**Batch: B-1 Roll No.: 16010122104**

**Experiment / assignment / tutorial No.05**

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

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :Vector** |

**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 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 Programing 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](https://www.edureka.co/blog/constructor-in-java/) 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. Create a class `Employee` with attributes E-Name, E-Id, and E-Salary.

2. Create a class `EmployeeManager` to manage employee records.

- Initialize an empty `Vector` named `employees` to store employee objects.

- Implement the following methods:

- `create(List<Employee> empList)`:

- Accept a list of Employee objects (`empList`).

- Add all employees from `empList` to the `employees` Vector.

- Sort the `employees` Vector based on E-Salary in ascending order.

- `insert(Employee emp)`:

- Accept an Employee object (`emp`).

- Find the appropriate index to insert `emp` in the `employees` Vector based on E-Salary.

- Insert `emp` at the found index.

- `deleteByName(String name)`:

- Accept a name (`name`) to be deleted.

- Remove all employees from the `employees` Vector whose E-Name matches the given `name`.

- `deleteById(int id)`:

- Accept an ID (`id`) to be deleted.

- Remove the employee from the `employees` Vector whose E-Id matches the given `id`.

- `display()`:

- Iterate through the `employees` Vector.

- Display each employee's E-Id, E-Name, and E-Salary.

3. In the `main` function:

- Initialize a `Scanner` object to read user input.

- Prompt the user to enter the number of employees (`n`).

- Create an empty list `empList` to store employee objects.

- For each employee (from 1 to `n`):

- Prompt the user to enter the employee's name, ID, and salary.

- Create an `Employee` object with the provided details and add it to `empList`.

- Create an `EmployeeManager` object (`manager`).

- Call `create(empList)` on `manager` to create and arrange employee records.

- Display the initial employee records using `display()`.

- Prompt the user to enter details for a new employee (name, ID, and salary).

- Call `insert(newEmployee)` on `manager` to insert the new employee.

- Display the updated employee records using `display()`.

- Prompt the user to enter the name of an employee to delete.

- Call `deleteByName(deleteName)` on `manager` to delete employees by name.

- Display the employee records after deletion using `display()`.

- Prompt the user to enter the ID of an employee to delete.

- Call `deleteById(deleteId)` on `manager` to delete employees by ID.

- Display the final employee records after deletion using `display()`.

4. Close the `Scanner`.

**Implementation details:**

import java.util.\*;

class Employee {

    String E\_Name;

    int E\_Id;

    double E\_Salary;

    Employee(String name, int id, double salary) {

        E\_Name = name;

        E\_Id = id;

        E\_Salary = salary;

    }

    @Override

    public String toString() {

        return "E-Id: " + E\_Id + ", E-Name: " + E\_Name + ", E-Salary: " + E\_Salary;

    }

}

public class EmployeeManager {

    private Vector<Employee> employees = new Vector<>();

    // Function to create and arrange Employee records in sorted order by E-Salary

    public void create(List<Employee> empList) {

        employees.addAll(empList);

        Collections.sort(employees, Comparator.comparingDouble(employee -> employee.E\_Salary));

    }

    // Function to insert an Employee record at the appropriate index based on E-Salary

    public void insert(Employee emp) {

        int index = Collections.binarySearch(employees, emp, Comparator.comparingDouble(employee -> employee.E\_Salary));

        if (index < 0) {

            index = -index - 1;

        }

        employees.add(index, emp);

    }

    // Function to delete an Employee record by E-Name

    public void deleteByName(String name) {

        employees.removeIf(emp -> emp.E\_Name.equals(name));

    }

    // Function to delete an Employee record by E-Id

    public void deleteById(int id) {

        employees.removeIf(emp -> emp.E\_Id == id);

    }

    // Function to display all Employee records

    public void display() {

        for (Employee emp : employees) {

            System.out.println(emp);

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        EmployeeManager manager = new EmployeeManager();

        int n;

        System.out.print("Enter the number of employees: ");

        n = scanner.nextInt();

