Nama : Deanissa Sherly Sabilla

Kelas / Absen : 1B SIB / 06

JOBSHEET IX

LINKED LIST

2.1 Pembuatan Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, akan dilakukan implementasi pembuatan linked list menggunakan array dan penambahan node ke dalam linked list

1. Buat folder baru Praktikum09
2. Tambahkan class-class berikut:
    a. Node.java
    b. LinkedList.java
    c. SLLMain.java
3. Deklarasikan class Node yang memiliki atribut data untuk menyimpan elemen dan atribut next bertipe Node untuk menyimpan node berikutnya. Tambahkan constructor berparameter untuk mempermudah inisialisasi

```
public class Node {
    int data;
    Node next;

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

4. Deklarasikan class LinkedList yang memiliki atribut head. Atribut head menyimpan node pertama pada linked list

```
public class LinkedList {
    Node head;
}
```

5. Sebagai langkah berikutnya, akan diimplementasikan method-method yang terdapat pada class LinkedList.
6. Tambahkan method **isEmpty()**

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

7. Implementasi method print() untuk mencetak dengan menggunakan proses traverse.

```
public void print() {
    if (!isEmpty()) {
        System.out.print("Isi linked list: ");
        Node currentNode = head;

        while (currentNode != null) {
            System.out.print(currentNode.data + "\t");
            currentNode = currentNode.next;
        }

        System.out.println("");
    } else {
        System.out.println("Linked list kosong");
    }
}
```

8. Implementasikan method **addFirst()** untuk menambahkan node baru di awal linked list

```
public void addFirst(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
}
```

9. Implementasikan method **addLast()** untuk menambahkan node baru di akhir linked list

```
public void addLast(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            currentNode = currentNode.next;
        }

        currentNode.next = newNode;
    }
}
```

10. Implementasikan method **insertAfter()** menambahkan node baru pada posisi setelah node yang berisi data tertentu (key)

```
public void insertAfter(int key, int input) {
    Node newNode = new Node(input, null);

    if (!isEmpty()) {
        Node currentNode = head;

        do {
            if (currentNode.data == key) {
                newNode.next = currentNode.next;
                currentNode.next = newNode;
                break;
            }

            currentNode = currentNode.next;
        } while (currentNode != null);
    } else {
        System.out.print("Linked list kosong");
    }
}
```

11. Pada class SLLMain, buatlah fungsi **main**, kemudian buat object myLinkedList bertipe LinkedList. Lakukan penambahan beberapa data. Untuk melihat efeknya terhadap object myLinkedList, panggil method print()

```
public static void main(String[] args) {
    LinkedList myLinkedList = new LinkedList();
    myLinkedList.print();
    myLinkedList.addFirst(800);
    myLinkedList.print();
    myLinkedList.addFirst(700);
    myLinkedList.print();
    myLinkedList.addLast(500);
    myLinkedList.print();
    myLinkedList.insertAfter(700, 300);
    myLinkedList.print();
}
```

2.1.1 Verifikasi Hasil Percobaan

Cocokkan hasil run program Anda dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
```

➢ **INPUT :**

Class Node

```java
public class Node06 {
int data;
Node06 next;

public Node06 (int data, Node06 next) {
    this.data = data;
    this.next = next;
}
}
```

Class LinkedList

```java
public class LinkedList06 {
    Node06 head;
    public boolean isEmpty() {
        return (head == null);
    }
    public void print () {
        if (!isEmpty()) {
            System.out.print(s:"Isi linked list : ");
            Node06 currentNode = head;
            while (currentNode != null) {
                System.out.print(currentNode.data + "\t");
                currentNode = currentNode.next;
            }
            System.out.println(x:"");
        } else {
            System.out.println(x:"Linked list kosong");
        }
    }
    public void addFirst(int input) {
        Node06 newNode = new Node06(input, next:null);
        if (isEmpty()) {
            head = newNode;
        } else {
            newNode.next = head;
            head = newNode;
        }
```

```java
    public void addLast (int input) {
        Node06 newNode = new Node06 (input, next:null);
        if (isEmpty()) {
            head = newNode;
        } else {
            Node06 currentNode = head;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = newNode;
        }
    }
```

```java
    public void insertAfter(int key, int input) {
        Node06 newNode = new Node06 (input, next:null);
        if (!isEmpty()) {
            Node06 currentNode = head;
            do {
                if (currentNode.data == key) {
                    newNode.next = currentNode.next;
                    currentNode.next = newNode;
                    break;
                }
                currentNode = currentNode.next;
            } while (currentNode != null);
        } else {
            System.out.print(s:"Linked list kosong");
        }
    }
}
```

Class Main

```
1   public class SLLMain06 {
     Run | Debug
2   public static void main(String[] args) {
3      LinkedList06 myLinkedList = new LinkedList06();
4      myLinkedList.print();
5      myLinkedList.addFirst(input:800);
6      myLinkedList.print();
7      myLinkedList.addFirst(input:700);
8      myLinkedList.print();
9      myLinkedList.addLast(input:500);
10     myLinkedList.print();
11     myLinkedList.insertAfter(key:700, input:300);
12     myLinkedList.print();
```

