**Exercise 4: Employee Management System**

**1.Understand Array Representation**

**How Arrays Work in Memory:**

* Arrays are stored in **contiguous memory blocks**.
* Each element is accessed via an **index**, which gives **O(1) time complexity** for access.
* They are fixed in size, so once you define the array length, it cannot grow or shrink dynamically.

**Advantages:**

* Fast access using index.
* Easy to implement and lightweight.
* Works great for small or fixed-size datasets.

**2. Setup**

Create a class named Employee with attributes:

* employeeId
* name
* position
* salary

We’ll use an **array** to store multiple employee objects.

**3. Implementation**

* Employee.java:

public class Employee {

int employeeId;

String name;

String position;

double salary;

public Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

@Override

public String toString() {

return employeeId + " - " + name + " - " + position + " - ₹" + salary;

}

}

* EmployeeManager.java

public class EmployeeManager {

Employee[] employees = new Employee[100];

int count = 0;

public void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

}

}

public Employee searchEmployee(int id) {

for (int i = 0; i < count; i++) {

if (employees[i].employeeId == id) {

return employees[i];

}

}

return null;

}

public void traverseEmployees() {

for (int i = 0; i < count; i++) {

System.out.println(employees[i]);

}

}

public void deleteEmployee(int id) {

for (int i = 0; i < count; i++) {

if (employees[i].employeeId == id) {

for (int j = i; j < count - 1; j++) {

employees[j] = employees[j + 1];

}

employees[--count] = null;

break;

}

}

}

}

* EmployeeTest.java

public class EmployeeTest {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager();

manager.addEmployee(new Employee(1, "Alice", "Manager", 80000));

manager.addEmployee(new Employee(2, "Bob", "Developer", 60000));

manager.addEmployee(new Employee(3, "Charlie", "Designer", 55000));

System.out.println("All Employees:");

manager.traverseEmployees();

System.out.println("\nSearching for Employee with ID 2:");

System.out.println(manager.searchEmployee(2));

System.out.println("\nDeleting Employee with ID 1:");

manager.deleteEmployee(1);

System.out.println("\nAll Employees After Deletion:");

manager.traverseEmployees();

}

}

**4. Analysis**

| **Operation** | **Time Complexity** | **Description** |
| --- | --- | --- |
| Add | O(1) | Just add to next free index |
| Search | O(n) | Linear search through array |
| Traverse | O(n) | Print all elements |
| Delete | O(n) | Requires shifting elements left |

**Limitations of Arrays**

* **Fixed Size**: Can’t grow dynamically.
* **Inefficient insert/delete in middle**: Need to shift elements manually.
* **Wastes space**: If you're not using all slots.

**When to Use Arrays**

* When number of elements is **known and fixed**.
* When you need **fast, indexed access**.
* When memory usage needs to be minimal and predictable.

**Output :**

