LAB 07

Singly Linked List

Lab Tasks

1. Write a program that can store 10 records of students in a link list manner and apply the following operations on it.
   a. View the list
   b. Insert the elements in different locations of linked list and view it.
   c. Search any element from the linked list
   d. Delete record again view the list after deletion.

Source Code :

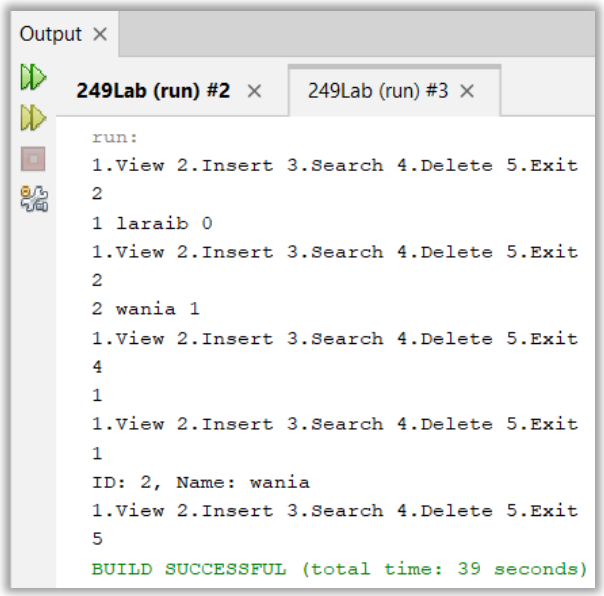
```
class Student {
    int id;
    String name;
    Student next;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}class StudentLinkedList {
    Student head;

    void viewList() {
 for (Student current = head; current != null; current = current.next)
 System.out.println("ID: " + current.id + ", Name: " + current.name);
    }
void insert(int id, String name, int pos) {
        Student newStudent = new Student(id, name);
        if (pos == 0) {
            newStudent.next = head;
            head = newStudent;
        } else {
            Student current = head;
            for (int i = 0; i < pos - 1 && current != null; i++)
                current = current.next;
            if (current != null) {
                newStudent.next = current.next;
                current.next = newStudent;
            }}}void search(int id) {
     for (Student current = head; current != null; current = current.next)
            if (current.id == id) {
 System.out.println("Found: ID " + current.id + ", Name " + current.name);
                return;
            }System.out.println("Not Found");
    } void delete(int id) {
        if (head != null && head.id == id) {
            head = head.next;
```

```
        } else {
            Student current = head;
            while (current != null && current.next != null &&
current.next.id != id)
                current = current.next;
            if (current != null && current.next != null)
                current.next = current.next.next;
        }}}public class lab249 {
    public static void main(String[] args) {
        StudentLinkedList list = new StudentLinkedList();
        Scanner sc = new Scanner(System.in);
        while (true) {
        System.out.println("1.View 2.Insert 3.Search 4.Delete 5.Exit");
            switch (sc.nextInt()) {
                case 1 -> list.viewList();
                case 2 -> list.insert(sc.nextInt(), sc.next(),
sc.nextInt());
                case 3 -> list.search(sc.nextInt());
                case 4 -> list.delete(sc.nextInt());
                case 5 -> { sc.close(); return; }
            }
        }
    }
}
```

Output :



