LAB # 07

Singly Linked List Implementation

Objective: Implementing singly linked list, associated operations and Runer technique.

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

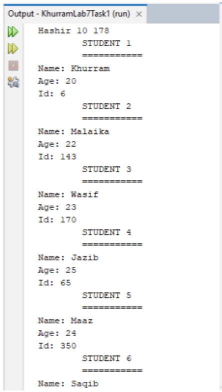
d. Delete record again view the list after deletion.

Source Code:

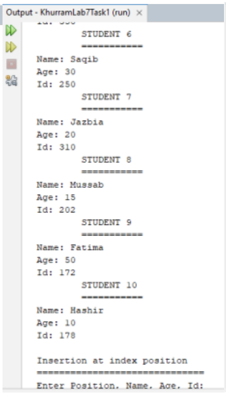
```
import java.util.Scanner;
public class KhurramLab7Task1 {
    static class LinkedList {
        static class Node {
            String studentName;
            int age, id;
            Node next;
            Node(String name, int age, int id) { this.studentName = name; this.age = age; this.id = id; }}
        private Node head, tail;
        void addNode(String name, int age, int id) {
            Node newNode = new Node(name, age, id);
            if (head == null) head = tail = newNode;
            else { tail.next = newNode; tail = newNode; }}
        void display() {
            if (head == null) { System.out.println("List is empty."); return; }
            Node current = head; int counter = 1;
            while (current != null) {
                System.out.printf("\tSTUDENT %d\n\t===========\nName: %s\nAge: %d\nId: %d\n", counter++,
current.studentName, current.age, current.id);
                current = current.next;}}
        void insertAt(int index, String name, int age, int id) {
            Node newNode = new Node(name, age, id);
            if (index == 0) { newNode.next = head; head = newNode; if (tail == null) tail = newNode; return; }
            Node current = head; int counter = 0;
            while (current != null && counter++ < index - 1) current = current.next;
            if (current != null) {
                newNode.next = current.next; current.next = newNode;
                if (newNode.next == null) tail = newNode;
            } else System.out.println("Index out of bounds.");}
        void search(String name) {
            Node current = head; int pos = 1;
```

```
      while (current != null) {
        if (current.studentName.equalsIgnoreCase(name)) {
          System.out.println(name + " found at position: " + pos);
          return;}
        current = current.next; pos++;}
      System.out.println("Student not found.");}
    void deleteAt(int index) {
      if (head == null) { System.out.println("List is empty."); return; }
      if (index == 0) { head = head.next; if (head == null) tail = null; return; }
      Node current = head; int counter = 0;
      while (current != null && counter++ < index - 1) current = current.next;
      if (current != null && current.next != null) {
        current.next = current.next.next;
        if (current.next == null) tail = current;
      } else System.out.println("Index out of bounds.");}}
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    LinkedList list = new LinkedList();
    System.out.println("Displaying Students Information of linked list");
    System.out.println("=================================================");
    System.out.println("Enter Records of 10 Students:");
    for (int i = 1; i <= 10; i++) {
      System.out.println("STUDENT " + i + ":\nEnter Name, Age, and Id:");
      list.addNode(sc.next(), sc.nextInt(), sc.nextInt());}
    list.display();
    System.out.println("\nInsertion at index position ");
    System.out.println("================================");
    System.out.println("Enter Position, Name, Age, Id:");
    list.insertAt(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextInt());
    list.display();
    System.out.println("\nSearching Student");
    System.out.println("===================");
    System.out.println("Enter the name of the student you want to search:");
    list.search(sc.next());
    System.out.println("\nDeletion at position ");
    System.out.println("=======================");
    System.out.println("Enter the position of the student you want to delete:");
    list.deleteAt(sc.nextInt());
    list.display();}}
```

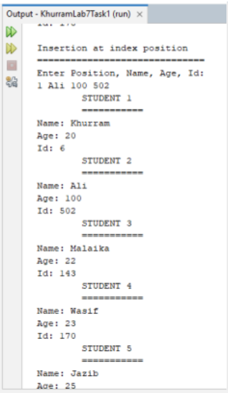
Output:


