MODUL 3. DOUBLY LINKED LIST

3.1 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:
1) Mengenal konsep Linked List pada Java
2) Mengenal konsep Doubly Linked List dan implementasinya pada Java
3) Memahami penggunaan iterator

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 (isi modulnya)

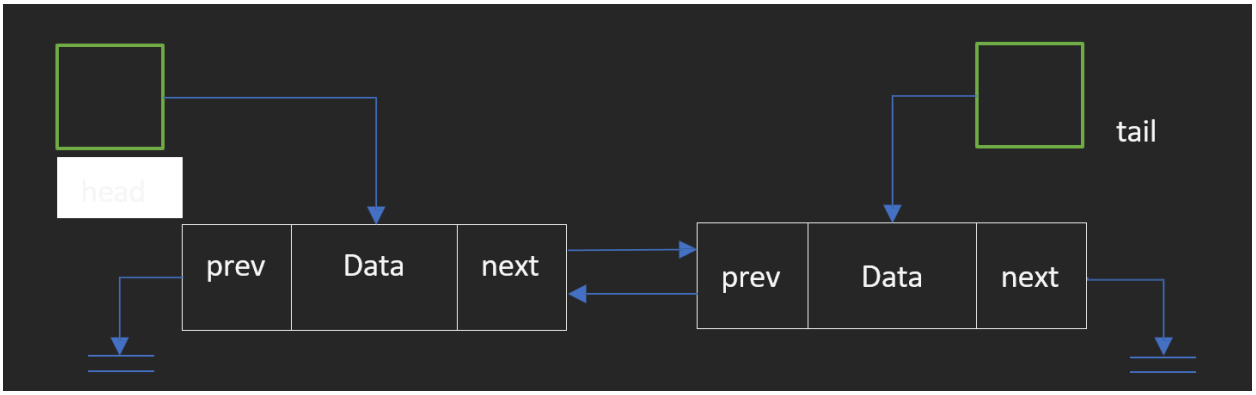
Java *Application Programming Interface* (API) menyediakan berbagai tipe struktur data, yang kita sebut collections! Struktur data ini berbentuk *class*, dan sudah diisi dengan *method-method* tinggal pakai untuk menyimpan dan mengatur data-data di dalamnya! Enak kan? Dengan begini, kita bisa langsung fokus ke pembangunan program dan aplikasi aja.

Dalam berbagai *class* di Collections, generics dipakai agar tipe data apapun bisa masuk ke dalam struktur-struktur data di dalam. Pada modul ini, kita akan melihat penggunaan salah satu struktur data dari *library* Collections, yaitu LinkedList yang merupakan implementasi Doubly Linked Listnya sendiri, tapi pakai yang udah disediain Java. Asik kan?

3.3.1 Doubly Linked List

Kalian inget kan kalau Linked List ini linear, dan terdiri atas objek-objek yang merujuk pada kelasnya sendiri. Kelas “Node” misalnya. Dari objek “Node” satu, dia bakal nunjuk ke objek “Node” lainnya, lewat atribut “next” biasanya. Atribut tersebut adalah “link” nya, yang menjadi nama “linked list” itu sendiri!

Kalau dalam linked list biasa, yang sebelumnya kita pelajari, dia cuma punya “next” kan? Tapi berbeda dengan Doubly Linked List! Dia punya referensi ke Node sebelumnya, lho! Biasanya disebut dengan atribut “prev” atau “previous”, yang berarti sebelumnya. Kalau “next” selanjutnya, maka “prev” sebelumnya, mudah kan? Kita bisa melihat struktur data tersebut di bawah ini!



Gambar 1 Doubly Linked List

Linked List Java Collection menerapkan implementasi struktur data Doubly Linked List. Kelas Linked List pada Java memiliki sifat sebagai berikut:

a. Kelas ini dapat mengandung elemen duplikat (dua elemen atau lebih dapat memiliki nilai yang sama)
b. Kelas ini mengatur urutan masuk elemen
c. Kelas ini adalah non-synchronized (dua atau lebih thread dapat mengakses kelas tersebut dalam waktu yang bersamaan[1]. Thread akan dipelajari pada mata kuliah tingkat berikutnya)
d. Manipulasi / perubahan pada susunan linked list dapat berlangsung dengan cepat karena tidak diperlukan pergeseran
e. Kelas ini dapat digunakan sebagai list, stack, atau queue.

