**Exercise 4: Employee Management System**

**Q) Explain how arrays are represented in memory and their advantages.**

In Java, an array is a contiguous block of memory where elements of the same data type are stored sequentially. When you create an array (e.g., Employee[] employees = new Employee[10];), the memory is allocated up front, and each element is accessed via its index, starting from 0.

Advantages of arrays:

* Accessing any element is done in constant time O(1) using its index.
* Contiguous memory layout improves performance due to spatial locality.
* Arrays are simple to declare and use, especially for fixed-size collections.

**Code:**

**Emloyee Class: -**

public class Employee {  
 private int employeeId;  
 private String name;  
 private String position;  
 private int salary;  
  
 public Employee(int employeeId,String name,String position,int salary){  
 this.employeeId=employeeId;  
 this.name=name;  
 this.position=position;  
 this.salary=salary;  
 }  
  
 public int getEmployeeId(){  
 return employeeId;  
 }  
  
 public void setEmployeeId(int employeeId){  
 this.employeeId=employeeId;  
 }  
  
 public String getName(){  
 return name;  
 }  
  
 public void setName(String name){  
 this.name=name;  
 }  
  
 public String getPosition(){  
 return position;  
 }  
  
 public void setPosition(String position){  
 this.position=position;  
 }  
  
 public int getSalary(){  
 return salary;  
 }  
  
 public void setSalary(int salary){  
 this.salary=salary;  
 }  
}

**EmployeeManagement Class: -**

public class EmployeeManagement {  
 public int add(Employee[] employees,Employee employee,int index){  
 if(index<employees.length) {  
 employees[index]=employee;  
 return index+1;  
 }else{  
 System.*out*.println("Array is full.");  
 return index;  
 }  
 }  
  
 public Employee search(Employee[] employees,int employeeID){  
 for(Employee employee1:employees){  
 if(employee1!=null && employee1.getEmployeeId()==employeeID){  
 System.*out*.println("Found: "+employee1.getName());  
 return employee1;  
 }  
 }  
 System.*out*.println("Not Found: ID "+employeeID);  
 return null;  
 }  
  
 public void traverse(Employee[] employees){  
 for(Employee e:employees){  
 if(e!=null){  
 System.*out*.println(e.getEmployeeId()+" . "+e.getName()+" , "+e.getPosition()+" , "+e.getSalary());  
 }  
 }  
 }  
  
 public int delete(Employee[] employees,int employeeId,int index){  
 int n=index;  
 for(int i=0;i<n;i++){  
 if(employees[i]!=null && employees[i].getEmployeeId()==employeeId){  
 for(int j=i;j<n-1;j++){  
 employees[j]=employees[j+1];  
 }  
 employees[n-1]=null;  
 System.*out*.println("Deleted employee with ID "+employeeId);  
 return index-1;  
 }  
 }  
 System.*out*.println("Employee not found with ID "+employeeId);  
 return index;  
 }  
}

**Main Class: -**

public class Main {  
 public static void main(String[] args){  
 Employee[] employees=new Employee[10];  
 EmployeeManagement management=new EmployeeManagement();  
 int index=0;  
  
 index=management.add(employees,new Employee(101,"Dhanush","CEO",120000),index);  
 index=management.add(employees,new Employee(102,"Priya","Manager",90000),index);  
 index=management.add(employees,new Employee(103,"Ram","Assistant Manager",60000),index);  
 index=management.add(employees,new Employee(104,"Rajeev","Backend Developer",50000),index);  
 index=management.add(employees,new Employee(105,"Ramya","Frontend Developer",40000),index);  
  
 System.*out*.println("\nAll Employees:");  
 management.traverse(employees);  
  
 System.*out*.println("\nSearching for Employee:");  
 management.search(employees,103);  
  
 System.*out*.println("\nDeleting Employee:");  
 index = management.delete(employees,102,index);  
  
 System.*out*.println("\nAll Employees:");  
 management.traverse(employees);  
 }  
}

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Q) Analyze the time complexity of each operation (add, search, traverse, delete).**

* Add Employee:  
  Time complexity: O(1) (if index is known and within bounds)  
  You insert the employee directly at the given index.
* Search Employee (by ID):  
  Time complexity: O(n)  
  Since the array is unsorted, one has to search till the match is found.
* Traverse Employees:  
  Time complexity: O(n)  
  You loop through the entire array to print or process each employee.
* Delete Employee:  
  Time complexity: O(n)  
  After finding the employee (O(n)), we should shift all the subsequent elements left by one position, which also takes O(n) in the worst case.

**Q)** **Discuss the limitations of arrays and when to use them.**

**-> Limitations of arrays:**

* Fixed size: Once declared, the size of the array cannot be changed. You must estimate the maximum size in advance.
* Inefficient insertions/deletions: Inserting or deleting elements in the middle of the array requires shifting elements, which takes O(n) time.
* Wasted space: If the array is not fully utilized, it leads to unused memory slots.

**->When to use arrays:**

* When the number of elements is known and fixed.
* When fast access by index is required.
* When memory allocation upfront is acceptable or preferred.