**Exercise – 4 Employee Management System**

**Representation of Arrays in memory**

Arrays are represented in memory as a **contiguous block of memory locations** where each element is stored one after another. The **starting address** of the array points to the first element, and the rest of the elements are accessed using an offset based on the index.

**Main.java**

public class Main{

    static Employee[] employees = new Employee[5];

    static int curr = 0;

    public static void main(String[] args) {

        addEmployee(new Employee(1,"Alice","Manager",5000));

        addEmployee(new Employee(2,"Bob","Developer",4000));

        addEmployee(new Employee(3,"Charlie", "Designer", 4500));

        System.out.println("\nEmployees:-\n");

        traverseEmployees();

        Employee found = searchEmployee(2);

        if(found!=null){

            System.out.println("Found "+found);

            System.out.println();

        }

        else{

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

            System.out.println();

        }

        deleteEmployee(2);

        System.out.println();

        traverseEmployees();

    }

    static void addEmployee(Employee e){

        if(curr<employees.length){

            employees[curr++] = e;

            System.out.println("Added the Employee"+e);

        }

    }

    static Employee searchEmployee(int empId){

        for(int i=0;i<curr;i++){

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

                return employees[i];

            }

        }

        return null;

    }

    static void traverseEmployees(){

        if(curr==0){

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

        }

        else{

            for(int i=0;i<curr;i++){

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

            }

        }

    }

    static void deleteEmployee(int empId){

        for(int i=0;i<curr;i++){

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

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

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

                }

                employees[--curr] = null;

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

                return;

            }

        }

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

    }

}

class Employee{

    int employeeId;

    String name;

    String position;

    int salary;

    Employee(int employeeId, String name, String position, int salary){

        this.employeeId = employeeId;

        this.name = name;

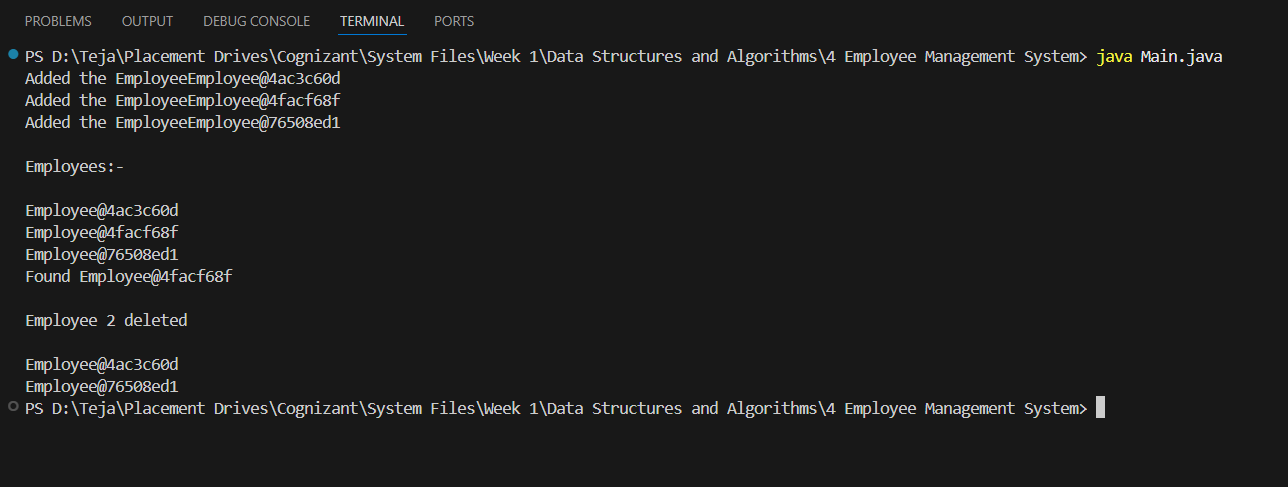
        this.position = position;

        this.salary = salary;

    }

}

**Output**



**Time Complexities of the operations:**

addEmployee: O(1)

searchEmployee: O(n)

traverseEmployee: O(n)

deleteEmployee: O(n)

**Limitations of Arrays:**

1. Fixed Size: Once an array is declared, its size is fixed and cannot be changed during runtime. This means the number of elements must be known in advance. If more elements are needed later, a new larger array must be created and the existing elements copied over.

2. Inefficient Insertions and Deletions: Inserting or deleting an element, especially from the beginning or middle of an array, requires shifting the subsequent elements. This leads to a time complexity of O(n) for these operations.

3. Wasted or Insufficient Memory: Overestimating the required size of an array leads to memory wastage, while underestimating it can cause overflow issues, requiring manual resizing and copying of data to a new array.

4. Lack of Flexibility: Arrays do not support dynamic resizing or provide built-in methods for common tasks such as searching, sorting, or filtering, which must be implemented manually by the programmer.

**Where to use Arrays:**

1. The number of elements is known and fixed.

2. Fast indexed access is required.

3. Memory layout and performance is important.

4. Simplicity is needed.