Output - KhurramLab7Task1 (run)

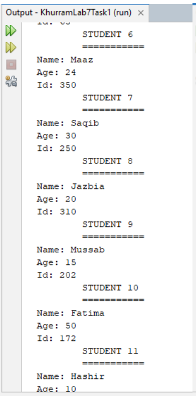
```
Hashir 10 178
           STUDENT 1
           ===========
Name: Khurram
Age: 20
Id: 6
           STUDENT 2
           ===========
Name: Malaika
Age: 22
Id: 143
           STUDENT 3
           ===========
Name: Wasif
Age: 23
Id: 170
           STUDENT 4
           ===========
Name: Jazib
Age: 25
Id: 65
           STUDENT 5
           ===========
Name: Maaz
Age: 24
Id: 350
           STUDENT 6
           ===========
Name: Saqib
```


Output - KhurramLab7Task1 (run)

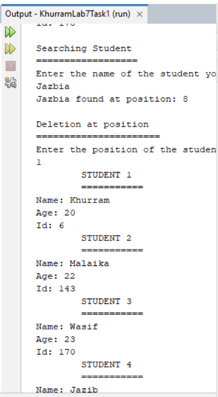
```
           STUDENT 6
           ===========
Name: Saqib
Age: 30
Id: 250
           STUDENT 7
           ===========
Name: Jazbia
Age: 20
Id: 310
           STUDENT 8
           ===========
Name: Mussab
Age: 15
Id: 202
           STUDENT 9
           ===========
Name: Fatima
Age: 50
Id: 172
           STUDENT 10
           ===========
Name: Hashir
Age: 10
Id: 178

Insertion at index position
================================
Enter Position, Name, Age, Id:
```


Output - KhurramLab7Task1 (run)

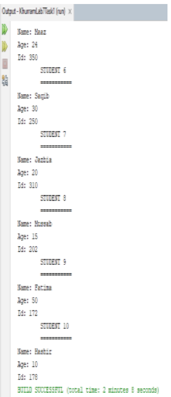
```
Insertion at index position
================================
Enter Position, Name, Age, Id:
1 Ali 100 502
           STUDENT 1
           ===========
Name: Khurram
Age: 20
Id: 6
           STUDENT 2
           ===========
Name: Ali
Age: 100
Id: 502
           STUDENT 3
           ===========
Name: Malaika
Age: 22
Id: 143
           STUDENT 4
           ===========
Name: Wasif
Age: 23
Id: 170
           STUDENT 5
           ===========
Name: Jazib
Age: 25
```


Output - KhurramLab7Task1 (run)

```
           STUDENT 6
           ===========
Name: Maaz
Age: 24
Id: 350
           STUDENT 7
           ===========
Name: Saqib
Age: 30
Id: 250
           STUDENT 8
           ===========
Name: Jazbia
Age: 20
Id: 310
           STUDENT 9
           ===========
Name: Mussab
Age: 15
Id: 202
           STUDENT 10
           ===========
Name: Fatima
Age: 50
Id: 172
           STUDENT 11
           ===========
Name: Hashir
Age: 10
```


Output - KhurramLab7Task1 (run)

```
Searching Student
===================
Enter the name of the student yo
Jazbia
Jazbia found at position: 8

Deletion at position
=======================
Enter the position of the studen
1
           STUDENT 1
           ===========
Name: Khurram
Age: 20
Id: 6
           STUDENT 2
           ===========
Name: Malaika
Age: 22
Id: 143
           STUDENT 3
           ===========
Name: Wasif
Age: 23
Id: 170
           STUDENT 4
           ===========
Name: Jazib
```


Output - KhurramLab7Task1 (run)