3.3.2 Method pada kelas Linked List Java Collections.

Kelas LinkedList ini memiliki banyak method berguna yang bisa kita pakai untuk memudahkan kita dalam memprogram, lho. Sama seperti *ArrayList* yang mana kita tinggal pakai method-method seperti .add() untuk menambahkan elemen dan .remove() untuk menghapus elemen, pada LinkedList, kita juga memiliki banyak method berguna untuk operasi manipulasi data.

Tabel 1 menunjukkan method-method yang tersedia pada kelas ini.

Method	Keterangan
LinkedList<String> lst = new LinkedList<String>();	Membuat linked list baru (linked list kosong)
addLast(entry)	Menambahkan node di belakang linked list. Sama seperti add
addFirst(entry)	Menambahkan node di depan linked list
getFirst()	Mengambil data yang disimpan pada node paling depan
getLast()	Mengambil data yang disimpan pada node paling belakang

removeFirst()	Menghapus elemen pertama pada list dan mengembalikannya.
removeLast()	Menghapus elemen terakhir pada list dan mengembalikannya.
ListIterator<String> iter = lst.listIterator()	Membuat iterator untuk menelusuri linked list

Sekarang, pertama-tama, mari kita membuat sebuah program kecil agar kita bisa tahu bagaimana method-method punya si Linked List ini bekerja!

```
import java.util.LinkedList;
```

Pertama-tama, kita import *class* LinkedListnya terlebih dahulu.

```
public class ContohList {
    public static void main(String[] args) {
        LinkedList<String> karakter = new LinkedList<>();
    }
}
```

Lalu kita membuat *class* utama yang namanya sama dengan nama file. Di dalamnya, ada method main. Mari kita buat LinkedList baru yang akan menyimpan String nama-nama karakter yang akan kita masukkan!

```
karakter.addLast("Reimu Hakurei");
karakter.add("Marisa Kirisame");
karakter.add("Sanae Kochiya");
```

Lalu setelahnya, setelah kita membuat sebuah objek “karakter” yang berupa linked list yang menerima String, kita jadi bisa deh memakai method-method punya LinkedList. Di antaranya adalah addLast() dan add(). Mereka berdua merupakan method dengan fungsionalitas sama, yaitu menambahkan elemen baru ke dalam list. Tapi kok bisa kegunaannya sama dan namanya doang yang beda? Itu karena sebenernya, Linked List ini mengimplementasikan sebuah *interface*, atau kayak “spesifikasi” gitu di belakangnya. Dia mengimplementasikan “Queue” dan “Dequeue” yang mana kedua “spesifikasi” tersebut punya method yang kegunaannya sama. Jadinya gitu deh!

Kalau gitu, kita coba print aja ya. Jangan lupa, untuk ngeprint Linked List ini sama ya kayak ArrayList. Kita harus telusurin per elemen dengan loop! Di sini biar gampang, kita pakai for each aja.

```
for (String name : karakter) {
    System.out.println(name);
}
```

Dan outputnya adalah:

```
Reimu Hakurei
Marisa Kirisame
Sanae Kochiya
```

Sekarang kita akan coba untuk pakai salah satu hal yang telah kita pelajari sebelumnya: Iterator.

```
import java.util.ListIterator;
```

Pertama, import dulu ListIterator.

```
ListIterator<String> iterator = karakter.listIterator();
iterator.next(); // Ini Reimu
```

Setelah itu, kita bisa memakai iterator untuk menelusuri List kita. Iterator ini, setiap kita pakai method .next(), maka si iterator itu bakal maju ke elemen selanjutnya dalam List kita, lho. Dan lagi, ketika kita panggil .next(), kita bisa masukin nilainya ke variabel. Di atas, kita pakai .next() satu kali, yang berarti dia bakalan masuk ke dalam listnya. Kode di atas bakal bikin si iterator ini nunjuk ke elemen pertama dalam list karakter, yaitu “Reimu Hakurei”.

