**EMPLOYEE MANAGEMENT SYSTEM**

**HOW ARRAYS ARE REPRESENTED IN MEMORY:**

- Arrays are stored in contiguous blocks of memory.

- Each element can be accessed using an index (O(1) time).

- Arrays are efficient for random access, but resizing and insertions/deletions (especially in the middle) can be expensive.

**ADVANTAGES:**

- Fast access by index.

- Memory efficiency (especially in low-level languages).

- Simplicity in implementation.

**PROGRAM:**

import java.util.Scanner;

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 "Employee(" + employeeId + ", " + name + ", " + position + ", " + salary + ")";

}

}

public class Main {

static final int MAX\_EMPLOYEES = 100;

static Employee[] employees = new Employee[MAX\_EMPLOYEES];

static int count = 0;

public static void addEmployee(int id, String name, String position, double salary) {

if (count < MAX\_EMPLOYEES) {

employees[count++] = new Employee(id, name, position, salary);

System.out.println("Employee added.");

} else {

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

}

}

public static void searchEmployee(int id) {

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

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

System.out.println("Found: " + employees[i]);

return;

}

}

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

}

public static void traverseEmployees() {

if (count == 0) {

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

return;

}

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

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

}

}

public static 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;

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

return;

}

}

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

}

public static void main(String[] args) {

addEmployee(1, "AAA", "Manager", 75000);

addEmployee(2, "BBB", "Engineer", 60000);

addEmployee(3, "CCC", "Technician", 45000);

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

traverseEmployees();

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

searchEmployee(2);

System.out.println("\nDeleting Employee ID 2:");

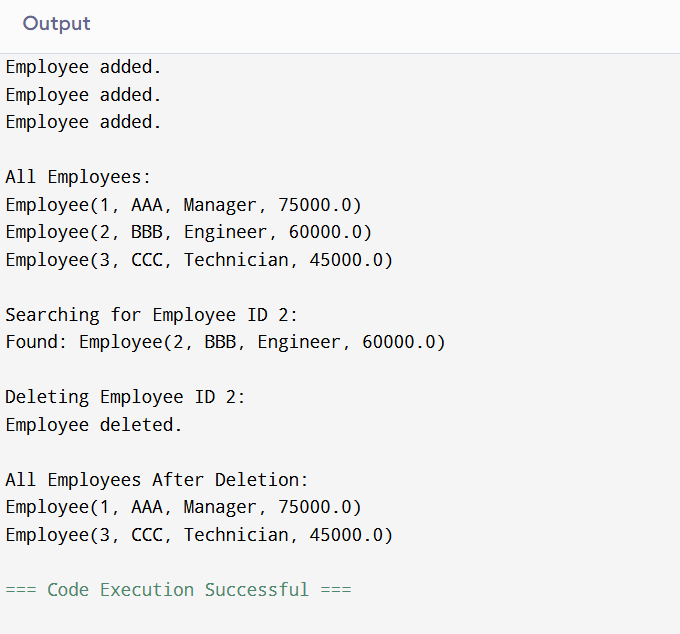
deleteEmployee(2);

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

traverseEmployees();

}

}



**TIME COMPLEXITY:**

- Add : O(1)

- Search (by ID): O(n)

- Traverse: O(n)

- Delete (by ID): O(n)

**LIMITATIONS OF ARRAYS:**

- Fixed size in low-level languages (like C/C++) unless using dynamic arrays.

- Insertion/deletion (except at end) is expensive (O(n) time).

- Inefficient for frequent insertions/deletions in middle or beginning.

**WHEN TO USE ARRAYS:**

- When you need fast random access.

- When the size of data is known or changes infrequently.

- For small to moderately sized datasets.

**ALTERNATIVES:**

- Linked lists (better for frequent insertions/deletions).

- Dictionaries/hash maps (faster search by key).