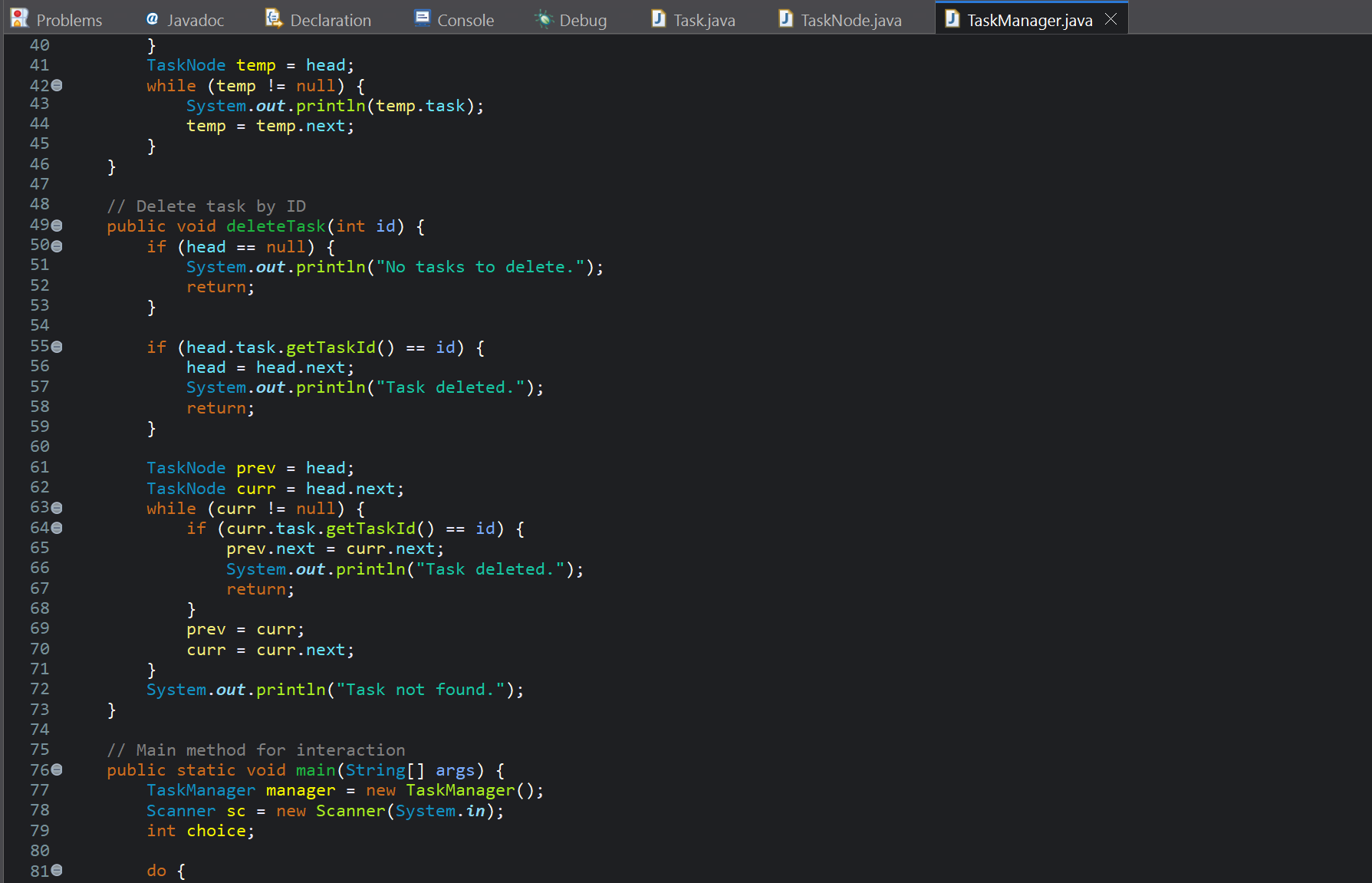
****

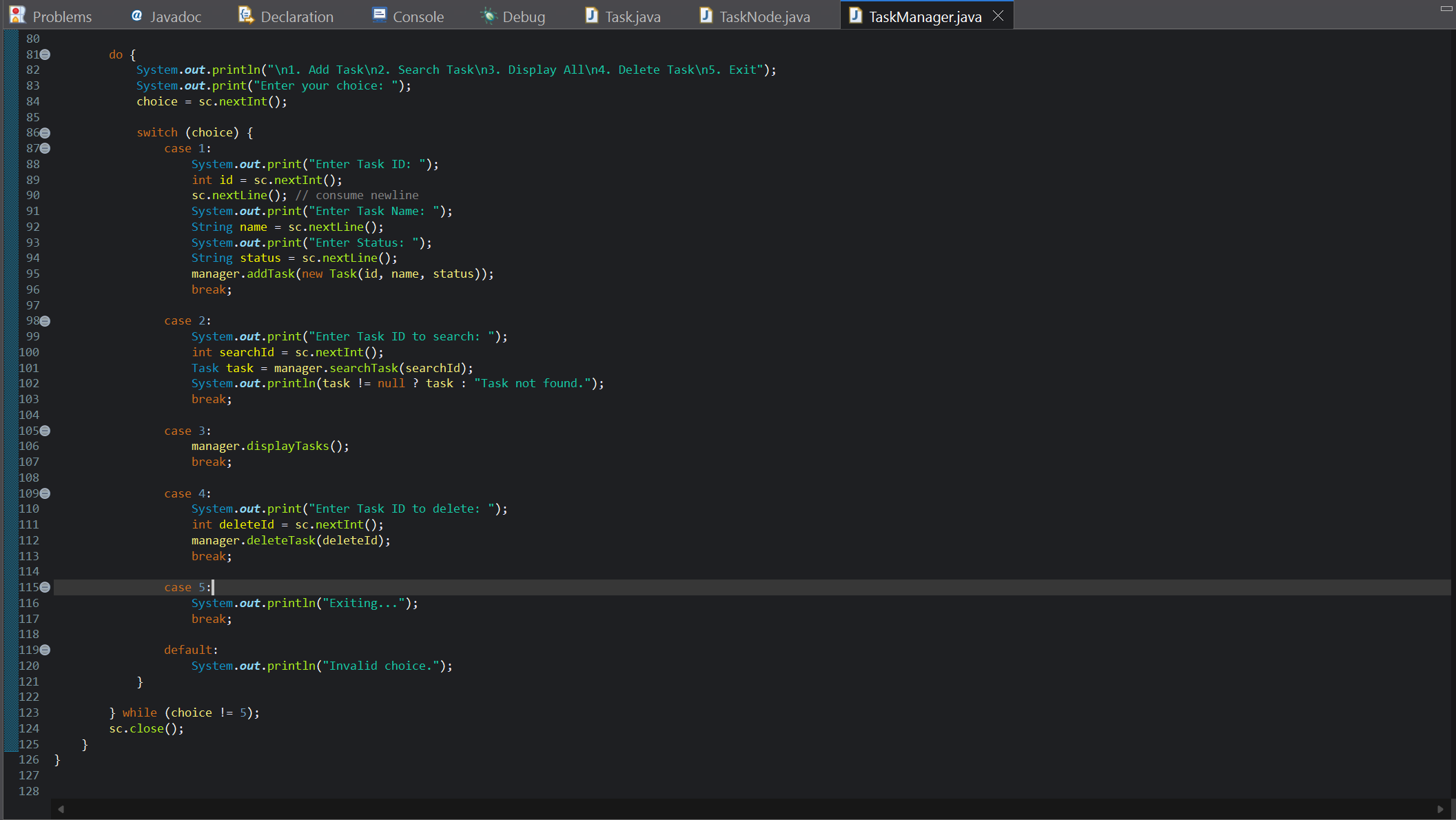
**TaskNode.java**

****

**TaskManager.java**







**Time Complexity of Operations**

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| **Add Task (at end)** | O(n) | Traverse to the end. |
| **Search Task** | O(n) | Linear search by ID. |
| **Traverse Tasks** | O(n) | Visit each node once. |
| **Delete Task** | O(n) | Need to find and unlink node. |

**Advantages of Linked Lists Over Arrays:**

| **Feature** | **Linked List** | **Array** |
| --- | --- | --- |
| **Dynamic size** | It grows dynamically | It is fixed once declared |
| **Insert/Delete at front** | TC: O(1) | TC: O(n)  It requires manual shifting of elements. |
| **Memory Usage** | It uses extra memory for pointers. | It uses Compact memory only. |
| **Access by index** | TC: O(n), when traversed | TC: O(1), when directly accessed |

Linked list are used when:

* We prioritize Frequent insertion and deletion operations.
* When the structure size needs to change dynamically.

