

Batch: B2 Roll No.: 1601024107

Experiment / assignment / tutorial No.06

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of the Staff In-charge with date

TITLE :Collection Framework

AIM: Create a class Employee which stores E-Name, E-Id and E-Salary of an Employee. Use class Vector to maintain an array of Employee with respect to the E-Salary. Provide the following functions

- 1) Create (): this function will accept the n Employee records in any order and will arrange them in the sorted order.
- 2) Insert (): to insert the given Employee record at appropriate index in the vector depending upon the E-Salary.
- 3) delete ByE-name(): to accept the name of the Employee and delete the record having given name
- 4) deleteByE-Id (): to accept the Id of the Employee and delete the record having given E-Id.

Provide the following functions

- 1) boolean add(E e) : This method appends the specified element to the end of this Vector.
- 2) void addElement(E obj) This method adds the specified component to the end of this vector, increasing its size by one.
- 3) int lastIndexOf(Object o, int index) This method returns the index of the last occurrence of the specified element in this vector, searching backwards from the index, or returns -1 if the element is not found.

4) `void removeElementAt(int index)` This method deletes the component at the specified index.

Expected OUTCOME of Experiment:

CO2: Explore arrays, vectors, classes and objects in C++ and Java.

Books/ Journals/ Websites referred:

1. Ralph Bravaco , Shai Simoson , “Java Programming From the Group Up” Tata McGraw-Hill.

2. Grady Booch, Object Oriented Analysis and Design .

Pre Lab/ Prior Concepts:

Vectors in Java are one of the most commonly used data structures. Similar to Arrays data structures which hold the data in a linear fashion. Vectors also store the data in a linear fashion, but unlike Arrays, they do not have a fixed size. Instead, their size can be increased on demand.

Vector class is a child class of `AbstractList` class and implements on `List` interface. To use Vectors, we first have to import Vector class from `java.util` package:

```
import java.util.Vector;
```

Access Elements in Vector:

We can access the data members simply by using the index of the element, just like we access the elements in Arrays.

Example- If we want to access the third element in a vector `v`, we simply refer to it as `v[3]`.

Vectors Constructors

Listed below are the multiple variations of vector [constructors](#) available to use:

1. **`Vector(int initialCapacity, int Increment)`** – Constructs a vector with given `initialCapacity` and its `Increment` in size.
2. **`Vector(int initialCapacity)`** – Constructs an empty vector with given `initialCapacity`. In this case, `Increment` is zero.
3. **`Vector()`** – Constructs a default vector of capacity 10.

4. **Vector(Collection c)** – Constructs a vector with a given collection, the order of the elements is same as returned by the collection's iterator.

There are also three protected parameters in vectors

- **Int capacityIncrement()**- It automatically increases the capacity of the vector when the size becomes greater than capacity.
- **Int elementCount()** – tell number of elements in the vector
- **Object[] elementData()** – array in which elements of vector are stored

Memory allocation of vectors:

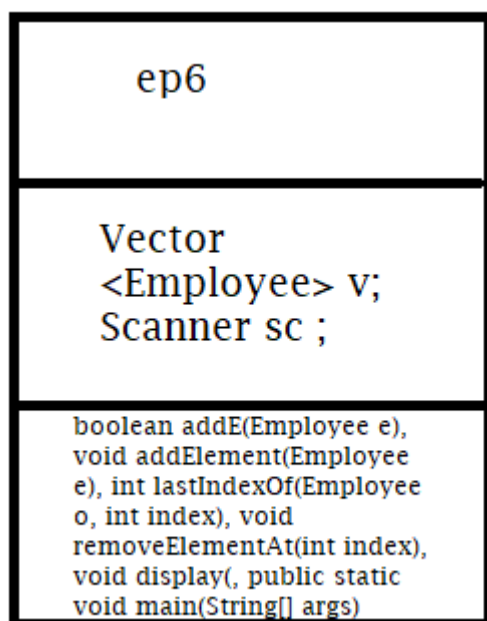
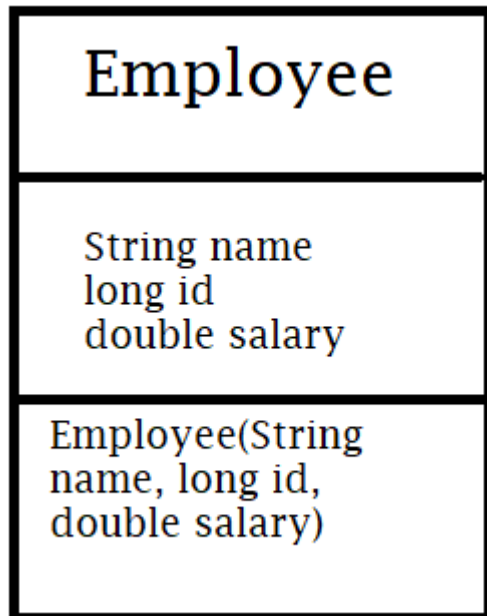
Vectors do not have a fixed size, instead, they have the ability to change their size dynamically. One might think that the vectors allocate indefinite long space to store objects. But this is not the case. Vectors can change their size based on two fields 'capacity' and 'capacityIncrement'. Initially, a size equal to 'capacity' field is allocated when a vector is declared. We can insert the elements equal to the capacity. But as soon as the next element is inserted, it increases the size of the array by size 'capacityIncrement'. Hence, it is able to change its size dynamically.

