

MODUL 3. SINGLY *LINKED LIST*

3.1. Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

- 1) Mengetahui konsep *Linked list* pada Java
- 2) Mengetahui konsep *Singly Linked list* dan implementasinya pada Java

3.2. Alat dan Bahan

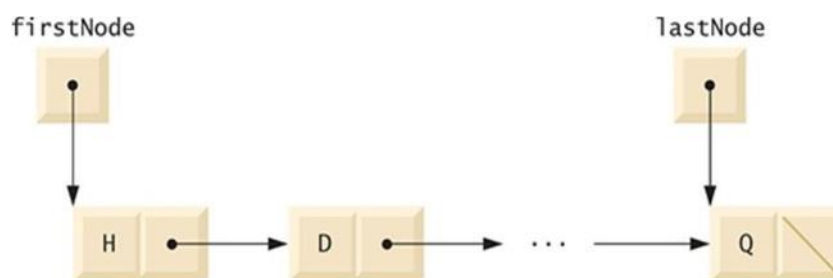
Alat & Bahan Yang digunakan adalah hardware perangkat PC beserta Kelengkapannya berjumlah 40 PC serta Software IntelliJ IDEA yang telah terinstall pada masing-masing PC

3.3. Dasar Teori

Linked list adalah struktur data linear, dan digunakan untuk mengumpulkan barisan objek yang memungkinkan terjadinya penambahan maupun penghapusan elemen di tengah barisan. *Linked list* terdiri atas objek-objek yang merujuk pada kelas nya sendiri, biasa disebut nodes, dan dihubungkan dengan suatu link (nama *linked list* merujuk pada hal ini).

Program mengakses *linked list* melalui referensi ke node pertama. Kemudian, node-node pada *linked list* diakses melalui referensi link pada node sebelumnya. Secara konvensi, referensi link pada node terakhir di-set menjadi null untuk menunjukkan akhir list. Data disimpan dan dihapus dari *linked list* secara dinamis; program menambah dan menghapus node sesuai kebutuhan.

Node pada *linked list* tidak disimpan secara berurutan pada memori. Node hanya terurut secara logika. Gambar 1 mengilustrasikan *singly linked list* dengan setiap node memiliki referensi pada node selanjutnya. Selain *singly*, jenis lain dari *linked list* adalah *doubly linked list* dimana setiap node memiliki referensi baik untuk node selanjutnya maupun node sebelumnya. *Doubly linked list* akan dipelajari pada modul berikutnya.



Gambar 1 Singly Linked list (Deitel, Deitel 2012)

Berikut contoh program implementasi Singly *Linked list* pada Java (Deitel & Deitel, 2017 dan Java2Blog). Kita akan melakukan implemetasi secara bertahap, yang dimulai dari pembentukan kelas POJO (karena objek yang akan dimasukkan merupakan tipe data bentukan), dilanjutkan dengan pembentukan kelas node, kemudian kelas singly *linked list*, yang akhirnya akan dipanggil pada Main.

3.3.1. Pembuatan POJO

Seperti pada modul 2 (Generic Class), kita akan membuat kelas POJO untuk digunakan pada kelas Node kelak. Object yang akan kita bentuk adalah objek Mahasiswa, yang memiliki tiga atribut, yaitu NIM, Nama, dan Kelas.

```
public class
    Mahasiswa{
        private String
        nim; private
        String nama;
        private String
        kelas;

        public Mahasiswa(String nim, String nama, String kelas) {
            this.nim = nim;
            this.nama =
            nama;
            this.kelas =
            kelas;
        }

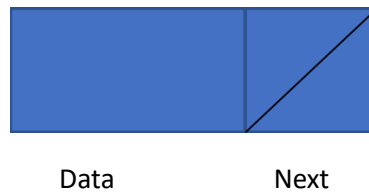
        public String getNim() {
            return nim;
        }

        @Override
        public String toString() {
            return "nim='" + nim + '\'' +
                ", nama='" + nama + '\'' +
                ", kelas='" + kelas + '\''
                ;
        }
    }
```

Atribut dari kelas Mahasiswa memiliki akses modifier private agar tidak dapat diubah dari luar kelas. Pembahasan mengenai akses modifier ini akan dilanjutkan lebih dalam pada mata kuliah Pemrograman Berorientasi Object (PBO). Konstruktor Mahasiswa menerima tiga parameter, sesuai dengan atribut yang dimiliki oleh kelas Mahasiswa. Method `toString()` dibutuhkan agar object mahasiswa dapat dicetak jika dibutuhkan.

3.3.2. Pembentukan Node

Pada struktur data Singly *Linked list*, suatu node terdiri atas bagian data, yang menampung objek dari `LinkedList`, dan next, yang merupakan “pointer” yang akan merujuk pada node berikut pada *Linked list* (Gambar 2).



Gambar 2 Node pada Singly Linked list

Kelas `ListNode` ini hanya memiliki dua atribut, yaitu `data` dan `next`. Perhatikan, tipe data dari atribut `data` adalah `E` yang berarti ini adalah tipe data generik. Demikian juga, tipe data dari `ListNode` adalah generik. Hal ini dikarenakan node disiapkan untuk bisa menerima berbagai tipe data, termasuk tipe data bentukan (dalam contoh kita adalah tipe data Mahasiswa). Sementara itu, tipe data dari `next` adalah `ListNode`. Ini terjadi karena `next` harus bisa merujuk pada node lain.

```
public class ListNode<E> {
    E data;
    ListNode<E> next;

    public ListNode(E object) {
        data = object;
        this.next = null;
    }

    public E getData() {
        return data;
    }
}
```

