**Exercise 4: Employee Management System**

**1. Understand Array Representation**

**Arrays in memory:**

Arrays are contiguous blocks of memory where each element can be accessed directly using its index. This allows for fast access times but makes resizing and certain operations more complex.

**2. Setup**

**Create an Employee class:**

public class Employee {

private int employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

// Getters and Setters

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPosition() {

return position;

}

public void setPosition(String position) {

this.position = position;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

}

**3. Implementation**

**Use an array to store employee records and implement methods:**

public class EmployeeManagement {

private Employee[] employees;

private int size;

public EmployeeManagement(int capacity) {

employees = new Employee[capacity];

size = 0;

}

public void addEmployee(Employee employee) {

if (size >= employees.length) {

resize();

}

employees[size++] = employee;

}

public Employee searchEmployee(int employeeId) {

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

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

return employees[i];

}

}

return null;

}

public void deleteEmployee(int employeeId) {

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

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

employees[i] = employees[--size];

employees[size] = null;

break;

}

}

}

public void traverseEmployees() {

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

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

}

}

private void resize() {

Employee[] newEmployees = new Employee[employees.length \* 2];

System.arraycopy(employees, 0, newEmployees, 0, employees.length);

employees = newEmployees;

}

public static void main(String[] args) {

EmployeeManagement management = new EmployeeManagement(2);

management.addEmployee(new Employee(1, "John", "Manager", 50000));

management.addEmployee(new Employee(2, "Jane", "Developer", 40000));

System.out.println("Traverse Employees:");

management.traverseEmployees();

System.out.println("Search Employee:");

System.out.println(management.searchEmployee(2).getName());

management.deleteEmployee(1);

System.out.println("After Deletion:");

management.traverseEmployees();

}

}

**4. Analysis**

**Time complexity:**

* **Add:** O(1) (Amortized due to resizing)
* **Search:** O(n)
* **Traverse:** O(n)
* **Delete:** O(n)

**Limitations and usage:**

Arrays are limited by their fixed size and inefficient operations like deletion and insertion in the middle. They are best used when the number of elements is known in advance and minimal modifications are needed.