For a default constructor, the capacity is doubled whenever the capacity is full and a new element is to be inserted.

Methods of Vectors :

- Adding elements
- Removing elements
- Changing elements
- Iterating the vector

Class Diagram:



Algorithm:

1. Start
2. Input number of employees n.
3. Repeat n times: input employee name, id, and salary and add employee to list.

4. Sort the list by salary in ascending order.
5. To insert an employee, input name, id, and salary.
6. Find the correct position in the list where the salary fits.
7. Insert the employee at that position.
8. To delete by name, input the employee name.
9. Search the list for an employee with a matching name.
10. If found, remove that employee; else print "Employee not found".
11. To delete by ID, input the employee id.
12. Search the list for an employee with a matching id.
13. If found, remove that employee; else print "Employee not found".
14. To display employees, if the list is empty, print "No employees".
15. Otherwise, print the name, id, and salary of each employee in the list
16. End

Implementation details:

```
import java.util.*;

class Employee{

    String name;

    long id;

    double salary;

    Employee(String name, long id, double salary)

    {

        this.name = name;

        this.id = id;

        this.salary = salary;

    }

}
```

```
class ep6{

    static Scanner sc = new Scanner(System.in);

    Vector <Employee> v = new Vector<>();

    void create()
    {

        System.out.println("Enter number of elements to add: ");

        int n = sc.nextInt();

        sc.nextLine();

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

            System.out.println("Enter name, ID, and salary. ");

            String name = sc.nextLine();

            long id = sc.nextLong();

            double salary = sc.nextDouble();

            sc.nextLine();

            Employee e = new Employee(name,id,salary);

            v.add(e);

        }

        //sort by salary

        n = v.size();

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

            for(int j=1+i;j<n;j++)
```

```
{  
  
    double s1 = v.get(i).salary;  
  
    double s2 = v.get(j).salary;  
  
    if(s1>s2)  
    {  
  
        Employee temp = v.get(j);  
  
        v.set(j,v.get(i));  
  
        v.set(i,temp);  
  
    }  
  
}  
  
}  
  
}  
  
}  
  
void Insert()  
  
{  
  
    int n = v.size();  
  
    System.out.println("Enter name, ID, and salary. ");  
  
    String name = sc.nextLine();  
  
    long id = sc.nextLong();  
  
    double salary = sc.nextDouble();  
  
    sc.nextLine();  
  
    Employee e = new Employee(name,id,salary);  
  
  
    if (n == 0 || salary <= v.get(0).salary) {
```

```
        v.add(0, e);

        return;
    }

    if (salary >= v.get(n - 1).salary) {

        v.add(e);

        return;
    }

    for(int i=0;i<n-1;i++)
    {

        double s1 = v.get(i).salary;

        double s3 = v.get(i+1).salary;

        double s2 = e.salary;

        if(s1<=s2 && s2<=s3)
        {

            v.add(i+1,e);

            break;
        }
    }
}

void deleteByENAME(String s)
{
```



```
int n = v.size();

for(int j=0;j<n;j++)
{
    String name = v.get(j).name;

    if(name.equalsIgnoreCase(s))
    {
        v.remove(j);

        return;
    }
}

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

