**Exercise 4: Employee Management System**

**1. Understanding Array Representation:**

Arrays are represented in memory as contiguous blocks, allowing efficient access through indexing. They offer constant-time access and are simple to use, making them suitable for storing fixed-size collections of similar elements.

**2. Setup:**

Create a class Employee with attributes such as employeeId, name, position, and salary to encapsulate employee data.

class Employee {

int id;

String name;

String position;

double salary;

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

this.id = id;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

return "Employee{id=" + id + ", name='" + name + "', position='" + position + "'}";

}

public String tostring() {

return "Employee Id: " + id + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;

}

}

**3. Implementation:**

Use an array to store employee records, implementing methods to add, search, traverse, and delete employees, ensuring efficient management of the data.

public class EmployeeManager {

private Employee[] employees;

private int count;

public EmployeeManager(int capacity) {

employees = new Employee[capacity];

count = 0;

}

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

if (count >= employees.length) {

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

return;

}

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

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

}

public Employee searchEmployee(int id) {

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

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

return employees[i];

}

}

return null;

}

public void deleteEmployee(int id) {

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

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

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

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

}

employees[count--] = null;

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

return;

}

}

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

}

public void traverseEmployee() {

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

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

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

EmployeeManager employeemanager = new EmployeeManager(20);

while (true) {

System.out.println("---------------------");

System.out.println("1. ADD EMPLOYEE");

System.out.println("2. SEARCH EMPLOYEE");

System.out.println("3. DELETE EMPLOYEE");

System.out.println("4. TRAVERSE EMPLOYEES");

System.out.println("----------------------");

System.out.println("Enter choice");

int ch = sc.nextInt();

sc.nextLine();

switch (ch) {

case 1: {

System.out.println("Enter employee id: ");

int id = sc.nextInt();

sc.nextLine();

System.out.println("Enter employee name: ");

String name = sc.nextLine();

System.out.println("Enter employee salary: ");

double salary = sc.nextDouble();

sc.nextLine();

System.out.println("Enter employee position: ");

String position = sc.nextLine();

employeemanager.addEmployee(id, name, position, salary);

break;

}

case 2: {

System.out.println("Enter employee id to search: ");

int id = sc.nextInt();

Employee employee = employeemanager.searchEmployee(id);

if (employee != null) {

System.out.println("Found employee: " + id);

} else {

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

}

break;

}

case 3: {

System.out.println("Enter employee id to delete: ");

int id = sc.nextInt();

employeemanager.deleteEmployee(id);

break;

}

case 4: {

employeemanager.traverseEmployee();

break;

}

default: {

System.out.println("Invalid choices");

break;

}

}

}

}

}

**4. Analysis**

* **Add**: O(1) if there's space; O(n) if resizing is needed.
* **Search**: O(n) in the worst case.
* **Traverse**: O(n) as each element is accessed once.
* **Delete**: O(n) due to shifting elements after removal.

Arrays are limited by their fixed size and inefficient resizing, making them ideal for scenarios with a known number of elements.