**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.**

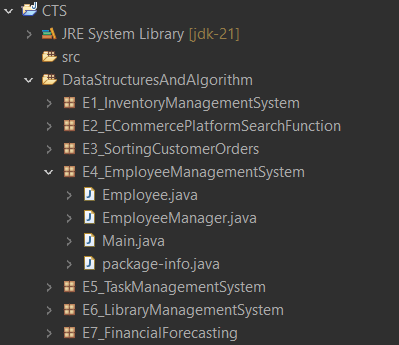
**How Arrays Work in Memory:**

* Stored in contiguous memory blocks.
* Elements can be accessed directly using indexes.
* Very fast access time: O(1) for arr[index].

**Advantages of Arrays:**

* Fast random access
* Simple and easy to use
* Memory-efficient for fixed-size data

1. **Setup:**
   * **Create a class Employee with attributes like employeeId, name, position, and salary.**

****

1. **Implementation:**
   * **Use an array to store employee records.**
   * **Implement methods to add, search, traverse, and delete employees in the array.**

**Employee.java**

package E4\_EmployeeManagementSystem;

public class Employee {

int employeeId;

String name;

String position;

double salary;

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

this.employeeId = id;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

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

}

}

**EmployeeManager.java**

package E4\_EmployeeManagementSystem;

public class EmployeeManager {

private Employee[] employees;

private int count;

public EmployeeManager(int size) {

employees = new Employee[size];

count = 0;

}

// Add employee

public void add(Employee emp) {

if (count < employees.length) {

employees[count++] = emp;

} else {

System.***out***.println("Employee list is full.");

}

}

// Search by ID

public Employee search(int id) {

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

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

return employees[i];

}

}

return null;

}

// Traverse

public void showAll() {

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

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

}

}

// Delete by ID

public void delete(int id) {

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

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

// Shift elements left

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

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

}

employees[--count] = null; // remove last duplicate

System.***out***.println("Employee deleted.");

return;

}

}

System.***out***.println("Employee not found.");

}

}

**Main.java**

package E4\_EmployeeManagementSystem;

public class Main {

public static void main(String[] args) {

EmployeeManager em = new EmployeeManager(5);

em.add(new Employee(101, "Sona", "Manager", 50000));

em.add(new Employee(102, "Sri", "Developer", 40000));

em.add(new Employee(103, "Dev", "Analyst", 35000));

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

em.showAll();

System.*out*.println("\nSearching for Employee with ID 102:");

Employee found = em.search(102);

System.*out*.println(found != null ? found : "Not found");

System.*out*.println("\nDeleting Employee with ID 101:");

em.delete(101);

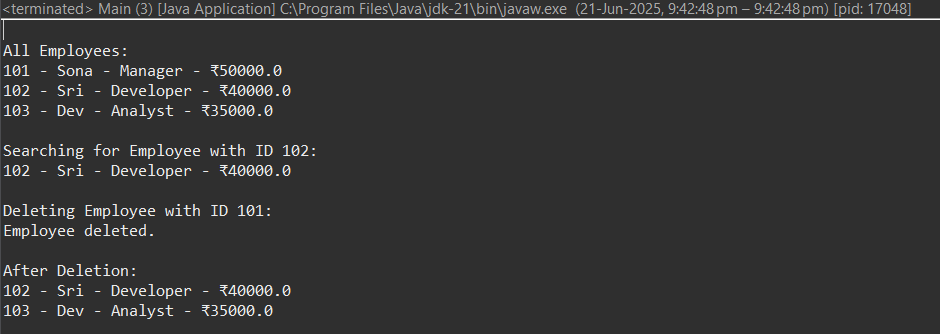
System.*out*.println("\nAfter Deletion:");

em.showAll();

}

}

**Output:**



1. **Analysis:**
   * **Analyze the time complexity of each operation (add, search, traverse, delete).**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Time Complexity** | **Explanation** |
| **Add** | O(1) | Adds at the end of array |
| **Search** | O(n) | Linear search |
| **Traverse** | O(n) | Visits each employee |
| **Delete** | O(n) | Finds and shifts elements |

* + **Discuss the limitations of arrays and when to use them.**

**Limitations of Arrays**

* **Fixed size**: Cannot grow dynamically
* **Costly deletion/insertion**: Elements must be shifted
* **Not suitable** for unknown or large number of records

**When to Use Arrays**

* When size is fixed or small
* When you need **fast access by index**
* When memory layout matters (e.g., embedded systems)