POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

DIKTISAINTEK BERDAMPAK KAMPUS INOVASI



COLLECTION API

PBO

Yunia Ikawati

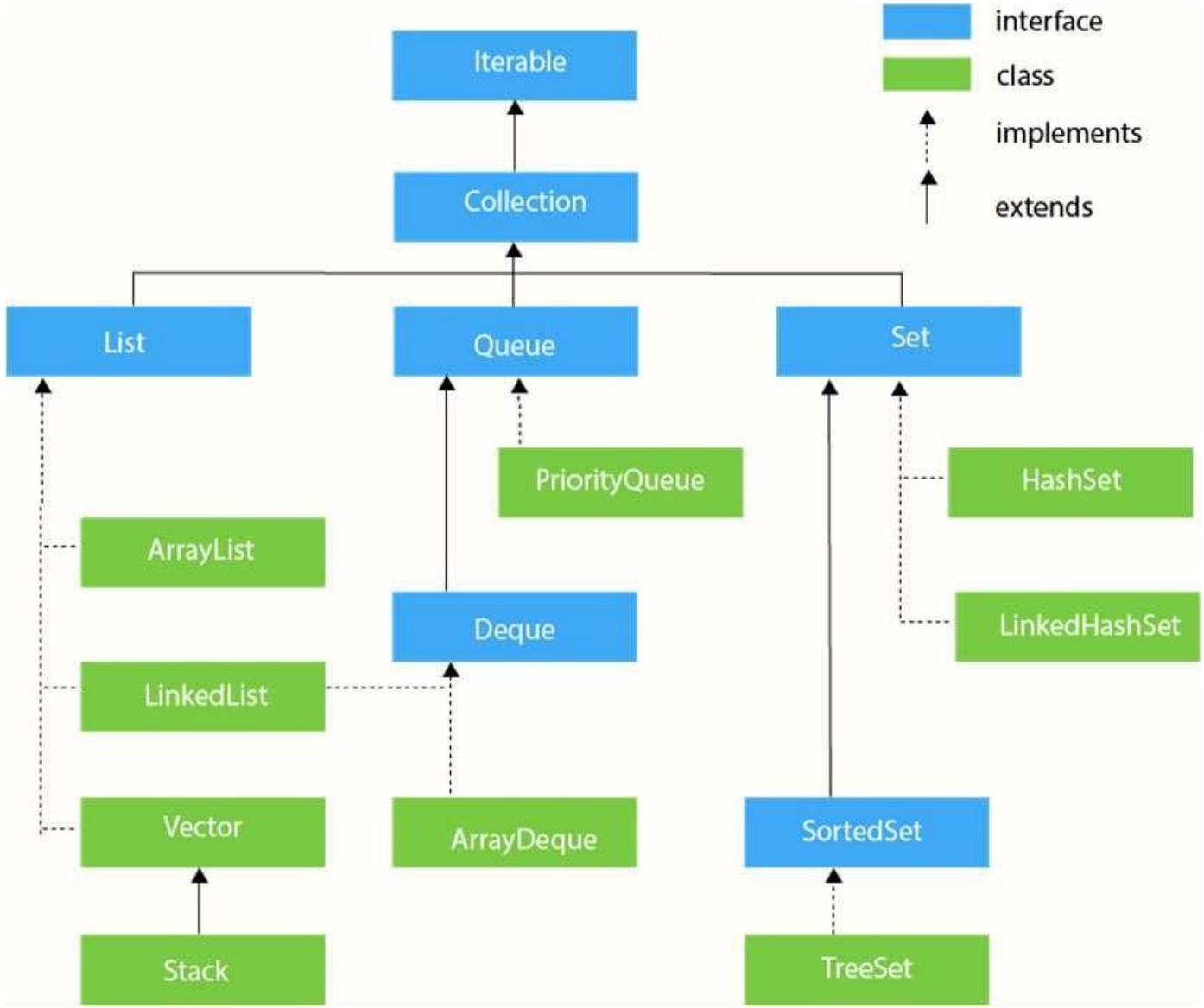
Teknik Informatika-PENS

Collection

- Merupakan sebuah framework yang menyediakan arsitektur untuk menyimpan dan memanipulasi sekumpulan objek.
- Sekumpulan interface dan class di Java yang digunakan untuk menyimpan dan memanipulasi kumpulan objek. Berada dalam **package java.util.**
- Java Collection Framework menyediakan banyak interface (Set, List, Queue, Deque) dan class (ArrayList, Vector, LinkedList, HastSet, TreeSet)
- Mempermudah pengelolaan data (list, set, map) tanpa harus membuat struktur data dari nol.

