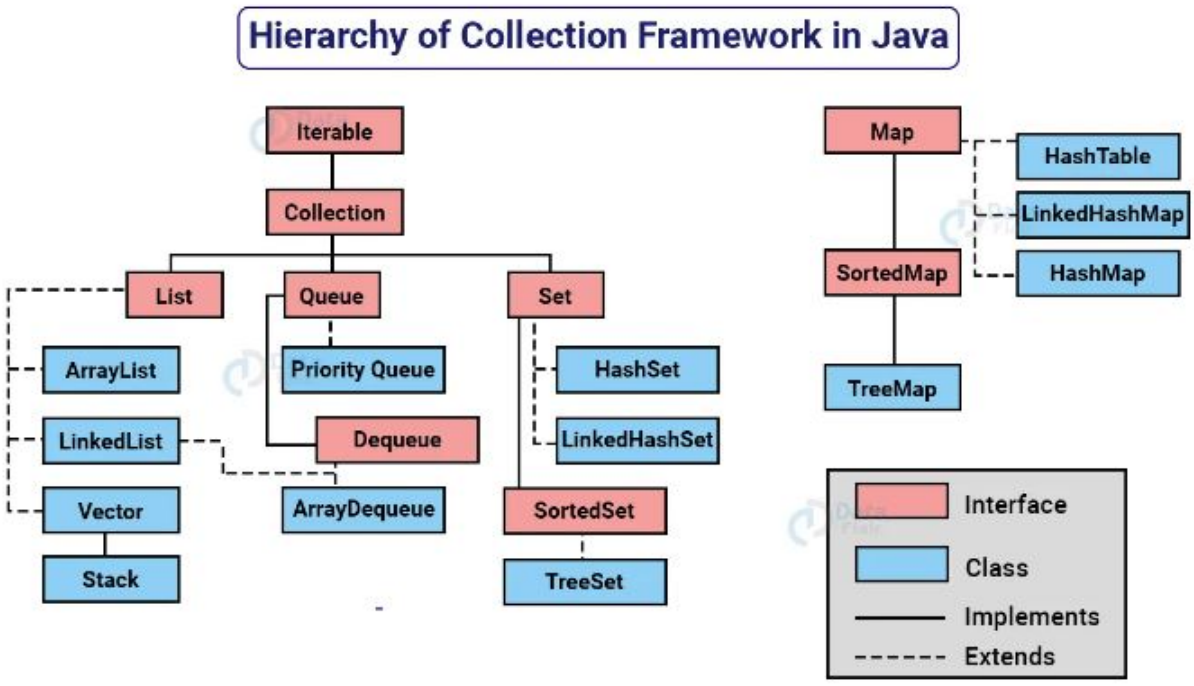
01.07.0003 Java Collection Framework

Java collection merupakan mekanisme pada Java untuk memanipulasi kumpulan object (dengan tipe data yang sama) menjadi sebuah kesatuan. Secara mudahnya, ini bisa dianggap pengembangan lebih lanjut dari Array dengan penambahan beberapa method yang memudahkan untuk manipulasi kumpulan object tersebut secara lebih mudah dan efisien.

Hierarki level java collection, termasuk interface dan implementasinya ditunjukkan pada gambar berikut:



Secara garis besar, ada 3 jenis (*interface*) collection + 1 map yaitu:

- `List` : menyimpan kumpulan object (dengan tipe data yang sama) dan dapat diakses menggunakan index yang dimulai dari 0 hingga n - 1
- `Queue` : menyimpan kumpulan object yang diakses dengan urutan First In First Out (FIFO), artinya data hanya bisa ditambahkan di akhir rangkaian dan hanya dapat diproses dari awal rangkaian. Pengembangan selanjutnya dari `Queue` adalah `Dequeue` (double ended queue) yang memungkinkan penambahan dan pemrosesan baik dari awal atau akhir rangkaian.
- `Set` : menyimpan kumpulan object secara unik, artinya object yang sama hanya dapat ditambahkan 1 kali saja ke dalam tipe data set.
- `Map` : menyimpan kumpulan object dengan mekanisme key & value yang dapat didefinisikan sendiri. Alih-alih menyimpan dengan index angka, Map memungkinkan kita mendefinisikan key untuk referensi pengaksesan value yang tersimpan.

