## **Exercise 4: Employee Management System**

## **How Arrays Are Represented in Memory**

* In Java (and most languages), an **array is a continuous block of memory**.
* All the elements of the array are stored **next to each other** in memory.
* Each element is accessed by its **index number**, starting from **0**.
* The address of any element can be directly calculated using:

**Address = Base Address + (Index × Size of each element)**

This is why accessing an array element by index is very fast — it takes **constant time O(1)**.

## **Advantages of Arrays**

**Fast Access:**

We can directly access any element using its index in **O(1) time**.

**Easy to Use:**

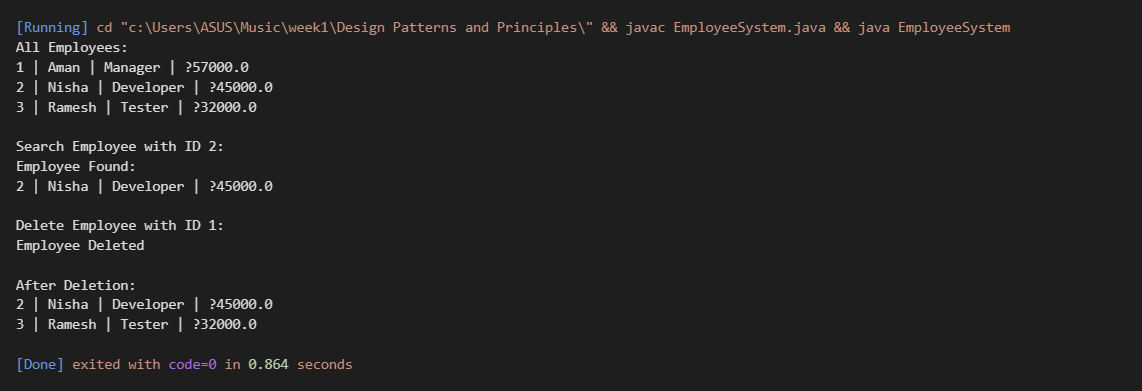
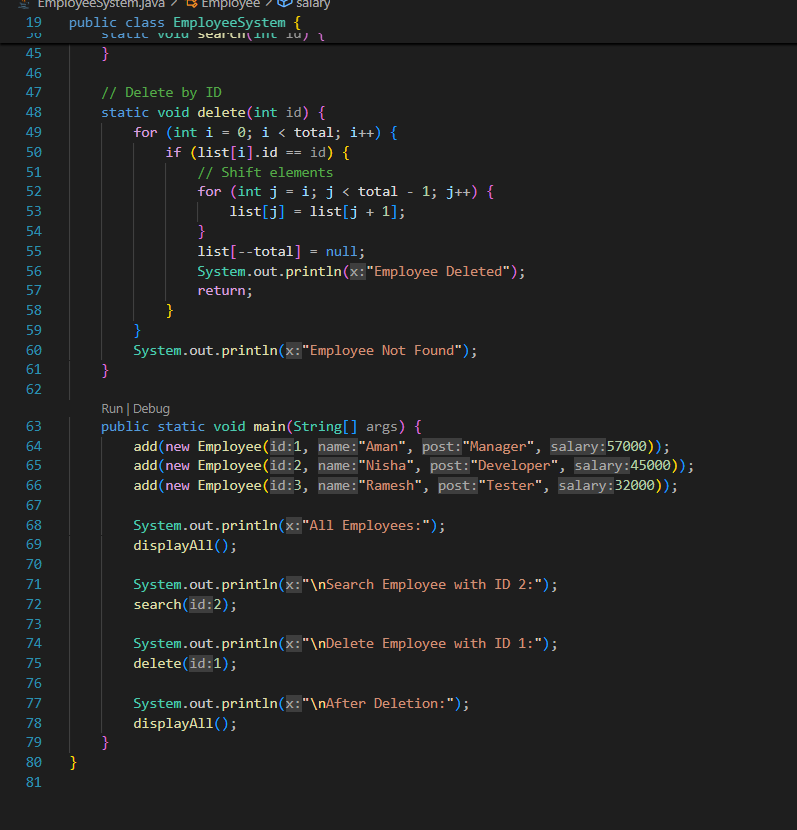
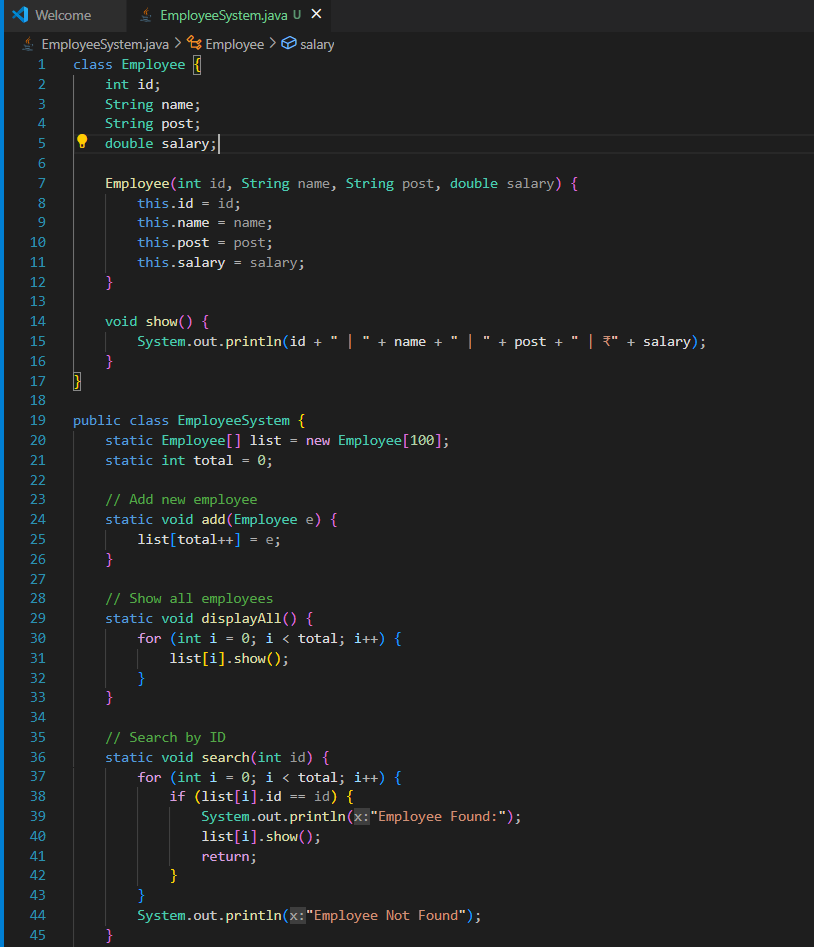
Simple to declare and use for storing multiple values of the same type.

**Memory Efficiency:**

Elements are stored in a continuous memory block, which makes memory management easier.

**Better Performance in Some Cases:**

When the number of elements is known and fixed, arrays are more efficient than dynamic structures like ArrayList or LinkedList.



## **Time Complexity of Each Array Operation**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Time Complexity** | **Explanation** |
| **Add** | O(1) *(if adding at the end and space is available)* | Directly placing the new element at the next index. |
| **Search** | O(n) | Need to check each element one by one (linear search). |
| **Traverse** | O(n) | Visit and display all elements one by one. |
| **Delete** | O(n) | After deletion, shift elements to fill the gap. |

## **Limitations of Arrays**

* **Fixed Size:**
* Once an array is created, its size can’t be changed during program execution.
* **Wasted Memory:**

If the array size is too big and not fully used, memory is wasted.

* **Inefficient Insertion/Deletion:**

Adding or removing elements (except at the end) requires shifting other elements, which takes extra time (O(n)).

* **Same Data Type Only:**

Arrays can only store elements of one data type.

## **📌 When to Use Arrays**

* When the **number of elements is known in advance** and fixed.
* When **fast direct access by index** is required.
* When the data is **simple and sequential** (like marks of students, ages, prices etc.).