NO	Key	Arrays	Collection
1	Size	Memiliki ukuran (size) yang tetap (fix). Sekali array dideklarasikan dengan ukuran tertentu kita tidak dapat merubah ukuran array tersebut.	Memiliki ukuran yang dinamis
2	Memory Consumption	Array dieksekusi dengan cepat dan memiliki kinerja yang baik, namun menghabiskan lebih banyak memori.	Konsumsi memori lebih rendah namun performa lebih rendah dibandingkan dengan Array
3	Data Type	Hanya dapat menampung data yang sejenis atau memiliki tipe data yang sama. (homogeneous data type)	Dapat menyimpan data yang homogen ataupun heterogen
4	Primitives Storage	Array dapat menampung baik tipe data primitive ataupun object	Hanya dapat menyimpan data bertipe object
5	Performance	Karena impementasi dan penyimpanannya internal, Array memiliki kinerja yang lebih baik.	Memiliki performa yang lebih rendah dibanding array

DIAGRAM HIERARKI COLLECTION FRAMEWORK DI JAVA



Collection

Interface Utama:
Collection → induk dari semua koleksi.

Turunan Penting:
List → urutan elemen, bisa duplikat (contoh: ArrayList, LinkedList).
Set → tidak boleh duplikat (contoh: HashSet, TreeSet).
Queue → antrian (contoh: PriorityQueue).

Interface dan Implementasi

List:

ArrayList → berbasis array, akses cepat.

LinkedList → berbasis node, cocok untuk operasi tambah/hapus.

Set:

HashSet → tidak berurutan, cepat.

TreeSet → terurut.

Contoh ArrayList

```java
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        System.out.println("ArrayList: " + fruits);

        // Akses elemen cepat
        System.out.println("Elemen ke-2: " + fruits.get(1));
    }
}
```

```
ArrayList: [Apple, Banana, Cherry]
Elemen ke-2: Banana
```

Contoh LinkedList

```java
import java.util.LinkedList;

public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<String> animals = new LinkedList<>();
        animals.add("Cat");
        animals.add("Dog");
        animals.add("Rabbit");

        System.out.println("LinkedList: " + animals);

        animals.addFirst("Elephant");
        animals.addLast("Tiger");

        System.out.println("Setelah tambah: " + animals);

        animals.remove("Dog");
        System.out.println("Setelah hapus Dog: " + animals);
    }
}
```

```
LinkedList: [Cat, Dog, Rabbit]
Setelah tambah: [Elephant, Cat, Dog, Rabbit, Tiger]
Setelah hapus Dog: [Elephant, Cat, Rabbit, Tiger]
```

Contoh HashSet

```java
import java.util.HashSet;
import java.util.Set;

public class HashSetExample {
    public static void main(String[] args) {
        Set<String> hashSet = new HashSet<>();
        hashSet.add("Apple");
        hashSet.add("Banana");
        hashSet.add("Cherry");
        hashSet.add("Apple"); // duplikat, tidak akan ditambahkan

        System.out.println("HashSet (tidak berurutan): " + hashSet);
    }
}
```

```
HashSet (tidak berurutan): [Banana, Cherry, Apple]
```

Contoh TreeSet

```java
import java.util.TreeSet;
import java.util.Set;

public class TreeSetExample {
    public static void main(String[] args) {
        Set<String> treeSet = new TreeSet<>();
        treeSet.add("Apple");
        treeSet.add("Banana");
        treeSet.add("Cherry");
        treeSet.add("Apple"); // duplikat, tidak akan ditambahkan

        System.out.println("TreeSet (terurut): " + treeSet);
    }
}
```

```
TreeSet (terurut): [Apple, Banana, Cherry]
```

Queue

- **Queue** dalam Java adalah struktur data yang mengikuti prinsip **FIFO (First In, First Out)**, artinya elemen yang pertama masuk akan diproses pertama.
- Queue digunakan untuk antrian data.

Contoh Queue

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

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        // Menambahkan elemen ke antrian
        queue.add("Yunia");
        queue.add("Ikawati");
        queue.add("Java");

        System.out.println("Isi Queue: " + queue);

        // Mengambil elemen pertama
        String first = queue.poll();
        System.out.println("Elemen yang diambil: " + first);
        System.out.println("Queue setelah poll: " + queue);

        // Melihat elemen pertama tanpa menghapus
        String peekElement = queue.peek();
        System.out.println("Elemen pertama sekarang: " + peekElement);
    }
}
```