Karena bersifat interface, keempat jenis collection tersebut tidak langsung dapat digunakan, namun harus diimplementasikan ke dalam sebuah class kongkrit terlebih dahulu. Java sudah menyiapkan beberapa implementasi dari masing-masing collection, namun di sini kita akan membahas 3 implementasi dari collection tersebut yaitu `ArrayList` , `HashSet` , dan `HashMap` .

ArrayList

ArrayList sangat mirip dengan tipe array, namun jumlah datanya tidak dibatasi secara statis bisa bertambah secara dinamis sesuai dengan kebutuhan. Cara deklarasi ArrayList adalah sebagai berikut:

```
// import java.util.ArrayList;
ArrayList<tipe_data> namaVariabel = new ArrayList<>();
```

Adapun beberapa method yang banyak digunakan untuk manipulasi ArrayList adalah:

- `add(object)` : menambahkan data
- `get(index)` : mendapatkan data di index tertentu
- `set(index, object)` : mengubah data pada index tertentu
- `remove(index)` : menghapus data pada index tertentu
- `clear()` : menghapus semua data pada ArrayList
- `isEmpty()` : mengecek apakah sebuah ArrayList kosong atau tidak
- `size()` : mendapatkan jumlah object yang tersimpan pada ArrayList

Berikut adalah contoh koding menggunakan ArrayList:

```
ArrayList<String> listDemotivasi = new ArrayList<>();

listDemotivasi.add("Jangan pesimislah, apalagi optimis!");
listDemotivasi.add("Kalau orang lain bisa, kenapa harus saya?");
listDemotivasi.add("Jangan mengulangi kesalahan yang sama! Kesalahan yang lain masih
banyak.");
listDemotivasi.add("Uang tidak bisa membawa kebahagiaan kalau uangnya tidak ada di
rekening kita.");
listDemotivasi.add("Pekerjaan seberat apa pun akan terasa ringan, apabila tidak
dikerjakan.");
listDemotivasi.add("Namanya hidup, pasti banyak cobaan. Kalau banyak cucian, Loundry
namanya.");
listDemotivasi.add("Nggak perlu merasa jago, karena anda bukan ayam.");
listDemotivasi.add("Cantik itu relatif, tergantung posisi kamera dan intensitas cahaya.");

System.out.println(listDemotivasi.get(5));
listDemotivasi.set(5, "Diam itu takkan menyelesaikan masalah. Tapi diam juga takkan
menimbulkan masalah.");
listDemotivasi.remove(3);

for(String demotivasi: listDemotivasi){
        System.out.println(demotivasi);
}

System.out.println(listDemotivasi.isEmpty()+" : "+listDemotivasi.size());
```

Traversal dengan `.forEach()`

Pada contoh sebelumnya, kita bisa melakukan operasi traversal (melakukan sebuah operasi ke semua elemen list) menggunakan **for** atau **for-each** bawaan Java. Selain cara tersebut, dalam `ArrayList` kita juga bisa memanggil method `forEach()` untuk melakukan fungsi traversal.

Sebagai contoh, untuk mencetak semua quotes demotivasi di atas, kita bisa melakukannya dengan cara seperti di bawah ini:

```
listDemotivasi.forEach(quotes -> System.out.println());
```

> *Catatan:* method `forEach` memiliki beberapa keterbatasan dibandingkan cara traversal biasa dengan **for** atau **for-each** bawaan dari Java, sehingga method tersebut lebih cocok digunakan untuk eksekusi perintah sederhana.

Mengubah Array Menjadi List

Kita bisa dengan mudah mengubah sebuah ArrayList menjadi Array dengan memanggil method `toArray()`. Adapun untuk operasi sebaliknya, yaitu mengubah Array biasa menjadi ArrayList (atau List lebih tepatnya), kita perlu menggunakan bantuan method static `Arrays.asList()`. Berikut adalah contohnya:

```
String[] textArray = {"Merah", "Kuning", "Hijau", "Biru", "Ungu"};
List<String> textList =  Arrays.asList(textArray);
textList.forEach(teks -> System.out.println(teks));
```

> *Catatan:* Seperti ditunjukkan pada gambar sebelumnya, `List` adalah induk dari `ArrayList`. Class ini biasa digunakan untuk penampungan tipe list yang lebih umum.

