



# Java Collection

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 12+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)





# Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- LinkedIn : <https://www.linkedin.com/company/programmer-zaman-now/>
- Facebook : [fb.com/ProgrammerZamanNow](https://fb.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Tiktok : <https://tiktok.com/@programmerzamannow>
- Email : [echo.khannedy@gmail.com](mailto:echo.khannedy@gmail.com)



# Sebelum Belajar

- Java Dasar
- Java Object Oriented Programming
- Java Standard Classes
- Java Generic
- <https://www.udemy.com/course/pemrograman-java-pemula-sampai-mahir/?referralCode=E97428FBE9A6F3590D8D>



# Agenda

- Pengenalan Collection
- Iterable
- List
- Set
- Queue
- Deque
- Map
- Dan lain-lain

---

# Pengenalan Collection



# Pengenalan Collection

- Collection adalah hal umum yang biasa dimiliki di bahasa pemrograman, seperti PHP, Python, Ruby dan lain-lain
- Collection atau kadang disebut container, adalah object yang mengumpulkan atau berisikan data-data, mirip seperti Array
- Java telah menyediakan class-class collection yang sudah bisa langsung kita gunakan, tanpa tambahan library
- Semua class-class collection di Java adalah generic class, sehingga kita bisa menggunakan Java collection dengan data apapun



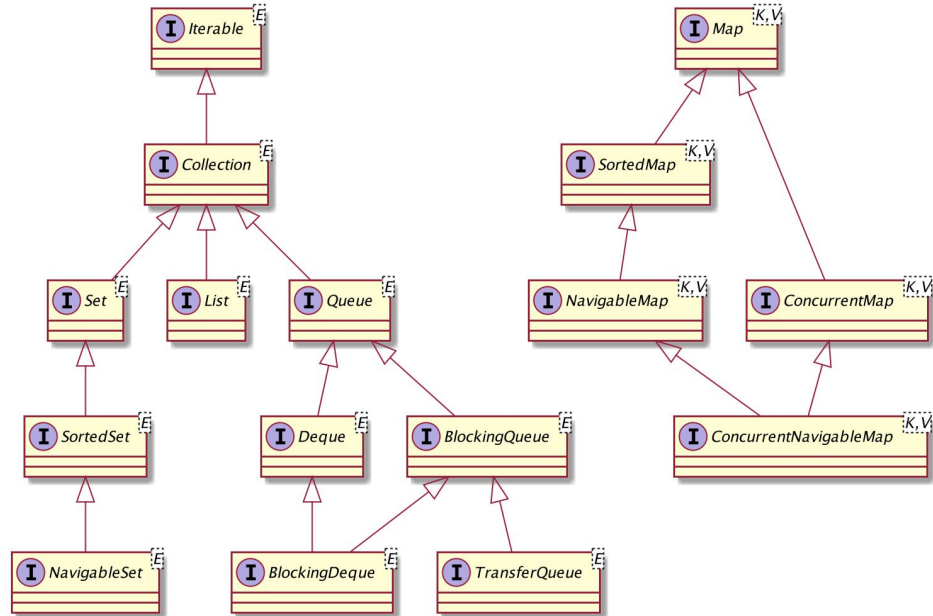
# Java Collection

Java collection telah menyediakan semuanya yang berhubungan dengan collection, sehingga kita hanya cukup tinggal menggunakannya

- Interfaces, ini adalah contract representasi dari collection. Semua collection di Java memiliki kontrak interface, sehingga jika kita mau, kita juga bisa membuat implementasinya sendiri
- Implementations, tidak perlu khawatir, kita juga bisa menggunakan implementasi yang sudah dibuat oleh Java, semua interface collection sudah ada implementasi class nya di Java collection
- Algorithms, Java juga sudah menyediakan algoritma-algoritma yang umum digunakan di collection, seperti pencarian dan pengurutan data di collection



# Java Collection Interface



---

# Iterable & Iterator Interface



# Iterable Interface

- Iterable adalah parent untuk semua collection di Java, kecuali Map
- Iterable sendiri sangat sederhana, hanya digunakan agar mendukung for-each loop
- Karena semua collection pasti implement iterable, secara otomatis maka semua collection di Java mendukung perulangan for-each, jadi bukan cuma Array



## Kode : Iterable

```
7  
8     Iterable<String> names = List.of("Eko", "Kurniawan", "Khannedy");  
9
```

```
10     for (var name : names) {  
11         System.out.println(name);  
12     }  
13
```

```
14 }  
15 }  
16 |
```



# Iterator Interface

- Tidak ada magic di Java, sebenarnya for-each di Iterable bisa terjadi karena ada method iterator() yang mengembalikan object Iterator
- Iterator adalah interface yang mendefinisikan cara kita melakukan mengakses element di collection secara sequential
- For-each sendiri muncul sejak Java 5, sebelum Java 5 untuk melakukan iterasi collection, biasanya dilakukan manual menggunakan Iterator object



## Kode : Iterator

```
8
9     Iterable<String> names = List.of("Eko", "Kurniawan", "Khannedy");
10    Iterator<String> iterator = names.iterator();
11
12    while (iterator.hasNext()) {
13        String name = iterator.next();
14        System.out.println(name);
15    }
16
17 }
18 }
19 |
```

---

# Collection Interface



# Collection Interface

- Selain Iterable interface, parent class semua collection di Java adalah Collection
- Kalo Iterable interface digunakan sebagai kontrak untuk meng-iterasi data secara sequential
- Collection merupakan kontrak untuk memanipulasi data collection, seperti menambah, menghapus dan mengecek isi data collection
- Tidak ada direct implementation untuk Collection, karena collection nanti akan dibagi lagi menjadi List, Set dan Queue



# Method di Collection

```
▼ Collection
  ▶ Iterable
  ▶ Object
    (m) size(): int
    (m) isEmpty(): boolean
    (m) contains(Object): boolean
    (m) toArray(): Object[]
    (m) toArray(T[]): T[]
    (m) toArray(IntFunction<T[]>): T[]
    (m) add(E): boolean
    (m) remove(Object): boolean
    (m) containsAll(Collection<?>): boolean
    (m) addAll(Collection<? extends E>): boolean
    (m) removeAll(Collection<?>): boolean
    (m) removeAll(Predicate<? super E>): boolean
    (m) retainAll(Collection<?>): boolean
    (m) clear(): void
    (m) stream(): Stream<E>
    (m) parallelStream(): Stream<E>
```



## Kode : Menambah Data ke Collection

```
9
10 Collection<String> names = new ArrayList<>();
11 names.add("Eko");
12 names.add("Kurniawan");
13 names.addAll(Arrays.asList("Kurniawan", "Programmer", "Zaman", "Now"));
14
15 for (var name : names) {
16     System.out.println(name);
17 }
18
19 }
20 }
```

## Kode : Menghapus Data di Collection

```
9  
10 Collection<String> names = new ArrayList<>();  
11 names.add("Eko");  
12 names.add("Kurniawan");  
13 names.addAll(Arrays.asList("Kurniawan", "Programmer", "Zaman", "Now"));  
14  
15 names.remove("Eko");  
16 names.removeAll(Arrays.asList("Zaman", "Programmer"));  
17  
18 for (var name : names) {  
19     System.out.println(name);  
20 }  
21
```



## Kode : Mengecek Data di Collection

```
11 names.add("Eko");
12 names.add("Kurniawan");
13 names.addAll(Arrays.asList("Kurniawan", "Programmer", "Zaman", "Now"));
14
15 System.out.println(names.contains("Khannedy"));
16 System.out.println(names.containsAll(Arrays.asList("Eko", "Now")));
17
18 names.remove("Eko");
19 names.removeAll(Arrays.asList("Zaman", "Programmer"));
20
21 System.out.println(names.contains("Eko"));
22 System.out.println(names.containsAll(Arrays.asList("Eko", "Now")));
```