```
void deleteByEID(long id)
{
    int n = v.size();

    for(int j=0;j<n;j++)
    {
        long identity = v.get(j).id;

        if(identity == id)
        {
            v.remove(j);

            return;
        }
    }
}
```

```
    }

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

boolean addE(Employee e)
{
    v.add(e);

    return true;
}

void addElement(Employee e)
{
    v.add(e);
}

int lastIndexOf(Employee o, int index)
{
    long myid = o.id;

    for(int i=index;i>=0;i--)
    {
        if(myid == v.get(i).id)
        {
            return i;
        }
    }
}
```

```
    }

    return -1;
}

void removeElementAt(int index)
{
    v.remove(index);
}

void display() {
    if(v.isEmpty()) {
        System.out.println("No employees to display.");
        return;
    }

    System.out.println("Employees:");

    for(Employee e : v) {
        System.out.println("Name: " + e.name + ", ID: " + e.id + ",
Salary: " + e.salary);
    }
}

public static void main(String[] args) {

    ep6 obj = new ep6();
```

```
Scanner sc = new Scanner(System.in);

while(true) {

    System.out.println("\nMenu:");

    System.out.println("1. Create Employees");

    System.out.println("2. Insert Employee (sorted by salary)");

    System.out.println("3. Delete Employee by Name");

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

    System.out.println("5. Add Employee");

    System.out.println("6. Find last index of Employee by ID");

    System.out.println("7. Remove Element at index");

    System.out.println("8. Display Employees");

    System.out.println("9. Exit");

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

    int choice = sc.nextInt();

    sc.nextLine();

    switch(choice) {

        case 1:

            obj.create();

            break;

        case 2:

            obj.Insert();

            break;
```

```
case 3:

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

    String name = sc.nextLine();

    obj.deleteByENAME(name);

    break;

case 4:

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

    long id = sc.nextLong();

    sc.nextLine();

    obj.deleteByEID(id);

    break;

case 5:

    System.out.println("Enter name, ID and salary to add
using addE:");

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

    String nameAddE = sc.nextLine();

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

    long idAddE = sc.nextLong();

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

    double salaryAddE = sc.nextDouble();

    sc.nextLine();

    Employee eAddE = new Employee(nameAddE, idAddE,
salaryAddE);

    obj.addE(eAddE);
```

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

        break;

    case 6:

        System.out.println("Enter ID of employee to find last
index:");

        long idFind = sc.nextLong();

        sc.nextLine();

        Employee dummy = new Employee("", idFind, 0);

        System.out.print("Enter index to start searching
backward from (0-based): ");

        int startIndex = sc.nextInt();

        sc.nextLine();

        int idx = obj.lastIndexOf(dummy, startIndex);

        if(idx != -1)

            System.out.println("Last index of employee with
ID " + idFind + " is: " + idx);

        else

            System.out.println("Employee with ID " + idFind +
" not found before index " + startIndex);

        break;

    case 7:

        System.out.print("Enter index to remove element at:

");

        int remIndex = sc.nextInt();

        sc.nextLine();

        if(remIndex >= 0 && remIndex < obj.v.size()) {
```

```
        obj.removeElementAt(remIndex);

        System.out.println("Element removed at index " +
remIndex);

    } else {

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

    }

    break;

case 8:

    obj.display();

    break;

case 9:

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

    return;

default:

    System.out.println("Invalid choice, try again.");

}

}

}

}
```

Output:

```
at ep6.main(ep6.java:178)
PS C:\Users\STUDENT\Documents\ashwera> java ep6
```

Menu:

1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit

Enter choice: 1

Enter number of elements to add:

2

Enter name, ID, and salary.

ashwera

427

10000

Enter name, ID, and salary.

riya

481

100000

Menu:

1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit

Enter choice: 1


Enter number of elements to add:

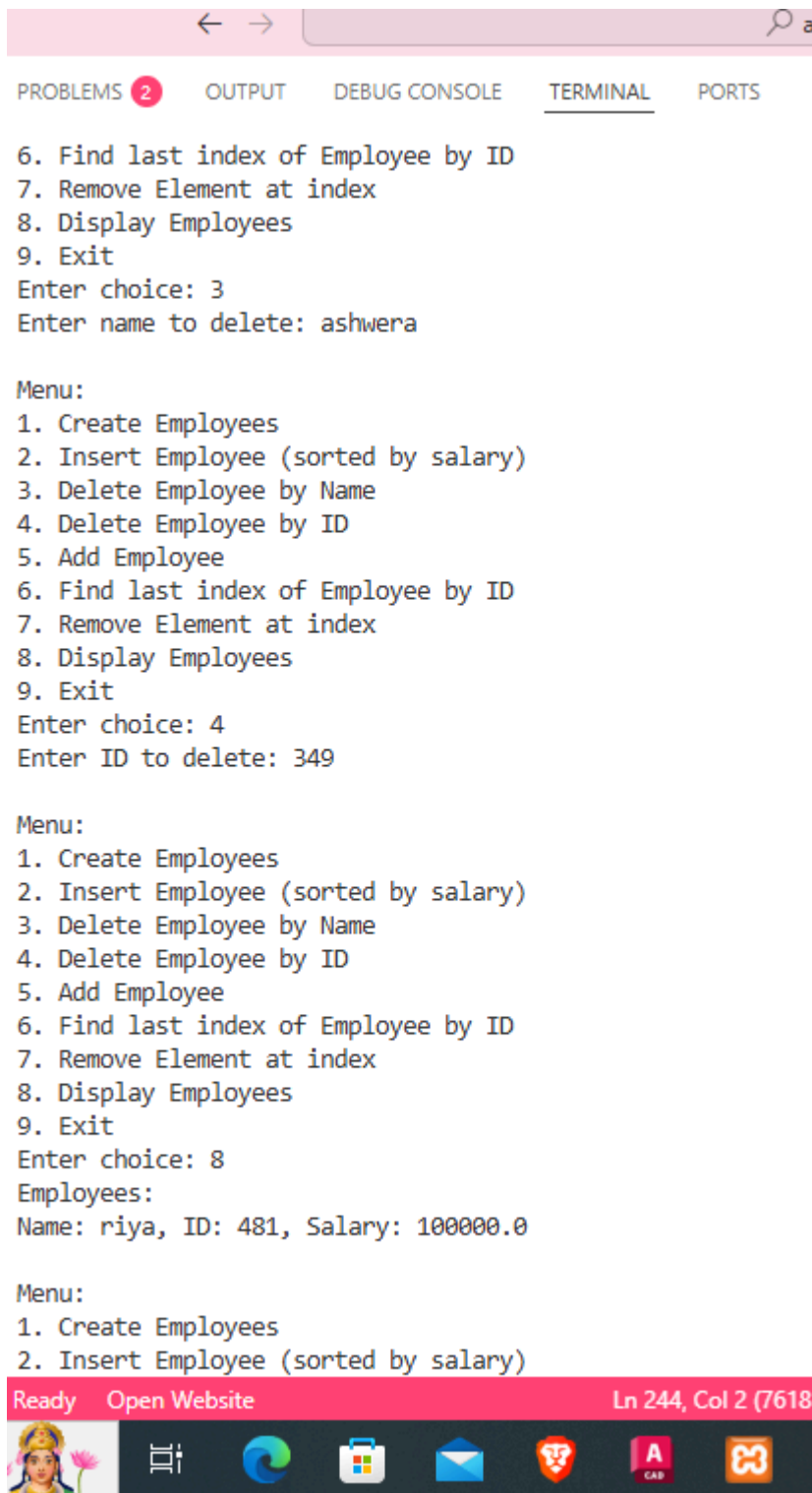

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERM

8. Display Employees
9. Exit
Enter choice: 1
Enter number of elements to add:
1
Enter name, ID, and salary.
molaika
349
100321

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 8
Employees:
Name: ashwera, ID: 427, Salary: 10000.0
Name: riya, ID: 481, Salary: 100000.0
Name: molaika, ID: 349, Salary: 100321.0

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 3
```





```
← → 🔍 a
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS


6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 3
Enter name to delete: ashwera

