LAPORAN HASIL PRAKTIKUM ALGORITMA DAN STRUKTUR DATA PRAKTIKUM 9

Oleh:
RANDA HERU KUSUMA

NIM. 2341760009

SIB-1F / 25

D-IV SISTEM INFORMASI BISNIS JURUSAN

TEKNOLOGI INFORMASI

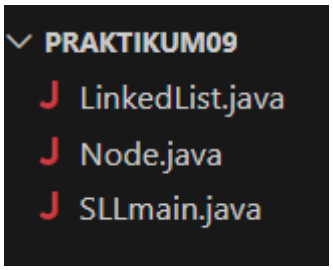
POLITEKNIK NEGERI MALANG

Link github: https://github.com/randaheru/Praktikum9.git

2.1 Pembuatan Linked List

Waktu percobaan: 50 menit

Buat folder baru Praktikum09



Tambahkan class-class berikut:

a. Node.java
b. LinkedList.java
c. SLLMain.java

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;
    }
}
```

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

```
class LinkedList {
    Node head;
```

Tambahkan method isEmpty()

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

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

```
public void print() {
    if (!isEmpty()) {
        System.out.print(s:"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(x:"Linked list kosong");
    }
}
```

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

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

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

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

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

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

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()

```java
public class SLLmain {
    Run | Debug
    public static void main(String[] args) {
        LinkedList myLinkedList = new LinkedList();
        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();
    }
}
```

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
```

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

Kelas LinkedList tidak memerlukan metode isFull() seperti halnya Stack dan Queue karena perbedaan mendasar dalam cara penyimpanan dan batasan kapasitas dari struktur data tersebut.

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

Kelas **LinkedList** hanya memiliki atribut **head** yang menyimpan informasi node pertama karena setiap node dalam linked list mengandung dua informasi penting

Dengan mengakses head, kita dapat mengikuti alur koneksi antar node menggunakan pointer next untuk mengakses node lainnya dalam linked list.

3. Pada langkah, jelaskan kegunaan kode berikut

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

Memeriksa apakah nilai data dari node saat ini (**currentNode.data**) sama dengan nilai kunci yang ingin kita sisipkan setelahnya (**key**). Jika kondisi tersebut terpenuhi, berarti kita telah menemukan node dengan nilai yang sesuai dengan kunci yang ingin kita sisipkan setelahnya.

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

```
// Insert a new node at a specific index
public void insertAt(int index, int key) {
    if (index < 0) {
        System.out.println(x:"Indeks tidak valid");
        return;
    }
```

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

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

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

    for (int i = 0; i < index; i++) {
        if (currentNode == null) {
            // Jika currentNode null, berarti indeks melebihi panjang list
            throw new IndexOutOfBoundsException(s:"Indeks melebihi panjang list");
        }
        currentNode = currentNode.next;
    }

    if (currentNode != null) {
        return currentNode.data;
    } else {
        // Jika currentNode null, berarti indeks melebihi panjang list
        throw new IndexOutOfBoundsException(s:"Indeks melebihi panjang list");
    }
}
```

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;
    }
}
```

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

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

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

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

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

```
public void remove(int key) {
    if (isEmpty()) {
        System.out.println(x:"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;
        }
    }
}
```

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(key:300));

    myLinkedList.remove(key:300);
    myLinkedList.print();
    myLinkedList.removeFirst();
    myLinkedList.print();
    myLinkedList.removeLast();
    myLinkedList.print();
}
```

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
```

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;
}
```

Untuk menghapus node yang memiliki nilai data yang sama dengan **key** dari linked list.

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

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

if-else block pada method **indexOf()** bertanggung jawab untuk menentukan indeks dari node yang memiliki nilai data yang sama dengan **key**, atau mengembalikan nilai **-1** jika **key** tidak ditemukan dalam linked list.

3. Error apa yang muncul jika argumen method getData() lebih besar dari jumlah node pada linked list? Modifikasi kode program untuk menghandle hal tersebut.

IndexOutOfBoundsException yang menyatakan bahwa indeks melebihi panjang list.