---

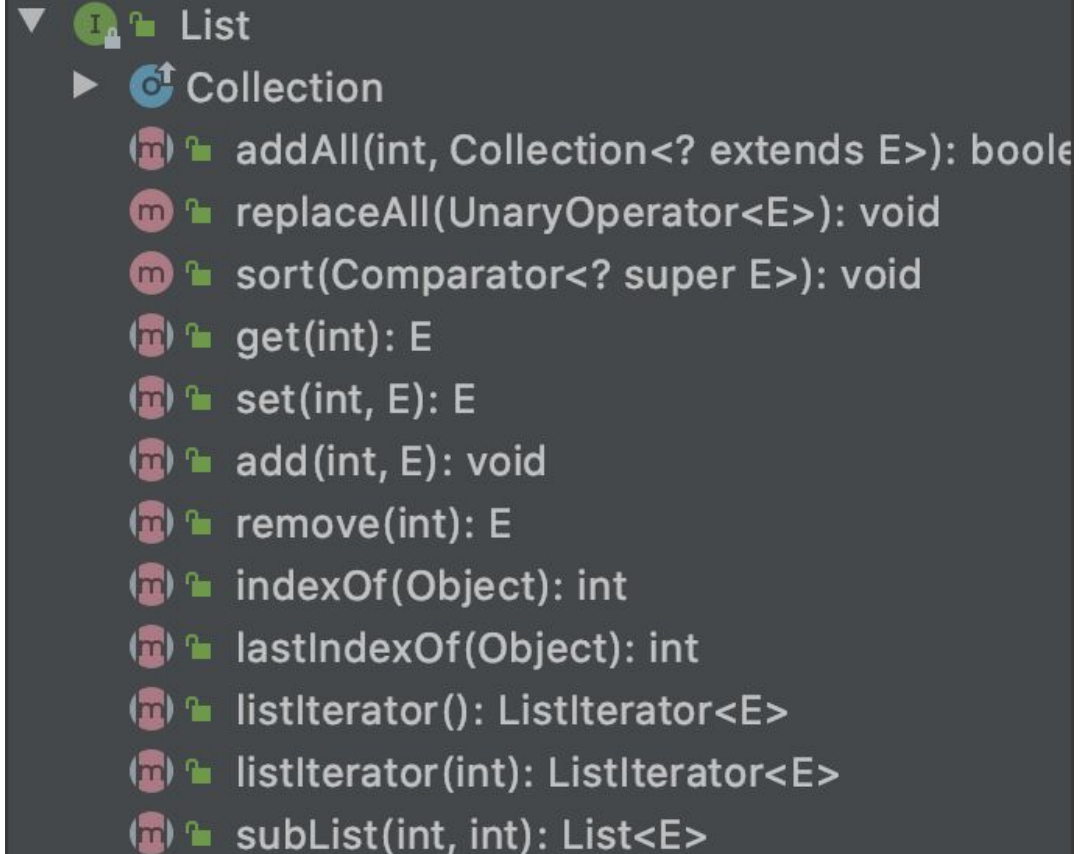
# List Interface



# List Interface

- List adalah struktur data collection yang memiliki sifat sebagai berikut
  - Elemen di list bisa duplikat, artinya bisa memasukkan data yang sama
  - Data list berurut sesuai dengan posisi kita memasukkan data
  - List memiliki index, sehingga kita bisa menggunakan nomor index untuk mendapatkan element di list
- Di Java ada beberapa implementasi List, dan kita bisa memilih sesuai dengan kebutuhan kita

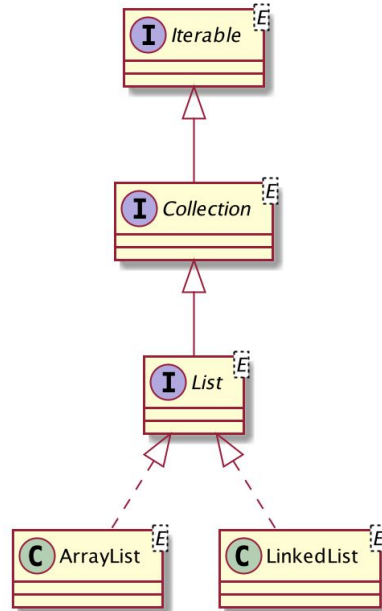
## Method di List



The screenshot shows a code editor with the following structure:

- ▼ List
  - ▶ Collection
    - (m) addAll(int, Collection<? extends E>): boolean
    - (m) replaceAll(UnaryOperator<E>): void
    - (m) sort(Comparator<? super E>): void
    - (m) get(int): E
    - (m) set(int, E): E
    - (m) add(int, E): void
    - (m) remove(int): E
    - (m) indexOf(Object): int
    - (m) lastIndexOf(Object): int
    - (m) listIterator(): ListIterator<E>
    - (m) listIterator(int): ListIterator<E>
    - (m) subList(int, int): List<E>

# Implementasi List





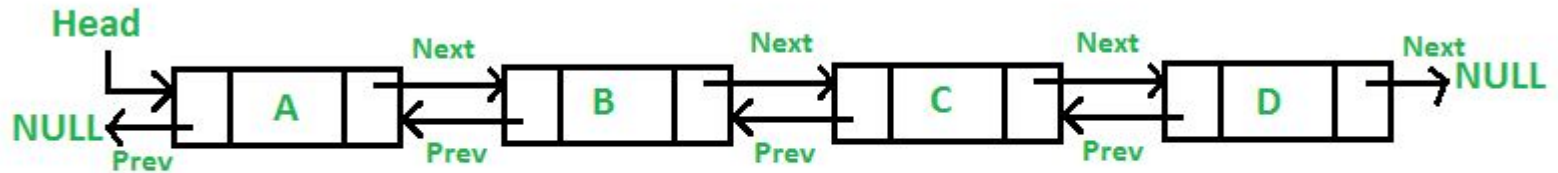


# ArrayList Class

- ArrayList adalah implementasi dari List menggunakan array
- Default kapasitas array di ArrayList adalah 10
- Namun ketika kita memasukkan data dan array sudah penuh, maka secara otomatis ArrayList akan membuat array baru dengan kapasitas baru dengan ukuran kapasitas lama + data baru

# LinkedList Class

- LinkedList adalah implementasi List dengan struktur data Double Linked List
- Bagi yang sudah belajar tentang struktur data di sekolah / kampus pasti tau apa itu double linked list





## ArrayList vs LinkedList

Operasi	ArrayList	LinkedList
add	Cepat jika kapasitas Array masih cukup, lambat jika sudah penuh	Cepat karena hanya menambah node di akhir
get	Cepat karena tinggal gunakan index array	Lambat karena harus di cek dari node awal sampai ketemu index nya
set	Cepat karena tinggal gunakan index array	Lambat karena harus di cek dari node awal sampai ketemu
remove	Lambat karena harus menggeser data di belakang yang dihapus	Cepat karena tinggal ubah prev dan next di node sebelah yang dihapus



## Kode : List

```
9
10 List<String> names = new ArrayList<>();
11 // List<String> names = new LinkedList<>();
12
13 names.add("Eko");
14 names.add("Kurniawan");
15 names.add("Khannedy");
16
17 names.set(0, "Programmer");
18
19 System.out.println(names.get(0));
20 System.out.println(names.get(1));
21
```

---

# Immutable List



# Immutable List

- Secara default, List di Java baik itu ArrayList ataupun LinkedList datanya bersifat mutable (Bisa diubah)
- Di Java mendukung pembuatan Immutable List, artinya datanya tidak bisa diubah lagi.
- Sekali List dibuat, datanya tidak bisa diubah lagi, alias final.
- Tapi ingat, data list nya yang tidak bisa diubah, bukan reference object element nya. Kalo misal kiat membuat Immutable List berisikan data Person, field data Person tetap bisa diubah, tapi isi elemen dari Immutable List tidak bisa diubah lagi
- Ini cocok ketika kasus-kasus misal, tidak sembarangan code yang boleh mengubah element di List



