|  |
| --- |
| NAME: Ayush Vinod Upadhyay  ROLL NO: I025  SAP ID: 60003220131  BRANCH: Information Technology  BATCH: 1 |

**EXPERIMENT NO. 01**

**CO/LO: CO1-** Modify the behaviour of methods, classes, and interfaces at runtime.

**AIM / OBJECTIVE:** To implement different collection types

**AIM:** Implementation of Stack and Queues using Arrays.

**DESCRIPTION OF EXPERIMENT:**

Collection in Java

**PROCEDURE / ALGORITHM:**

1. Write a Java program to create a vector that stores names of 10 employees, later add 2 new employees, then remove eighth employee from vector, copy the vector to another vector, display the values and size of new vector

**Code:**

import java.util.\*;

public class vector {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Vector<String> v1 = new Vector<String>();

        // Vector<String> v2 = new Vector<String>();

        System.out.println("Enter names of 10 employees:");

        String s = "";

        for (int i = 0; i < 10; i++) {

            s = sc.nextLine();

            v1.add(s);

        }

v1.add("e11");

        v1.add("e12");

        System.out.println("two employees are added");

        v1.remove(7);

        System.out.println("employee 8 is deleted");

        Object v2 = v1.clone();

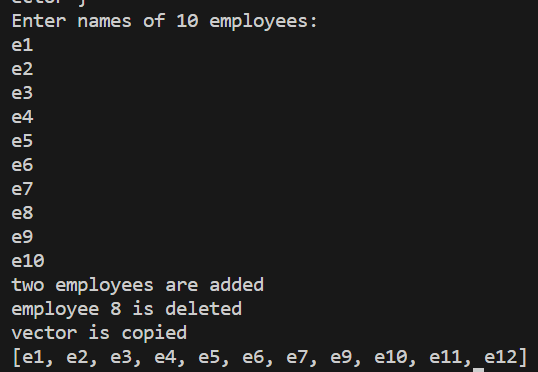
        System.out.println("vector is copied");

        System.out.println(v2);

    }

}

**OUTPUT:**

****

1. In this problem we have given you three classes in the editor:

· Student class

· Rockstar class

· Hacker class

In the main method, we populated an ArrayList with several instances of these classes. count method calculates how many instances of each type is present in the ArrayList. (Use instanceof)

*Sample Input*

Student

Student

Rockstar

Student

Hacker

*Sample Output*

3 1

**Code:**

import java.util.\*;

class Student {}

class Rockstar {}

class Hacker {}

public class P2 {

    public static void main(String[] args) {

        ArrayList<Object> people = new ArrayList<Object>();

        people.add(new Student());

        people.add(new Student());

        people.add(new Hacker());

        people.add(new Student());

        people.add(new Rockstar());

        count(people);

    }

    public static void count(ArrayList<Object> people) {

        int studentCount = 0;

        int rockstarCount = 0;

        int hackerCount = 0;

        for (Object person : people) {

            if (person instanceof Student) {

                studentCount++;

            } else if (person instanceof Rockstar) {

                rockstarCount++;

            } else if (person instanceof Hacker) {

                hackerCount++;

            }

        }

        System.out.println("Student count: " + studentCount);

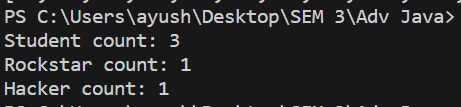
        System.out.println("Rockstar count: " + rockstarCount);

        System.out.println("Hacker count: " + hackerCount);

    }

}

**OUTPUT:**

****

1. 3. You are given n pairs of strings. Two pairs (a,b) and (c,d) are identical if a=c and b=d. That also implies (a,b) is not same as (b,a). After taking each pair as input, you need to print a number of unique pairs you currently have. Print n lines. In the ith line, print number of unique pairs you have after taking ith pair as input.