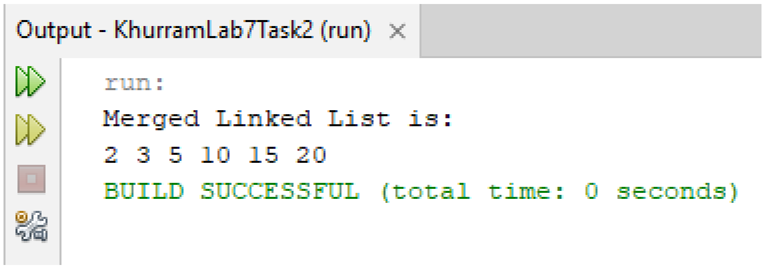
```
Name: Maaz
Age: 24
Id: 350
           STUDENT 6
           ==========
Name: Saqib
Age: 30
Id: 250
           STUDENT 7
           ==========
Name: Jazbia
Age: 20
Id: 310
           STUDENT 8
           ==========
Name: Mussab
Age: 15
Id: 202
           STUDENT 9
           ==========
Name: Fatima
Age: 50
Id: 172
           STUDENT 10
           ==========
Name: Hashir
Age: 10
Id: 178
BUILD SUCCESSFUL (total time: 2 minutes 8 seconds)
```

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

Source Code:

```
package khurramlab7task2;
public class KhurramLab7Task2 {
    public static void main(String[] args) {
        MergeLists llist1 = new MergeLists();
        MergeLists llist2 = new MergeLists();
        llist1.add(5, 10, 15);
        llist2.add(2, 3, 20);
        llist1.head = new Gfg().sortedMerge(llist1.head, llist2.head);
        System.out.println("Merged Linked List is:");
        llist1.printList();}}
class Node {
    int data;
    Node next;
    Node(int d) { data = d; }}
class MergeLists {
    Node head;
    void add(int... values) {
        for (int val : values) addToLast(new Node(val));}
    void addToLast(Node node) {
        if (head == null) head = node;
        else {
            Node temp = head;
            while (temp.next != null) temp = temp.next;
            temp.next = node;}}
    void printList() {
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
        System.out.println();}}
class Gfg {
    Node sortedMerge(Node headA, Node headB) {
        Node dummyNode = new Node(0), tail = dummyNode;
        while (headA != null && headB != null) {
            if (headA.data <= headB.data) {
                tail.next = headA; headA = headA.next;
            } else {
                tail.next = headB; headB = headB.next;}
            tail = tail.next;}
        tail.next = (headA != null) ? headA : headB;
        return dummyNode.next;}}
```

Output:



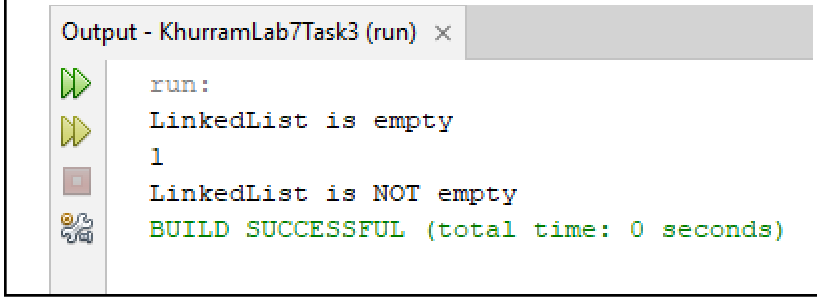
```
run:
Merged Linked List is:
2 3 5 10 15 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

Source Code:

```
          package khurramlab7task3;
          import java.util.LinkedList;
          public class KhurramLab7Task3 {
             public static void main(String[] args) {
                LinkedList<String> fruitsList = new LinkedList<>();
                System.out.println(fruitsList.isEmpty() ? "LinkedList is empty" : "LinkedList is NOT empty");
                fruitsList.add("APPLE");
                System.out.println(fruitsList.size());
     System.out.println(fruitsList.isEmpty() ? "LinkedList is empty" : "LinkedList is NOT empty");}}
```

Output:



```
run:
LinkedList is empty
1
LinkedList is NOT empty
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