Pengurutan dengan `.sort()`

Pengurutan objek dalam java memerlukan mekanisme khusus (*interface*) yang disebut `Comparator`. Beberapa tipe data bawaan Java seperti `String`, `Integer`, dan `Double` sudah memiliki mekanisme tersebut. Sehingga untuk mengurutkannya kita tinggal memanggil method `.sort()` dengan pengurutan yang diinginkan sepertin ditunjukkan contoh di bawah ini.

```
List<String> textList =  Arrays.asList("Merah", "Kuning", "Hijau", "Biru", "Ungu");
textList.sort(Comparator.naturalOrder());
textList.forEach(teks -> System.out.println(teks));
```

HashSet

`HashSet` mirip dengan `ArrayList` hanya saja data yang disimpan bersifat unik dan otomatis terurut, artinya jika data yang sama dimasukkan lebih dari satu kali, maka hanya 1 yang tersimpan. Berikut contoh kodingnya:

```
HashSet<String> dataBuah = new HashSet<>();
dataBuah.add("Anggur");
dataBuah.add("Durian");
dataBuah.add("Melon");
dataBuah.add("Lemon");
dataBuah.add("Durian");
dataBuah.add("Melon");
dataBuah.add("Strawbery");

for (String buah : dataBuah) {
        System.out.println(buah);
}
```

Tips Mencari Nilai Unik pada ArrayList:

- Karena HashSet mempunyai karakteristik yang mirip dengan ArrayList, jika kita ingin mencari nilai unik pada ArrayList, kita bisa langsung memasukkannya pada HashSet.
- Berikut contoh kodingnya:

```
ArrayList<String> dataMhs = new ArrayList<>();
dataMhs.addAll( Arrays.asList("A1234567", "A2345678", "B1234567",
                                                    "A2345678", "C123456",
"C123456") );
HashSet<String> setMhs = new HashSet<>(dataMhs);
setMhs.forEach((mhs) -> System.out.println(mhs));
```

> Set dan list memiliki banyak kesamaan karakteristik termasuk kesamaan method seperti `remove()`, `clear()`, `isEmpty()`, dan `size()`.

HashMap

`HashMap` memungkinkan kita menyimpan kumpulan object dalam bentuk key-value, artinya kita dapat mendefinisikan sendiri key untuk setiap penyimpanan objectnya.

Beberapa method yang banyak digunakan pada HashMap adalah:

- `put(key, value)` : menambah dan merubah nilai dengan key tertentu.
- `remove(key)` : menghapus element map dengan key tertentu.
- `entrySet()` : mengembalikan semua key yang tersimpan dalam bentuk HashSet. Method ini dapat juga digunakan untuk traversal menggunakan for-each loop.
- `get(key)` : mengecek apakah sebuah key ada pada HashMap.
- `clear()` : menghapus semua data pada HashMap.
- `isEmpty()` : mengecek apakah sebuah HashMap kosong atau tidak.
- `size()` : mendapatkan jumlah object yang tersimpan pada HashMap.

Berikut adalah contoh koding HashMap dengan key berupa String dan value juga berupa String:

```
HashMap<String, String> webSettings = new HashMap<>();
webSettings.put("url", "dinus.ac.id");
webSettings.put("debug_level", "ERROR_ALL");
webSettings.put("method", "GET|POST");
webSettings.put("address", "132.190.12.49");
webSettings.put("webserver", "Nginx");

System.out.println("Jumlah entry: "+webSettings.size());

System.out.println("IP Address: "+webSettings.getOrDefault("address", "tidak
terdefinisi"));
System.out.println(webSettings.get("webserver"));
webSettings.put("webserver", "Apache");

for(Map.Entry<String,String> entry: webSettings.entrySet()){
        System.out.println(entry.getKey()+" : "+entry.getValue());
}
```

Pengenalan Stream

Stream merupakan urutan proses atau objek yang dapat digunakan untuk manipulasi lebih lanjut dari collection yang sudah ada. Contoh operasi yang dapat dilakukan dengan Stream adalah operasi traversal, operasi agregasi, operasi filter, dan lain sebagainya.

Secara konseptual, semua proses yang dapat dilakukan menggunakan Stream juga dapat dilakukan menggunakan for loop. Meski demikian, proses yang dilakukan menggunakan Stream relatif lebih efektif dan dapat menerapkan paralelisasi proses. Selain itu, Stream juga memungkinkan kita mengadopsi teknik pemrograman fungsional pada Java.

