LAPORAN PRAKTIKUM JOBSHEET 10
LINKED LIST
MATA KULIAH ALGORITMA DAN STRUKTUR DATA

Disusun Oleh :

Jami'atul Afifah (2341760102)

SIB-1F

PROGRAM STUDI D4 SISTEM INFOEMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

PRAKTIKUM

Pembuatan Linked List

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. Node14.java
    b. LinkedList14.java
    c. SLLMain14.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

```java
public class Node14 {
    int data;
    Node14 next;

    public Node14(int data, Node14 next) {
        this.data = data;
        this.next = next;
    }
}
```

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

```java
public class LinkedList14 {
    Node14 head;
```

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

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

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

```java
    // Method print
    public void print() {
        if (!isEmpty()) {
            System.out.print(s:"Isi linked list: ");
            Node14 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");
        }
    }
```

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

```
// Method addFirst
public void addFirst(int input) {
    Node14 newNode = new Node14(input, next:null);
    if (isEmpty()) {
        head = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
}
```

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

```
// Method addLast
public void addLast(int input) {
    Node14 newNode = new Node14(input, next:null);
    if (isEmpty()) {
        head = newNode;
    } else {
        Node14 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)

```
// Method insertAfter
public void insertAfter(int key, int input) {
    Node14 newNode = new Node14(input, next:null);
    if (!isEmpty()) {
        Node14 currentNode = head;
        while (currentNode != null) {
            if (currentNode.data == key) {
                newNode.next = currentNode.next;
                currentNode.next = newNode;
                break;
            }
            currentNode = currentNode.next;
        }
    } else {
        System.out.println(x:"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()

```java
public class SLLMain14 {
    Run | Debug
    public static void main(String[] args) {
        LinkedList14 myLinkedList = new LinkedList14();
        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();
    }
}
```

Hasil

```
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
PS D:\Matkul\SEM 2\ASD\Jobsheet10>
```

Pertanyaan

1. Mengapa class LinkedList tidak memerlukan method isFull() seperti halnya Stack dan Queue?
   LinkedList tidak memerlukan method isFull() karena LinkedList secara dinamis mengalokasikan memori untuk setiap elemen baru yang ditambahkan. Tidak ada batasan ukuran yang tetap pada LinkedList selama masih ada memori yang tersedia di sistem. Berbeda dengan Stack dan Queue yang diimplementasikan menggunakan array yang memiliki ukuran tetap, LinkedList tidak memiliki batasan kapasitas yang tetap sehingga tidak perlu memeriksa apakah struktur data tersebut penuh
2. Mengapa class LinkedList hanya memiliki atribut head yang menyimpan informasi node pertama? Bagaimana informasi node kedua dan lainnya diakses?
   LinkedList hanya membutuhkan atribut head untuk menyimpan informasi tentang node pertama karena setiap node dalam LinkedList menyimpan referensi (pointer) ke node berikutnya. Informasi node kedua dan seterusnya diakses dengan mengikuti referensi next dari node pertama, dan seterusnya hingga akhir daftar. Ini adalah cara traversal

LinkedList: mulai dari head, dan terus bergerak ke node berikutnya menggunakan referensi next sampai mencapai node terakhir yang referensi next-nya adalah null.

3. Pada langkah, jelaskan kegunaan kode berikut

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

Kode tersebut digunakan dalam metode insertAfter untuk menyisipkan node baru setelah node yang memiliki nilai data tertentu (key). Prosesnya adalah sebagai berikut:

- if (currentNode.data == key): Mengecek apakah node saat ini (currentNode) memiliki nilai data yang sama dengan key.
- newNode.next = currentNode.next: Mengatur node baru (newNode) untuk menunjuk ke node yang saat ini ditunjuk oleh currentNode.next. Ini menghubungkan node baru ke bagian sisa dari LinkedList.
- currentNode.next = newNode: Mengatur node saat ini (currentNode) untuk menunjuk ke node baru (newNode). Ini menyisipkan node baru setelah node saat ini.
- break: Menghentikan loop setelah node baru disisipkan, karena tugas telah selesai.