## Kode : Problem Mutable List (1)

```
6 public class Person {
7
8     private String name;
9
10    private List<String> hobbies;
11
12    public void addHobby(String hobby) {
13        hobbies.add(hobby);
14    }
15
16    public List<String> getHobbies() {
17        return hobbies;
18    }
19 }
```

## Kode : Problem Mutable List (2)

```
9   Person person = new Person( name: "Eko");
10  person.addHobby( hobby: "Coding");
11  person.addHobby( hobby: "Gaming");
12  doSomethingWithHobbies(person.getHobbies());
13  for (String hobby : person.getHobbies()) {
14      System.out.println(hobby);
15  }
16 }
17
18 @ public static void doSomethingWithHobbies(List<String> hobbies) {
19     hobbies.add("Eaaa");
20 }
```





## Kode : Konversi ke Immutable List

```
10
11     private List<String> hobbies;
12
13     public void addHobby(String hobby) {
14         hobbies.add(hobby);
15     }
16
17     public List<String> getHobbies() {
18         return Collections.unmodifiableList(hobbies);
19     }
20
21     public Person(String name) {
```



## Membuat Immutable List

Method	Keterangan
<code>Collections.emptyList()</code>	Membuat immutable list kosong
<code>Collections.singletonList(e)</code>	Membuat immutable list berisi 1 element
<code>Collections.unmodifiableList(list)</code>	Mengubah mutable list menjadi immutable
<code>List.of(e...)</code>	Membuat immutable list dari element - element



## Kode : Immutable List

```
9
10 List<String> names = List.of("Eko", "Kurniawan", "Khannedy");
11
12 names.set(0, "Ups"); // error
13 names.remove(0: "Eko"); // error
14 names.add("Programmer"); // error
15
16 }
17 }
18
```

---

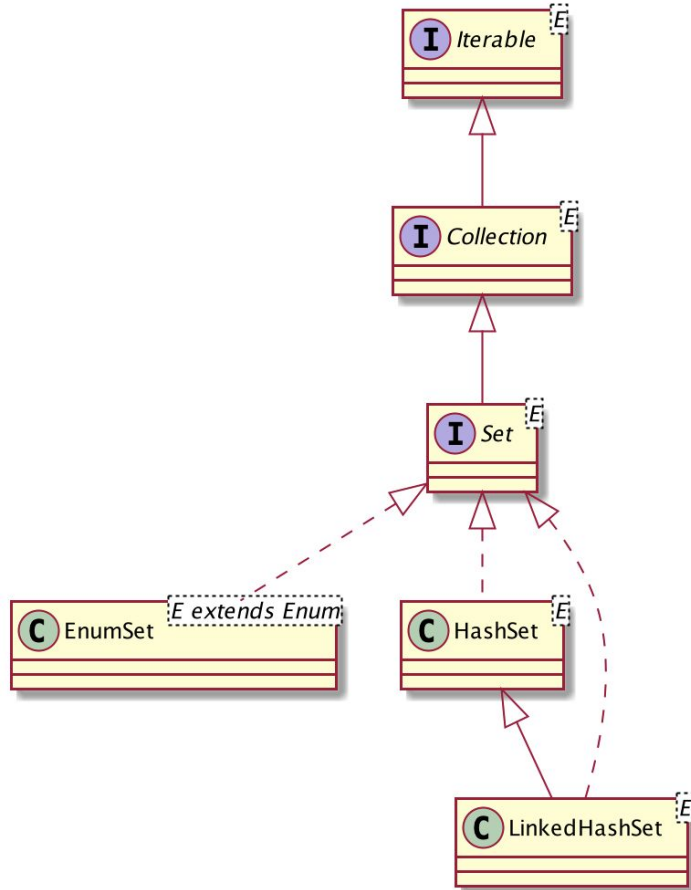
# Set Interface



# Set Interface

- Set adalah salah satu collection yang berisikan elemen-elemen yang unik, atau tidak boleh duplicate
- Set tidak memiliki index seperti di List, oleh karena itu tidak ada jaminan data yang ada di Set itu akan terurut sesuai dengan waktu kita memasukkan data ke Set
- Set tidak memiliki method baru, jadi hanya menggunakan method yang ada di interface parent nya, yaitu Collection dan Iterable
- Karena tidak memiliki index, untuk mengambil data di Set juga kita harus melakukan iterasi satu per satu

# Implementasi Set





# HashSet vs LinkedHashSet

- Di belakang HashSet dan LinkedHashSet sebenarnya sama-sama ada hash table, dimana data disimpan dalam sebuah hash table dengan mengkalkulasi hashCode() function
- Yang membedakan adalah, HashSet tidak menjamin data terurut sesuai dengan waktu kita menambahkan data, sedangkan LinkedHashSet menjamin data terurut sesuai dengan waktu kita menambahkan data.
- Urutan data di LinkedHashSet bisa dijaga karena di belakang nya menggunakan double linked list



## Kode : HashSet

```
8 Set<String> names = new HashSet<>();
9 names.add("Eko");
10 names.add("Kurniawan");
11 names.add("Khannedy");
12
13 for (var name : names) {
14     System.out.println(name);
15 }
16 }
17 }
18 }
```





## Kode : LinkedHashSet

```
8 Set<String> names = new LinkedHashSet<>();
9 names.add("Eko");
10 names.add("Kurniawan");
11 names.add("Khannedy");
12
13 for (var name : names) {
14     System.out.println(name);
15 }
16
17 }
```



# EnumSet

- EnumSet adalah Set yang elemen datanya harus Enum
- Karena data Enum itu unik, sehingga sangat cocok menggunakan Set dibandingkan List



## Kode : EnumSet

```
6
7  public static enum Gender {
8      MALE, FEMALE, NOT_MENTION
9  }
10
11  public static void main(String[] args) {
12
13      EnumSet<Gender> genders = EnumSet.allOf(Gender.class);
14      for (var gender : genders) {
15          System.out.println(gender);
16      }
17  }
```

---

# Immutable Set



# Immutable Set

- Sama seperti List, Set pun memiliki tipe data Immutable
- Cara pembuatan immutable Set di Java mirip dengan pembuatan immutable List



## Membuat Immutable Set

Method	Keterangan
<code>Collections.emptySet()</code>	Membuat immutable set kosong
<code>Collections.singleton(e)</code>	Membuat immutable set berisi 1 element
<code>Collections.unmodifiableSet(set)</code>	Mengubah mutable set menjadi immutable
<code>Set.of(e...)</code>	Membuat immutable set dari element - element



## Kode : Immutable Set

```
8      Set<String> names = Set.of("Eko", "Kurniawan", "Khannedy");
9
10     names.add("Eko"); // error
11     names.remove(o: "Eko"); // error
12 }
13 }
14
```