- **OUTPUT :**

```
Linked list kosong
Isi linked list : 800
Isi linked list : 700   800
Isi linked list : 700   800     500
Isi linked list : 700   300     800     500
```

2.1.2 Pertanyaan

1. Mengapa class LinkedList tidak memerlukan method isFull() seperti halnya Stack dan Queue?

- class LinkedList tidak memerlukan method isFull() karena LinkedList biasanya tidak memiliki batasan maksimum pada jumlah elemen yang dapat disimpan di dalamnya, seperti pada Stack atau Queue yang memiliki ukuran terbatas (terbatas oleh ukuran array yang digunakan dalam implementasinya).

2. Mengapa class LinkedList hanya memiliki atribut head yang menyimpan informasi node pertama? Bagaimana informasi node kedua dan lainnya diakses?

- karena node selanjutnya dapat diakses secara berurutan dari node pertama menggunakan referensi next yang disimpan dalam setiap node.

3. Pada langkah, jelaskan kegunaan kode berikut

```
if (currentNode.data == key) {
    newNode.next = currentNode.next;
    currentNode.next = newNode;
    break;
}
```

- if (currentNode.data == key) kondisi yang mengecek apakah data dari node saat ini (currentNode.data) sama dengan nilai kunci (key).
- newNode.next = currentNode.next; mengatur referensi next dari newNode agar menunjuk ke node yang sebelumnya ditunjuk oleh currentNode.next.
- currentNode.next = newNode; menyisipkan newNode setelah currentNode dengan mengatur referensi next dari currentNode untuk menunjuk ke newNode.

- break; setelah menemukan tempat di mana newNode harus disisipkan, keluar dari loop dengan menggunakan pernyataan break;

4. Implementasikan method insertAt(int index, int key) dari tugas mata kuliah ASD (Teori)
   - Input dari implementai method insertAt (int index, int key)

```
129     public void insertAt(int index, int input) {
130         Node06 newNode = new Node06(input, next:null);
131         if (index < 0) {
132             System.out.println(x:"Index tidak valid");
133             return;
134         }
135         if (index == 0) {
136             addFirst(input);
137             return;
138         }
139         Node06 currentNode = head;
140 for (int i = 0; i < index - 1; i++) {
141     if (currentNode == null) {
142         System.out.println(x:"Index melebihi panjang LinkedList");
143         return;
144     }
145     currentNode = currentNode.next;
146 }
147 if (currentNode == null) {
148     System.out.println(x:"Index melebihi panjang LinkedList");
149     return;
150 }
151 newNode.next = currentNode.next;
152 currentNode.next = newNode;
```

   - Hasil Output :

```
Linked list kosong
Isi linked list : 800
Isi linked list : 700   800
Isi linked list : 700   800     500
Isi linked list : 700   300     800     500
Data pada index ke-1 : 300
Data 300 berada pada index ke : 1
Isi linked list : 700   800     500
Isi linked list : 800   500
Isi linked list : 800
Isi linked list : 800   400
PS C:\Users\TOSHIBA\Semester 2\Pratikum09>
```

-

2.2 Mengakses dan menghapus node pada Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, kita akan mengimplementasikan method untuk melakukan pengaksesan dan penghapusan data pada linked list

2.2.1 Langkah-langkah Percobaan

1. Tambahkan method getData() untuk mengembalikan nilai elemen di dalam node pada index tertentu

```
public int getData(int index) {
    Node currentNode = head;

    for (int i = 0; i < index; i++) {
        currentNode = currentNode.next;
    }

    return currentNode.data;
}
```

2. Tambahkan method indexOf() untuk mengetahu index dari node dengan elemen tertentu

```
public int indexOf(int key) {
    Node currentNode = head;
    int index = 0;

    while (currentNode != null && currentNode.data != key) {
        currentNode = currentNode.next;
        index++;
    }

    if (currentNode == null) {
        return -1;
    } else {
        return index;
    }
}
```

3. Tambahkan method removeFirst() untuk menghapus node pertama pada linked list

```
public void removeFirst() {
    if (!isEmpty()) {
        head = head.next;
    } else {
        System.out.println("Linked list kosong");
    }
}
```

4. Tambahkan method removeLast() untuk menghapus node terakhir pada linked list

```
public void removeLast() {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.next == null) {
        head = null;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.next == null) {
                currentNode.next = null;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

5. Method remove() digunakan untuk mengapus node yang berisi elemen tertentu

```
public void remove(int key) {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.data == key) {
        removeFirst();
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.data == key) {
                currentNode.next = currentNode.next.next;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

