**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.

class Employee {

int employeeId;

String name;

String position;

double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

}

public class EmployeeManagementSystem {

private Employee[] employees;

private int size;

public EmployeeManagementSystem(int capacity) {

employees = new Employee[capacity];

size = 0;

}

public void addEmployee(Employee employee) {

if (size < employees.length) {

employees[size] = employee;

size++;

} else {

System.***out***.println("Array is full");

}

}

public Employee searchEmployeeById(int employeeId) {

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

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

return employees[i];

}

}

return null;

}

public void displayEmployees() {

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

System.***out***.println(employees[i].employeeId + ", " + employees[i].name + ", " + employees[i].position + ", " + employees[i].salary);

}

}

public void deleteEmployee(int employeeId) {

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

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

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

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

}

size--;

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

return;

}

}

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

}

public static void main(String[] args) {

EmployeeManagementSystem system = new EmployeeManagementSystem(5);

system.addEmployee(new Employee(1, "Neetu", "Manager", 90000));

system.addEmployee(new Employee(2, "Yash", "Developer", 100000));

system.addEmployee(new Employee(3, "Harya", "Designer", 85000));

system.displayEmployees();

Employee foundEmployee = system.searchEmployeeById(2);

if (foundEmployee != null) {

System.***out***.println("Found employee: " + foundEmployee.name);

}

system.deleteEmployee(2);

system.displayEmployees();

}

}

OUTPUT:

1, Neetu, Manager, 90000.0

2, Yash, Developer, 100000.0

3, Harya, Designer, 85000.0

Found employee: Yash

Employee deleted

1, Neetu, Manager, 90000.0

3, Harya, Designer, 85000.0

1.Explain how Arrays Are Represented in Memory and Their Advantages?

Arrays are a collection of elements stored in contiguous memory locations. They allow for efficient access to elements using an index. Advantages include:

-constant-time access (O(1)) and ease of iteration.

-Simple to implement.

-Efficient for random access.

-Good for storing data that needs to be accessed sequentially.

2.Analysis

**Time complexity:**

addEmployee: O(1) on average if there's space, otherwise O(n) for resizing

searchEmployeeById: O(n) as we need to iterate through the entire array in the worst case.

displayEmployees: O(n) as we iterate through all employees.

deleteEmployee: O(n) due to shifting elements.

**3.Limitations of arrays:**

Fixed size, which can lead to issues if the number of employees grows beyond the initial capacity.

Inefficient for insertions and deletions as elements need to be shifted.

**4.When to use arrays:**

When the size of the data is known in advance and doesn't change frequently.

When random access to elements is required.

For simple data structures where performance is not critical.

For larger and more dynamic employee management systems, data structures like ArrayLists or HashMaps would be more suitable due to their flexibility and better performance for insertions and deletions.