---

# SortedSet Interface

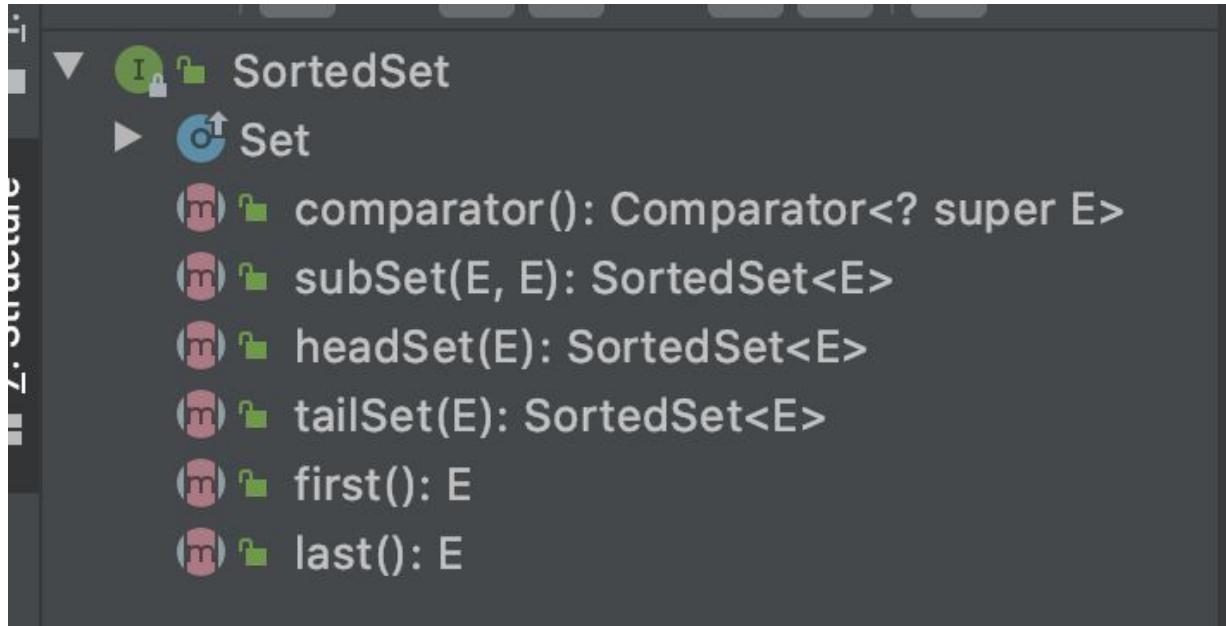




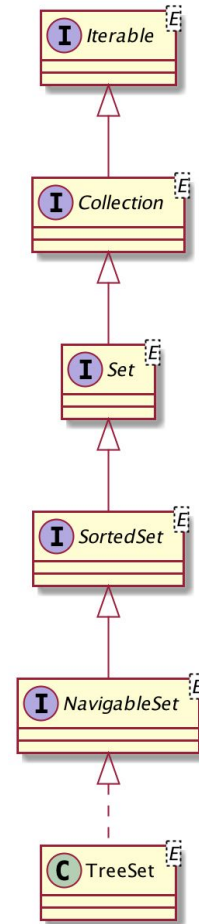
# SortedSet Interface

- SortedSet adalah turunan dari Set, namun di SortedSet elemen-elemen yang dimasukkan kedalam SortedSet akan otomatis diurutkan
- Jika element adalah turunan dari interface Comparable, maka secara otomatis akan diurutkan menggunakan comparable tersebut
- Jika element bukan turunan dari interface Comparable, maka kita bisa menggunakan Comparator untuk memberi tahu si SortedSet bagaimana cara mengurutkan elemen-elemen nya

## Method di SortedSet Interface



# Implementasi SortedSet





## Kode : Person Comparator

```
4
5  public class PersonComparator implements Comparator<Person> {
6
7      @Override
8  @ public int compare(Person o1, Person o2) {
9      return o1.getName().compareTo(o2.getName());
10 }
11 }
12
```



## Kode : SortedSet

```
SortedSet<Person> people = new TreeSet<>(new PersonComparator());
people.add(new Person( name: "Eko"));
people.add(new Person( name: "Budi"));
people.add(new Person( name: "Joko"));

for (var person : people) {
    System.out.println(person.getName());
}
}
```



## Membuat Immutable SortedSet

Method	Keterangan
<code>Collections.emptySortedSet()</code>	Membuat immutable sorted set kosong
<code>Collections.unmodifiableSortedSet(set)</code>	Mengubah mutable sorted set menjadi immutable

---

# NavigableSet Interface

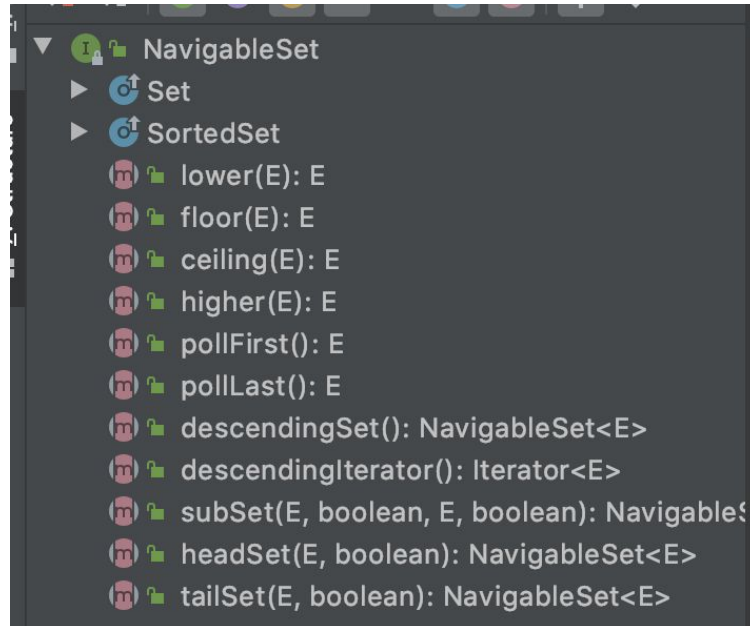


# NavigableSet Interface

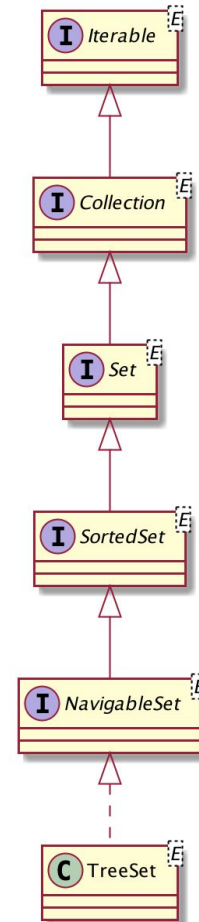
- NavigableSet adalah turunan dari SortedSet
- NavigableSet menambah method-method untuk melakukan navigasi pencarian element, seperti mencari elemen yang lebih besar dari, kurang dari, membalikkan urutan set, dan lain-lain



# Method di NavigableSet



# Implementasi NavigableSet





## Kode : NavigableSet

```
9
10 NavigableSet<String> names = new TreeSet<>();
11 names.addAll(Set.of("Eko", "Kurniawan", "Khannedy", "Programmer", "Zaman", "Now"));
12
13 NavigableSet<String> namesDesc = names.descendingSet();
14 NavigableSet<String> kurniawan = names.headSet(toElement: "Kurniawan", inclusive: true);
15
16 for (var name : names) {
17     System.out.println(name);
18 }
19
20
21
```



## Membuat Immutable SortedSet

Method	Keterangan
<code>Collections.emptyNavigableSet()</code>	Membuat immutable navigable set kosong
<code>Collections.unmodifiableNavigableSet(set)</code>	Mengubah mutable navigable set menjadi immutable

