Lab Sheet 07

02). Answer

```
public class LinkedListStats {

    private class Node {
        int data;
        Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    private Node head;
    private int size;

    public LinkedListStats() {
        this.head = null;
        this.size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public void insertLast(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
        size++;
    }
```

```
    public int getSize() {
        return size;
    }

    public double getMean() {
        if (isEmpty()) return 0;
        Node current = head;
        double sum = 0;
        while (current != null) {
            sum += current.data;
            current = current.next;
        }
        return sum / size;
    }

    public double getMedian() {
        if (isEmpty()) return 0;
        bubbleSort();
        Node middle = getNodeAtPosition(size / 2);
        if (size % 2 == 0) {
            Node middlePrev = getNodeAtPosition(size / 2 - 1);
            return (middlePrev.data + middle.data) / 2.0;
        } else {
            return middle.data;
        }
    }

    public int getMode() {
        if (isEmpty()) return 0;
        bubbleSort();
        Node current = head;
        int mode = current.data;
        int maxCount = 0;
        int currentCount = 0;
        int currentValue = current.data;

        while (current != null) {
            if (current.data == currentValue) {
                currentCount++;
            } else {
                if (currentCount > maxCount) {
                    maxCount = currentCount;
                    mode = currentValue;
                }
                currentValue = current.data;
```

```
                currentCount = 1;
            }
            current = current.next;
        }

        if (currentCount > maxCount) {
            mode = currentValue;
        }

        return mode;
    }

    public int getRange() {
        if (isEmpty()) return 0;
        Node current = head;
        int min = current.data;
        int max = current.data;
        while (current != null) {
            if (current.data < min) {
                min = current.data;
            }
            if (current.data > max) {
                max = current.data;
            }
            current = current.next;
        }
        return max - min;
    }

    private Node getNodeAtPosition(int position) {
        Node current = head;
        for (int i = 0; i < position; i++) {
            current = current.next;
        }
        return current;
    }

    private void bubbleSort() {
        if (isEmpty() || head.next == null) {
            return;
        }
        boolean swapped;
        do {
            Node current = head;
            Node prev = null;
```

```
            Node next = null;
            swapped = false;

            while (current.next != null) {
                next = current.next;
                if (current.data > next.data) {
                    swapped = true;
                    if (prev != null) {
                        Node temp = next.next;
                        prev.next = next;
                        next.next = current;
                        current.next = temp;
                    } else {
                        Node temp = next.next;
                        head = next;
                        next.next = current;
                        current.next = temp;
                    }
                    prev = next;
                } else {
                    prev = current;
                    current = next;
                }
            }
        } while (swapped);
    }

    public static void main(String[] args) {
        LinkedListStats list = new LinkedListStats();
        list.insertLast(10);
        list.insertLast(9);
        list.insertLast(52);
        list.insertLast(24);
        list.insertLast(35);
        list.insertLast(11);
        list.insertLast(9);
        list.insertLast(12);
        list.insertLast(3);
        list.insertLast(11);
        list.insertLast(25);
        list.insertLast(24);
        list.insertLast(8);
        list.insertLast(11);
        list.insertLast(42);
```

```
        System.out.println("Mean: " + list.getMean());
        System.out.println("");
        System.out.println("Median: " + list.getMedian());
        System.out.println("");
        System.out.println("Mode: " + list.getMode());
        System.out.println("");
        System.out.println("Range: " + list.getRange());
        System.out.println("");
    }
}
```

Output:-

```
 '-cp' 'C:\Users\desit\AppData\Roaming\Code\User\workspaceSto
Mean: 19.066666666666666

Median: 11.0

Mode: 11

Range: 49

PS D:\MY University Doc UOK\2nd Year 1st Semester\COSC 21063
```