**Exercise 7: Financial Forecasting (Mandatory)**

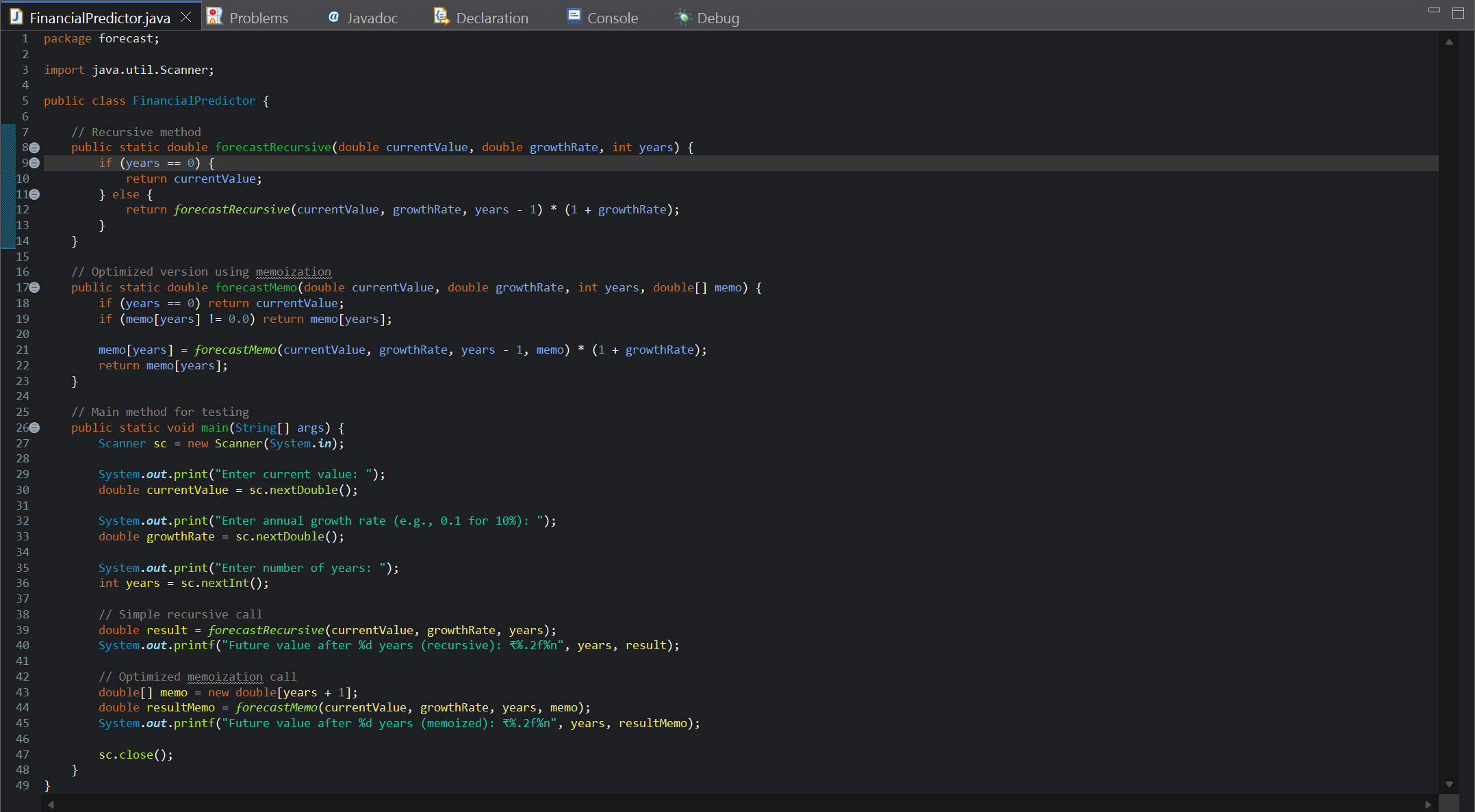
**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

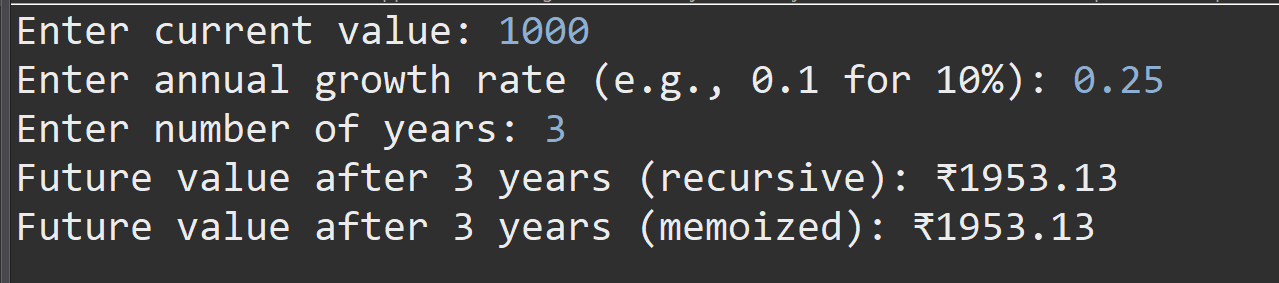
**Recursion:**

A programming technique in which the function calls itself repeatedly to solve a specific problem by dividing the whole problem into sub tasks is called Recursion. Recursion is used when we have repetitive sub structure in the problems like Factorial, Fibonacci or Financial Forecasting.

For this scenario, I created a class called FinancialPredictor.java which contains both recursive and memorization method (most effective).



Output:



**Time Complexity:**

| **Version** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| Simple Recursion | O(n) | O(n) |
| Memoization | O(n) | O(n) |

**How to optimize the recursive solution to avoid excessive computation:**

1. When the input is large, the recursive calls can be time taking and expensive.
2. In absence of memoization, the function computes the values repeatedly when called.