---

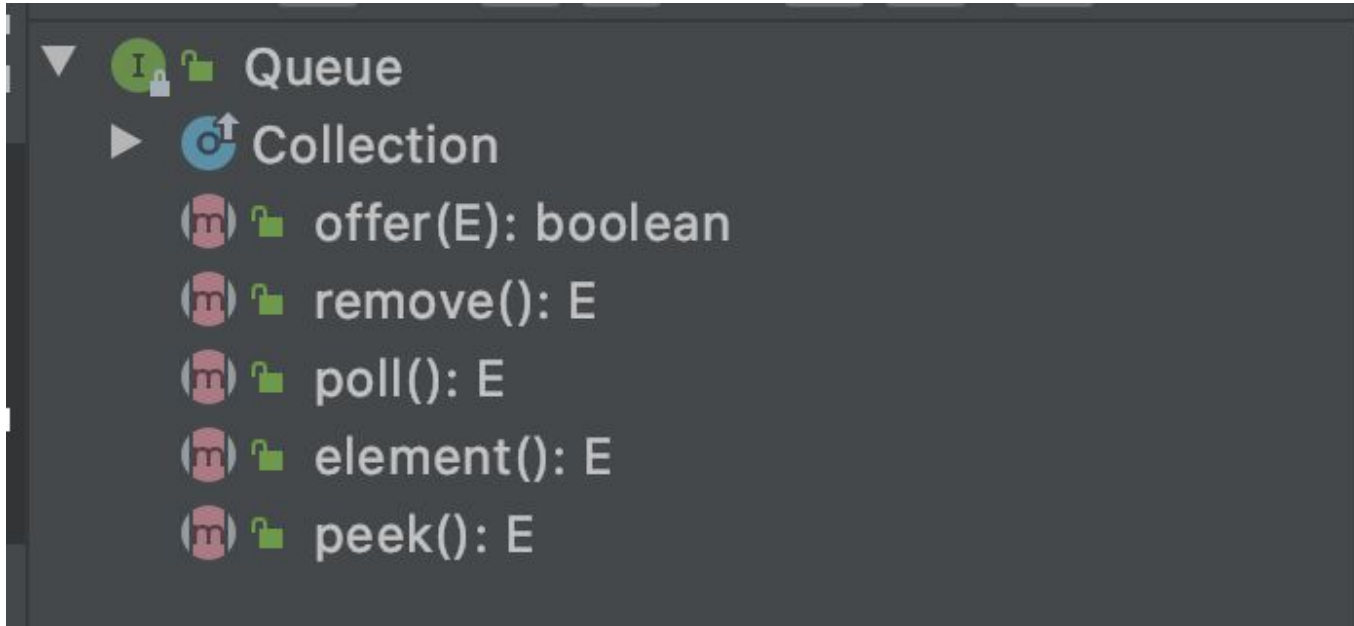
# Queue Interface



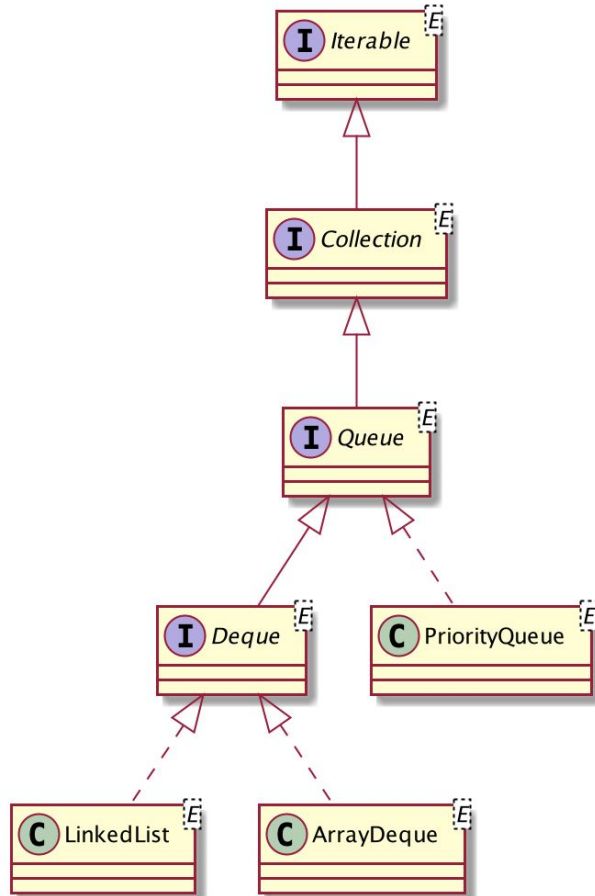
# Queue Interface

- Queue adalah implementasi dari struktur data Antrian atau FIFO (First In First Out)

## Method di Queue Interface



# Implementasi Queue







# ArrayDeque vs LinkedList vs PriorityQueue

- ArrayDeque menggunakan array sebagai implementasi queue nya
- LinkedList menggunakan double linked list sebagai implementasi queue nya
- PriorityQueue menggunakan array sebagai implementasi queue nya, namun diurutkan menggunakan Comparable atau Comparator



## Kode : Queue

```
9
10 Queue<String> queue = new ArrayDeque<>( numElements: 10);
11 queue.offer( e: "Eko");
12 queue.offer( e: "Kurniawan");
13 queue.offer( e: "Khannedy");
14
15 for (String next = queue.poll(); next != null; next = queue.poll()) {
16     System.out.println(next);
17 }
18
19 System.out.println(queue.size());
20
21
```



## Kode : PriorityQueue

```
9 Queue<String> queue = new PriorityQueue<>();
10 queue.offer( e: "Eko");
11 queue.offer( e: "Budi");
12 queue.offer( e: "Joko");
13
14 for (String next = queue.poll(); next != null; next = queue.poll()) {
15     System.out.println(next);
16 }
17
18 System.out.println(queue.size());
19
20 }
```

---

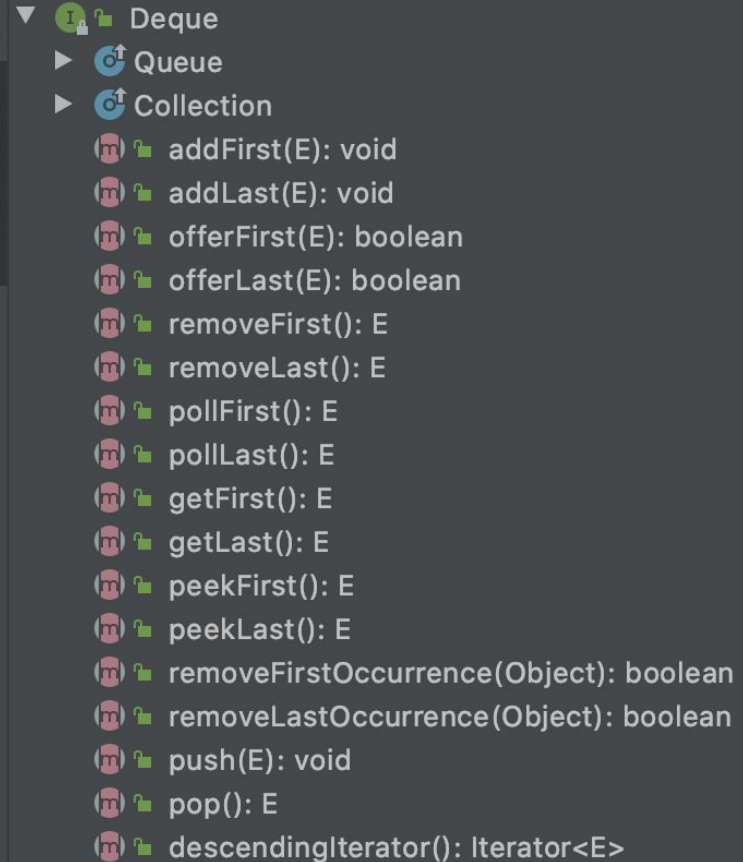
# Deque Interface



# Deque Interface

- Deque singkatan dari double ended queue, artinya queue yang bisa beroperasi dari depan atau belakang
- Jika pada queue, operasi yang didukung ada FIFO, namu pada deque, tidak hanya FIFO, naun juga mendukung LIFO (Last In First Out)
- Bisa dibilang deque adalah implementasi struktur data antrian dan stack (tumpukan)

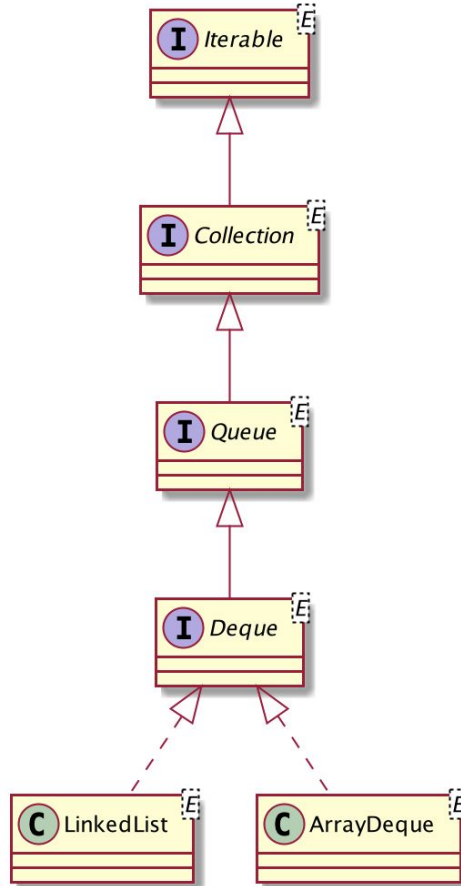
# Method di Deque



The screenshot shows a Java IDE with the following structure:

- ▼ Deque
  - ▶ Queue
    - ▶ Collection
      - addFirst(E): void
      - addLast(E): void
      - offerFirst(E): boolean
      - offerLast(E): boolean
      - removeFirst(): E
      - removeLast(): E
      - pollFirst(): E
      - pollLast(): E
      - getFirst(): E
      - getLast(): E
      - peekFirst(): E
      - peekLast(): E
      - removeFirstOccurrence(Object): boolean
      - removeLastOccurrence(Object): boolean
      - push(E): void
      - pop(): E
      - descendingIterator(): Iterator<E>

# Implementasi Deque



## Kode : Stack Menggunakan Deque

```
9      Deque<String> stack = new LinkedList<>();
10
11      stack.offerLast( e: "Eko");
12      stack.offerLast( e: "Kurniawan");
13      stack.offerLast( e: "Khannedy");
14
15      for (var item = stack.pollLast(); item != null; item = stack.pollLast()) {
16          System.out.println(item);
17      }
18
19  }
20 }
```



---

# Map Interface



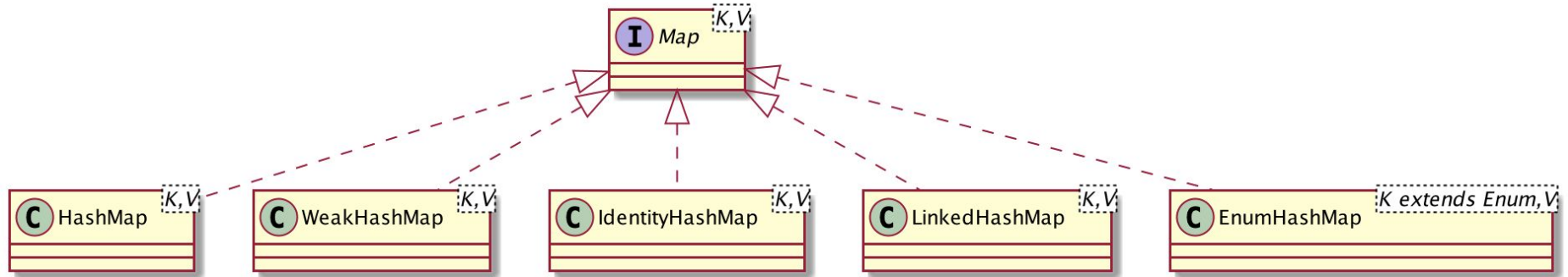
# Map Interface

- Map adalah struktur data collection yang berisikan mapping antara key dan value
- Dimana key di map itu harus unik, tidak boleh duplikat, dan satu key cuma boleh mapping ke satu key
- Map sebenarnya mirip dengan Array, cuma bedanya kalo di Array, key adalah index (integer), sedangkan di Map, key nya bebas kita tentukan sesuai keinginan kita

# Method di Map

```
▼ Map
  ▶ Entry
  ▶ Object
    (m) size(): int
    (m) isEmpty(): boolean
    (m) containsKey(Object): boolean
    (m) containsValue(Object): boolean
    (m) get(Object): V
    (m) put(K, V): V
    (m) remove(Object): V
    (m) putAll(Map<? extends K, ? extends V>): void
    (m) clear(): void
    (m) keySet(): Set<K>
    (m) values(): Collection<V>
    (m) entrySet(): Set<Entry<K, V>>
    (m) getOrDefault(Object, V): V
    (m) forEach(BiConsumer<? super K, ? super V>): void
    (m) replaceAll(BiFunction<? super K, ? super V, ? extends V>): void
    (m) putIfAbsent(K, V): V
    (m) remove(Object, Object): boolean
    (m) replace(K, V, V): boolean
    (m) replace(K, V): V
    (m) computeIfAbsent(K, Function<? super K, ? extends V>): V
    (m) computeIfPresent(K, BiFunction<? super K, ? super V, ? extends V>): V
    (m) compute(K, BiFunction<? super K, ? super V, ? extends V>): V
    (m) merge(K, V, BiFunction<? super V, ? super V, ? extends V>): V
```

# Implementasi Map





# HashMap

- HashMap adalah implementasi Map yang melakukan distribusi key menggunakan hashCode() function
- Karena HashMap sangat bergantung dengan hashCode() function, jadi pastikan kita harus membuat function hashCode seunik mungkin, karena jika terlalu banyak nilai hashCode() yang sama, maka pendistribusian key nya tidak akan optimal sehingga proses get data di Map akan semakin lambat
- Di HashMap pengecekan data duplikat dilakukan dengan menggunakan method equals nya



## Kode : HashMap

```
8
9      Map<String, String> map = new HashMap<>();
10     map.put("firstName", "Eko");
11     map.put("middleName", "Kurniawan");
12     map.put("lastName", "Khannedy");
13
14     System.out.println(map.get("firstName"));
15     System.out.println(map.get("middleName"));
16     System.out.println(map.get("lastName"));
17
18 }
19 }
```



# WeakHashMap

- WeakHashMap adalah implementasi Map mirip dengan HashMap
- Yang membedakan adalah WeakHashMap menggunakan weak key, dimana jika tidak digunakan lagi maka secara otomatis data di WeakHashMap akan dihapus
- Artinya, jika terjadi garbage collection di Java, bisa dimungkinkan data di WeakHashMap akan dihapus
- WeakHashMap cocok digunakan untuk menyimpan data cache di memory secara sementara



## Code : WeakHashMap

```
8
9  Map<Integer, Integer> map = new WeakHashMap<>();
10 for (int i = 0; i < 1000000; i++) {
11     map.put(i, i);
12 }
13
14 System.gc();
15
16 System.out.println(map.size());
17
18 }
19 }
```





# IdentityHashMap

- IdentityHashMap adalah implementasi Map sama seperti HashMap
- Yang membedakan adalah cara pengecekan kesamaan datanya, tidak menggunakan function equals, melainkan menggunakan operator == (reference equality)
- Artinya data dianggap sama, jika memang lokasi di memory tersebut sama



## Kode : IdentityHashMap

```
9      String key1 = "name.first";
10
11      String name = "name";
12      String first = "first";
13
14      String key2 = name + "." + first;
15
16      Map<String, String> map = new IdentityHashMap<>();
17      map.put(key1, "Eko Kurniawan");
18      map.put(key2, "Eko Kurniawan");
19
20      System.out.println(map.size());
```



# LinkedHashMap

- LinkedHashMap adalah implementasi Map dengan menggunakan double linked list
- Data di LinkedHashMap akan lebih terprediksi karena datanya akan disimpan berurutan dalam linked list sesuai urutan kita menyimpan data
- Namun perlu diperhatikan, proses get data di LinkedHashMap akan semakin lambat karena harus melakukan iterasi data linked list terlebih dahulu
- Gunakan LinkedHashMap jika memang kita lebih mementingkan iterasi data Map nya



## Kode : LinkedHashMap

```
9 Map<String, String> map = new LinkedHashMap<>();
10 map.put("Eko", "Eko");
11 map.put("Kurniawan", "Kurniawan");
12 map.put("Khannedy", "Khannedy");
13
14 for (var key : map.keySet()) {
15     System.out.println(key);
16 }
17
18 }
19
20 |
```



# EnumMap

- EnumMap adalah implementasi Map dimana key nya adalah enum
- Karena data enum sudah pasti unik, oleh karena itu cocok dijadikan key di Map
- Algoritma pendistribusian key dioptimalkan untuk enum, sehingga lebih optimal dibandingkan menggunakan hashCode() method