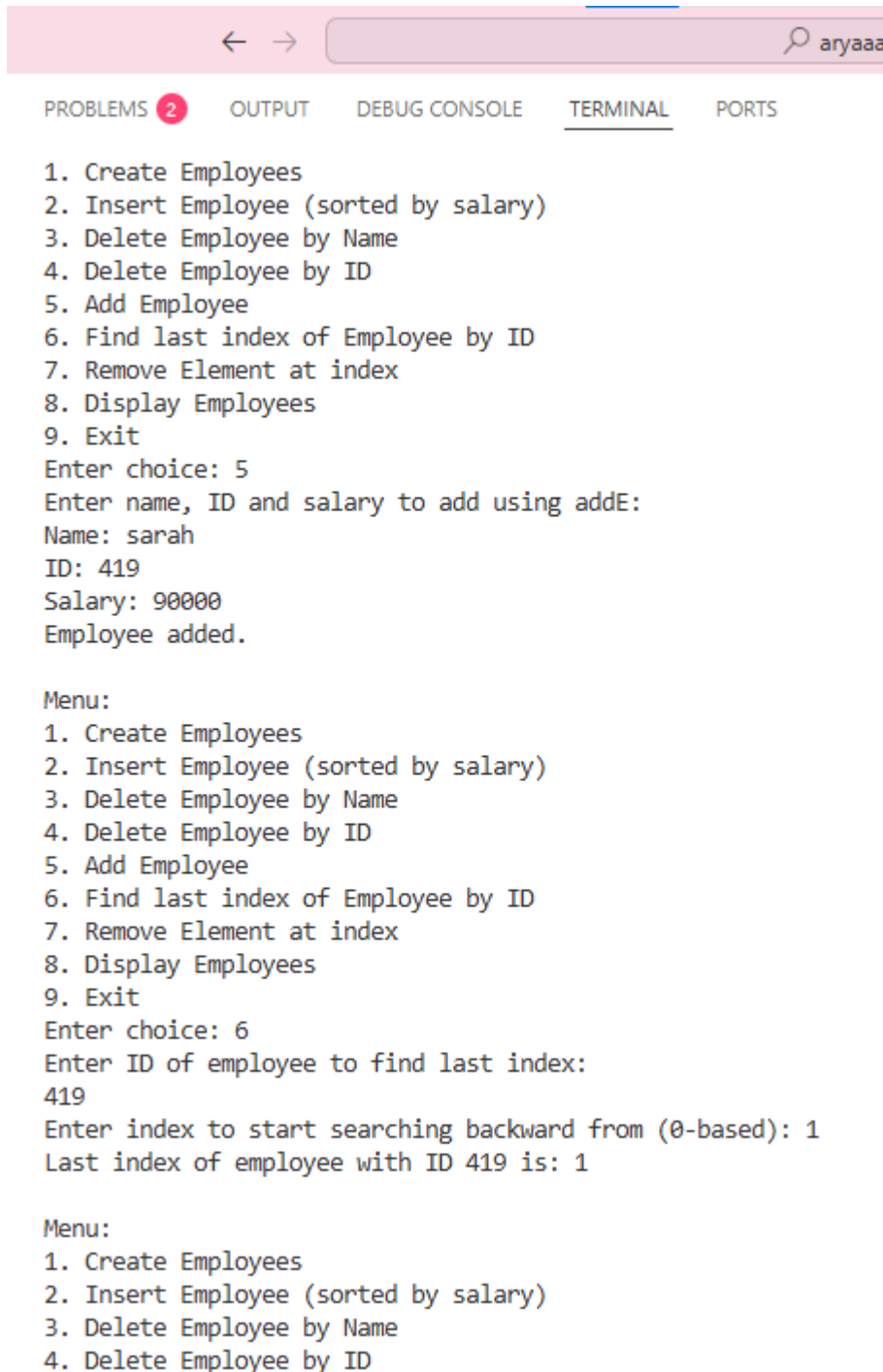
Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 4
Enter ID to delete: 349

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 8
Employees:
Name: riya, ID: 481, Salary: 100000.0

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
```

Ready Open Website Ln 244, Col 2 (7618)





```
← →  aryaaa
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 5
Enter name, ID and salary to add using addE:
Name: sarah
ID: 419
Salary: 90000
Employee added.

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 6
Enter ID of employee to find last index:
419
Enter index to start searching backward from (0-based): 1
Last index of employee with ID 419 is: 1