Studi Kasus: Class Mahasiswa

Pada tutorial ini kita akan menggunakan studi kasus Class Mahasiswa sederhana untuk mendemonstrasikan proses Stream yang lebih kompleks. Class mahasiswa yang digunakan pada tutorial ini adalah sebagai berikut:

```
public class Mahasiswa {
    private String nim;
    private String nama;
    private int sksTotal;
    private double ipk;

    public Mahasiswa(String nim, String nama, int sksTotal, double ipk) {
        this.nim = nim;
        this.nama = nama;
        this.sksTotal = sksTotal;
        this.ipk = ipk;
    }

    public String getNim() {
        return nim;
    }

    public String getNama() {
        return nama;
    }

    public int getSksTotal() {
        return sksTotal;
    }

    public double getIpk() {
        return ipk;
    }

        @Override
    public String toString() {
        return "| "+nim+" | "+nama+" ".repeat(20-nama.length())+" | "
                +sksTotal+" ".repeat(3-Integer.toString(sksTotal).length())+" | "
                +ipk+" ".repeat(4-Double.toString(ipk).length())+" |";
    }
}
```

Load Data dari CSV

Kita akan load data dari CSV untuk memudahkan input banyak data sekaligus. Adapun library yang kita gunakan untuk loading CSV adalah Apache Commons CSV. Jika kita menggunakan gradle, kita dapat menambahkan kode berikut di bagian `dependencies` pada file `build.gradle`

```
implementation 'org.apache.commons:commons-csv:1.10.0'
```

Selanjutnya kita membuat class baru dengan nama `Loader.java` dan isikan koding sebagai berikut:

```
public class Loader {
    public static ArrayList<Mahasiswa> csvMhsLoad(){
        String path = "./data-mahasiswa.csv";
        ArrayList<Mahasiswa> dataMhs = new ArrayList<>();
        String [] headers = {"nim", "nama", "sksTotal", "ipk"};
        dataMhs.clear();
        try {
            FileReader reader = new FileReader(path);
            Iterable<CSVRecord> recordData = CSVFormat.DEFAULT.builder().setH
                                                .setSkipHeaderRecord(true)
                                                .build().parse(reader);
            for(CSVRecord record: recordData){
                dataMhs.add(new Mahasiswa(record.get("nim"), record.get("nama"),
                                    Integer.parseInt( record.get("sksTotal") ),
                                    Double.parseDouble(record.get("ipk"))) );
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return dataMhs;
    }
}
```

`.sorted()`

Di bagian sebelumnya kita sudah melakukan pengurutan pada `List` menggunakan method `sort()`, hanya saja method tersebut akan melakukan perubahan urutan data pada data aslinya. Stream menyediakan method tambahan `sorted()` untuk mengurutkan data tanpa mengubah data aslinya, berikut adalah cara penggunaanya.

```
String[] textArray = {"Merah", "Kuning", "Hijau", "Biru", "Ungu"};
ArrayList<String> textList =  new ArrayList<>(Arrays.asList(textArray));
List<String> newList = textList.stream().sorted(Comparator.naturalOrder()).toList();
textList.forEach(teks -> System.out.println(teks));
System.out.println("=".repeat(20));
newList.forEach(teks -> System.out.println(teks));
```

Seperti dijelaskan sebelumnya, beberapa tipe data bawaan Java sudah memiliki mekanisme `Comparator`-nya sendiri, sehingga kita dapat langsung mengurutkannya. Sedangkan class `Mahasiswa` yang kita buat belum memiliki mekanisme tersebut, sehingga untuk pengurutannya kita perlu melakukan pekerjaan tambahan. Di bawah ini adalah contoh pengurutan class `Mahasiswa` dengan pilihan field patokan yang berbeda.

```
public List<Mahasiswa> sort(int sortBy){
        if(sortBy == 1){
                return dataMhs.stream()
                                .sorted(Comparator.comparing(Mahasiswa::getNama))
                                .toList();
        } else if(sortBy == 2){
                return dataMhs.stream()

.sorted(Comparator.comparing(Mahasiswa::getIpk).reversed())
                                .toList();
        } else {
                return dataMhs.stream()
                                .sorted((m1, m2) -> m1.getNim().compareTo(m2.getNim()))
```

```
                                .toList();
        }
}
```

Pada method `sort` di atas kita menggunakan method `stream()` untuk mempersiapkan operasi stream (dalam kasus ini adalah rangkaian pengurutan). Selanjutnya kita bisa menentukan field apa yang menjadi patokan pengurutan dengan menggunakan method static `Comparator.comparing` atau menggunakan pemanggilan method `compareTo()` seperti di contohkan pada blok `else`.