2. Write a java program to merge two equal linkedlists using runner technique.

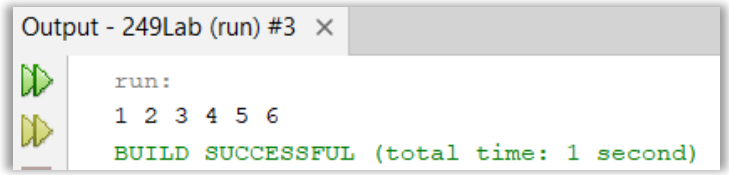
Source Code :

```
class Node {
    int value;
    Node next;

    Node(int value) {
        this.value = value;
    }}class LinkedList {
    Node head;
    void add(int value) {
```

```
        Node newNode = new Node(value);
        if (head == null) head = newNode;
        else {
            Node current = head;
            while (current.next != null) current = current.next;
            current.next = newNode;
        }
    }void printList() {
        for (Node current = head; current != null; current = current.next)
            System.out.print(current.value + " ");
        System.out.println();
    }static LinkedList merge(LinkedList l1, LinkedList l2) {
        LinkedList merged = new LinkedList();
        for (Node n1 = l1.head, n2 = l2.head; n1 != null && n2 != null; n1
= n1.next, n2 = n2.next) {
            merged.add(n1.value);
            merged.add(n2.value);
        }
        return merged;
    }
}public class lab249 {
    public static void main(String[] args) {
        LinkedList l1 = new LinkedList(), l2 = new LinkedList();
        l1.add(1); l1.add(3); l1.add(5);
        l2.add(2); l2.add(4); l2.add(6);
        LinkedList merged = LinkedList.merge(l1, l2);
        merged.printList();
    }
}
```

Output :



3. Write a program to check whether the linkedlist is empty or not.

Source Code :

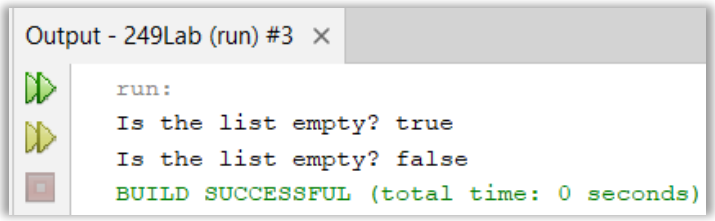
```
class Node {
    int value;
    Node next;
    Node(int value) {
        this.value = value;
    }
}class LinkedList {
    Node head;
    boolean isEmpty() {
        return head == null;
    }void add(int value) {
        Node newNode = new Node(value);
        if (head == null) head = newNode;
```

```
        else {
            Node current = head;
            while (current.next != null) current = current.next;
            current.next = newNode;
        }
    }}public class lab249 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        System.out.println("Is the list empty? " + list.isEmpty());

        list.add(1);
        System.out.println("Is the list empty? " + list.isEmpty());
    }
}
```

Output :



4. You are managing a list of integers in a class, and you need to implement a **Singly Linked List** with the following operations:

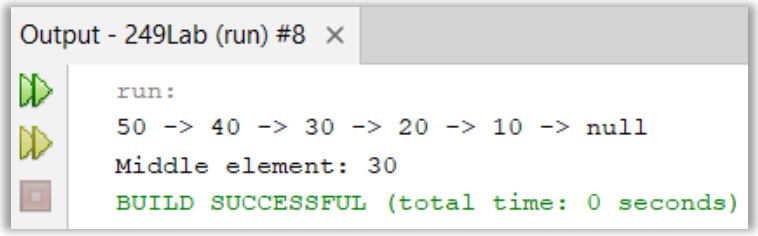
   a) **Insert** an integer at the **beginning** of the list.
   b) **Display** the list.
   c) Find the **middle element** of the list. If the list has an even number of elements, return the **first middle element**.

Source Code :

```
public class lab249 {
    private Node head;
    class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }public void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }public void displayList() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }Node current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
```

```
            current = current.next;
        }System.out.println("null");
    }public int findMiddle() {
    if (head == null) throw new IllegalStateException("The list is empty.");
        Node slowPointer = head, fastPointer = head;
        while (fastPointer != null && fastPointer.next != null) {
            slowPointer = slowPointer.next;
            fastPointer = fastPointer.next.next;
        }return slowPointer.data;
    }
public static void main(String[] args) {
        lab249 list = new lab249();
        list.insertAtBeginning(10);
        list.insertAtBeginning(20);
        list.insertAtBeginning(30);
        list.insertAtBeginning(40);
        list.insertAtBeginning(50);
        list.displayList();
        System.out.println("Middle element: " + list.findMiddle());
    }
}
```

Output :



Home Task for Singly linked list

1. Write a program that reads the name, age and salary of 10 persons and perform the following operations on it.
   a. Insert the elements in different locations of linked list and view it.
   b. Delete record and again view the list after deletion.