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
```

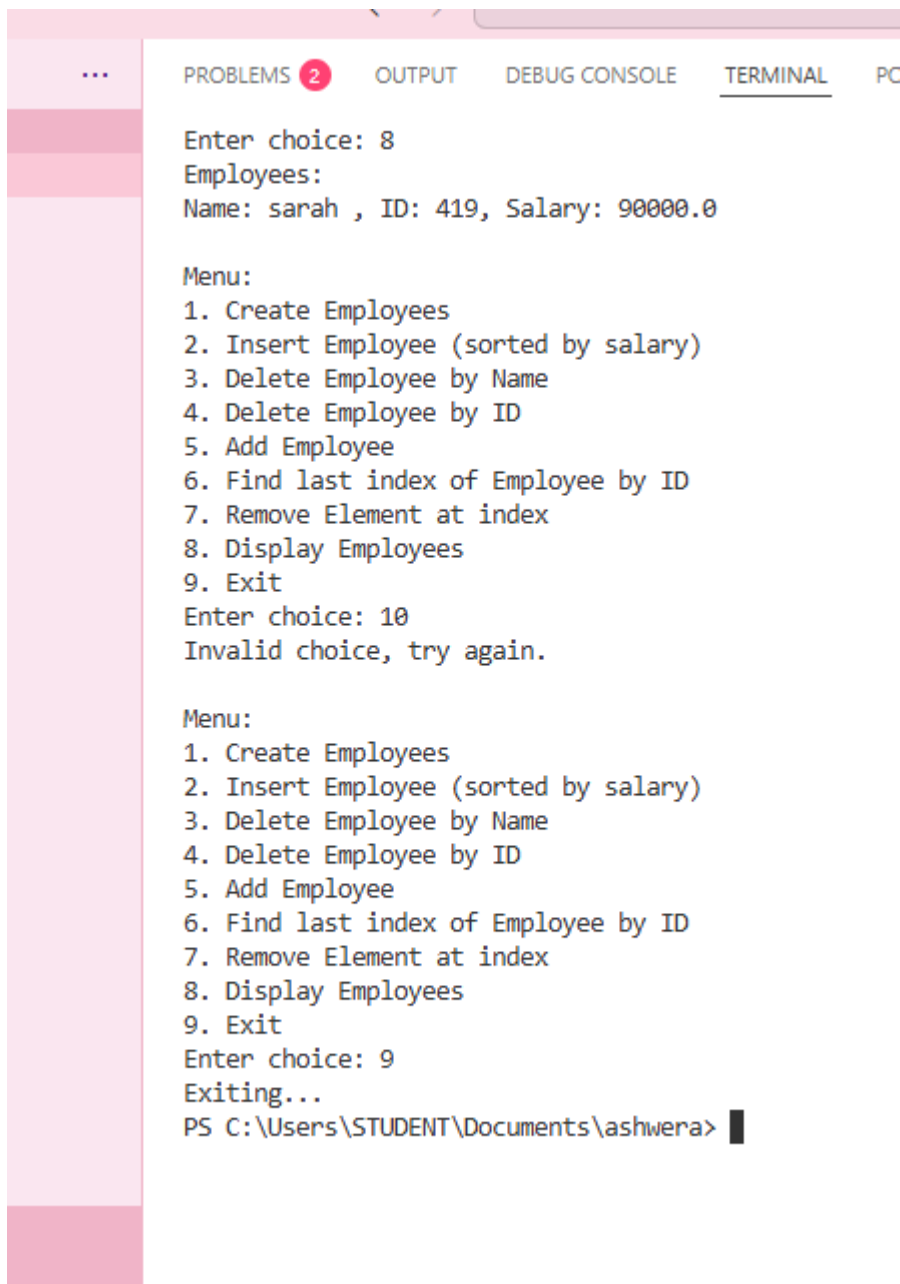
```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

8. Display Employees
9. Exit
Enter choice: 6
Enter ID of employee to find last index:
419
Enter index to start searching backward from (0-based): 1
Last index of employee with ID 419 is: 1

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 7
Enter index to remove element at: 0
Element removed at index 0

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 8
Employees:
Name: sarah , ID: 419, Salary: 90000.0

Menu:
```



```
... PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PC
Enter choice: 8
Employees:
Name: sarah , ID: 419, Salary: 90000.0

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 10
Invalid choice, try again.