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

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

Validasi Index

```
    Node14 newNode = new Node14(input, next:null);
```

Membuat Node Baru

```
    if (index == 0) {
        newNode.next = head;
        head = newNode;
```

Jika Indeks adalah 0

```
        Node14 currentNode = head;
        int currentIndex = 0;

        while (currentNode != null && currentIndex < index - 1) {
            currentNode = currentNode.next;
            currentIndex++;
        }
```

Traverse LinkedList

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

Menambahkan Node pada posisi yang ditentukan

Mengakses dan menghapus node pada Linked List

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

```
// Method getData
public int getData(int index) {
    if (index < 0 || isEmpty()) {
        System.out.println(x:"Index tidak valid atau linked list kosong");
        return -1; // Atau bisa melempar exception
    }

    Node14 currentNode = head;
    for (int i = 0; i < index; i++) {
        if (currentNode.next == null) {
            System.out.println(x:"Index melebihi panjang linked list");
            return -1; // Atau bisa melempar exception
        }
        currentNode = currentNode.next;
    }
    return currentNode.data;
}
```

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

```
// Method indexOf
public int indexOf(int key) {
    Node14 currentNode = head;
    int index = 0;
    while (currentNode != null) {
        if (currentNode.data == key) {
            return index;
        }
        currentNode = currentNode.next;
        index++;
    }
    return -1; // Key tidak ditemukan
}
```

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

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

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

```
// Method removeLast
public void removeLast() {
    if (isEmpty()) {
        System.out.println(x:"Linked list kosong");
    } else if (head.next == null) {
        head = null;
    } else {
        Node14 currentNode = head;
        while (currentNode.next.next != null) {
            currentNode = currentNode.next;
        }
        currentNode.next = null;
    }
}
```

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

```java
// Method remove
public void remove(int key) {
    if (isEmpty()) {
        System.out.println(x:"Linked list kosong");
    } else if (head.data == key) {
        removeFirst();
    } else {
        Node14 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

```java
// Mengakses data pada index tertentu
System.out.println("Data pada index ke-1: " + myLinkedList.getData(index:1));

// Mengetahui index dari node dengan elemen tertentu
System.out.println("Data 300 berada pada index ke: " + myLinkedList.indexOf(key:300));

// Menghapus node yang berisi elemen tertentu
myLinkedList.remove(key:300);
myLinkedList.print();

// Menghapus node pertama
myLinkedList.removeFirst();
myLinkedList.print();

// Menghapus node terakhir
myLinkedList.removeLast();
myLinkedList.print();
```

Hasil

```
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
PS D:\Matkul\SEM 2\ASD\Jobsheet10>
```

Pertanyaan

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

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

Potongan kode ini adalah bagian dari loop yang mencari node dengan elemen key dan menghapusnya dari linked list.

- if (currentNode.next.data == key): Mengecek apakah data di node berikutnya (currentNode.next) sama dengan key.
- currentNode.next = currentNode.next.next: Jika kondisi terpenuhi, mengubah referensi next dari node saat ini (currentNode) untuk melompati node berikutnya. Ini berarti node berikutnya yang memiliki data key dihapus dari linked list.
- break;: Menghentikan loop setelah node dengan data key ditemukan dan dihapus. Tanpa break, loop akan terus berjalan meskipun node yang dicari sudah ditemukan dan dihapus.

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

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

Potongan kode ini menentukan apa yang akan dikembalikan oleh method indexOf() setelah loop selesai:

- if (currentNode == null): Mengecek apakah loop telah mencapai akhir linked list (currentNode == null) tanpa menemukan node dengan data key.
- return -1;: Jika ya, mengembalikan -1 yang menandakan bahwa elemen key tidak ditemukan di linked list.
- else: Jika currentNode tidak null, berarti node dengan data key ditemukan.
- return index;: Mengembalikan indeks dari node yang memiliki data key.

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 index pada method getData() lebih besar dari jumlah node pada linked list, program akan mengakses node yang null dan menghasilkan NullPointerException. Berikut adalah modifikasi untuk menghandle hal tersebut:

```
public int getData(int index) {
    if (index < 0 || isEmpty()) {
        System.out.println(x:"Index tidak valid atau linked list kosong");
        return -1; // Atau bisa melempar exception
    }

    Node14 currentNode = head;
    for (int i = 0; i < index; i++) {
        if (currentNode.next == null) {
            System.out.println(x:"Index melebihi panjang linked list");
            return -1; // Atau bisa melempar exception
        }
        currentNode = currentNode.next;
    }
    return currentNode.data;
}
```

Penambahan pengecekan if (currentNode.next == null) memastikan bahwa jika index melebihi panjang linked list, method akan mencetak pesan kesalahan dan mengembalikan -1.

4. Apa fungsi keyword break pada method remove()? Bagaimana efeknya jika baris tersebut dihapus?
   Fungsi break pada method remove() adalah untuk menghentikan loop segera setelah node dengan data key ditemukan dan dihapus. Tanpa break, loop akan terus berjalan sampai akhir linked list, meskipun node yang dicari sudah ditemukan dan dihapus.
   Jika break dihapus:
   - Loop akan terus berjalan sampai mencapai akhir linked list.
   - Akibatnya, program mungkin mencoba mengakses currentNode.next yang sudah null, yang bisa menyebabkan NullPointerException.
   - Efisiensi program menurun karena traversal tidak berhenti setelah node yang dihapus ditemukan.

TUGAS

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

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

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

    Node14 newNode = new Node14(input, next:null);
    Node14 currentNode = head;
    while (currentNode.next != null && currentNode.next.data != key) {
        currentNode = currentNode.next;
    }

    if (currentNode.next == null) {
        System.out.println("Node dengan data " + key + " tidak ditemukan");
    } else {
        newNode.next = currentNode.next;
        currentNode.next = newNode;
    }
}
```

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

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

    Node14 newNode = new Node14(input, next:null);

    if (index == 0) {
        newNode.next = head;
        head = newNode;
    } else {
        Node14 currentNode = head;
        int currentIndex = 0;

        while (currentNode != null && currentIndex < index - 1) {
            currentNode = currentNode.next;
            currentIndex++;
        }

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

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

```java
public void removeAt(int index) {
    if (index < 0 || isEmpty()) {
        System.out.println(x:"Index tidak valid atau linked list kosong");
        return;
    }

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

    Node14 currentNode = head;
    int currentIndex = 0;

    while (currentNode != null && currentIndex < index - 1) {
        currentNode = currentNode.next;
        currentIndex++;
    }

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

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.

Point.java

```java
public class Point {
    String question;
    String answer;
    Point next;

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

ScavengerHunt.java

```java
public class ScavengerHunt {
    Point head;

    public ScavengerHunt() {
        head = null;
    }

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

    public void addPoint(String question, String answer) {
        Point newPoint = new Point(question, answer);
        if (isEmpty()) {
            head = newPoint;
        } else {
            Point current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newPoint;
        }
    }

    public void printPoints() {
        if (isEmpty()) {
            System.out.println(x:"Tidak ada point dalam scavenger hunt.");
        } else {
            Point current = head;
            while (current != null) {
                System.out.println("Pertanyaan: " + current.question);
                System.out.println("Jawaban: " + current.answer);
                current = current.next;
            }
```

ScavengerHuntMain.java

```java
public class ScavengerHuntMain {
    Run | Debug
    public static void main(String[] args) {
        ScavengerHunt hunt = new ScavengerHunt();

        hunt.addPoint(question:"Apa ibu kota Indonesia?", answer:"Jakarta");
        hunt.addPoint(question:"Siapa presiden pertama Indonesia?", answer:"Soekarno");
        hunt.addPoint(question:"Berapa jumlah provinsi di Indonesia?", answer:"34");

        hunt.printPoints();
    }
}
```

Hasil