        List<Employee> empList = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            System.out.print("Enter employee name: ");

            String name = scanner.next();

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

            int id = scanner.nextInt();

            System.out.print("Enter employee salary: ");

            double salary = scanner.nextDouble();

            empList.add(new Employee(name, id, salary));

        }

        manager.create(empList);

        System.out.println("Employee Records:");

        manager.display();

        System.out.print("Enter the details of a new employee:\nName: ");

        String name = scanner.next();

        System.out.print("ID: ");

        int id = scanner.nextInt();

        System.out.print("Salary: ");

        double salary = scanner.nextDouble();

        manager.insert(new Employee(name, id, salary));

        System.out.println("Updated Employee Records:");

        manager.display();

        System.out.print("Enter the name of an employee to delete: ");

        String deleteName = scanner.next();

        manager.deleteByName(deleteName);

        System.out.println("Employee Records after deletion:");

        manager.display();

        System.out.print("Enter the ID of an employee to delete: ");

        int deleteId = scanner.nextInt();

        manager.deleteById(deleteId);

        System.out.println("Final Employee Records after deletion:");

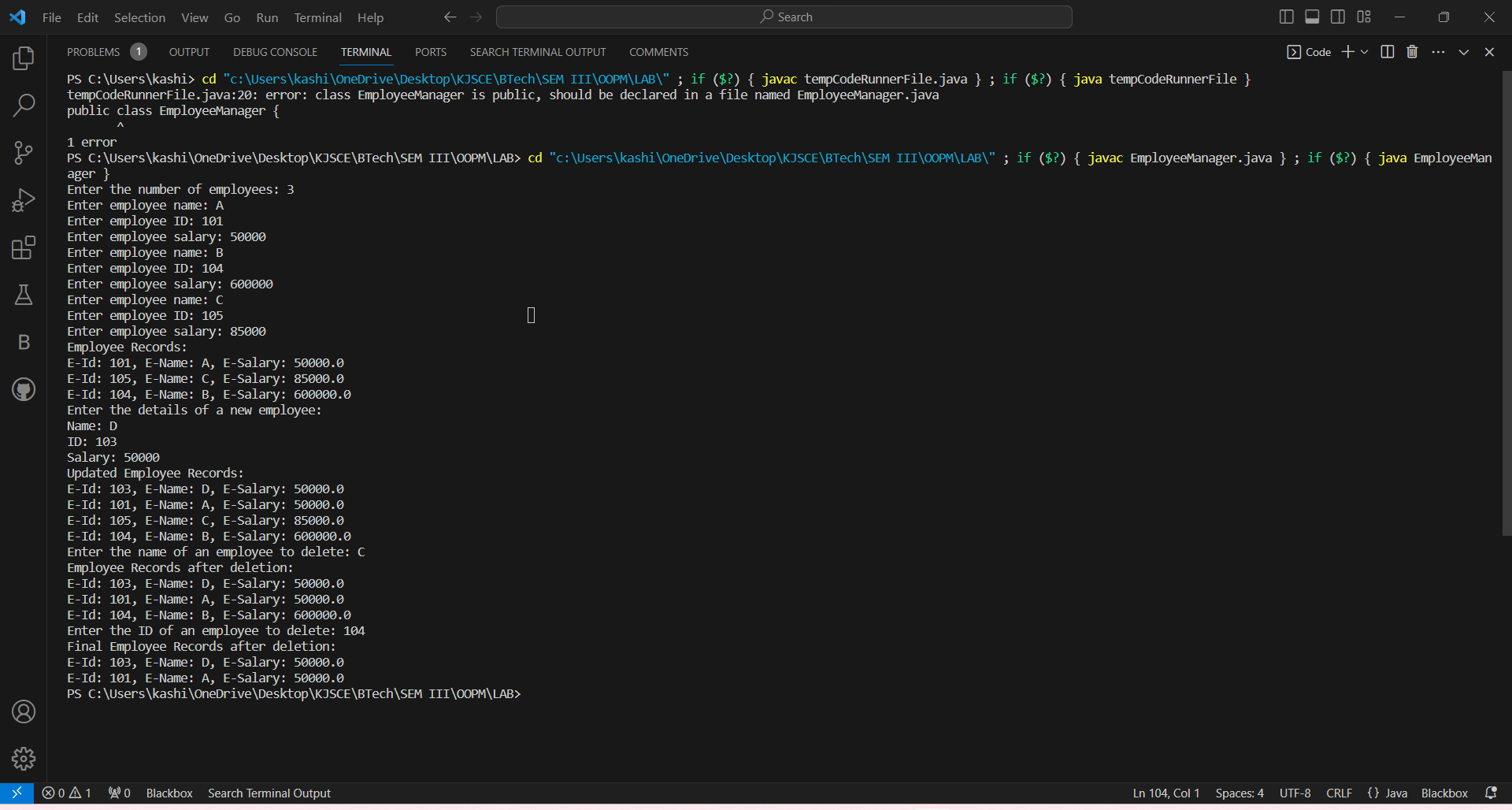
        manager.display();