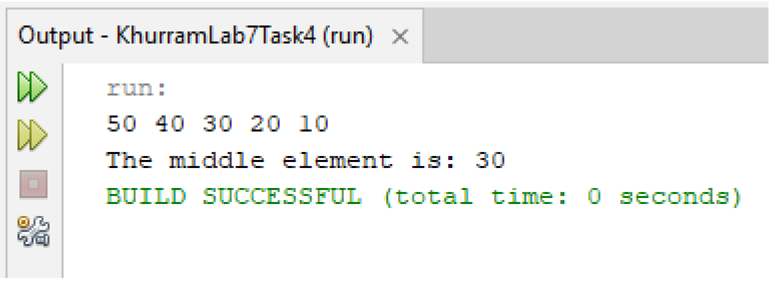
a) **Insert** an integer at the **beginning** of the list. b) **Display** the list. c) Find the **middle element** of the list. If the list has an even number of elements, return the **first middle element**.

Source Code:

```
package khurramlab7task4;

public class KhurramLab7Task4 {

   static class SinglyLinkedList {

      static class Node {

         int data;

         Node next;

         Node(int data) { this.data = data; }}

      private Node head;

      void insertAtBeginning(int data) {

         Node newNode = new Node(data);

         newNode.next = head;

         head = newNode;}
```

```
    void display() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;}
        for (Node temp = head; temp != null; temp = temp.next) {
            System.out.print(temp.data + " ");}
        System.out.println();}
    int findMiddle() {
        if (head == null) throw new IllegalStateException("The list is empty.");
        Node slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;}
        return slow.data;}}
  public static void main(String[] args) {
    SinglyLinkedList list = new SinglyLinkedList();
    list.insertAtBeginning(10);
    list.insertAtBeginning(20);
    list.insertAtBeginning(30);
    list.insertAtBeginning(40);
    list.insertAtBeginning(50);
    list.display();
    System.out.println("The middle element is: " + list.findMiddle());}}
```

Output:



```
run:
50 40 30 20 10
The middle element is: 30
BUILD SUCCESSFUL (total time: 0 seconds)
```

Home Tasks:

1. Write a program that reads the name, age and salary of 10 persons and perform the following operations on it.

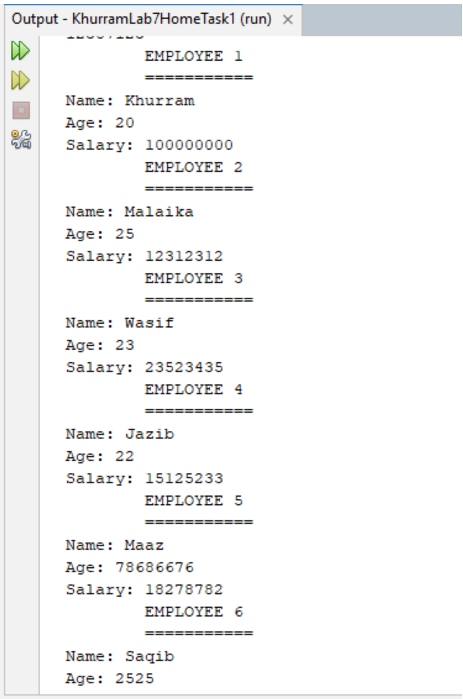
a. Insert the elements in different locations of linked list and view it.

b. Delete record and again view the list after deletion.

