```java
package LinkedListPracticum;

public class Node08 {
    int data;
    Node08 next;

    public Node08(int score, Node08 next){
        this.data=score;
        this.next=next;
    }
}
```

```java
package LinkedListPracticum;

public class MainSLL08 {
    public static void main(String[] args) {
        SingleLinkedList08 singLL1 = new SingleLinkedList08();

        singLL1.print();
        singLL1.addFirst(89);
        singLL1.print();
        singLL1.addLast(76);
        singLL1.print();
        singLL1.addFirst(70);
        singLL1.print();
        singLL1.insertAfter(70, 99);
        singLL1.print();
        singLL1.insertAt(3, 83);
        singLL1.print();
    }
}
```

```java
package LinkedListPracticum;

public class SingleLinkedList08 {
    Node08 head;
    Node08 tail;

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

    public void print() {
        if (!isEmpty()) {
            Node08 tmp = head;
            System.out.println("Data on Linked List:\t");
            while (tmp != null) {
```

```
            System.out.println(tmp.data + "\t");
            tmp = tmp.next;
        }
        System.out.println("");
    } else {
        System.out.println("Linked List is empty");
    }
}

public void addFirst(int input) {
    Node08 ndInput = new Node08(input, null);
    if (!isEmpty()) {
        head = ndInput;
        tail = ndInput;
    } else {
        ndInput.next = head;
        head= ndInput;
    }
}

public void addLast(int input){
    Node08 ndInput = new Node08(input, null);
    if (isEmpty()){
        head = ndInput;
        tail = ndInput;
    } else {
        tail.next = ndInput;
        tail = ndInput;
    }
}

public void insertAfter(int key, int input){
    Node08 ndInput = new Node08(input, null);
    Node08 temp = head;
    do {
        if(temp.data == key){
            ndInput.next = temp.next;
            temp.next = ndInput;
            if(ndInput.next == null) tail=ndInput;
            break;
        }
        temp = temp.next;
    } while (temp != null);
}

public void insertAt(int index,int input){
    if(index<0){
        System.out.println("wrong index");
```

```
    } else if (index == 0){
        addFirst(input);
    } else {
        Node08 temp = head;
        for (int i=0; i<index-1; i++){
            temp = temp.next;
        }
        temp.next = new Node08(input, temp.next);
        if(temp.next.next==null){
            tail=temp.next;
        }
    }
  }
}
```

```
System.out.println("indeks 1 data = " +singLL1.getData(1));
    System.out.println("Data with value = 76 in the index : " + singLL1.indexOf(76));
    singLL1.remove(990);
    singLL1.print();
    singLL1.remove(99);
    singLL1.print();
    singLL1.removeAt(0);
    singLL1.print();
    singLL1.removeFirst();
    singLL1.print();
    singLL1.removeLast();
    singLL1.print();

  }
}
```

```
public void removeFirst() {
    if (isEmpty()){
        System.out.println("Linked list is empty, cannot delete data!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
    }
  }

  public void removeLast(){
    if (isEmpty()){
        System.out.println("Linked list is empty, cannot delete data!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        Node08 temp = head;
```

```
        while (temp.next != tail){
            temp = temp.next;
        }
    }
}

public void remove (int key){
    if (isEmpty()){
        System.out.println("Linked List is empty, cannot delete data!");
    } else {
        Node08 temp = head;
        while (temp != null){
            if (temp.next==null){
                System.out.println("data to be deleted was not found");
                break;
            } else {
                if ((temp.data == key) && (temp == head)){
                    this.removeFirst();
                    break;
                }
            } temp = temp.next;
        }
    }
}

public void removeAt (int index){
    if(index == 0){
        removeFirst();
    } else {
        Node08 temp = head;
        for (int i=0; i<index-1; i++){
            temp = temp.next;
        }
        temp.next = temp.next.next;
        if (temp.next == null){
            tail = temp;
        }
    }
}
}
```

Question 1:

1. Why does the program code execution result in the first line produce “Linked List is Empty”?
   ⇨ because there is no input
2. In step 10, explain utility code following
   ```
   ndInput.next = temp.next;
   temp.next = ndInput;
   ```
   ⇨ advance temp by one element
3. Pay attention to SingleLinkedList00 class, in the insertAt method Explain the function of the following code

```
if(temp.next.next==null) {
    tail=temp.next;
}
```

⇨ if the condition is true then tail is advanced by 1 element

Question 2:

1. What is the function of the break keyword in the remove method? Explain!
   ⇨ to end an execution in a statement
2. Explain the function of the code below in the remove method

```
else if (temp.next.data == key) {
  temp.next = temp.next.next;
```