```java
public int getData(int index) {
if (index < 0) {
    throw new IndexOutOfBoundsException(s:"Indeks negatif tidak valid");
}

Node currentNode = head; // Mulai dari node pertama
int currentIndex = 0;

while (currentNode != null) {
    if (currentIndex == index) {
        return currentNode.data;
    }
    currentNode = currentNode.next;
    currentIndex++;
}

// Jika sampai di sini, berarti indeks melebihi panjang linked list
throw new IndexOutOfBoundsException(s:"Indeks melebihi panjang list");
```

4. Apa fungsi keyword break pada method remove()? Bagaimana efeknya jika baris tersebut dihapus?

   untuk menghentikan iterasi saat sudah ditemukan node yang akan dihapus. Ketika sebuah node dengan data yang sesuai dengan **key** ditemukan, maka node tersebut dihapus dengan cara mengubah pointer **next** pada node sebelumnya sehingga melewati node yang akan dihapus, Jika baris yang berisi **break** dihapus, maka iterasi akan terus berlanjut bahkan setelah node yang sesuai dengan **key** ditemukan. Hal ini akan menyebabkan pengulangan berlanjut tanpa henti hingga akhir linked list, bahkan setelah node yang sesuai telah dihapus.

3. Tugas

Waktu pengerjaan: 50 menit

1. Implementasikan method-method berikut pada class LinkedList:
   a. insertBefore() untuk menambahkan node sebelum keyword yang diinginkan

```java
public void insertBefore(int key, int input) {
    if (isEmpty()) {
        System.out.println(x:"Linked list kosong");
        return;
    }

    if (head.data == key) {
        addFirst(input);
        return;
    }

    Node currentNode = head;
    Node prevNode = null;

    while (currentNode != null) {
        if (currentNode.data == key) {
            Node newNode = new Node(input, currentNode);
            prevNode.next = newNode;
            return;
        }
        prevNode = currentNode;
        currentNode = currentNode.next;
    }

    System.out.println(x:"Keyword tidak ditemukan");
}
```

b. insertAt(int index, int key) untuk menambahkan node pada index tertentu

```java
public void insertAt(int index, int key) {
    if (index < 0) {
        System.out.println(x:"Indeks tidak valid");
        return;
    }

    // Buat node baru dengan nilai key
    Node newNode = new Node(key, next:null);

    // Jika indeks adalah 0, sisipkan node baru di awal linked list
    if (index == 0) {
        newNode.next = head;
        head = newNode;
        return;
    }

    // Penelusuran linked list untuk mencari posisi yang sesuai
    Node currentNode = head;
    int counter = 0;
    while (currentNode != null && counter < index - 1) {
        currentNode = currentNode.next;
        counter++;
    }

    // Jika indeks melebihi panjang linked list, cetak pesan kesalahan
    if (currentNode == null) {
        System.out.println(x:"Indeks melebihi panjang linked list");
```

c. removeAt(int index) untuk menghapus node pada index tertentu

```java
public void removeAt(int index) {
    if (index < 0) {
        System.out.println(x:"Indeks tidak valid");
        return;
    }

    if (index == 0) {
        removeFirst();
        return;
    }

    Node currentNode = head;
    Node prevNode = null;
    int counter = 0;
    while (currentNode != null && counter < index) {
        prevNode = currentNode;
        currentNode = currentNode.next;
        counter++;
    }

    if (currentNode == null) {
        System.out.println(x:"Indeks melebihi panjang linked list");
    } else {
        prevNode.next = currentNode.next;
    }
}
```

2. Dalam suatu game scavenger hunt, terdapat beberapa point yang harus dilalui peserta untuk menemukan harta karun. Setiap point memiliki soal yang harus dijawab, kunci jawaban, dan pointer ke point selanjutnya. Buatlah implementasi game tersebut dengan linked list.

Buat class point

```java
class Point {
    String question;
    String answer;
    Point nextPoint;
```

Tambahkan atribut sebagai berikut

```java
// Kelas Point untuk merepresentasikan setiap poin dalam linked list
class Point {
    String question;
    String answer;
    Point nextPoint;
```

Tambahkan constuktor

```java
    public Point(String question, String answer) {
        this.question = question;
        this.answer = answer;
        this.nextPoint = null;
    }
}
```

Buat class scavengerHunt

