JOBSHEET XI
LINKED LIST

1. Learning Objectives

After completing this practical session, students will be able to:

1. Construct a linked list data structure
2. Implement a linked list in a program
3. Identify problems that can be effectively solved using a linked list

2. Labs Activities

2.1 Experiment 1: Implementing Single Linked List

Allocated times: 30 minutes

In this experiment, we will practice how to create a Single Linked List using Node-based data representation, access the linked list, and implement data insertion methods.

1. In the project created for this semester, create a new package named **week12**.
2. Create the following classes:
   a. `Student00.java`
   b. `Node00.java`
   c. `SingleLinkedList00.java`
   d. `SLLMain00.java`

   Modify 00 with your number
3. Implement **Student** following this class diagram:

Student
nim: String name: String className: String gpa: double
Student() Student (nm: String, name: String, kls: String, ip: double) print(): void

4. The following code is the Student class implementation

```
public class Student {
    String nim, name, className;
    double gpa;

    public Student(){
    }
    public Student(String nm, String nama, String kls, double ip){
```

```
        nim = nm;
        name = nama;
        className = kls;
        gpa = ip;
    }
    void print(){
        System.out.println(nim+" - "+name+" - "+className+" - "+gpa);
    }
}
```

5. The following code snippet depicts the **Node** class implementation. Please make a note that the **Node** class manages the **Student** object data, that is why we create **Student** attribute within **Node** class. **You will need to modify it based on the data type of data that you manage.**

```
public class Node {
   Student data;
   Node next;
   public Node(){
   }
   public Node(Sudent data, Node next){
    this.data = data;
    this.next = next;
   }
}
```

6. Create a new class named **SingleLinkedList**
7. Add **head** attributes within **SingleLinkedList** class

```
public class SingleLinkedList {
    Node head;
    Node tail;

}
```

8. In the next step we will implement the basic method of single linked list
9. Create a new method, **`isEmpty()`**.

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

10. Implement traversal process for displaying all data managed in the single linked list

```
void print(){
        if(!isEmpty()){
            Node tmp = head;
```

```
            System.out.println("LinkedList Data:");
            while(tmp!=null){
                tmp.data.print();
                tmp = tmp.next;
            }
        }else{
            System.out.println("LinkedList is empty!!");
        }
    }
```

11. Add **`addFirst()`** method as follows:

```
void addFirst(Student std){
        Node newNode = new Node(std, null);
        if(isEmpty()){
            head = newNode;
            tail = newNode;
        }else{
            newNode.next = head;
            head = newNode;
        }
    }
```

12. Create a new method, **`addLast()`** as follows:

```
void addLast(Student std){
        Node newNode = new Node(std, null);
        if(isEmpty()){
            head = newNode;
            tail = newNode;
        }else{
            tail.next = newNode;
            tail = newNode;
        }
    }
```

13. Add **`insertAfter()`**, to add a new student data after a **key** specified. In this example, we use student name as the key.

```
void insertAfter(Student std, String key){
        Node newNode = new Node(std, null);
        Node temp = head;
        do {
            if (temp.data.name.equalsIgnoreCase(key)) {
                newNode.next = temp.next;
                temp.next = newNode;
                if (newNode.next == null) {
                    tail = newNode;
                }
```

```
                break;
            }
            temp = temp.next;
        } while (temp != null);
    }
```

14. Add these following codes to add a node based on defined index

```
public void insertAt(int index, Student std) {
        if (index < 0) {
            System.out.println("Wrong index!!");
        } else if (index == 0) {
            addFirst(std);
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = new Node(std, temp.next);
            if (temp.next.next == null) {
                tail = temp.next;
            }
        }
    }
```

15. Create a new **Main** class, with a **`main`** method inside. Then create a **SingleLinkedList** object.

```
public static void main(String[] args) {
        SingleLinkedList sll = new SingleLinkedList();

    }
```

16. Add 4 student objects, **std1**, **std2**, **std3** and **std4**.

```
Student std1 = new Student("001", "Student 1", "TI-1I", 3.89);
Student std2 = new Student("002", "Student 2", "TI-1I", 3.45);
Student std3 = new Student("003", "Student 3", "TI-1I", 3.20);
Student std4 = new Student("004", "Student 4", "TI-1I", 3.00);
```

17. Add the students object into SingleLinkedList object, following these scenarios:

```
sll.print();
sll.addFirst(std4);
sll.print();
sll.addLast(std1);
sll.print();
sll.insertAfter(std3, "Student 4");
sll.insertAt(2, std2);
sll.print();
```

2.1.1 Output Verification

Observe the output of your program and compare it with the following output!

```
LinkedList is empty!!
LinkedList Data:
004 - Student 4 - TI-1I - 3.0
LinkedList Data:
004 - Student 4 - TI-1I - 3.0
001 - Student 1 - TI-1I - 3.89
LinkedList Data:
004 - Student 4 - TI-1I - 3.0
003 - Student 3 - TI-1I - 3.2
002 - Student 2 - TI-1I - 3.45
001 - Student 1 - TI-1I - 3.89
```

2.1.2 Questions!

1. Why does compiling the program code result in the message "Linked List is Empty" on the first line?
2. Explain the general purpose of the variable temp in each method!
3. Modify the code so that data can be added via keyboard input!
4. What would happen if we did not use the **tail** attribute? Would it affect the code implementation? Please explain.