Kalau begitu, mari kita coba panggil si iterator.next() sekali lagi. Tapi kali ini, kita jadiin variabel!

```
String karakterSetelahReimu = iterator.next();
```

Pada kode di atas, ada dua hal yang bakal terjadi. Yang pertama adalah, dia majuin lokasi iterator sekarang ke elemen kedua, yaitu “Marisa Kirisame”. Yang kedua, dia bakal ngembaliin nilai. Si *method* .next() ini bakal sekalian ngembaliin nilai yang lagi dia tunjuk di list sekarang, yaitu “Marisa Kirisame”. Sehingga, variabel karakterSetelahReimu bisa keisi dengan “Marisa Kirisame”, deh!

```
System.out.println(karakterSetelahReimu);
```

Output:
Marisa Kirisame

Dan kalau kita print, hasilnya seperti yang kita bayangkan! Kita juga bisa, lho menghapus node yang lagi ditunjuk sama iteratornya. Sekarang kan iterator sedang berada di Marisa. Kalau kita memakai method .remove() dari iterator, maka dia akan menghapus data yang sedang dia tunjuk!

```
iterator.remove();
```

Dan kalau kita print lagi listnya dengan for each, maka datanya sekarang akan:

```
Reimu Hakurei
Sanae Kochiya
```

Marisa Kirisame sudah dihapus dari list!

3.3.3 Penggunaan LinkedList dengan POJO.

Sekarang, kita akan membuat sebuah doubly linked list menggunakan kelas LinkedList dari Java seperti sebelumnya. dengan contoh sebelumnya, pada program kali ini kita akan menyimpan tipe data bentukan Barang yang berisi atribut kode barang, nama barang, dan stok barang.

```
public class Barang {
    private String kodeBarang; private String namaBarang; int stok;

    public Barang(String kodeBarang, String namaBarang, int stok) {
        this.kodeBarang = kodeBarang; this.namaBarang = namaBarang;
this.stok = stok;
    }

    public String getKodeBarang() {
        return kodeBarang;
    }

    @Override
    public String toString() {
        return "kodeBarang='" + kodeBarang + '\'' + ", namaBarang='" +
namaBarang + '\'' + ", stok= " + stok;
    }
}
```

Lalu, pada *class* Main, kita akan menggunakan LinkedList untuk memanipulasi data Barang yang ada di dalamnya! Pertama-tama, import dulu semua yang dibutuhkan, kemudian buat *class* utamanya.

```
import java.util.LinkedList;
import java.util.ListIterator;

public class Main {
    public static void main(String[] args) {
```

```
    }
}
```

Selanjutnya, kita akan menginisialisasi list barang tersebut menggunakan LinkedList dari Java, dan memasukkan elemen-elemen ke dalamnya!

```
LinkedList<Barang> listBarang = new LinkedList<>();

listBarang.add(new Barang("001", "Tongkat Gohei", 10));
listBarang.addLast(new Barang("002", "Bros Kodok", 20));
listBarang.addFirst(new Barang("003", "Topi Penyihir", 15));
listBarang.add(2, new Barang("005", "Katana", 1));
```

Nah, di sini kita bisa melihat berbagai macam method untuk menambahkan item baru ke list! Pada method pertama, .add(), dia bakal langsung nambahin elemen ke bagian akhir list. Jadi, kalau kita pakai listBarang.add(new Barang(...)), barang tersebut bakal otomatis ditaruh di urutan terakhir.

Terus, ada .addLast(), yang sebenarnya sama aja kayak .add(), karena dua-duanya bakal nambahin elemen di akhir list, kayak yang sudah dibahas sebleumnya.

Lalu, ada .addFirst(), yang sesuai namanya, bakal langsung menaruh elemen di awal list. Jadi, barang yang kita tambahin dengan cara ini bakal jadi elemen pertama di list.

Terakhir, ada .add(index, elemen), yang memungkinkan kita menaruh elemen di posisi tertentu dalam list. Misalnya, di contoh tadi, listBarang.add(2, new Barang("005", "Katana", 1)) bakal memasukkan "Katana" ke posisi indeks 2, menggeser elemen yang ada di posisi itu dan setelahnya ke belakang. Jadi, yang awalnya posisinya adalah Topi Penyihir, Tongkat Gohei, dan Bros Kodok, kita geser Bros Kodok yang awalnya ada di indeks ke-2 dan kita ganti sama Katana. Alhasil, sekarang urutannya adalah Topi Penyihir, Tongkat Gohei, Katana, Bros Kodok.