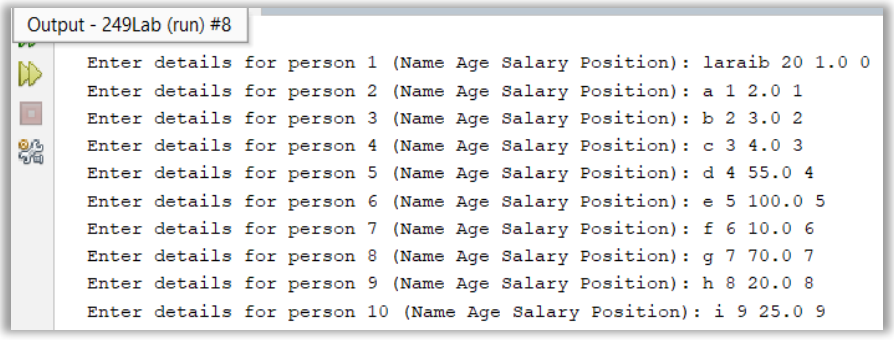
Source Code :

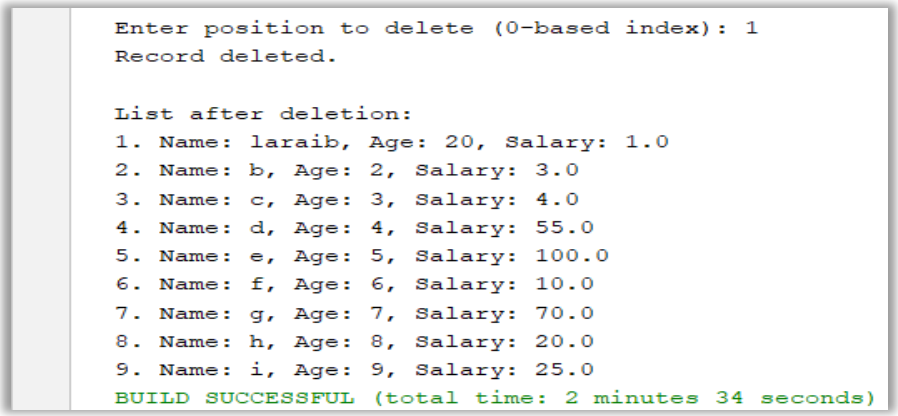
```
class Person {
    String name;
    int age;
    double salary;
    Person(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Salary: " + salary;
    }
}public class Lab249 {
    public static void main(String[] args) {
        LinkedList<Person> persons = new LinkedList<>();
```

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

        for (int i = 0; i < 10; i++) {
System.out.print("Enter details for person " + (i + 1) + " (Name Age Salary
Position): ");
            String[] details = scanner.nextLine().split(" ");
            String name = details[0];
            int age = Integer.parseInt(details[1]);
            double salary = Double.parseDouble(details[2]);
            int position = Integer.parseInt(details[3]);
            if (position < 0 || position > persons.size()) {
                System.out.println("Invalid position. Adding to the end.");
                position = persons.size();
            }persons.add(position, new Person(name, age, salary));
        }System.out.println("\nCurrent list:");
        displayList(persons);
        System.out.print("\nEnter position to delete (0-based index): ");
        int deletePosition = Integer.parseInt(scanner.nextLine());
        if (deletePosition >= 0 && deletePosition < persons.size()) {
            persons.remove(deletePosition);
            System.out.println("Record deleted.");
        } else {
            System.out.println("Invalid position. No record deleted.");
        }private static void displayList(LinkedList<Person> persons) {
        if (persons.isEmpty()) {
            System.out.println("The list is empty.");
        } else {
            for (int i = 0; i < persons.size(); i++) {
                System.out.println((i + 1) + ". " + persons.get(i));
            }
        }
    }
}
```

Output :





2. You are tasked with managing a list of **students' roll numbers** in a class. Initially, the list is empty. You have to implement a **Singly Linked List** with the following operations:

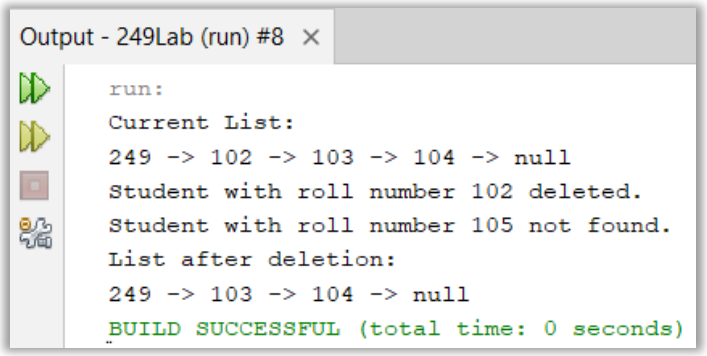
   a) **Add student roll number** at the **end** of the list.
   b) **Delete a student by roll number**.
   c) **Display the roll numbers** of all students in the class

Source Code :