2.2 Experiment 2: Accessing Element in Single Linked List

Allocated time: 30 minutes

In this practical session, we will learn how to access elements, retrieve their index, and perform data deletion in a Single Linked List.

2.2.1 Experiment Steps

1. Open **SingleLinkedList** class that is already created in Experiment 1.
2. Add a **getData()** method to get data at a specific index in the SingleLinkedList class.

```
Student getData(int idx){
        if(isEmpty()){
            System.out.println("LinkedList is empty!!");
            return null;
        }
        Node tmp = head;
        for(int i=0; i<idx;i++){
            tmp = tmp.next;
        }
        return tmp.data;
    }
```

3. Add a new method to get the index of a data specified. This method is named **`indexOf()`**.

```
int indexOf(String key){
        if(isEmpty()){
            System.out.println("LinkedList is empty!!");
            return -1;
        }
        Node tmp = head;
        int idx = 0;
        while(tmp != null && !tmp.data.name.equalsIgnoreCase(key)){
            tmp = tmp.next;
            idx++;
        }
        if(tmp == null){
            return -1;
        }else{
            return idx;
        }
    }
```

4. Add a new mwthod named **removeFirst()** to remove the first element in single linked list object

```
void removeFirst(){
        if(isEmpty()){
            System.out.println("LinkedList is empty!!");
        }else if(head==tail){
            head = tail = null;
        }else{
            head = head.next;
        }
    }
```

5. Add a method to remove the last element in the SingleLinkedList class, named **removelast()**

```
void removeLast(){
        if(isEmpty()){
            System.out.println("LinkedList is empty!!");
        }else if(head==tail){
            head = tail = null;
        }else{
            Node tmp = head;
            while(tmp.next != tail){
                tmp = tmp.next;
            }
            tmp.next = null;
            tail = tmp;
        }
```

```
    }
```

6. As the next step, the **remove()** method will be implemented.

```
public void remove(String key) {
        if (isEmpty()) {
            System.out.println("LinkedList is empty!!");
        } else {
            Node temp = head;
            while (temp != null) {
                if ((temp.data.name.equalsIgnoreCase(key)) && (temp ==
head)) {
                    removeFirst();
                    break;
                } else if (temp.next.data.name.equalsIgnoreCase(key)) {
                    temp.next = temp.next.next;
                    if (temp.next == null) {
                        tail = temp;
                    }
                    break;
                }
                temp = temp.next;
            }
        }
    }
```

7. Implement a method to **removeAt()** a node by a specified index.

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

8. Next, try accessing and deleting data in the **main** method of the **Main** class by adding the following code.

```
System.out.println("Data at index 1 is:");
Student data = sll.getData(1);
data.print();
```

```
int idx = sll.indexOf("Student 1");
System.out.println("Student 1 is located at index: "+idx);

sll.removeFirst();
sll.removeLast();
sll.print();
sll.removeAt(0);
sll.print();
```

9. Re-run the **Main** class

2.2.2 Output verification

Observe the output of your program and compare it with the following output!

```
Data at index 1 is:
003 - Student 3 - TI-1I - 3.2
Student 1 is located at index: 3
LinkedList Data:
003 - Student 3 - TI-1I - 3.2
002 - Student 2 - TI-1I - 3.45
LinkedList Data:
002 - Student 2 - TI-1I - 3.45
```

2.2.3 Questions

1. Why is the **break** keyword used in the remove function? Explain!
2. Explain the purpose of the code below in the remove method.

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

3. Assignments

Allocated times: 50 menit

Create a queue-based program for student service operations with the following requirements:

a. Implement the queue using a **Linked List-based Queue**.
b. The program should be a **new project**, not a modification of an existing example.
c. When a student wants to join the queue, they must **register their information**.
d. Include functions to **check if the queue is empty**, **check if it is full**, and **clear the queue**.
e. Implement **adding a student to the queue**.
f. Implement **calling the next student in the queue**.
g. Display the **first (front)** and **last (rear)** student in the queue.
h. Display the **total number of students** still in the queue.