**Sample Input:**

5

Ayush Dhruv

Ayush Advait

Advait Shivam

Shivam Dhruv

Ayush Dhruv

**Sample Output:**

**CODE:**

import java.util.\*;

public class P1 {

    public static void main(String[] args) {

        HashSet<String> hs = new HashSet<String>();

        System.out.println("enter number of strings");

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int count = -1;

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

            if (hs.add(sc.nextLine())) {

                count++;

            }

            System.out.println("unique strings:" + count);

        }

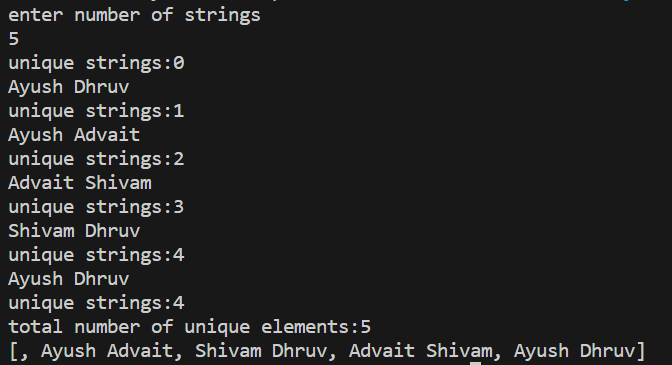
        System.out.println("total number of unique elements:" + hs.size());

        System.out.println(hs);

    }

}

**OUTPUT:**

****

1. WAP to convert an infix expression to postfix and evaluate the same. [input in the form of 5\*7+8-3]

**Code:**

import java.util.\*;

public class infix {

    private static boolean isOperator(char c) {

        return c == '+' || c == '-' || c == '\*' || c == '/';

    }

    private static int getPrecedence(char operator) {

        switch (operator) {

            case '+':

            case '-':

                return 1;

            case '\*':

            case '/':

                return 2;

            default:

                return 0;

        }

    }

    public static String infixToPostfix(String infix) {

        StringBuilder postfix = new StringBuilder();

        Stack<Character> operatorStack = new Stack<>();

        for (char c : infix.toCharArray()) {

            if (Character.isDigit(c)) {

                postfix.append(c);

            } else if (c == '(') {

                operatorStack.push(c);

            } else if (c == ')') {

                while (!operatorStack.isEmpty() && operatorStack.peek() != '(') {

                    postfix.append(operatorStack.pop());

                }

                operatorStack.pop(); // Pop '('

            } else if (isOperator(c)) {

                while (!operatorStack.isEmpty() && getPrecedence(c) <= getPrecedence(operatorStack.peek())) {

                    postfix.append(operatorStack.pop());

                }

                operatorStack.push(c);

            }

        }

        while (!operatorStack.isEmpty()) {

            postfix.append(operatorStack.pop());

        }

        return postfix.toString();

    }

    public static int evaluatePostfix(String postfix) {

        Stack<Integer> operandStack = new Stack<>();

        for (char c : postfix.toCharArray()) {

            if (Character.isDigit(c)) {

                operandStack.push(c - '0');

            } else if (isOperator(c)) {

                int operand2 = operandStack.pop();

                int operand1 = operandStack.pop();

                int result = performOperation(c, operand1, operand2);

                operandStack.push(result);

            }

        }

        return operandStack.pop();

    }

    private static int performOperation(char operator, int operand1, int operand2) {

        switch (operator) {

            case '+':

                return operand1 + operand2;

            case '-':

                return operand1 - operand2;

            case '\*':

                return operand1 \* operand2;

            case '/':

                if (operand2 == 0) {

                    throw new ArithmeticException("Division by zero");

                }

                return operand1 / operand2;

            default:

                throw new IllegalArgumentException("Invalid operator");

        }

    }

    public static void main(String[] args) {

        String infixExpression = "5\*7+8-3";

        String postfixExpression = infixToPostfix(infixExpression);