Menu:
1. Create Employees
2. Insert Employee (sorted by salary)
3. Delete Employee by Name
4. Delete Employee by ID
5. Add Employee
6. Find last index of Employee by ID
7. Remove Element at index
8. Display Employees
9. Exit
Enter choice: 9
Exiting...
PS C:\Users\STUDENT\Documents\ashwera>
```

Conclusion:

The employee management program demonstrates effective use of Java's Vector class to store and manipulate employee records dynamically. It supports core operations like creating a list of employees, inserting new employees while maintaining order, and deleting employees by name or ID. The program highlights how collections simplify data management compared to fixed-size arrays, providing flexibility and ease of use.

Overall, this approach offers a solid foundation for building more complex employee management systems.

Date: 22/09/2025

Signature of faculty in-charge

Post Lab Descriptive Questions

1) Write a note on collection framework.

The Java Collection Framework is a unified architecture that provides a set of interfaces, classes, and algorithms to store and manipulate groups of objects. It simplifies programming by offering ready-made data structures like lists, sets, queues, and maps, which can be used to manage collections of objects efficiently.

Features:

- Interfaces: Define common behaviors for collections (e.g., Collection, List, Set, Queue, Map).
- Implementations: Concrete classes that implement these interfaces (e.g., ArrayList, LinkedList, HashSet, TreeSet, HashMap).
- Algorithms: Methods to perform operations like sorting, searching, and manipulating collections.

Advantages:

- Reusability: Pre-built classes and interfaces reduce development time.
- Flexibility: Different implementations available for various needs (e.g., fast access, ordered data).
- Interoperability: Standardized interfaces allow easy switching between implementations.
- Efficiency: Optimized data structures and algorithms improve performance.

2) Explain any 10 methods of Vector class in detail with the help of example

1. Add method: Adds an object to the end of the vector
`Vector<String> v = new Vector<>();`
`v.add("Apple");`
`v.add("Banana");`
2. `add(int index, E element)`: Adds an object to a specific index
`v.add("Banana");`
`v.add(1, "Orange");`
Output: [Banana,Orange]
3. `get(int index)`: Returns value at index
`v.add("Apple");`
`v.add("Banana");`

- ```
System.out.println(v.get(1));
output: banana
```
4. `remove(int index)`: Removes value at index  

```
v.add("Apple");
v.add("Banana");
v.remove(0);
System.out.println(v); // Output: [Banana]
```
  5. `remove(Object o)`: Finds and removes object  

```
v.add("Apple");
v.add("Banana");
v.remove("Apple");
System.out.println(v); // Output: [Banana]
```
  6. `size()` : Returns size of a vector  

```
v.add("Apple");
v.add("Banana");
System.out.println(v.size()); // Output: 2
```
  7. `isEmpty()`: Checks if a vector is empty  

```
Vector<String> v = new Vector<>();
System.out.println(v.isEmpty()); // Output: true
v.add("Apple");
System.out.println(v.isEmpty()); // Output: false
```
  8. `clear()`: Clears all content of a vector  

```
Vector<String> v = new Vector<>();
v.add("Apple");
v.add("Banana");
v.clear();
System.out.println(v); // Output: []
```
  9. `indexOf(Object o)`: Returns the first index of the object  

```
v.add("Apple");
v.add("Banana");
v.add("Apple");
System.out.println(v.indexOf("Apple")); // Output: 0
System.out.println(v.indexOf("Banana")); // Output: 1
System.out.println(v.indexOf("Orange")); // Output: -1
```
  10. `lastIndexOf(Object o)` : Returns the last index of the object  

```
v.add("Apple");
v.add("Banana");
v.add("Apple");
```

```
System.out.println(v.lastIndexOf("Apple")); // Output: 2
System.out.println(v.lastIndexOf("Banana")); // Output: 1
System.out.println(v.lastIndexOf("Orange")); // Output: -1
```

**3) What is an ArrayList? How it differs from the array?**

ArrayList is a part of Java's Collection Framework and represents a resizable array — a dynamic list that can grow or shrink in size as needed. It stores elements in an ordered sequence and allows fast access by index.

| FEATURE     | ARRAY                                 | ARRAYLIST                                    |
|-------------|---------------------------------------|----------------------------------------------|
| Data Type   | Primitive/Object                      | Object                                       |
| Size        | Fixed                                 | Can change dynamically                       |
| Performance | Faster                                | Slow                                         |
| Syntax      | int[] arr = new int[5];               | ArrayList<Integer> list = new ArrayList<>(); |
| Methods     | Limited (basic array operations only) | Rich methods (add, remove, contains, etc.)   |

**4) Implement a menu driven program for the following:**

**Accepts a shopping list (name, price and quantity) from the command line and stores them in a vector.**

**To delete specific item (given by user) in the vector**

**Add item at the end of the vector**

**Add item at specific location**

**Print the contents of vector using enumeration interface.**

