**Exercise 4: Employee Management System**

**Code:**

**EmployeeManagementSystem.java**

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 Scanner sc = new Scanner(System.in);

static final int MAX = 100;

static Employee[] employees = new Employee[MAX];

static int count = 0;

public static void main(String[] args) {

while (true) {

System.out.println("\n--- Employee Management Menu ---");

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

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

System.out.println("3. Display All Employees");

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

System.out.println("5. View Time Complexity Analysis");

System.out.println("6. Exit");

System.out.print("Choose an option: ");

int choice = sc.nextInt();

switch (choice) {

case 1 -> addEmployee();

case 2 -> searchEmployee();

case 3 -> displayEmployees();

case 4 -> deleteEmployee();

case 5 -> printAnalysis();

case 6 -> {

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

return;

}

default -> System.out.println("Invalid option!");

}

}

}

static void addEmployee() {

if (count >= MAX) {

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

return;

}

sc.nextLine();

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

int id = sc.nextInt();

sc.nextLine();

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

String name = sc.nextLine();

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

String position = sc.nextLine();

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

double salary = sc.nextDouble();

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

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

}

static void searchEmployee() {

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

int id = sc.nextInt();

boolean found = false;

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

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

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

found = true;

break;

}

}

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

}

static void displayEmployees() {

if (count == 0) {

System.out.println("No employees to display.");

return;

}

System.out.println("\n--- Employee List ---");

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

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

}

}

static void deleteEmployee() {

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

int id = sc.nextInt();

int index = -1;

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

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

index = i;

break;

}

}

if (index == -1) {

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

return;

}

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

employees[i] = employees[i + 1];

}

employees[--count] = null;

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

}

static void printAnalysis() {

System.out.println("\n--- Time Complexity Analysis ---");

System.out.println("Add Employee -> O(1) [inserting at end]");

System.out.println("Search Employee -> O(n) [linear search]");

System.out.println("Traverse -> O(n)");

System.out.println("Delete Employee -> O(n) [search + shift]");

System.out.println("\n--- Limitations of Arrays ---");

System.out.println("1. Fixed size: Can't grow beyond the defined limit (" + MAX + ").");

System.out.println("2. Inefficient insert/delete operations at arbitrary positions.");

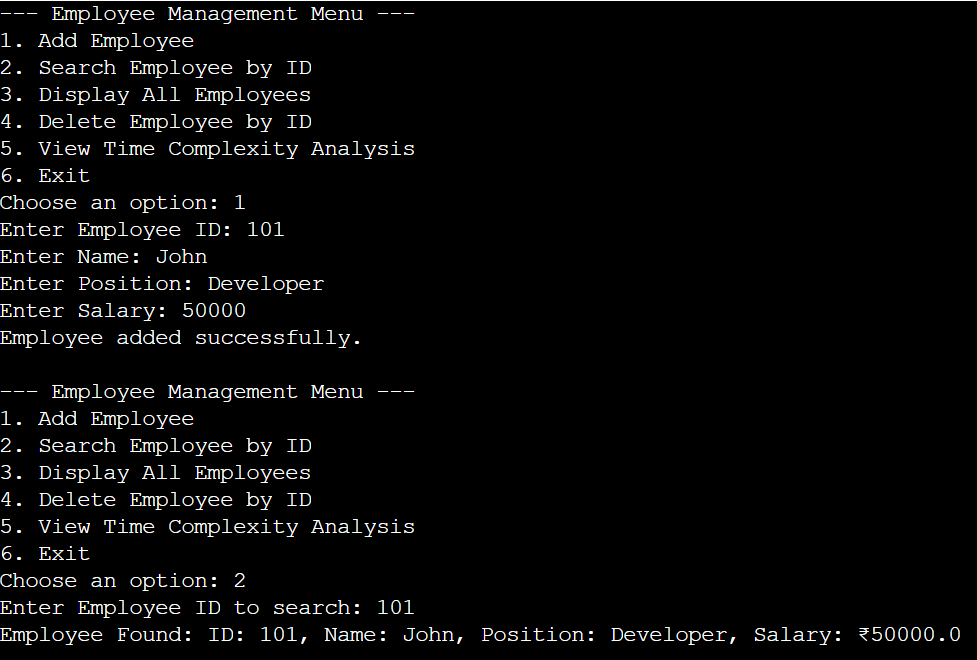
System.out.println("3. No built-in support for dynamic memory management.");