        scanner.close();

    }

}

**Output:**



**Conclusion:**

We learned the use of Vector function.

**Date: 25/09/2023 Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. **What is the output of the following Program**

|  |
| --- |
| import java.util.\*;  class demo2 {      public static void main(String[] args)      {          Vector v = new Vector(20);          v.addElement(“Geeksforgeeks”);          v.insertElementAt(“Java”, 2);          System.out.println(v.firstElement());      }  } |

**Output:**

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2 > 1

at java.base/java.util.Vector.insertElementAt(Vector.java:620)

at demo2.main(demo2.java:7)

1. **Expain any 10 methods of Vector class in detail with the help of example**

**Ans:**

1. **add(E element)**:

* This method is used to add an element to the end of the vector.
* Example:

Vector<String> names = new Vector<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

1. **add(int index, E element)**:

* Adds an element at a specified index in the vector, shifting the existing elements to the right.
* Example:

Vector<Integer> numbers = new Vector<>();

numbers.add(0, 1); // Insert 1 at index 0

numbers.add(1, 2); // Insert 2 at index 1

1. **remove(int index)**:

* Removes the element at the specified index and shifts the subsequent elements to the left.
* Example:

Vector<Double> prices = new Vector<>();

prices.add(10.5);

prices.add(15.0);

prices.remove(0); // Removes the element at index 0 (10.5)

1. **get(int index)**:

* Retrieves the element at the specified index.
* Example:

Vector<Integer> scores = new Vector<>();

scores.add(85);

scores.add(92);

int secondScore = scores.get(1); // Retrieves the element at index 1 (92)

1. **set(int index, E element)**:

* Replaces the element at the specified index with a new element.
* Example:

Vector<String> fruits = new Vector<>();

fruits.add("Apple");

fruits.add("Banana");

fruits.set(1, "Cherry"); // Replaces "Banana" with "Cherry" at index 1

1. **size()**:

* Returns the number of elements in the vector.
* Example:

Vector<Character> vowels = new Vector<>();

vowels.add('A');

vowels.add('E');

int size = vowels.size(); // Returns 2

1. **isEmpty()**:

* Checks if the vector is empty and returns a boolean value.
* Example:

Vector<Integer> data = new Vector<>();

boolean empty = data.isEmpty(); // Returns true

1. **contains(Object o)**:

* Checks if the vector contains a specified element and returns a boolean value.
* Example:

Vector<String> colors = new Vector<>();

colors.add("Red");

colors.add("Green");

boolean hasRed = colors.contains("Red"); // Returns true

1. **indexOf(Object o)**:

* Returns the index of the first occurrence of a specified element in the vector.
* Example:

Vector<Integer> nums = new Vector<>();

nums.add(10);

nums.add(20);

int index = nums.indexOf(20); // Returns 1

1. **clear()**:

* Removes all elements from the vector, leaving it empty.
* Example:

Vector<String> elements = new Vector<>();

elements.add("One");

elements.add("Two");

elements.clear(); // Clears all elements