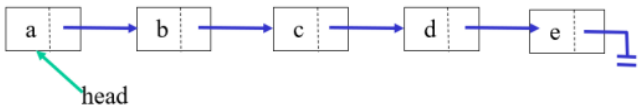
   ⇨ to delete the list corresponding to the keyword
3. What is the return value that can be returned in the indexOf method? Explain the meaning of each of these returns!
   ⇨ starts from 0 for the first character position. This method will return -1 if the character is not found

Assignment

1. Create insertBefore() method, this method is used to add a node before the desired keyword
   ⇨ Dibawah

```
void insertBefore(int key,Node input){
Node temp = head;
        while (temp != null){
                if ((temp.data == key)&&(temp == head)){
                        this.addFirst(input);
                        System.out.println("Insert data is succeed.");
                        break;
                }else if (temp.next.data == key){
                        input.next = temp.next;
                    temp.next = input;
                System.out.println("Insert data is succeed.");
                        break;
                }
                temp = temp.next;
        }
}
```

2. Implement The following linked list illustration . Use 4 kinds addition of data that has been studied previously for input data.




```
public void addFirst(int input) {
    Node08 ndInput = new Node08(input, null);
    if (!isEmpty()) {
      head = ndInput;
      tail = ndInput;
    } else {
      ndInput.next = head;
      head= ndInput;
    }
  }

  public void insertAt(int index,int input){
    if(index<0){
      System.out.println("wrong index");
```

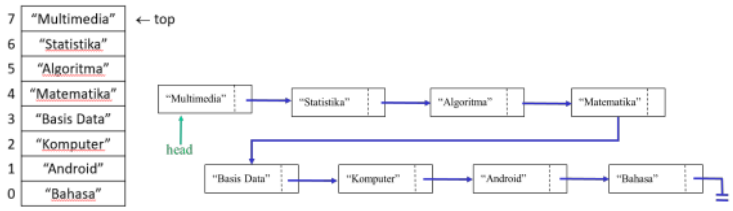
```
    } else if (index == 0){
      addFirst(input);
    } else {
      Node08 temp = head;
      for (int i=0; i<index-1; i++){
        temp = temp.next;
      }
      temp.next = new Node08(input, temp.next);
      if(temp.next.next==null){
        tail=temp.next;
      }
    }
  }
}

  public void insertAfter(int key, int input){
    Node08 temp = head;
    do {
      if(temp.data == key){
        Input.next = temp.next;
        temp.next = Input;
        if(Input.next == null) tail=Input;
        break;
      }
      temp = temp.next;
    } while (temp != null);
  }
```

```
Node temp = head;
        while (temp != null){
                if ((temp.data == key)&&(temp == head)){
                        this.addFirst(input);
                        System.out.println("Insert data is succeed.");
                        break;
                }else if (temp.next.data == key){
                        input.next = temp.next;
                    temp.next = input;
                System.out.println("Insert data is succeed.");
                        break;
                }
                temp = temp.next;
        }
}
```

```
  public void addLast(int input){
    Node08 ndInput = new Node08(input, null);
    if (isEmpty()){
      head = ndInput;
      tail = ndInput;
    } else {
```

```
        tail.next = ndInput;
        tail = ndInput;
    }
}
```

3. Create the following Stack Implementation using Single Linked List Concept.




```
1. import java.util.*;
2.
3. /*  Class Node  */
4. class Node
5. {
6.     protected int data;
7.     protected Node link;
8.
9.     /*  Constructor  */
10.     public Node()
11.     {
12.         link = null;
13.         data = 0;
14.     }
15.     /*  Constructor  */
16.     public Node(int d,Node n)
17.     {
18.         data = d;
19.         link = n;
20.     }
21.     /*  Function to set link to next Node  */
22.     public void setLink(Node n)
23.     {
24.         link = n;
25.     }
26.     /*  Function to set data to current Node  */
27.     public void setData(int d)
28.     {
29.         data = d;
30.     }
31.     /*  Function to get link to next node  */
32.     public Node getLink()
33.     {
34.         return link;
35.     }
```