## Kode : EnumMap

```
7 public static enum Level {  
8     FREE, STANDARD, PREMIUM, VIP  
9 }  
10  
11 public static void main(String[] args) {  
12  
13     EnumMap<Level, String> map = new EnumMap<>(Level.class);  
14     map.put(Level.FREE, "Gratis");  
15     map.put(Level.PREMIUM, "Bayar");  
16  
17     System.out.println(map.get(Level.FREE));  
18     System.out.println(map.get(Level.PREMIUM));  
}
```

---

# Immutable Map



# Immutable Map

- Sama seperti List dan Set, Map pun punya tipe data Immutable





## Membuat Immutable Map

Method	Keterangan
<code>Collections.emptyMap()</code>	Membuat immutable map kosong
<code>Collections.unmodifiableMap(map)</code>	Mengubah mutable map menjadi immutable
<code>Collections.singletonMap(key, value)</code>	Membuat map dengan satu jumlah key-value
<code>Map.of(...)</code>	Membuat immutable map dari key-value

## Kode : Immutable Map

```
7  ▶ public static void main(String[] args) {  
8  
9      Map<String, String> name = Map.of(  
10         k1: "firstName", v1: "Eko",  
11         k2: "middleName", v2: "Kurninawan",  
12         k3: "lastName", v3: "Khannedy"  
13     );  
14  
15     name.put("middleName", "Ups"); // error  
16  
17 }  
18 }
```

---

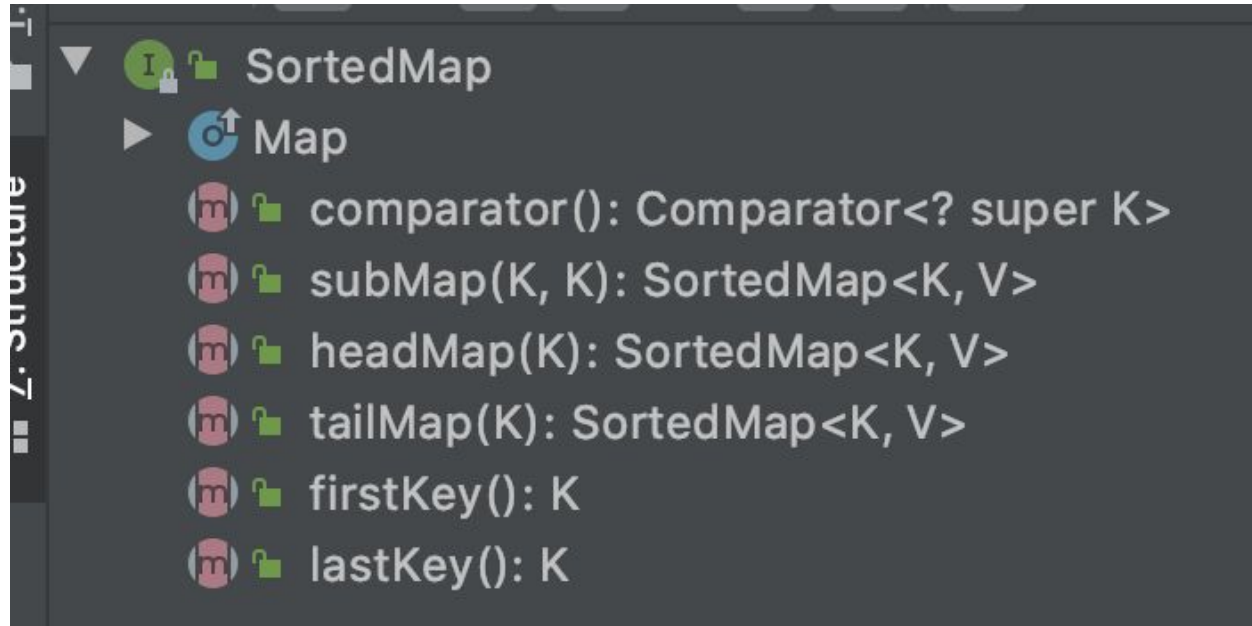
# SortedMap Interface



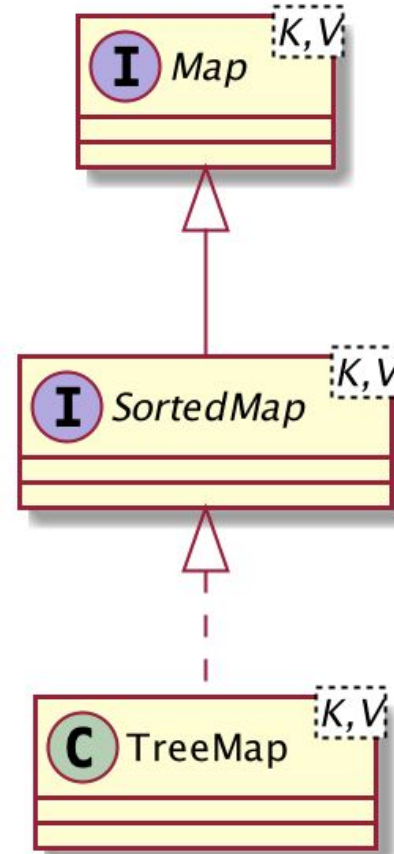
# SortedMap Interface

- SortedMap adalah implementasi Map dengan data key diurutkan sesuai dengan Comparable key atau bisa menggunakan Comparator
- SortedMap cocok untuk kasus yang posisi key pada Map harus berurut

## Method di SortedMap



# Implementasi SortedMap





## Kode : SortedMap

```
8
9 SortedMap<String, String> map = new TreeMap<>();
10 map.put("Eko", "Eko");
11 map.put("Budi", "Budi");
12 map.put("Joko", "Joko");
13
14 for (var key : map.entrySet()) {
15     System.out.println(key);
16 }
17
18 }
```

## Kode : SortedMap Menggunakan Comparator

```
12 SortedMap<Person, String> map = new TreeMap<>(new Comparator<Person>() {
13     @Override
14     public int compare(Person o1, Person o2) {
15         return o1.getName().compareTo(o2.getName());
16     }
17 });
18 map.put(new Person( name: "Eko"), "Eko");
19 map.put(new Person( name: "Budi"), "Budi");
20 map.put(new Person( name: "Joko"), "Joko");
21
22 for (var key : map.entrySet()) {
23     System.out.println(key);
```





## Membuat Immutable SortedMap

Method	Keterangan
<code>Collections.emptySortedMap()</code>	Membuat immutable sorted map kosong
<code>Collections.unmodifiableSortedMap(map)</code>	Mengubah mutable sorted map menjadi immutable