6. Kemudian, coba lakukan pengaksesan dan penghapusan data di method main pada class SLLMain dengan menambahkan kode berikut

```
System.out.println("Data pada index ke-1: " + myLinkedList.getData(1));
System.out.println("Data 300 berada pada index ke: " + myLinkedList.indexOf(300));

myLinkedList.remove(300);
myLinkedList.print();
myLinkedList.removeFirst();
myLinkedList.print();
myLinkedList.removeLast();
myLinkedList.print();
```

7. Compile dan run program kemudian amati hasilnya

2.2.2 Verifikasi Hasil Percobaan

Cocokkan hasil run program dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 300
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```

- **INPUT**

  Class LinkedList (kode tambahan)

```
57          public int getData (int index) {
58              Node06 currentNode = head;
59              for (int i = 0; i < index; i++) {
60                  currentNode = currentNode.next;
61              }
62              return currentNode.data;
63          }
64          public int indexOf(int key) {
65              Node06 currentNode = head;
66              int index = 0;
67              while (currentNode != null && currentNode.data != key) {
68                  currentNode = currentNode.next;
69                  index++;
70              }
71              if (currentNode == null) {
72                  return -1;
73              } else {
74                  return index;
75              }
76          }
```

```java
    public void removeFirst () {
        if (!isEmpty()) {
            head = head.next;
        } else {
            System.out.println(x:"Linked list kosong");
        }
    }
    public void removeLast() {
        if (isEmpty()) {
            System.out.println(x:"Linked list kosong");
        } else if (head.next == null) {
            head = null;
        } else {
            Node06 currentNode = head;
            while (currentNode.next.next == null) {
                currentNode.next = null;
                break;
            }
            currentNode = currentNode.next;
        }
    }
```

```java
    public void remove( int key) {
        if (isEmpty()){
            System.out.println(x:"Linked list kosong");
        } else if (head.data == key) {
            removeFirst();
        } else {
            Node06 currentNode = head;
            while (currentNode.next != null) {
                if (currentNode.next.data == key) {
                    currentNode.next = currentNode.next.next;
                    break;
                }
                currentNode = currentNode.next;
            }
        }
    }
}
```

Class Main (kode tambahan)

```java
public class SLLMain06 {
    Run | Debug
    public static void main(String[] args) {
        LinkedList06 myLinkedList = new LinkedList06();
        myLinkedList.print();
        myLinkedList.addFirst(input:800);
        myLinkedList.print();
        myLinkedList.addFirst(input:700);
        myLinkedList.print();
        myLinkedList.addLast(input:500);
        myLinkedList.print();
        myLinkedList.insertAfter(key:700, input:300);
        myLinkedList.print();
        System.out.println("Data pada index ke-1 : " +myLinkedList.getData(index:1));
        System.out.println("Data 300 berada pada index ke : " +myLinkedList.indexOf(key:300));
        myLinkedList.remove(key:300);
        myLinkedList.print();
        myLinkedList.removeFirst();
        myLinkedList.print();
        myLinkedList.removeLast();
        myLinkedList.print();
```

➢ **OUTPUT**

```
Linked list kosong
Isi linked list : 800
Isi linked list : 700   800
Isi linked list : 700   800     500
Isi linked list : 700   300     800     500
Data pada index ke-1 : 300
Data 300 berada pada index ke : 1
Isi linked list : 700   800     500
Isi linked list : 800   500
Isi linked list : 800
```

2.2.3 Pertanyaan

1. Jelaskan maksud potongan kode di bawah pada method remove()

```
if (currentNode.next.data == key) {
    currentNode.next = currentNode.next.next;
    break;
}
```

   - Potongan kode tersebut untuk menghapus node pertama dari LinkedList jika nilainya sama dengan nilai kunci (key) yang diberikan.

2. Jelaskan maksud if-else block pada method indexOf() berikut

```
if (currentNode == null) {
    return -1;
} else {
    return index;
}
```

   - Potongan kode tersebut untuk untuk menentukan nilai yang akan dikembalikan berdasarkan apakah elemen yang dicari ditemukan atau tidak dalam LinkedList.

3. Error apa yang muncul jika argumen method getData() lebih besar dari jumlah node pada linked list? Modifikasi kode program untuk menghandle hal tersebut.
   - Jika argumen yang diberikan lebih besar dari jumlah node pada LinkedList, maka akan muncul error NullPointerException.
   - Modifikasi :

```
57
58      public int getData(int index) {
59          Node06 currentNode = head;
60          for (int i = 0; i < index; i++) {
61              if (currentNode == null) {
62                  System.out.println(x:"Index melebihi panjang LinkedList");
63              }
64              currentNode = currentNode.next;
65          }
66          if (currentNode == null) {
67              System.out.println(x:"Index melebihi panjang LinkedList");
68          }
69          return currentNode.data;
70      }
```

4. Apa fungsi keyword break pada method remove()? Bagaimana efeknya jika baris tersebut dihapus?
   - untuk menghentikan iterasi melalui linked list setelah operasi penghapusan dilakukan.