```
import java.util.*;

class Item {
 String name;
 double price;
 int quantity;

 Item(String name, double price, int quantity) {
```



```
 this.name = name;
 this.price = price;
 this.quantity = quantity;
 }
}

class e6plq {
 static Scanner sc = new Scanner(System.in);
 Vector<Item> v = new Vector<>();

 void create() {
 System.out.println("Enter number of items to add:");
 int n = sc.nextInt();
 sc.nextLine();
 for(int i = 0; i < n; i++) {
 System.out.println("Enter name, price, and quantity:");
 String name = sc.nextLine();
 double price = sc.nextDouble();
 int quantity = sc.nextInt();
 sc.nextLine();
 Item item = new Item(name, price, quantity);
 v.add(item);
 }
 }

 void deleteItem() {
 System.out.println("Enter name of item to delete:");
 String name = sc.nextLine();
 for(int i = 0; i < v.size(); i++) {
 if(v.get(i).name.equalsIgnoreCase(name)) {
 v.remove(i);
 System.out.println("Item deleted");
 return;
 }
 }
 System.out.println("Item not found");
 }
}
```

```
void addItemEnd() {
 System.out.println("Enter name, price, and quantity to add at
end:");
 String name = sc.nextLine();
 double price = sc.nextDouble();
 int quantity = sc.nextInt();
 sc.nextLine();
 Item item = new Item(name, price, quantity);
 v.add(item);
 System.out.println("Item added at end");
}

void addItemAtPosition() {
 System.out.println("Enter position to insert item:");
 int pos = sc.nextInt();
 sc.nextLine();
 if(pos < 0 || pos > v.size()) {
 System.out.println("Invalid position");
 return;
 }
 System.out.println("Enter name, price, and quantity:");
 String name = sc.nextLine();
 double price = sc.nextDouble();
 int quantity = sc.nextInt();
 sc.nextLine();
 Item item = new Item(name, price, quantity);
 v.add(pos, item);
 System.out.println("Item added at position " + pos);
}

void printItems() {
 Enumeration<Item> e = v.elements();
 if(!e.hasMoreElements()) {
 System.out.println("No items in the list");
 return;
 }
 while(e.hasMoreElements()) {
 Item item = e.nextElement();
 }
}
```

```
 System.out.println("Name: " + item.name + ", Price: " +
item.price + ", Quantity: " + item.quantity);
 }
}

public static void main(String[] args) {
 e6plq list = new e6plq();
 list.create();

 while(true) {
 System.out.println("\nMenu:");
 System.out.println("1. Delete item");
 System.out.println("2. Add item at end");
 System.out.println("3. Add item at specific position");
 System.out.println("4. Print shopping list");
 System.out.println("5. Exit");
 System.out.print("Enter choice: ");
 int choice = sc.nextInt();
 sc.nextLine();
 switch(choice) {
 case 1:
 list.deleteItem();
 break;
 case 2:
 list.addItemEnd();
 break;
 case 3:
 list.addItemAtPosition();
 break;
 case 4:
 list.printItems();
 break;
 case 5:
 System.exit(0);
 default:
 System.out.println("Invalid choice");
 }
 }
}
```

```
}
}
```

```
PS C:\Users\STUDENT\Documents\ashwera> java e6plq` `
Enter number of items to add:
1
Enter name, price, and quantity:
soap
54.3
2

Menu:
1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit
Enter choice: 1
Enter name of item to delete:
shampoo
Item not found

Menu:
1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit
Enter choice: 1
Enter name of item to delete:
soap
Item not found

Menu:
1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
```

Java: Ready - Open Website

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PC

1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit
Enter choice: 3
Enter position to insert item:
2
Invalid position

Menu:
1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit
Enter choice: 3
Enter position to insert item:
1
Enter name, price, and quantity:
jaggert
43.5
23
Item added at position 1

Menu:
1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit
Enter choice: 4
Name: soap , Price: 54.3, Quantity: 2
Name: jaggert, Price: 43.5, Quantity: 23

Menu:
1. Delete item

Java: Ready Open Website
```

```
name: jagger!, Price: 43.5, Quantity: 23
```

```
Menu:
```

1. Delete item
2. Add item at end
3. Add item at specific position
4. Print shopping list
5. Exit

```
Enter choice: 5
```

```
PS C:\Users\STUDENT\Documents\ashwera>
```