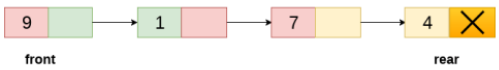
```
36.     /*  Function to get data from current Node  */
37.     public int getData()
38.     {
39.         return data;
40.     }
41. }
42.
43. /*  Class linkedStack  */
44. class linkedStack
45. {
46.     protected Node top ;
47.     protected int size ;
48.
49.     /*  Constructor  */
50.     public linkedStack()
51.     {
52.         top = null;
53.         size = 0;
54.     }
55.     /*  Function to check if stack is empty */
56.     public boolean isEmpty()
57.     {
58.         return top == null;
59.     }
60.     /*  Function to get the size of the stack */
61.     public int getSize()
62.     {
63.         return size;
64.     }
65.     /*  Function to push an element to the stack */
66.     public void push(int data)
67.     {
68.         Node nptr = new Node (data, null);
69.         if (top == null)
70.             top = nptr;
71.         else
72.         {
73.             nptr.setLink(top);
74.             top = nptr;
75.         }
76.         size++ ;
77.     }
78.     /*  Function to pop an element from the stack */
79.     public int pop()
80.     {
81.         if (isEmpty() )
```

```
            throw new NoSuchElementException("Underflow Exception") ;
        Node ptr = top;
        top = ptr.getLink();
        size-- ;
        return ptr.getData();
    }
    /*  Function to check the top element of the stack */
    public int peek()
    {
        if (isEmpty() )
            throw new NoSuchElementException("Underflow Exception") ;
        return top.getData();
    }
    /*  Function to display the status of the stack */
    public void display()
    {
        System.out.print("\nStack = ");
        if (size == 0)
                {
                    System.out.print("Empty\n");
                    return ;
                }
                Node ptr = top;
                while (ptr != null)
                {
                    System.out.print(ptr.getData()+" ");
                    ptr = ptr.getLink();
                }
                System.out.println();
            }
        }

        /* Class LinkedStackImplement */
        public class LinkedStackImplement
        {
            public static void main(String[] args)
            {
                Scanner scan = new Scanner(System.in);
                /* Creating object of class linkedStack */
                linkedStack ls = new linkedStack();
                /* Perform Stack Operations */
                System.out.println("Linked Stack Test\n");
                char ch;
                do
                {
                    System.out.println("\nLinked Stack Operations");
```

```
                System.out.println("1. push");
                System.out.println("2. pop");
                System.out.println("3. peek");
                System.out.println("4. check empty");
                System.out.println("5. size");
                int choice = scan.nextInt();
                switch (choice)
                {
                case 1 :
                    System.out.println("Enter integer element to push");
                    ls.push( scan.nextInt() );
                    break;
                case 2 :
                    try
                    {
                        System.out.println("Popped Element = "+ ls.pop());
                    }
                    catch (Exception e)
                    {
                        System.out.println("Error : " + e.getMessage());
                    }
                    break;
                case 3 :
                    try
                    {
                        System.out.println("Peek Element = "+ ls.peek());
                    }
                    catch (Exception e)
                    {
                        System.out.println("Error : " + e.getMessage());
                    }
                    break;
                case 4 :
                    System.out.println("Empty status = "+ ls.isEmpty());
                    break;
                case 5 :
                    System.out.println("Size = "+ ls.getSize());
                    break;
                case 6 :
                    System.out.println("Stack = ");
                    ls.display();
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
                }
```

```
                    /* display stack */
                    ls.display();
                    System.out.println("\nDo you want to continue (Type y or n) \n");
                    ch = scan.next().charAt(0);

                } while (ch == 'Y'|| ch == 'y');
            }
        }
```

4. Create queue program implementation for illustrate student who is request sign KRS hands on DPA lecturers on campus on assignments worksheets 8 using LinkedList. Implement Queue on queue college student with use LinkedList concept ! Illustration of linked queue




```
import java.util.Scanner;
public class std {
    int id;
    String nama,perlu;
    stf next;
    static Scanner in=new Scanner(System.in);
    static Scanner str=new Scanner(System.in);
    public void input(){
        System.out.print("Enter NIM  : ");
        nim=in.nextInt();
        System.out.print("Enter Name : ");
        name=str.nextLine();
        System.out.print("Enter need : ");
        need=str.nextLine();
        next=null;
    }
    public void read(){
        System.out.println("|| "+NIM+" \t|| "+name+" \t|| "+need+" \t||");
    }
    public static void main(String[] args){
        int menu=0;
        linked que=new linked();
        while(menu!=4){
            System.out.print("1.Enqueue\n2.Dequeue\n3.View\n4.Exit\n : ");
            menu=in.nextInt();
            if(menu==1)que.enque();
            else if(menu==2)que.deque();
            else if(menu==3)que.view();
            else if(menu==4)System.out.println("- PRINT  -");
            else System.out.println("- WRONG -");
            System.out.println("");
```

```
            }
        }
    }
class linked{
    std head,tail;
    public linked(){
        head=null;
        tail=null;
    }
    public void enque(){
        std baru=new std();
        baru.input();
        if(head==null)head=baru;
        else tail.next=baru;
        tail=baru;
    }
    public void deque(){
        if(head==null)System.out.println("- Empty -");
        else{
            System.out.println("Output data KRS with NIM : "+head.id);
            head=head.next;
        }
    }
    public void view(){
        if(head==null)System.out.println("- Empty -");
        else{
            System.out.println("|| NIM \t|| Name \t|| Need \t||");
            for(std a=head; a!=null; a=a.next) a.read();
        }
    }
}
```