**Exercise 4: Employee Management System**

**Scenario:**

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

**Steps:**

1. **Understand Array Representation:**
   * Explain how arrays are represented in memory and their advantages.
2. **Setup:**
   * Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.
3. **Implementation:**
   * Use an array to store employee records.
   * Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, search, traverse, delete).
   * Discuss the limitations of arrays and when to use them.

**1: Array Representation in Memory**

**Explanation:**

* Arrays in memory are **contiguous blocks** of memory where elements are stored in sequence.
* Each element can be accessed using an index:  
  address of element = base address + (index \* size\_of\_element)
* **Advantages:**
  + Fast access time: O(1) for access by index.
  + Simple structure and easy to implement.

**CODE:-**

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 void display() {

        System.out.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: " + salary);

    }

}

public class EmployeeManagementSystem {

    private static final int MAX\_EMPLOYEES = 100;

    private Employee[] employees = new Employee[MAX\_EMPLOYEES];

    private int count = 0;

    // Add employee

    public void addEmployee(Employee emp) {

        if (count < MAX\_EMPLOYEES) {

            employees[count++] = emp;

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

        } else {

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

        }

    }

    // Search employee by ID

    public Employee searchEmployee(int id) {

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

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

                return employees[i];

            }

        }

        return null;

    }

    // Traverse (display all)

    public void traverseEmployees() {

        if (count == 0) {

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

            return;

        }

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

            employees[i].display();

        }

    }

    // Delete employee by ID

    public void deleteEmployee(int id) {

        int index = -1;

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

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

                index = i;

                break;

            }

        }

        if (index != -1) {

            // Shift remaining employees

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

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

            }

            employees[--count] = null;

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

        } else {

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

        }

    }

    // Main method for testing

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        EmployeeManagementSystem system = new EmployeeManagementSystem();

        while (true) {

            System.out.println("\n1. Add Employee\n2. Search Employee\n3. Display All\n4. Delete Employee\n5. Exit");

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

            int choice = sc.nextInt();

            switch (choice) {

                case 1:

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

                    int id = sc.nextInt();

                    sc.nextLine();  // Consume newline

                    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();

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

                    break;

                case 2:

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

                    int searchId = sc.nextInt();

                    Employee emp = system.searchEmployee(searchId);

                    if (emp != null) emp.display();

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

                    break;

                case 3:

                    system.traverseEmployees();

                    break;

                case 4:

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

                    int delId = sc.nextInt();

                    system.deleteEmployee(delId);

                    break;

                case 5:

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

                    sc.close();

                    return;

                default:

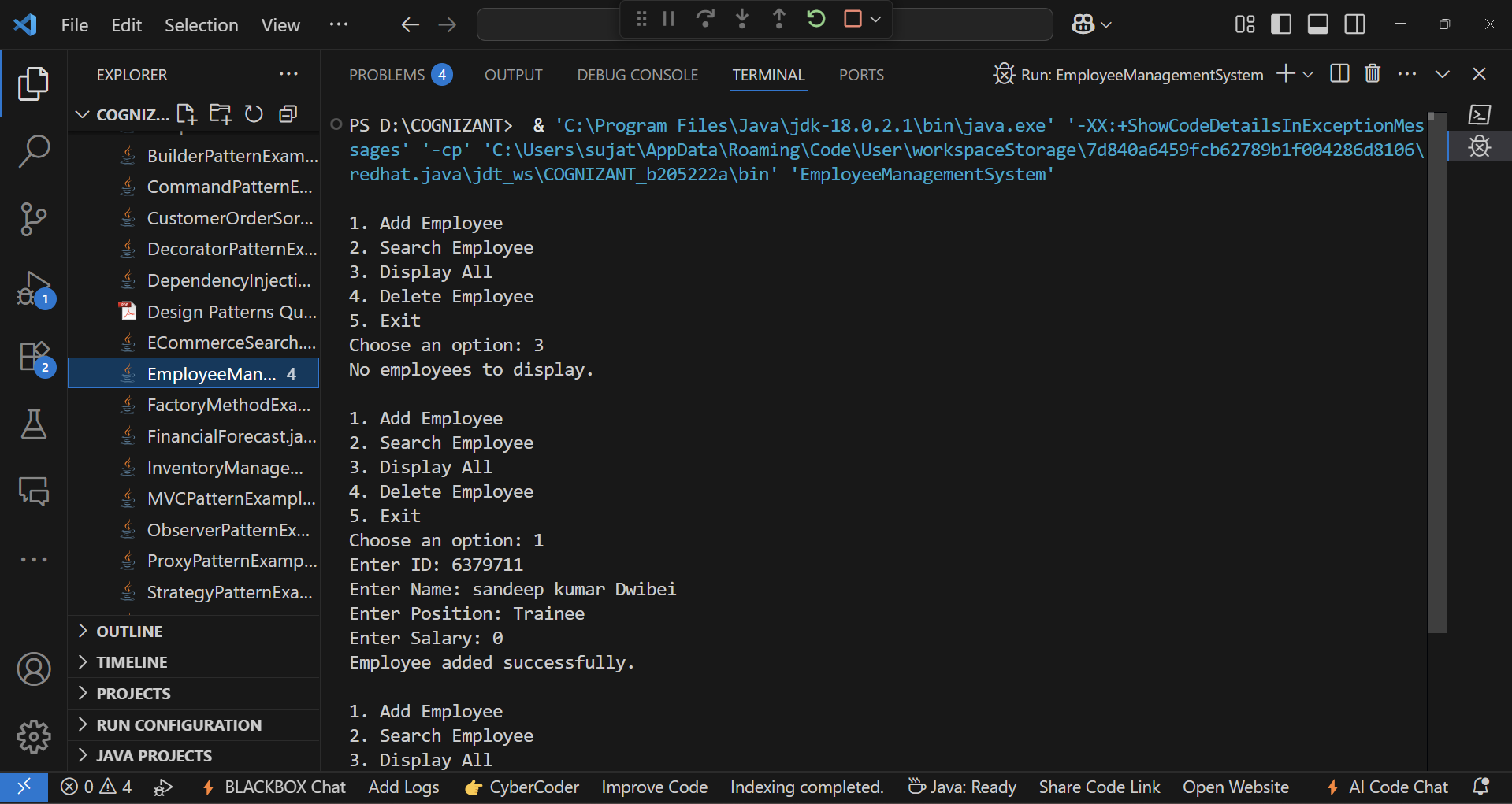
                    System.out.println("Invalid choice.");

            }

        }

    }

}

**OUTPUT:-**

**4: Time Complexity & Limitations**

Time Complexities:

* Add: O(1) — Inserting at the end of the array.
* Search: O(n) — Linear search for employee ID.
* Traverse: O(n) — Looping through all employees.
* Delete: O(n) — Finding index and shifting elements.

Limitations of Arrays:

* Fixed size: Cannot dynamically grow or shrink (solution: use ArrayList in real-world).
* Costly insertions/deletions: Especially if not at the end.
* Wasted memory: If many unused elements.

When to Use Arrays:

* When the number of elements is fixed or predictable.
* When fast indexed access is required.
* When memory allocation overhead needs to be minimized.