**Exercise 4: Employee Management System**

**What Is an Array?**

An array is a **collection of elements stored in a fixed-size, continuous block of memory**. Each element in the array is of the same data type and can be directly accessed using an index (starting from 0).

For example:

int[] numbers = {10, 20, 30};

Here, numbers[0] gives 10, and numbers[2] gives 30. The elements are stored side-by-side in memory, which makes lookup by index very fast.

**🔹 Advantages of Arrays**

* **Fast Access**: Accessing any element using its index is done in constant time – O(1).
* **Compact Memory Use**: Arrays use a minimal amount of memory without extra pointers or references.
* **Simple to Use**: Arrays are easy to declare and manage, especially for fixed-size data like 10 employees or 100 student records

**Source Code:**

**Employee.java**

public class Employee {

private final String employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String getEmployeeId() { return employeeId; }

public String getName() { return name; }

public String getPosition() { return position; }

public double getSalary() { return salary; }

public void setName(String name) { this.name = name; }

public void setPosition(String position) { this.position = position; }

public void setSalary(double salary) { this.salary = salary; }

@Override

public String toString() {

return employeeId + " | " + name + " | " + position + " | " + salary;

}

}

**EmployeeManager.java**

public class EmployeeManager {

private Employee[] employees;

private int count;

public EmployeeManager(int size) {

employees = new Employee[size];

count = 0;

}

public boolean add(Employee emp) {

if (count >= employees.length) return false;

employees[count++] = emp;

return true;

}

public Employee search(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

return employees[i];

}

}

return null;

}

public void traverse() {

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

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

}

}

public boolean delete(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

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

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

}

employees[--count] = null;

return true;

}

}

return false;

}

}

**Demo.java**

public class Demo {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(10);

manager.add(new Employee("E001", "Alice", "Manager", 75000));

manager.add(new Employee("E002", "Bob", "Developer", 55000));

manager.add(new Employee("E003", "Charlie", "Designer", 50000));

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

manager.traverse();

System.out.println("\nSearch E002:");

System.out.println(manager.search("E002"));

System.out.println("\nDelete E001:");

manager.delete("E001");

System.out.println("\nRemaining Employees:");

manager.traverse();

}

}

**Output:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**Time Complexity Analysis**

| **Operation** | **Description** | **Time Complexity** |
| --- | --- | --- |
| **Add** | Add at the next available index | O(1) |
| **Search** | Linear search by employee ID | O(n) |
| **Traverse** | Print all employees | O(n) |
| **Delete** | Find and shift remaining items | O(n) |

**Limitations of Using Arrays**

**🔹 Fixed Size**

Once an array is created, its size cannot grow. If your employee list grows beyond that size, you need to create a larger array manually.

**🔹 Costly Insertions and Deletions**

Adding in the middle or deleting an employee requires **shifting elements**, which slows performance as the size increases.

**🔹 Inefficient Search by ID**

To find an employee by ID, you must **scan through the array linearly**, which takes O(n) time. A better choice would be a HashMap for instant lookup.

**When Arrays Are a Good Fit**

Use arrays when:

* The number of employees is small and fixed.
* You need simple and fast indexing.
* You don’t need frequent insertions/deletions in the middle.
* You're working on simple projects, exercises, or prototypes.

**Conclusion**

Arrays are simple and efficient for fixed-size employee management. You can add, search, list, and delete employees using basic array operations. However, as data size and complexity grow, arrays become limited. That’s when dynamic structures like ArrayList or HashMap are better suited.