**Exercise – 4 Employee Management System**

**Representation of Arrays in memory**

Arrays in Java are stored as a **contiguous block of memory**, meaning all elements are placed next to each other in memory. Each element can be accessed in **constant time O(1)** using its index by calculating the offset from the base (starting) memory address.

**Main.java**

public class Main {

static Employee[] employees = new Employee[5];

static int curr = 0;

public static void main(String[] args) {

addEmployee(new Employee(1, "Alice", "Manager", 5000));

addEmployee(new Employee(2, "Bob", "Developer", 4000));

addEmployee(new Employee(3, "Charlie", "Designer", 4500));

System.out.println("\nEmployees List:");

traverseEmployees();

Employee found = searchEmployee(2);

if (found != null) {

System.out.println("\nFound: " + found + "\n");

} else {

System.out.println("\nEmployee not found\n");

}

deleteEmployee(2);

System.out.println("\nUpdated Employees List:");

traverseEmployees();

}

static void addEmployee(Employee e) {

if (curr < employees.length) {

employees[curr++] = e;

System.out.println("Added: " + e);

} else {

System.out.println("Cannot add more employees. Array is full.");

}

}

static Employee searchEmployee(int empId) {

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

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

return employees[i];

}

}

return null;

}

static void traverseEmployees() {

if (curr == 0) {

System.out.println("No employees to display.");

} else {

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

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

}

}

}

static void deleteEmployee(int empId) {

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

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

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

employees[j] = employees[j + 1]; // Shift left

}

employees[--curr] = null;

System.out.println("Deleted Employee with ID: " + empId);

return;

}

}

System.out.println("Employee with ID " + empId + " not found.");

}

}

**Employee.java**

class Employee {

int employeeId;

String name;

String position;

int salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;

}

}

**Output**

**A computer screen with white text

AI-generated content may be incorrect.**

**Time Complexities of the operations:**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Time Complexity** | **Explanation** |
| addEmployee() | O(1) | Adds at the end of the array using curr pointer. |
| searchEmployee() | O(n) | Linear search through the array. |
| traverseEmployees() | O(n) | Simple loop through all current elements. |
| deleteEmployee() | O(n) | Requires shifting elements after deletion. |

**Limitations in Arrays:**

1. **Fixed Size**: You must define the array size at the start. Once full, it can’t grow dynamically.
2. **Inefficient Insertions/Deletions**: Removing or adding elements from anywhere other than the end requires shifting.
3. **Wasted or Insufficient Memory**: You may allocate too much or too little memory.
4. **No Built-in Utilities**: Arrays don’t provide built-in methods like add(), remove(), etc.

**When to use Arrays:**

1. When the number of employees is **known and fixed**.
2. When **fast index-based access** is needed.
3. When **memory layout and performance** are important.
4. When you want a **simple and minimalistic solution**.