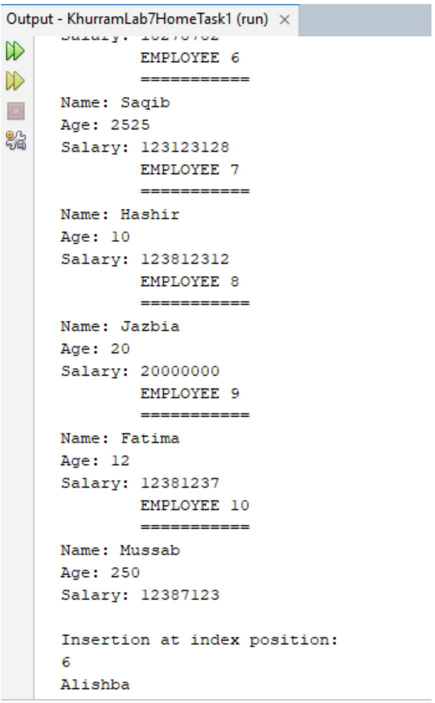
Source Code:

```
import java.util.Scanner;
public class KhurramLab7HomeTask1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedList list = new LinkedList();
        System.out.println("Enter Records of 10 Employees:");
        for (int i = 1; i <= 10; i++) {
            System.out.println("EMPLOYEE " + i + ":");
            list.addNode(sc.next(), sc.nextInt(), sc.nextInt());}
        list.display();
        System.out.println("\nInsertion at index position:");
        list.insertAt(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextInt());
        list.display();
        System.out.println("\nDeletion at position:");
        list.deleteAt(sc.nextInt());}
    static class LinkedList {
        class Node {
            String name;
            int age, salary;
            Node next;
            Node(String name, int age, int salary) { this.name = name; this.age = age; this.salary = salary; }}
        private Node head = null;
        void addNode(String name, int age, int salary) {
            Node newNode = new Node(name, age, salary);
            if (head == null) head = newNode;
            else {
                Node temp = head;
                while (temp.next != null) temp = temp.next;
                temp.next = newNode;}}
        void display() {
            if (head == null) { System.out.println("List is empty"); return; }
            Node current = head;
            int counter = 1;
            while (current != null) {
                System.out.printf("\tEMPLOYEE %d\n\t===========\nName: %s\nAge: %d\nSalary: %d\n",
counter++, current.name, current.age, current.salary);
                current = current.next;}}
        void insertAt(int index, String name, int age, int salary) {
            if (index < 1) { System.out.println("Invalid index."); return; }
            Node newNode = new Node(name, age, salary);
            if (index == 1) { newNode.next = head; head = newNode; return; }
            Node current = head;
            for (int i = 1; current != null && i < index - 1; i++) current = current.next;
            if (current == null) { System.out.println("Index out of bounds."); return; }
            newNode.next = current.next;
            current.next = newNode;}
        void deleteAt(int index) {
            if (index < 1 || head == null) { System.out.println("Invalid index or empty list."); return; }
            if (index == 1) { head = head.next; return; }
            Node current = head;
            for (int i = 1; current != null && i < index - 1; i++) current = current.next;
            if (current == null || current.next == null) { System.out.println("Index out of bounds."); return; }
            current.next = current.next.next;}}}
```

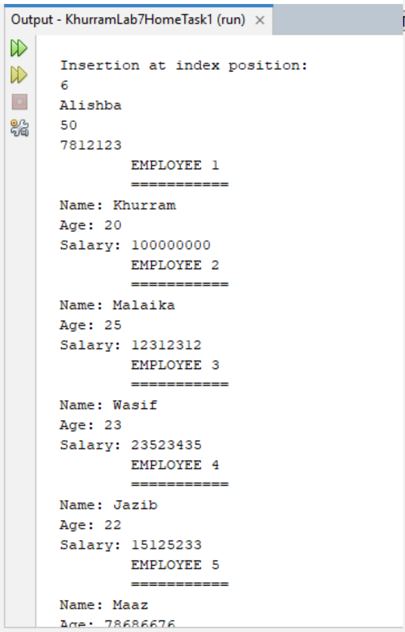
Output:



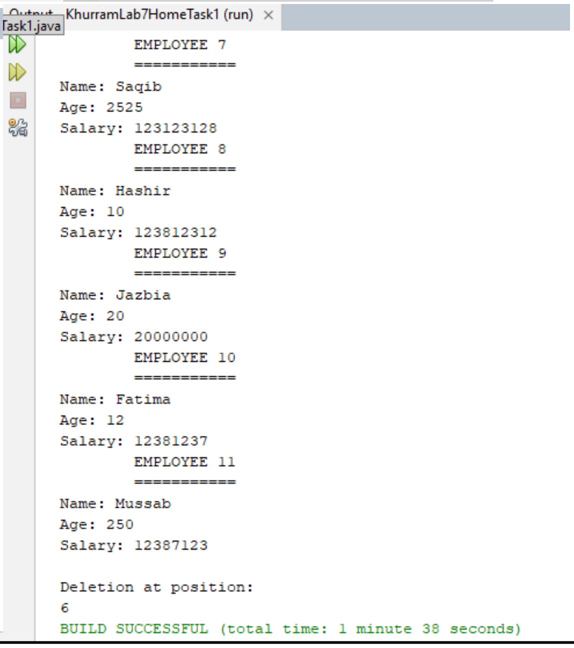
```
 12387123
          EMPLOYEE 1
          ===========
Name: Khurram
Age: 20
Salary: 100000000
          EMPLOYEE 2
          ===========
Name: Malaika
Age: 25
Salary: 12312312
          EMPLOYEE 3
          ===========
Name: Wasif
Age: 23
Salary: 23523435
          EMPLOYEE 4
          ===========
Name: Jazib
Age: 22
Salary: 15125233
          EMPLOYEE 5
          ===========
Name: Maaz
Age: 78686676
Salary: 18278782
          EMPLOYEE 6
          ===========
Name: Saqib
Age: 2525
```



```
Salary: 18278782
          EMPLOYEE 6
          ===========
Name: Saqib
Age: 2525
Salary: 123123128
          EMPLOYEE 7
          ===========
Name: Hashir
Age: 10
Salary: 123812312
          EMPLOYEE 8
          ===========
Name: Jazbia
Age: 20
Salary: 20000000
          EMPLOYEE 9
          ===========
Name: Fatima
Age: 12
Salary: 12381237
          EMPLOYEE 10
          ===========
Name: Mussab
Age: 250
Salary: 12387123

Insertion at index position:
6
Alishba
```



```
Insertion at index position:
6
Alishba
50
7812123
         EMPLOYEE 1
         ===========
Name: Khurram
Age: 20
Salary: 100000000
         EMPLOYEE 2
         ===========
Name: Malaika
Age: 25
Salary: 12312312
         EMPLOYEE 3
         ===========
Name: Wasif
Age: 23
Salary: 23523435
         EMPLOYEE 4
         ===========
Name: Jazib
Age: 22
Salary: 15125233
         EMPLOYEE 5
         ===========
Name: Maaz
Age: 78686676
```



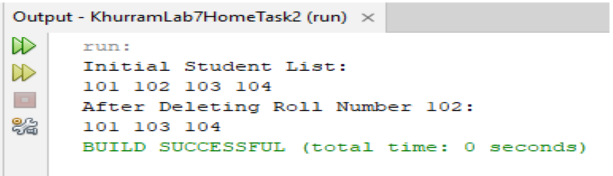
```
         EMPLOYEE 7
         ===========
Name: Saqib
Age: 2525
Salary: 123123128
         EMPLOYEE 8
         ===========
Name: Hashir
Age: 10
Salary: 123812312
         EMPLOYEE 9
         ===========
Name: Jazbia
Age: 20
Salary: 20000000
         EMPLOYEE 10
         ===========
Name: Fatima
Age: 12
Salary: 12381237
         EMPLOYEE 11
         ===========
Name: Mussab
Age: 250
Salary: 12387123

Deletion at position:
6
BUILD SUCCESSFUL (total time: 1 minute 38 seconds)
```