02) Answer

```
public class ScoreCalculator {
    private class Node {
        String name;
        int score;
        int time;
        Node next;

        public Node(String name, int score, int time) {
            this.name = name;
            this.score = score;
            this.time = time;
            this.next = null;
        }
    }

    private Node head;
    private int size;

    public ScoreCalculator(){
        this.head = null;
        this.size = 0;
    }

    public boolean isEmpty(){
        return head == null;
    }

    public void insertLast(String name, int score, int time){
        Node newNode = new Node(name, score, time);
        if(isEmpty()){
            head = newNode;
        }else{
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
        size++;
    }
```

```
    public int ListSize(){
        return size;
    }

    public void traverseList(){
        if(isEmpty()){
            System.out.println("List is empty");
        }else{
            Node current = head;
            System.out.println("Participant\tScore\tTime (minutes)");
            while (current != null) {
                System.out.println(current.name + "\t\t" + current.score + "\t" +
current.time );
                current = current.next;
            }
            System.out.println("");
        }
    }

    public String getHighestScorer(){
        if(isEmpty()){
            return "List is empty";
        }else{
            Node current = head;
            Node highestScorer = head;
            while (current != null) {
                if(current.score > highestScorer.score){
                    highestScorer = current;
                }
                current = current.next;
            }
            return highestScorer.name + " with a score of " +
highestScorer.score;
        }
    }

    public String getFastestParticipant(){
        if(isEmpty()){
            return "List is empty";
        }else{
            Node current = head;
            Node fastestParticipant = head;
            while (current != null) {
                if(current.time < fastestParticipant.time){
                    fastestParticipant = current;
```

```
                }
                current = current.next;
            }
            return fastestParticipant.name + " with a time of " +
fastestParticipant.time + " minutes";
        }
    }
    public void displayEfficiency(){
        if(isEmpty()){
            System.out.println("List is empty");
        }else{
            Node current = head;
            System.out.println("Participant\tEfficiency");
            while (current != null) {
                double efficiency = (double) current.score / current.time;
                System.out.println(current.name + "\t\t" + efficiency);
                current = current.next;
            }
            System.out.println("");
        }
    }
    public String getMostEfficientParticipant() {
        if(isEmpty()){
            return "List is empty";
        } else {
            Node current = head;
            Node mostEfficient = head;
            double highestEfficiency = (double) head.score / head.time;
            while (current != null) {
                double currentEfficiency = (double) current.score / current.time;
                if (currentEfficiency > highestEfficiency) {
                    highestEfficiency = currentEfficiency;
                    mostEfficient = current;
                }
                current = current.next;
            }
            return mostEfficient.name + " with an efficiency of " +
highestEfficiency;
        }
    }

    public static void main(String[] args) {
        ScoreCalculator Slist = new ScoreCalculator();
        Slist.insertLast("Bob", 35, 40);
        Slist.insertLast("Diana", 94, 57);
```

```
        Slist.insertLast("Jon", 90, 60);
        Slist.insertLast("Mary", 56, 49);
        Slist.insertLast("Charlie", 87, 52);

        Slist.traverseList();
        System.out.println("(02) a)-------Answer");
        System.out.println("Highest score: " + Slist.getHighestScorer());
        System.out.println("");
        System.out.println("(02) b)-------Answer");
        System.out.println("");
        System.out.println("Fastest completion time: " +
Slist.getFastestParticipant());
        System.out.println("");
        System.out.println("(02) c)-------Answer");
        System.out.println("");
        Slist.displayEfficiency();
        System.out.println("");
        System.out.println("(02) d)-------Answer");
        System.out.println("");
        System.out.println("Most efficient participant: " +
Slist.getMostEfficientParticipant());
        System.out.println("");
    }
}
```

Output:-

```
orithms\Peactical Tutorial\Labsheet-07'; & 'C:\Program Files\Java\jdk-17\bin\java.e
 '-cp' 'C:\Users\desit\AppData\Roaming\Code\User\workspaceStorage\2edcaff97c6a79466
Participant     Score   Time (minutes)
Bob             35      40
Diana           94      57
Jon             90      60
Mary            56      49
Charlie         87      52

(02) a)-------Answer
Highest score: Diana with a score of 94

(02) b)-------Answer

Fastest completion time: Bob with a time of 40 minutes

(02) c)-------Answer

Participant     Efficiency
Bob             0.875
Diana           1.6491228070175439
Jon             1.5
Mary            1.1428571428571428
Charlie         1.6730769230769231


(02) d)-------Answer

Most efficient participant: Charlie with an efficiency of 1.6730769230769231

PS D:\MY University Doc UOK\2nd Year 1st Semester\COSC 21063 Data Structure & Algor
```

Link:- https://github.com/Dp-Sathsara/Labsheet-07.git