```
public class lab249 {
 private Node head;
 class Node {
     int rollNumber;
     Node next;
     public Node(int rollNumber) {
         this.rollNumber = rollNumber;
         this.next = null;
     }
 }public void addStudent(int rollNumber) {
     Node newNode = new Node(rollNumber);
     if (head == null) {
         head = newNode;
     } else {
         Node current = head;
         while (current.next != null) {
             current = current.next;
         }current.next = newNode;
     }
 }public void deleteStudent(int rollNumber) {
     if (head == null) {
         System.out.println("The list is empty.");
         return;
     }if (head.rollNumber == rollNumber) {
         head = head.next;
System.out.println("Student with roll number " +rollNumber + " deleted.");
         return;
     }Node current = head;
while (current.next != null && current.next.rollNumber != rollNumber) {
         current = current.next;
     }
if (current.next == null) {
System.out.println("Student with roll number " +rollNumber+ "not found.");
     } else {
         current.next = current.next.next;
System.out.println("Student with roll number " +rollNumber + " deleted.");
     }
 }public void displayList() {
     if (head == null) {
         System.out.println("The list is empty.");
         return;
     }Node current = head;
```

```
        while (current != null) {
            System.out.print(current.rollNumber + " -> ");
            current = current.next;
        }System.out.println("null");
    }
    public static void main(String[] args) {
        lab249 list = new lab249();
        list.addStudent(249);
        list.addStudent(102);
        list.addStudent(103);
        list.addStudent(104);
        System.out.println("Current List:");
        list.displayList();
        list.deleteStudent(102);
        list.deleteStudent(105);
        System.out.println("List after deletion:");
        list.displayList();
    }
}
```

Output :



3. You are managing two **singly linked lists** representing **two groups of students**. Your task is to:

a) **Append** the second list to the first list (i.e., add all elements of the second list to the end of the first list).
b) **Count the number of students** in the final list (i.e., the total number of nodes in the list).
c) **Display the final list** after the append operation.

Source Code :

```
public class lab249 {
    private Node head;
    class Node {
        int rollNumber;
        Node next;
        public Node(int rollNumber) {
            this.rollNumber = rollNumber;
            this.next = null;
        }
    }public lab249() {
        head = null;
    }public void append(int rollNumber) {
```

```
        Node newNode = new Node(rollNumber);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) current = current.next;
            current.next = newNode;
        }
    }public void appendList(lab249 otherList) {
        if (head == null) head = otherList.head;
        else {
            Node current = head;
            while (current.next != null) current = current.next;
            current.next = otherList.head;
        }
    }public int countStudents() {
        int count = 0;
        Node current = head;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }public void displayList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.rollNumber + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }public static void main(String[] args) {
        lab249 list1 = new lab249();
        lab249 list2 = new lab249();
        list1.append(101);
        list1.append(102);
        list2.append(103);
        list2.append(104);
        System.out.println("List 1:");
        list1.displayList();
        System.out.println("List 2:");
        list2.displayList();
        list1.appendList(list2);
        System.out.println("\nFinal List:");
        list1.displayList();
        System.out.println("\nTotal students: " + list1.countStudents());
    }
}
```

Output :