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:

```
package khurramlab7hometask2;
public class KhurramLab7HomeTask2 {
    static class Node {
        int rollNumber;
        Node next;
        Node(int rollNumber) { this.rollNumber = rollNumber; }}
    static Node head = null;
    static void addStudent(int rollNumber) {
        Node newNode = new Node(rollNumber);
        if (head == null) head = newNode;
        else {
            Node temp = head;
            while (temp.next != null) temp = temp.next;
            temp.next = newNode;}}
    static void deleteStudent(int rollNumber) {
        if (head == null) return;
        if (head.rollNumber == rollNumber) head = head.next;
        else {
            Node temp = head;
            while (temp.next != null && temp.next.rollNumber != rollNumber) temp = temp.next;
            if (temp.next != null) temp.next = temp.next.next;}}
    static void displayList() {
        if (head == null) System.out.println("No students in the list.");
        else {
            for (Node temp = head; temp != null; temp = temp.next) {
                System.out.print(temp.rollNumber + " ");}
            System.out.println();}}
    public static void main(String[] args) {
        addStudent(101);
        addStudent(102);
        addStudent(103);
        addStudent(104);
        System.out.println("Initial Student List:");
        displayList();
        deleteStudent(102);
        System.out.println("After Deleting Roll Number 102:");
        displayList();}}
```

Output:



```
run:
Initial Student List:
101 102 103 104
After Deleting Roll Number 102:
101 103 104
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

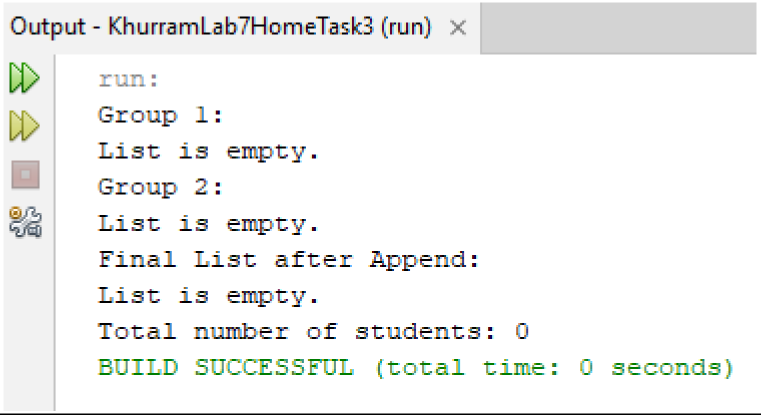
A doubly linked list/2-way LL is often more convenient.

- Nodes store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes.

Source Code:

```
package khurramlab7hometask3;
public class KhurramLab7HomeTask3 {
  static class Node {
    int rollNumber;
    Node next, prev;
    Node(int rollNumber) { this.rollNumber = rollNumber; }}
  static class StudentGroupManagement {
    Node head1 = null, head2 = null;
    void addStudentToGroup(Node head, int rollNumber) {
      Node newNode = new Node(rollNumber);
      if (head == null) head = newNode;
      else {
        Node temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = newNode;
        newNode.prev = temp;}}
    void appendLists() {
      if (head1 == null) head1 = head2;
      else {
        Node temp = head1;
        while (temp.next != null) temp = temp.next;
        temp.next = head2;
        if (head2 != null) head2.prev = temp;}}
    int countStudents() {
      int count = 0;
      for (Node temp = head1; temp != null; temp = temp.next) count++;
      return count;}
    void displayList(Node head) {
      if (head == null) {
        System.out.println("List is empty.");
        return;}
      for (Node temp = head; temp != null; temp = temp.next) {
        System.out.print(temp.rollNumber + " ");}
      System.out.println();}}
    public static void main(String[] args) {
        StudentGroupManagement manager = new StudentGroupManagement();
        manager.addStudentToGroup(manager.head1, 101);
        manager.addStudentToGroup(manager.head1, 102);
        manager.addStudentToGroup(manager.head1, 103);
        manager.addStudentToGroup(manager.head2, 201);
        manager.addStudentToGroup(manager.head2, 202);
        System.out.println("Group 1:");
        manager.displayList(manager.head1);
        System.out.println("Group 2:");
        manager.displayList(manager.head2);
        manager.appendLists();
        System.out.println("Final List after Append:");
        manager.displayList(manager.head1);
  System.out.println("Total number of students: " + manager.countStudents());}}
```

Output:



```
run:
Group 1:
List is empty.
Group 2:
List is empty.
Final List after Append:
List is empty.
Total number of students: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```