Algorithm and Data Structure Practicum - Jobsheet IX

- Name: Dicha Zelianivan Arkana
- Class: 1i
- NIM: 2241720002

Lab Activities #1

Questions

1. Why the output of the program in the first line is "Linked List is Empty"?
2. Please explain the usage of the following code in:

   ```
   ndInput.next = temp.next;
   temp.next = ndInput;
   ```

   It's used to insert a new node in between 2 nodes. In this case it's the `temp` node and its neighbour.
3. In **SingleLinkedList**, what is the usage of the following code in **insertAt**?

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

   It's to set the tail if the next 2 node is null.

Lab Activities #2

Questions

1. Why we use the `break` keyword in remove function? Please explain! We use the `break` keyword because we want to stop traversing the list after removing the node that we want
2. Please explain why we implement these following code in method remove?

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

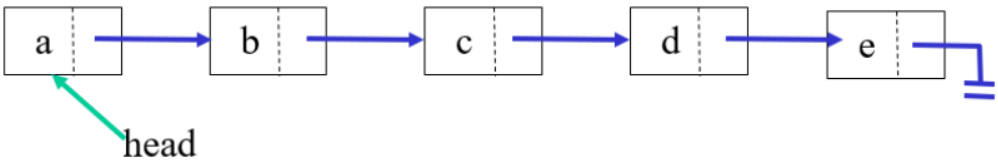
   If the next node data matches what we're looking for, we assign the current node neighbour to be the next's node neighbour to skip the node, effectively removing the next node, which is what we want.
3. What are the outputs of method `indexOf`? Please explain each of the output! It outputs the position of the node that contains the data that we're looking for.

Assignments

1. Create a method `insertBefore()` to add node before the desired keyword!

```
public void insertBefore(int key, int input) {
    if (isEmpty()) {
        head = tail = new Node(input, null);
        return;
    }

    if (head == tail) {
        head = new Node(input, head);
        return;
    }

    Node current = head;
    while (current.next != null) {
        if (current.next.data == key) {
            Node tmp = current.next;
            current.next = new Node(input, tmp);
            return;
        }
        current = current.next;
    }
}
```

2. Implement the linked list from this following:



- `Node.java`

```
public class Node<TData> {
    public TData value;
    public Node<TData> next;

    public Node(TData value) {
        this.value = value;
        this.next = null;
    }

    public Node(TData value, Node<TData> next) {
        this.value = value;
        this.next = next;
    }
}
```

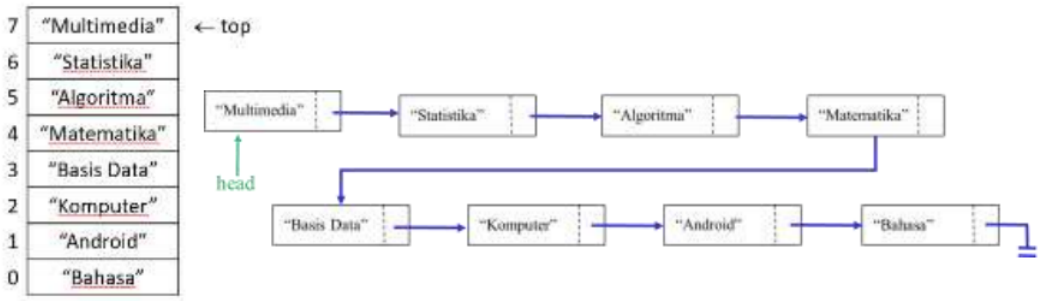
- **LinkedList.java**

```
public class LinkedList<TData> {
    Node<TData> head;
    Node<TData> tail;

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

```
public void print() {
    if (!isEmpty()) {
        Node<TData> tmp = head;
        System.out.println("Linked list content: \t");
        while (tmp != null) {
            System.out.print(tmp.data + "\t");
            tmp = tmp.next;
        }
        System.out.println("");
    } else {
        System.out.println("Linked list is empty");
    }
}

public void addFirst(TData input) {
    Node<TData> ndInput = new Node<TData>(input, null);
    if (isEmpty()) {
        head = ndInput;
        tail = ndInput;
    } else {
        ndInput.next = head;
        head = ndInput;
    }
}

public void addLast(TData input) {
    Node<TData> ndInput = new Node<TData>(input, null);
    if (isEmpty()) {
        head = ndInput;
        tail = ndInput;
    } else {
        tail.next = ndInput;
        tail = ndInput;
    }
}

public TData getData(int index) {
    Node<TData> tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}

public void removeFirst() {
    if (isEmpty()) {
        System.out.println("Linked list is empty. Cannot remove a 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 remove a data");
            } else if (head == tail) {
                head = tail = null;
            } else {
                Node<TData> temp = head;
                while (temp.next == null) {
                    temp = temp.next;
                }
                temp.next = null;
                tail = temp;
            }
        }
    }
```

3. Create this following `Stack` implementation using Linked List implementation




```
public class Stack {
    private final SingleLinkedList<String> list = new SingleLinkedList<>();

    public void push(String data) {
        list.addFirst(data);
    }

    public String pop() {
        String data = list.getData(0);
        list.removeFirst();
        return data;
    }

    public void print() {
        list.print();
    }
}
```

1. Create a program that helps bank customer using linked list with data are as follows:
   - name
   - address
   - customerAccountNumber

```
public class Customer {
    String name;
    String address;
    String customerAccountNumber;

    public Customer(String name, String address, String customerAccountNumber) {
        this.name = name;
        this.address = address;
        this.customerAccountNumber = customerAccountNumber;
    }
}
```

2. Implement **Queue** in previous number with **Linked List** concept

```
public class Queue {
    private final SingleLinkedList<Customer> list = new SingleLinkedList<>();

    public void enqueue(Customer data) {
        list.addLast(data);
    }

    public Customer dequeue() {
        Customer data = list.getData(0);
        list.removeFirst();
        return data;
    }

    public void print() {
        Node<Customer> current = list.head;
        while (current != null) {
            System.out.println("----");
            System.out.println("Name: " + current.data.name);
            System.out.println("Address: " + current.data.address);
            System.out.println(
                "Account Number: " +
                current.data.customerAccountNumber
            );
            current = current.next;
        }
    }
}
```