Selanjutnya, kita akan mencetak mereka. Berbeda seperti sebelumnya, kali ini kita mencetak mereka menggunakan iterator, dan while, bukan for each loop. Kita menggunakan *while* untuk terus melakukan loop selama si iterator ini punya elemen selanjutnya. Jadi kalau elemen selanjutnya udah nggak ada, ya otomatis itu adalah elemen terakhir di listnya.

```
System.out.println("Daftar Barang:");
ListIterator<Barang> iterator = listBarang.listIterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

Nah, outputnya akan berupa setiap elemen yang telah kita masukkan tadi, dengan urutan sesuai bagaimana kita memasukkannya: Topi Penyihir, Tongkat Gohei, Katana, Bros Kodok.

Kemudian, sekarang pastinya iterator karena sudah kita panggil .next() bakal berubah jadi nunjuk ke elemen paling terakhir, bukan? Karena itu, kita bisa cetak secara mundur sekarang!

```
System.out.println("\nCetak mundur list:");
while (iterator.hasPrevious()) {
    System.out.println(iterator.previous());
}
```

Dan outputnya akan berupa urutan item-item kita tadi secara mundur!

```
Cetak mundur list:
kodeBarang='002', namaBarang='Bros Kodok', stok= 20
kodeBarang='005', namaBarang='Katana', stok= 1
kodeBarang='001', namaBarang='Tongkat Gohei', stok= 10
kodeBarang='003', namaBarang='Topi Penyihir', stok= 15
```

Terakhir, kita akan coba untuk menghapus barang dengan ID 005 dari list.

```
ListIterator<Barang> iter = listBarang.listIterator();
while (iter.hasNext()) {
    Barang ob = iter.next();
    if (ob.getKodeBarang().equals("005")) {
        iter.remove();
    }
}
```

Di sini, kita menggunakan sebuah variabel ob untuk mengambil tiap elemen dalam list, lalu kita pakai variabelnya untuk mendapatkan id dari data yang sedang kita ambil tersebut, lalu kita bandingkan dengan “005”. Kalau ternyata benar, maka kita panggil method .remove() yang kita telah pelajari sebelumnya.

```
System.out.println("\nSesudah hapus data:");
for (Barang item : listBarang) {
    System.out.println(item);
}
```

Bila kita print, maka outputnya sekarang akan berupa:

```
Sesudah hapus data:
kodeBarang='003', namaBarang='Topi Penyihir', stok= 15
kodeBarang='001', namaBarang='Tongkat Gohei', stok= 10
kodeBarang='002', namaBarang='Bros Kodok', stok= 20
```

Lalu gimana kalau kita ingin edit data barang? Nah, kita bisa mengambil objek yang sedang ditunjuk oleh iterator dengan menjadikannya variabel, kemudian langsung kita ubah aja nilai di dalamnya! Di sini, kita akan mengubah stok. Karena stok tidak private, maka kita tidak perlu menggunakan setter. Kita akan langsung memanggil atributnya, seperti di bawah.

```
if (iter.hasPrevious()) {
    Barang ob = iter.previous();
    ob.stok = 5;
}
```

Lalu apa alasan kita memakai .hasPrevious() dan .previous()? Itu karena kita sudah memanggil `while (iter.hasNext())` sebelumnya, yang menyebabkan iterator sekarang berada atau menunjuk pada data terakhir dalam LinkedList. Bila kita print kembali data-data kita, maka kita bisa melihat perubahan jumlah stok dari item Bros Kodok, yang merupakan item terakhir kita dalam list!

```
Sesudah edit data:
kodeBarang='003', namaBarang='Topi Penyihir', stok= 15
kodeBarang='001', namaBarang='Tongkat Gohei', stok= 10
kodeBarang='002', namaBarang='Bros Kodok', stok= 5
```

Dan begitulah untuk modul kali ini! Selamat belajar!