```
class ScavengerHunt {
    Point startPoint;
    Point currentPoint;

    public ScavengerHunt() {
        startPoint = null;
        currentPoint = null;
    }
```

Tambahkan metod addpoint

```
    // Method untuk menambahkan poin baru ke linked list
    public void addPoint(String question, String answer) {
        Point newPoint = new Point(question, answer);
        if (startPoint == null) {
            startPoint = newPoint;
            currentPoint = startPoint;
        } else {
            Point temp = startPoint;
            while (temp.nextPoint != null) {
                temp = temp.nextPoint;
            }
            temp.nextPoint = newPoint;
        }
    }
```

Tambahan metod startHunt

```
// Method untuk memulai perburuan
public void startHunt() {
    Scanner scanner = new Scanner(System.in);
    System.out.println(x:"Selamat datang di Scavenger Hunt!");
    System.out.println(x:"Setiap pertanyaan memiliki sebuah kunci jawaban. Jawablah dengan benar untuk melanjutkan perburuan.");
    System.out.println(x:"Mari kita mulai!");
    currentPoint = startPoint;
    while (currentPoint != null) {
        System.out.println("\n" + currentPoint.question);
        System.out.print(s:"Jawaban: ");
        String userAnswer = scanner.nextLine().trim().toLowerCase();
        if (userAnswer.equals(currentPoint.answer)) {
            System.out.println(x:"Selamat! Jawaban kamu benar!");
            if (currentPoint.nextPoint != null) {
                System.out.println(x:"Langsung ke pertanyaan berikutnya...");
                currentPoint = currentPoint.nextPoint;
            } else {
                System.out.println(x:"Selamat! Kamu telah menemukan harta karun!");
                break;
            }
        } else {
            System.out.println(x:"Jawaban kamu salah. Coba lagi!");
        }
    }
    scanner.close();
}
```

Buat class main

```
// Kelas Main untuk menjalankan program
public class Main {
    Run | Debug
```

Buat metod main

```
    Run | Debug
    public static void main(String[] args) {
        // Inisialisasi perburuan
```

Membuat objek **hunt** dari kelas **ScavengerHunt** menggunakan konstruktor **ScavengerHunt()**. Ini menginisialisasi permainan scavenger hunt.

```
        // Inisialisasi perburuan
        ScavengerHunt hunt = new ScavengerHunt();
```

Menambahkan poin-poin perburuan dengan memanggil method **addPoint** dari objek **hunt** yang telah kita buat sebelumnya. Setiap poin memiliki pertanyaan dan jawaban yang sesuai.

```
// Menambahkan poin-poin perburuan
hunt.addPoint(question:"Di dalam gua yang gelap, terdapat seekor binatang dengan dua mata tapi tidak memiliki mulut. Apakah itu?", answer:"k
hunt.addPoint(question:"Di kebun belakang rumah, terdapat pohon dengan buah yang berwarna kuning ketika sudah matang. Apakah nama buah itu?"
hunt.addPoint(question:"Di bawah jembatan yang terbuat dari batu, terdapat sebuah peti yang tertutup rapat. Buka peti itu dan ambil kunci un
```

Ini memulai permainan scavenger hunt dengan memanggil method **startHunt** dari objek **hunt**.

```
        // Memulai perburuan
        hunt.startHunt();
    }
}
```

Hasil run program

```
Di dalam gua yang gelap, terdapat seekor binatang dengan dua mata tapi tidak memiliki mulut. Apakah itu?
Jawaban: katak
Selamat! Jawaban kamu benar!
Langsung ke pertanyaan berikutnya...

Di kebun belakang rumah, terdapat pohon dengan buah yang berwarna kuning ketika sudah matang. Apakah nama buah itu?
Jawaban: pisang
Selamat! Jawaban kamu benar!
Langsung ke pertanyaan berikutnya...

Di bawah jembatan yang terbuat dari batu, terdapat sebuah peti yang tertutup rapat. Buka peti itu dan ambil kunci untuk menuju point selanjutnya.
Jawaban: peti
Selamat! Jawaban kamu benar!
Selamat! Kamu telah menemukan harta karun!
PS C:\Users\ACER\Documents\Praktikum09>
```