3.3.3. Pembentukan kelas *Singly Linked list*

Setelah data yang akan dimasukkan dan node dibentuk, barulah dibuat kelas *Singly Linked list* yang membentuk *linked list*. Untuk kemudahan, method yang terdapat pada kelas ini hanya sisip depan, sisip belakang, hapus belakang dan tampil list.

```
import java.util.NoSuchElementException;

public class SinglyLinkedList<E> {
    private ListNode<E> firstNode;
    private ListNode<E> lastNode;
    private String name;

    public SinglyLinkedList() {
        this("linked list");
    }

    public SinglyLinkedList(String listName) {
        name = listName;
        firstNode = lastNode = null;
    }

    public void insertAtFront(E insertItem) {
        ListNode newNode = new ListNode(insertItem);

        if (isEmpty()) { // firstNode and lastNode refer to same object
            firstNode = lastNode = new ListNode<E>(insertItem);
        }
    }
}
```

```

    } else { // firstNode refers to new node
        newNode.next = firstNode;
        firstNode = newNode;
    }
}

public void insertAtBack(E insertItem) {
    ListNode newNode = new ListNode(insertItem);

    if (isEmpty()) { // firstNode and lastNode refer to same object
        firstNode = lastNode = new ListNode<E>(insertItem);
    } else { // lastNode refers to new node
        lastNode.next = newNode;
        lastNode = newNode;
    }
}

public E removeFromBack() throws NoSuchElementException {
    if (isEmpty()) { // throw exception if List is empty
        throw new NoSuchElementException(name + " is empty");
    }

    E removedItem = lastNode.data; // retrieve data being removed
    // update references firstNode and lastNode
    if (firstNode == lastNode) {
        firstNode = lastNode = null;
    } else { // locate new last node
        ListNode<E> current = firstNode;

        // loop while current node does not refer to lastNode
        while (current.next != lastNode) {
            current = current.next;
        }

        lastNode = current; // current is new lastNode
        current.next = null;
    }

    return removedItem; // return removed node data
}

private boolean isEmpty() {
    return firstNode == null;
}

public void print() {
    if (isEmpty()) {
        System.out.printf("Empty %s\n", name);
        return;
    }

    System.out.printf("The %s is: %n", name);
    ListNode<E> current = firstNode;
    // while not at end of list, output current node's data
    while (current != null) {
        System.out.printf("%s ", current.data);
        current = current.next;
    }
}

```

```

        System.out.println();
    }
}

```

Linked list dirujuk oleh dua referensi, yaitu *firstNode* (head) dan *lastNode* (tail). Kelas ini menggunakan tipe data generic agar dapat digunakan oleh tipe data objek apapun. Pemanggilan kelas *Singly Linked list* dilakukan pada kelas *main*.

```

public class Main {

    public static void main(String[] args) {
        SinglyLinkedList<Mahasiswa> linkedList =
            new SinglyLinkedList<>();

        linkedList.insertAtFront(new Mahasiswa("6706123456", "Yulia", "D3IF
45-01"));
        linkedList.insertAtBack(new Mahasiswa("670678901", "Riki", "D3IF
45-02"));
        linkedList.insertAtFront(new Mahasiswa("6706123458", "Tina", "D3IF
45-03"));

        linkedList.print();

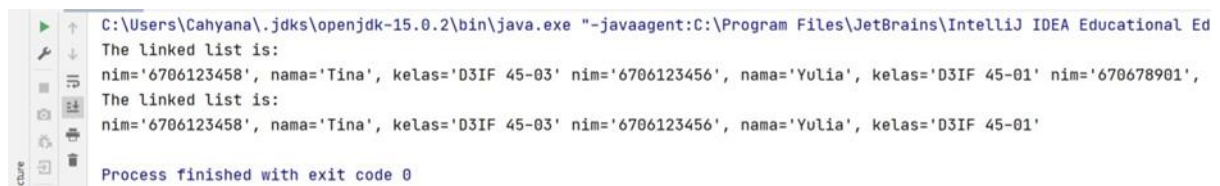
        linkedList.removeFromBack();

        linkedList.print();

    }
}

```

Hasil program di atas ketika dijalankan adalah sebagai berikut (node terakhir terpotong dari tampilan screenshot).



```

C:\Users\Cahyana\.jdk\openjdk-15.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Educational Ed
The linked list is:
nim='6706123458', nama='Tina', kelas='D3IF 45-03' nim='6706123456', nama='Yulia', kelas='D3IF 45-01' nim='670678901',
The linked list is:
nim='6706123458', nama='Tina', kelas='D3IF 45-03' nim='6706123456', nama='Yulia', kelas='D3IF 45-01'
Process finished with exit code 0

```

3.4. Studi Kasus

Contoh bisa mengikuti kode yang di atas.

- ✓ Buatlah aplikasi sederhana untuk menginput baju seseorang seperti tipe baju (string), ukuran baju (String), dan Warna baju (String).
- ✓ Aplikasi sederhana ini dapat menyimpan data baju seseorang dan menampilkan baju yang telah di input oleh user
- ✓ **Challenge** tambahkan pengoperasian menghapus depan dan belakang pada aplikasi sederhana ini (**Optional**)

Contoh app sederhana

```
1.Tambah Baju
2.Tampilkan Baju
Pilih Operasi:
```

Gambar 1 Tampilan awal

```
1.Tambah Baju
2.Tampilkan Baju
Pilih Operasi:
1
Masukan Baju di Depan/Belakang? 1/2
1
Tipe Baju:
kaos
Ukuran Baju:
m
Warna Baju:
merah
```

Gambar 2 Ketika Memilih operasi tambah baju

```
Baju yang tersimpan:
Baju [tipe=kemeja, ukuran=m, warna=hitam]
```

Gambar 3 Tampilan Ketika sudah menginput baju dan mengoperasikan Tampilkan