**Exercise 4: Employee Management System**

**Scenario:**  
You are developing an employee management system for a company. Efficiently managing employee records is crucial.

**1. Understand Array Representation**

**Array Representation in Memory:**  
Arrays are stored in contiguous blocks of memory. Each element is accessed via an index, enabling fast read/write operations. This layout allows for constant time complexity (O(1)) when accessing elements by index.

**Advantages:**

* Fast element access using indices (O(1))
* Simple and efficient memory usage
* Easy to implement

**Disadvantages:**

* Fixed size (not dynamic)
* Costly insertions/deletions at specific positions (O(n))

**2. Setup**

**Employee Class:**

public 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 [ID=" + employeeId + ", Name=" + name + ", Position=" + position + ", Salary=" + salary + "]";

}

}

**3. Implementation**

**EmployeeManagement Class:**

public class EmployeeManagement {

private Employee[] employees;

private int size;

public EmployeeManagement(int capacity) {

employees = new Employee[capacity];

size = 0;

}

public void addEmployee(Employee emp) {

if (size < employees.length) {

employees[size++] = emp;

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

} else {

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

}

}

public void searchEmployee(int empId) {

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

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

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

return;

}

}

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

}

public void traverseEmployees() {

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

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

}

}

public void deleteEmployee(int empId) {

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

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

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

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

}

employees[--size] = null;

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

return;

}

}

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

}

}

**Main Class:**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

EmployeeManagement system = new EmployeeManagement(100);

int choice;

do {

System.out.println("\nEmployee Management System:");

System.out.println("1. Add Employee");

System.out.println("2. Search Employee");

System.out.println("3. Traverse Employees");

System.out.println("4. Delete Employee");

System.out.println("0. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter ID: ");

int id = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Position: ");

String position = scanner.nextLine();

System.out.print("Enter Salary: ");

double salary = scanner.nextDouble();

system.addEmployee(new Employee(id, name, position, salary));

break;

case 2:

System.out.print("Enter Employee ID to search: ");

system.searchEmployee(scanner.nextInt());

break;

case 3:

system.traverseEmployees();

break;

case 4:

System.out.print("Enter Employee ID to delete: ");

system.deleteEmployee(scanner.nextInt());

break;

case 0:

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

break;

default:

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

}

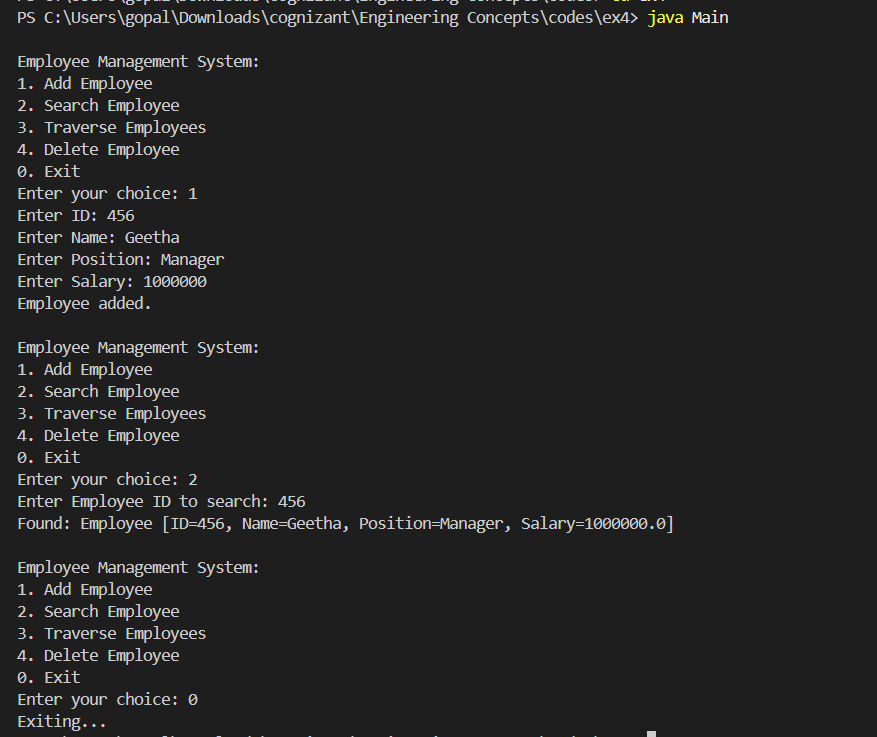
} while (choice != 0);

scanner.close();

}

}

**Output:**



**4. Analysis**

**Time Complexity:**

* Add: O(1) if space is available
* Search: O(n)
* Traverse: O(n)
* Delete: O(n) due to shifting elements

**Limitations of Arrays:**

* Fixed size: must define capacity upfront
* Inefficient for insertions and deletions in the middle
* Does not scale well with dynamic data

**When to Use Arrays:**

* When the number of elements is known and fixed
* When fast access by index is needed
* When memory usage needs to be controlled tightly