        System.out.println("Postfix expression:" + postfixExpression);

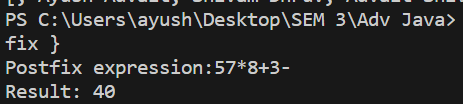
        int result = evaluatePostfix(postfixExpression);

        System.out.println("Result: " + result);

    }

}

**OUTPUT:**

****

1. Write a java program that maintains a separate linked list denoting marks of students from two divisions. Perform the following operations on Linked List:

a. Merge both the list

b. Sort

c. Minimum and maximum

d. Split failed and passed

e. Count number of students getting above 90 marks

**CODE:**

import java.util.\*;

class Node {

    int data;

    Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class LinkedList {

    Node head;

    public LinkedList() {

        this.head = null;

    }

    public void insert(int data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

    }

    public void display() {

        Node current = head;

        while (current != null) {

            System.out.print(current.data + " ");

            current = current.next;

        }

        System.out.println();

    }

    public void merge(LinkedList otherList) {

        Node current = head;

        while (current.next != null) {

            current = current.next;

        }

        current.next = otherList.head;

    }

    public void sort() {

        if (head == null || head.next == null) {

            return;

        }

        Node current = head;

        while (current != null) {

            Node index = current.next;

            while (index != null) {

                if (current.data > index.data) {

                    int temp = current.data;

                    current.data = index.data;

                    index.data = temp;

                }

                index = index.next;

            }

            current = current.next;

        }

    }

    public int getMinimum() {

        if (head == null) {

            return -1;

        }

        int min = head.data;

        Node current = head.next;

        while (current != null) {

            if (current.data < min) {

                min = current.data;

            }

            current = current.next;

        }

        return min;

    }

    public int getMaximum() {

        if (head == null) {

            return -1;

        }

        int max = head.data;

        Node current = head.next;

        while (current != null) {

            if (current.data > max) {

                max = current.data;

            }

            current = current.next;

        }

        return max;

    }

    public LinkedList splitPassedAndFailed(int passThreshold) {

        LinkedList passedList = new LinkedList();

        Node current = head;

        while (current != null) {

            if (current.data >= passThreshold) {

                passedList.insert(current.data);

            }

            current = current.next;

        }

        current = head;

        while (current != null) {

            if (current.data < passThreshold) {

                current.data = -1;

            }

            current = current.next;

        }

        return passedList;

    }

    public int countAboveThreshold(int threshold) {

        int count = 0;

        Node current = head;

        while (current != null) {

            if (current.data > threshold) {

                count++;

            }

            current = current.next;

        }

        return count;

    }

}

public class P4 {

    public static void main(String[] args) {

        LinkedList division1 = new LinkedList();

        division1.insert(85);

        division1.insert(95);

        division1.insert(70);

        division1.insert(60);

        division1.insert(80);

        LinkedList division2 = new LinkedList();

        division2.insert(75);

        division2.insert(90);

        division2.insert(65);

        division2.insert(55);

        division2.insert(100);

        System.out.println("Division 1 marks:");

        division1.display();

        System.out.println("Division 2 marks:");

        division2.display();

        division1.merge(division2);

        System.out.println("Merged marks:");

        division1.display();

        division1.sort();

        System.out.println("Sorted marks:");

        division1.display();

        int minMark = division1.getMinimum();

        int maxMark = division1.getMaximum();

        System.out.println("Minimum Mark: " + minMark);

        System.out.println("Maximum Mark: " + maxMark);

        int passThreshold = 60;

        LinkedList passedList = division1.splitPassedAndFailed(passThreshold);

        System.out.println("Passed Marks:");

        passedList.display();

        System.out.println("Failed Marks:");

        division1.display();

        int above90Count = division1.countAboveThreshold(90);