```
Isi Queue: [Yunia, Ikawati, Java]
Elemen yang diambil: Yunia
Queue setelah poll: [Ikawati, Java]
Elemen pertama sekarang: Ikawati
```

Operasi Dasar pada Collection

- Menambah elemen: add()
- Menghapus elemen: remove()
- Mengecek isi: contains()
- Ukuran: size()
- Iterasi:
  - Menggunakan for-each
  - Menggunakan Iterator

Contoh Operasi Dasar pada Collection

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class CollectionOperations {
    public static void main(String[] args) {
        // Membuat List
        List<String> names = new ArrayList<>();

        // Menambah elemen: add()
        names.add("Yunia");
        names.add("Ikawati");
        names.add("Java");
        System.out.println("Setelah add(): " + names);

        // Menghapus elemen: remove()
        names.remove("Java");
        System.out.println("Setelah remove(): " + names);

        // Mengecek isi: contains()
        boolean hasYunia = names.contains("Yunia");
        System.out.println("Apakah mengandung 'Yunia'? " + hasYunia);

        // Ukuran: size()
        System.out.println("Ukuran List: " + names.size());

        // Iterasi menggunakan for-each
        System.out.println("Iterasi dengan for-each:");
        for (String name : names) {
            System.out.println("- " + name);
        }

        // Iterasi menggunakan Iterator
        System.out.println("Iterasi dengan Iterator:");
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            System.out.println("* " + iterator.next());
        }
    }
}
```

```
Setelah add(): [Yunia, Ikawati, Java]
Setelah remove(): [Yunia, Ikawati]
Apakah mengandung 'Yunia'? true
Ukuran List: 2
Iterasi dengan for-each:
- Yunia
- Ikawati
Iterasi dengan Iterator:
* Yunia
* Ikawati
```

Comparator dan Comparable

- Comparable → digunakan untuk natural order (satu aturan bawaan).
- Comparator → digunakan untuk custom order (bisa banyak aturan).

Contoh Comparator dan Comparable

```java
import java.util.*;

class Mahasiswa implements Comparable<Mahasiswa> {
    String nama;
    int umur;

    Mahasiswa(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
    }

    // Natural order: berdasarkan nama (ascending)
    @Override
    public int compareTo(Mahasiswa m) {
        return this.nama.compareTo(m.nama);
    }

    @Override
    public String toString() {
        return nama + " (" + umur + ")";
    }
}
```

```java
public class SortingExample {
    public static void main(String[] args) {
        List<Mahasiswa> list = new ArrayList<>();
        list.add(new Mahasiswa("Yunia", 22));
        list.add(new Mahasiswa("Ikawati", 21));
        list.add(new Mahasiswa("Java", 23));

        // Sorting natural order (nama) menggunakan Comparable
        Collections.sort(list);
        System.out.println("Sorting Natural Order (nama): " + list);

        // Sorting custom order (umur) menggunakan Comparator
        Collections.sort(list, new Comparator<Mahasiswa>() {
            @Override
            public int compare(Mahasiswa m1, Mahasiswa m2) {
                return Integer.compare(m1.umur, m2.umur);
            }
        });
        System.out.println("Custom Sorting (umur): " + list);
    }
}
```

```
Sorting Natural Order (nama): [Ikawati (21), Java (23), Yunia (22)]
Custom Sorting (umur): [Ikawati (21), Yunia (22), Java (23)]
```

Utility Class: Collections

1. **sort()**
   - Mengurutkan elemen dalam koleksi (misalnya List) secara ascending (menaik).
   - Disediakan oleh Collections.sort().
2. **reverse()**
   - Membalik urutan elemen dalam koleksi.
   - Disediakan oleh Collections.reverse().
3. **shuffle()**
   - Mengacak urutan elemen dalam koleksi.
   - Disediakan oleh Collections.shuffle().
4. **min()**
   - Mengembalikan elemen minimum dari koleksi.
   - Disediakan oleh Collections.min().
5. **max()**
   - Mengembalikan elemen maksimum dari koleksi.
   - Disediakan oleh Collections.max().

Contoh Utility Class: Collections

```java
import java.util.*;

public class CollectionDemo {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>(Arrays.asList(10, 5, 20, 15, 30));

        // sort()
        Collections.sort(numbers);
        System.out.println("Sorted: " + numbers);

        // reverse()
        Collections.reverse(numbers);
        System.out.println("Reversed: " + numbers);

        // shuffle()
        Collections.shuffle(numbers);
        System.out.println("Shuffled: " + numbers);

        // min() dan max()
        int minValue = Collections.min(numbers);
        int maxValue = Collections.max(numbers);
        System.out.println("Minimum: " + minValue);
        System.out.println("Maximum: " + maxValue);
    }
}
```

```
Sorted: [5, 10, 15, 20, 30]
Reversed: [30, 20, 15, 10, 5]
Shuffled: [15, 30, 10, 5, 20]  // urutan acak
Minimum: 5
Maximum: 30
```