**Exercise 4: Employee Management System**

**Scenario:**

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

**Steps:**

1. **Understand Array Representation:**
   * Explain how arrays are represented in memory and their advantages.
2. **Setup:**
   * Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.
3. **Implementation:**
   * Use an array to store employee records.
   * Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, search, traverse, delete).
   * Discuss the limitations of arrays and when to use them.

**Solution:**

**1. Understand Array Representation How Arrays Are Represented in Memory:**

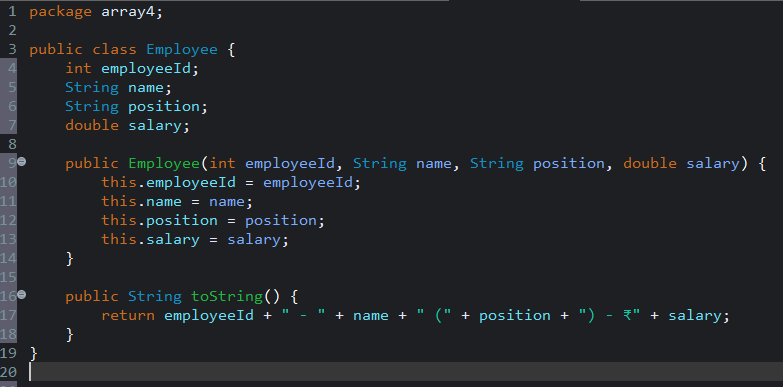
* Arrays are stored in contiguous memory locations.
* Each element in the array can be accessed via an index starting from 0.
* The address of an element is calculated as:

**address = base\_address + (index \* size\_of\_element)**

**Advantages of Arrays:**

* Fast access (O(1)) using index.
* Simple and easy to use.
* Efficient for storing fixed-size data collections**.**

**2. Setup: Employee Class:**

****

**3. Implementation: Array-Based Employee Management**

****

**4. Analysis**

**Time Complexity of Operations:**

| Operation | Time Complexity |
| --- | --- |
| Add Employee | O(1) |
| Search Employee | O(n) |
| Traverse | O(n) |
| Delete Employee | O(n) (due to shifting) |

**Limitations of Arrays:**

| **Limitation** | **Explanation** |
| --- | --- |
| Fixed Size | You must define the array size initially. Cannot grow dynamically. |
| Inefficient Deletion | Requires shifting elements after deletion. |
| Inefficient Insertion | Inserting at specific positions also requires shifting. |
| No Built-in Flexibility | Unlike ArrayList, no automatic resizing or high-level operations. |

**When to Use Arrays:**

* When the number of records is known and fixed.
* When fast indexed access (O(1)) is needed.
* For low-level performance optimization.

For dynamic and scalable systems, ArrayList or other collection frameworks like HashMap or LinkedList are more suitable.