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.

→ Step 1: Understand Array Representation

How Arrays Work in Memory:

- Arrays are **contiguous blocks of memory** where elements are stored.
- Each element is accessed using an **index**, starting from 0.
- The memory address of each element is calculated as:
- base_address + (index * size_of_each_element)

Advantages of Arrays:

- Fast **random access**: O(1) time to access any element.
- Easy to **traverse** and implement.
- Memory-efficient for **fixed-size** data sets.

Step 2: Setup Employee Class

```
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;
    }
    public String toString() {
        return "Employee[ID=" + employeeId + ", Name=" + name + ", Position=" + position + ",
Salary=$" + salary + "]";
    }
}
```

Step 3: Implementation using Array

```
public class EmployeeManagementSystem {
    private Employee[] employees;
    private int count;
    public EmployeeManagementSystem(int size) {
        employees = new Employee[size];
        count = 0;
    }
    // Add employee
    public void addEmployee(Employee emp) {
        if (count < employees.length) {
            employees[count++] = emp;
        } else {
            System.out.println("Employee array is full.");
        }
    }
    // Search employee by ID
    public Employee searchEmployee(int empId) {
        for (int i = 0; i < count; i++) {
```

```
            if (employees[i].employeeId == empId) {
                return employees[i];
            }
        }
        return null;
    }
    // Traverse all employees
    public void displayAllEmployees() {
        for (int i = 0; i < count; i++) {
            System.out.println(employees[i]);
        }
    }
    // Delete employee by ID
    public void deleteEmployee(int empId) {
        for (int i = 0; i < count; i++) {
            if (employees[i].employeeId == empId) {
                for (int j = i; j < count - 1; j++) {
                    employees[j] = employees[j + 1];
                }
                employees[--count] = null;
                System.out.println("Employee with ID " + empId + " deleted.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }
}
```

Main Class

```
public class Main {
    public static void main(String[] args) {
        EmployeeManagementSystem ems = new EmployeeManagementSystem(5);
```

```
        ems.addEmployee(new Employee(101, "Alice", "Manager", 75000));
        ems.addEmployee(new Employee(102, "Bob", "Engineer", 60000));
        ems.addEmployee(new Employee(103, "Charlie", "Technician", 40000));
        System.out.println("All Employees:");
        ems.displayAllEmployees();
        System.out.println("\nSearch Employee ID 102:");
        System.out.println(ems.searchEmployee(102));
        System.out.println("\nDeleting Employee ID 102:");
        ems.deleteEmployee(102);
        System.out.println("\nAll Employees After Deletion:");
        ems.displayAllEmployees();
    }
}
```

Step 4: Time Complexity Analysis

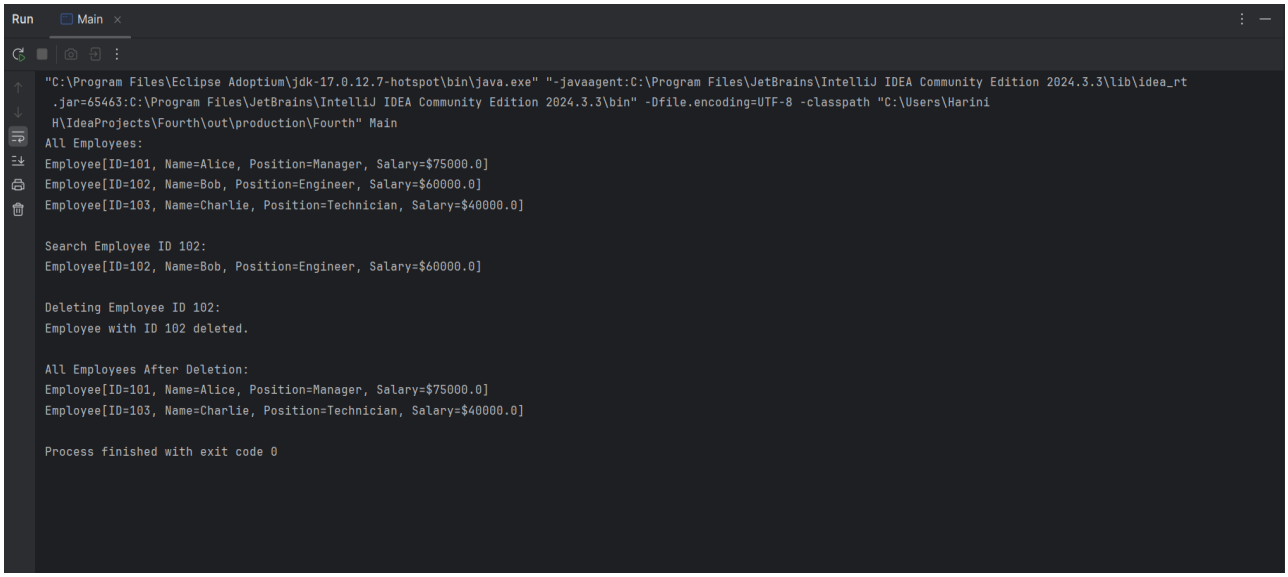
Operation	Time Complexity	Explanation
Add	O(1)	Direct insert at end (if space available)
Search	O(n)	Linear search through array
Traverse	O(n)	Print all records one by one
Delete	O(n)	Need to shift elements after deletion

Limitations of Arrays

- **Fixed size**: Cannot grow dynamically.
- **Costly insertions/deletions**: Especially from the middle.
- **Wasted memory**: If array size > actual data.
- **Inefficient search**: Linear time unless sorted.

When to Use Arrays:

- When the number of records is **known and fixed**.
- When **fast access by index** is needed.
- For **simple, small datasets** where overhead is minimal.


Run
Main
"C:\Program Files\Eclipse Adoptium\jdk-17.0.12.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.3\lib\idea_rt
.jar=65463:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Users\Harini
H\IdeaProjects\Fourth\out\production\Fourth" Main
All Employees:
Employee[ID=101, Name=Alice, Position=Manager, Salary=$75000.0]
Employee[ID=102, Name=Bob, Position=Engineer, Salary=$60000.0]
Employee[ID=103, Name=Charlie, Position=Technician, Salary=$40000.0]
Search Employee ID 102:
Employee[ID=102, Name=Bob, Position=Engineer, Salary=$60000.0]
Deleting Employee ID 102:
Employee with ID 102 deleted.
All Employees After Deletion:
Employee[ID=101, Name=Alice, Position=Manager, Salary=$75000.0]
Employee[ID=103, Name=Charlie, Position=Technician, Salary=$40000.0]
Process finished with exit code 0