Fungsi yang kita buat tersebut, kemudian dapat kita panggil dengan cara berikut:

```
ArrayList<Mahasiswa> dataMhs = (ArrayList<Mahasiswa>) this.sort(2);
dataMhs.forEach(mahasiswa -> System.out.println(mahasiswa));
```

> Java menyediakan beberapa cara untuk sorting untuk fleksibilitas pengurutan dari cara yang paling sederhana hingga yang sangat rumit. Silahkan merujuk ke dokumentasi untuk mendapatkan referensi yang sesuai.

`.map()`

Mirip dengan `.forEach()`, method `.map()` dapat digunakan untuk melakukan operasi pada semua elemen list. Perbedaannya, kita hanya melakukan operasi saja tanpa melakukan perubahan pada elemen list maka kita menggunakan `.forEach()`, sedangkan jika kita ingin melakukan operasi dengan mengaplikasikan perubahan pada elemen list maka kita menggunakan `.map()`. Berikut contoh implementasinya pada tipe data Integer.

```
List<Integer> dataNilai = Arrays.asList(78, 90, 30, 50, 66, 87, 70);
dataNilai.forEach(nilai -> System.out.print(nilai+" "));
System.out.println();
List<Integer> newNilai = dataNilai.stream().map((nilai) -> nilai+10)

.collect(Collectors.toList());
newNilai.forEach((nilai) -> System.out.print(nilai+" "));
```

Implementasi pada object mahasiswa untuk memproses nama mahasiswa menjadi huruf kapital adalah sebagai berikut:

```
List<String> listNama = dataMhs.stream().map(mhs -> mhs.getNama().toUpperCase())

.collect(Collectors.toList());
listNama.forEach(nama -> System.out.println(nama));
```

> Pada koding di atas, method `.map()` akan menghasilkan List of String dengan nama yang sudah dijadikan kapital. Jika kita ingin merubah nama di dalam objek `Mahasiswa` semua berubah menjadi kapital, maka method `forEach()` untuk memanggil method setter lebih cocok digunakan.

`.reduce()`

Method `.reduce()` akan melakukan pemrosesan terhadap semua elemen serta menggabungkannya menjadi satu nilai tunggal. Method ini bisa dibilang dasar dari operasi agregasi seperti count, sum, max, min, dan average. Berikut adalah contoh penggunaan reduce untuk mendapatkan total tagihan SKS dari list jumlah SKS yang diambil beberapa mahasiswa.

```
List<Integer> listSks = Arrays.asList(20, 18, 22, 24);
int total = listSks.stream().reduce(0, (hasil, sks) -> hasil + (sks*200000));
System.out.println("Total tagihan: "+total);
```

Adapun untuk pemrosesan terhadap object yang kita buat sendiri, seringkali kita memerlukan bantuan method `.map()` untuk mendapatkan list nilai yang diinginkan, baru kemudian gunakan `.reduce()` untuk memproses list tersebut. Berikut adalah contoh koding untuk mendapatkan rerata IPK dari semua mahasiswa.

```
double rerata = dataMhs.stream().map(mhs -> mhs.getIpk())
                                        .reduce(0.0, (sum, ipk) -> sum + ipk) /
dataMhs.size();
System.out.println("Rata-rata IPK: "+(rerata));
```

`.filter()`

Method `.filter()` dapat mengembalikan list baru dari list lama yang sudah diseleksi dengan kriteria tertentu. Berikut adalah contoh filter untuk mendapatkan hanya nilai genap dari list integer.

```
List<Integer> dataAngka = Arrays.asList(78, 12, 33, 67, 23, 51, 32, 50);
List<Integer> angkaGenap = dataAngka.stream().filter(angka -> (angka % 2) == 0)

.collect(Collectors.toList());
angkaGenap.forEach(angka -> System.out.print(angka+" "));
```

Berikut adalah contoh filter untuk mendapatkan list mahasiswa yang memiliki IPK di atas atau sama dengan 3.5 saja.

```
List<Mahasiswa> mhsCumlaude = dataMhs.stream().filter(mhs -> (mhs.getIpk()) >= 3.5)

.collect(Collectors.toList());
mhsCumlaude.forEach(mhs -> System.out.println(mhs));
```