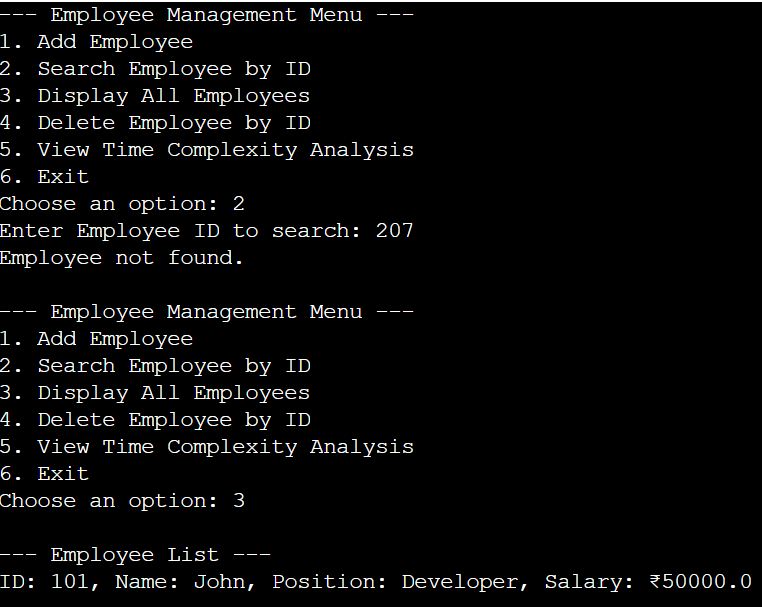
System.out.println("4. Better alternatives: ArrayList, LinkedList, or databases for large systems.");

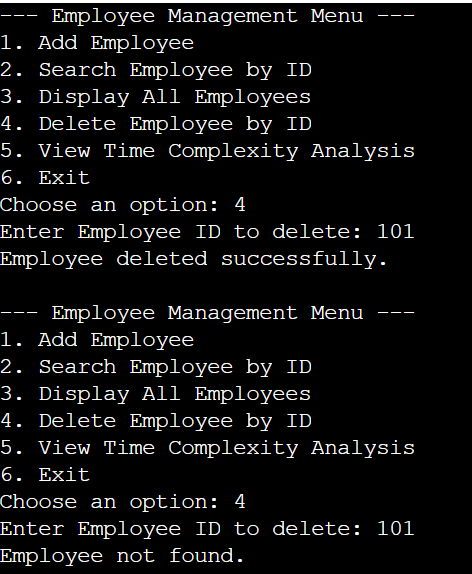
}

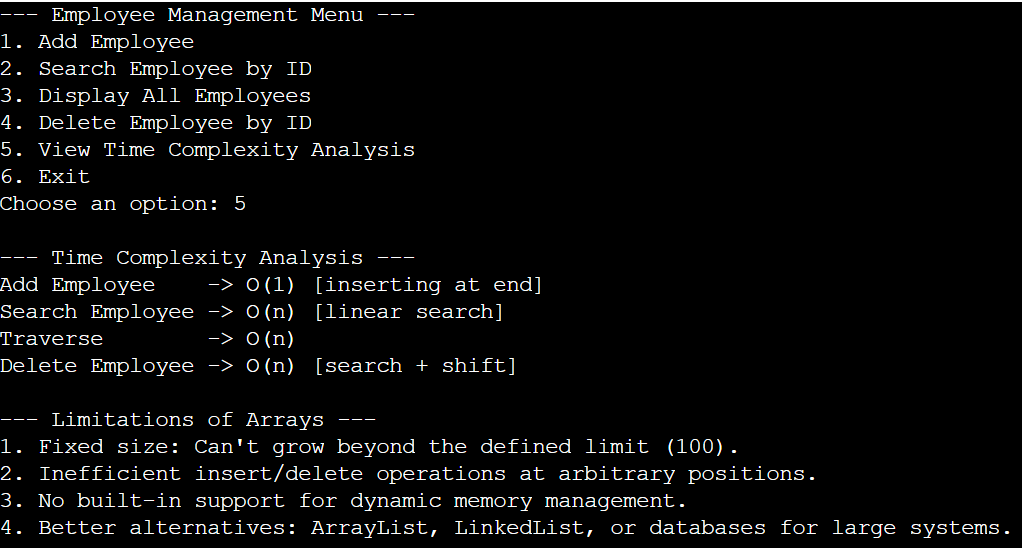
}

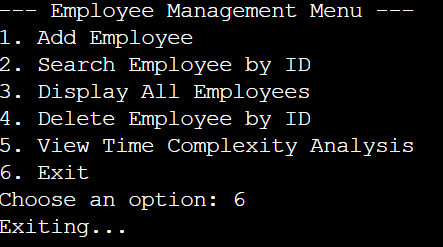
**Output:**

****

****

****

****

****