Exercise 4: Employee Management System

✅ 1. Understand Array Representation

Arrays are contiguous memory blocks where each element is placed next to the previous one. This allows:

Constant-time access using an index (O(1)) Efficient traversal

Memory locality benefits (cache-friendly) However, arrays have limitations:

Fixed size (you must declare the size upfront) Insertion/deletion requires shifting elements

✅ 2. Setup

Create a class Employee with the following attributes: employeeId (int)

name (String) position (String) salary (double)

✅ 3. Java Code Implementation

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;

}

public String toString() {

return "ID: " + employeeId + ", Name: " + name + ", Position: " +

position + ", Salary:

}

₹" + salary;

}

public class EmployeeManagementSystem {

static Employee[] employees = new Employee[100]; static int count = 0;

static Scanner sc = new Scanner(System.in);

public static void addEmployee() {

System.out.print("Enter ID, Name, Position, Salary: "); int id = sc.nextInt();

String name = sc.next(); String position = sc.next();

double salary = sc.nextDouble();

employees[count++] = new Employee(id, name, position, salary); System.out.println("Employee added.");

}

public static void searchEmployee() { System.out.print("Enter employee ID to search: "); int id = sc.nextInt();

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() { System.out.println("Employee List:"); for (int i = 0; i < count; i++) {

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

}

}

public static void deleteEmployee() { System.out.print("Enter employee ID to delete: "); int id = sc.nextInt();

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) { while (true) {

System.out.println("\n1. Add 2. Search 3. Display All 4. Delete 5.

Exit");

int choice = sc.nextInt(); switch (choice) {

case 1: addEmployee(); break; case 2: searchEmployee(); break;

case 3: traverseEmployees(); break; case 4: deleteEmployee(); break; case 5: return;

}

}

}

}

✅ 4. Analysis

Operation Time Complexity Explanation

Add O(1) Insert at the end using count index

|  |  |  |
| --- | --- | --- |
| Search | O(n) | Linear search through the array |
| Traverse | O(n) | Visit each element to print |
| Delete | O(n) | Find and shift elements to maintain order |

❌ Limitations of Arrays:

Fixed size (waste of space or overflow)

Slow insertions/deletions in the middle Not dynamic — need manual resizing

✅ When to Use Arrays:

When data size is known in advance For fast index-based access

For memory efficiency in simple static datasets Sample Input and Output:

1. Add 2. Search 3. Display All 4. Delete 5. Exit 1

Enter ID, Name, Position, Salary: 26 Deepika

Technician 5,000

Employee added.

1. Add 2. Search 3. Display All 4. Delete 5. Exit 3

Employee List:

ID: 26, Name: Deepika, Position: Technician, Salary:

₹5000.0

1. Add 2. Search 3. Display All 4. Delete 5. Exit 1

Enter ID, Name, Position, Salary: 26 Deepak

Technician 15000

Employee added.

1. Add 2. Search 3. Display All 4. Delete 5. Exit 3

Employee List:

ID: 26, Name: Deepika, Position: Technician, Salary: ID: 26, Name: Deepak, Position: Technician, Salary:

₹5000.0

₹15000.0

1. Add 2. Search 3. Display All 4. Delete 5. Exit