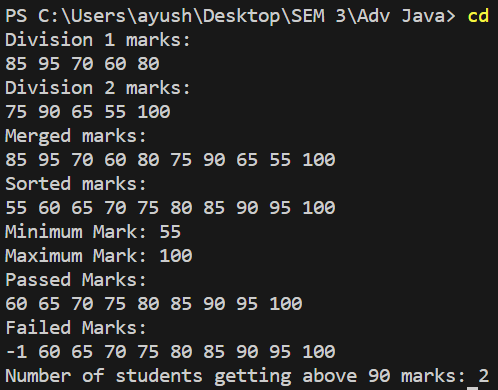
        System.out.println("Number of students getting above 90 marks: " +

                above90Count);

    }

}

**OUTPUT:**

****

1. There are a number of students in a school who wait to be served. Two types of events, ENTER and SERVED, can take place which are described below. ENTER: A student with some priority enters the queue to be served. SERVED: The student with the highest priority is served (removed) from the queue. A unique id is assigned to each student entering the queue. The queue serves the students based on the following criteria (priority criteria): The student having the highest Cumulative Grade Point Average (CGPA) is served first. Any students having the same CGPA will be served by name in ascending case-sensitive alphabetical order. Any students having the same CGPA and name will be served in ascending order of the id. Create the following two classes: The Student class should implement: The constructor Student(int id, String name, double cgpa). The method int getID() to return the id of the student. The method String getName() to return the name of the student. The method double getCGPA() to return the CGPA of the student. The Priorities class should implement the method List getStudents(List events) to process all the given events and return all the students yet to be served in the priority order.

**CODE:**

import java.util.\*;

class Student {

    private int id;

    private String name;

    private double cgpa;

    public Student(int id, String name, double cgpa) {

        this.id = id;

        this.name = name;

        this.cgpa = cgpa;

    }

    public int getID() {

        return id;

    }

    public String getName() {

        return name;

    }

    public double getCGPA() {

        return cgpa;

    }

}

class Priorities {

    public List<Student> getStudents(List<String> events) {

        PriorityQueue<Student> priorityQueue = new PriorityQueue<>(

                (a, b) -> {

                    if (a.getCGPA() != b.getCGPA()) {

                        return Double.compare(b.getCGPA(), a.getCGPA()); // Higher CGPA first

                    } else if (!a.getName().equals(b.getName())) {

                        return a.getName().compareTo(b.getName()); // Alphabetical order

                    } else {

                        return Integer.compare(a.getID(), b.getID()); // Lower ID first

                    }

                });

        for (String event : events) {

            String[] parts = event.split(" ");

            if (parts[0].equals("ENTER")) {

                String name = parts[1];

                double cgpa = Double.parseDouble(parts[2]);

                int id = Integer.parseInt(parts[3]);

                Student student = new Student(id, name, cgpa);

                priorityQueue.add(student);

            } else if (parts[0].equals("SERVED")) {

                priorityQueue.poll(); // Remove the highest priority student

            }

        }

        List<Student> students = new ArrayList<>(priorityQueue.size());

        while (!priorityQueue.isEmpty()) {

            students.add(priorityQueue.poll());

        }

        return students;

    }

}

public class P5 {

    public static void main(String[] args) {

        Priorities priorities = new Priorities();

        List<String> events = Arrays.asList(

                "ENTER John 3.75 50",

                "ENTER Mark 3.5 45",

                "ENTER Steve 3.75 35",

                "SERVED",

                "SERVED",

                "ENTER Andy 3.85 55",

                "ENTER Bob 3.65 44",

                "SERVED");

        List<Student> students = priorities.getStudents(events);

        for (Student student : students) {

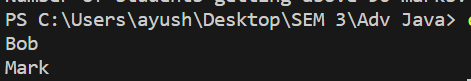
            System.out.println(student.getName());

        }

    }

}

**OUTPUT:**

****

**OBSERVATION:**

**CONCLUSION:** I learned the implementation of Collection in JAVA and its its application via performing the experiments.