**EXERCISE 4:**

**EMPLOYEE MANAGEMENT SYSTEM**

Arrays are a collection of elements stored in contiguous memory locations.

The memory address of the first element (base address) is used to calculate the addresses of subsequent elements.

**Advantages of Arrays:**

* Constant time access (O(1)) to elements using their index.
* Easy to iterate through elements.
* Memory is allocated contiguously, reducing memory overhead**.**

**Add Employee:**

Best Case: O(1)

Worst Case: O(1)

**Search Employee:**

Best Case: O(1)

Worst Case: O(n)

**Traverse Employees:**

Best Case: O(n)

Worst Case: O(n)

**Delete Employee:**

Best Case: O(1)

Worst Case: O(n)

class Employee {  
 private int employeeId;  
 private String name;  
 private String position;  
 private 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 int getEmployeeId() { return employeeId; }  
 public String getName() { return name; }  
 public String getPosition() { return position; }  
 public double getSalary() { return salary; }  
  
 @Override  
 public String toString() {  
 return "Employee{" +  
 "employeeId=" + employeeId +  
 ", name='" + name + '\'' +  
 ", position='" + position + '\'' +  
 ", salary=" + salary +  
 '}';  
 }  
}  
  
class EmployeeManagement {  
 private Employee[] employees;  
 private int size;  
  
 public EmployeeManagement(int capacity) {  
 employees = new Employee[capacity];  
 size = 0;  
 }  
  
 // Add an employee  
 public boolean addEmployee(Employee employee) {  
 if (size == employees.length) {  
 System.*out*.println("Array is full. Cannot add more employees.");  
 return false;  
 }  
 employees[size++] = employee;  
 return true;  
 }  
  
 // Search an employee by ID  
 public Employee searchEmployee(int employeeId) {  
 for (int i = 0; i < size; i++) {  
 if (employees[i].getEmployeeId() == employeeId) {  
 return employees[i];  
 }  
 }  
 return null;  
 }  
  
 // Traverse all employees  
 public void traverseEmployees() {  
 for (int i = 0; i < size; i++) {  
 System.*out*.println(employees[i]);  
 }  
 }  
  
 // Delete an employee by ID  
 public boolean deleteEmployee(int employeeId) {  
 for (int i = 0; i < size; i++) {  
 if (employees[i].getEmployeeId() == employeeId) {  
 for (int j = i; j < size - 1; j++) {  
 employees[j] = employees[j + 1];  
 }  
 employees[--size] = null; // Decrease size and remove last element  
 return true;  
 }  
 }  
 return false;  
 }  
}  
  
public class EmployeeManagementSystem {  
 public static void main(String[] args) {  
 EmployeeManagement empManagement = new EmployeeManagement(10);  
  
 // Add employees  
 empManagement.addEmployee(new Employee(1, "Alice", "Manager", 80000));  
 empManagement.addEmployee(new Employee(2, "Bob", "Developer", 60000));  
 empManagement.addEmployee(new Employee(3, "Charlie", "Designer", 50000));  
  
 // Traverse employees  
 System.*out*.println("All Employees:");  
 empManagement.traverseEmployees();  
  
 // Search for an employee  
 int searchId = 2;  
 Employee emp = empManagement.searchEmployee(searchId);  
 if (emp != null) {  
 System.*out*.println("Employee found: " + emp);  
 } else {  
 System.*out*.println("Employee not found.");  
 }  
  
 // Delete an employee  
 int deleteId = 1;  
 if (empManagement.deleteEmployee(deleteId)) {  
 System.*out*.println("Employee with ID " + deleteId + " deleted successfully.");  
 } else {  
 System.*out*.println("Employee with ID " + deleteId + " not found.");  
 }  
  
 // Traverse employees after deletion  
 System.*out*.println("All Employees after deletion:");  
 empManagement.traverseEmployees();  
 }  
}