---

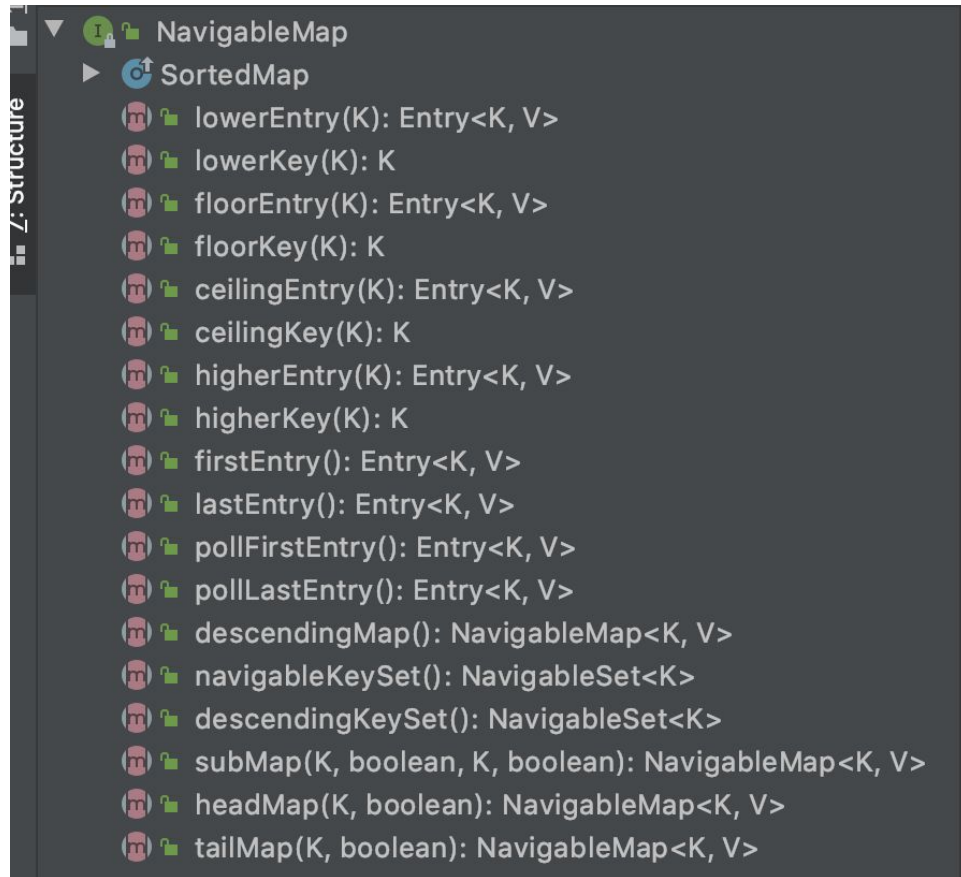
# NavigableMap Interface



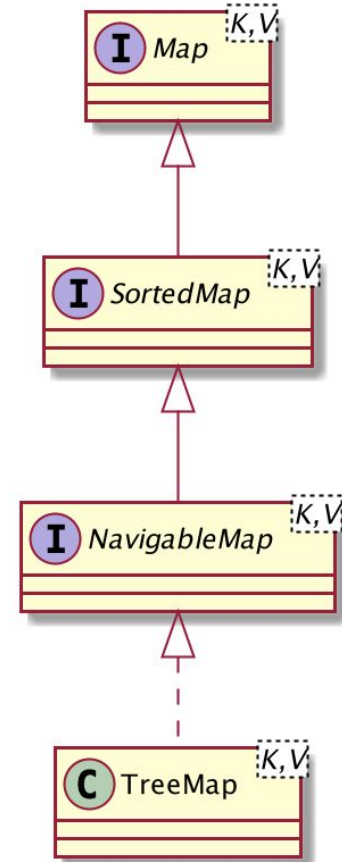
# NavigableMap Interface

- NavigableMap adalah turunan dari SortedMap
- Namun NavigableMap memiliki kemampuan navigasi berdasarkan operasi kurang dari, lebih dari dan sejenisnya
- Misal, kita ingin mengambil data yang lebih dari key x atau kurang dari key y, ini bisa dilakukan di NavigableMap

# Method di NavigableMap



# Implementasi NavigableMap





## Kode : NavigableMap

```
8
9     NavigableMap<String, String> map = new TreeMap<>();
10
11     map.put("Eko", "Eko");
12     map.put("Budi", "Budi");
13     map.put("Joko", "Joko");
14
15     System.out.println(map.lowerKey(key: "Eko"));
16     System.out.println(map.higherKey(key: "Eko"));
17 }
18 }
19
```



# Membuat NavigableMap

Method	Keterangan
<code>Collections.emptyNavigableMap()</code>	Membuat immutable navigable map kosong
<code>Collections.unmodifiableNavigableMap(map)</code>	Mengubah mutable navigable map menjadi immutable

---

# Entry Map

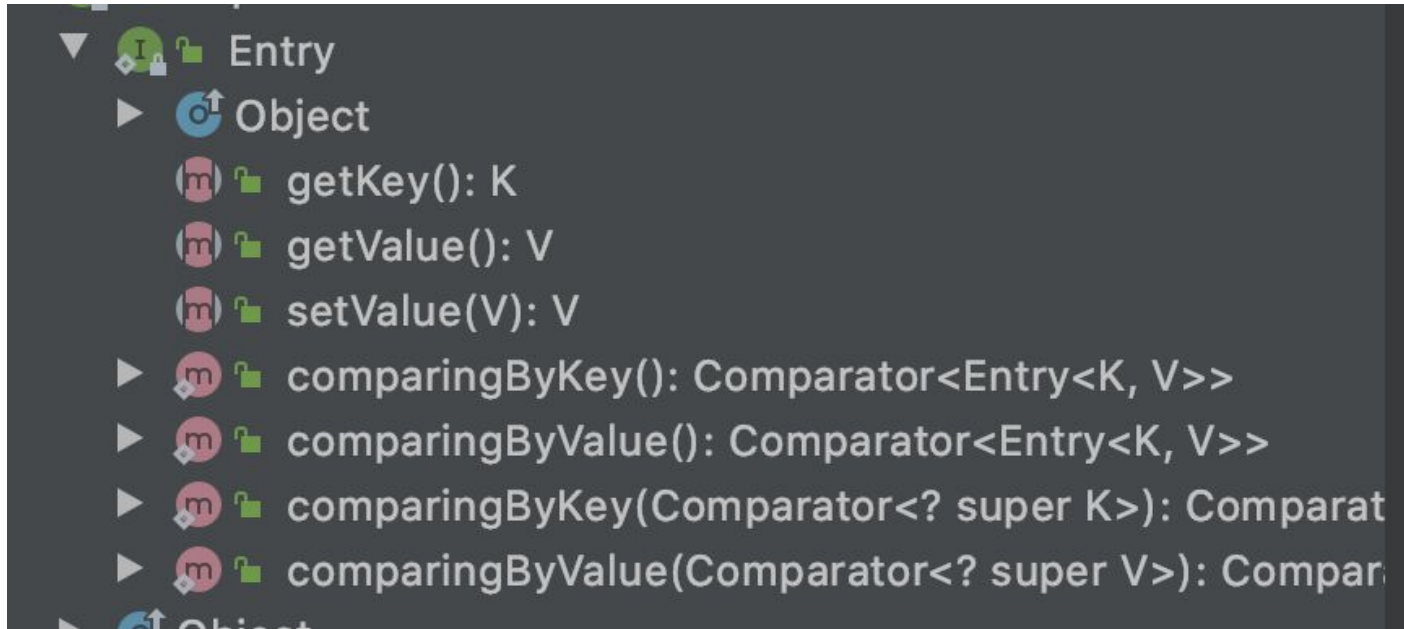




# Entry Interface

- Saat kita menyimpan data di Map, data disimpan dalam pair (key-value)
- Di Java collection, implementasi Pair di Map bernama Entry
- Entry adalah interface sederhana yang berisikan method untuk mengambil key dan value

## Method di Entry Interface





## Kode : Menggunakan Entry

```
10 Map<String, String> map = new HashMap<>();
11 map.put("Eko", "Eko");
12 map.put("Kurniawan", "Kurniawan");
13 map.put("Khanendy", "Khannedy");
14
15 Set<Map.Entry<String, String>> entries = map.entrySet();
16
17 for (var entry : entries) {
18     System.out.println("====");
19     System.out.println("Key : " + entry.getKey());
20     System.out.println("Value : " + entry.getValue());
21 }
```

---

# Legacy Collection



# Legacy Collection

- Collection sudah ada sejak Java versi 1, namun semakin kesini, Java Collection semakin berkembang
- Sebenarnya ada beberapa legacy collection (collection jadul) yang belum kita bahas, namun jarang sekali digunakan sekarang ini



# Vector Class

- Vector class adalah implementasi dari interface List
- Cara kerja Vector mirip dengan ArrayList, yang membedakan adalah semua method di Vector menggunakan kata kunci synchronized yang artinya dia thread safe
- Namun problemnya adalah, karena semua method menggunakan kata kunci synchronized, secara otomatis impact nya ke-performance yang menjadi lambat dibandingkan menggunakan ArrayList
- Lantai bagaimana jika kita ingin membuat List yang bisa digunakan di proses paralel? Di versi Java baru, sudah disediakan yang lebih canggih untuk itu, oleh karena itu penggunaan Vector sudah jarang sekali ditemui sekarang



## Kode : Vector

```
8
9  List<String> names = new Vector<>();
10 names.add("Eko");
11 names.add("Kurniawa");
12 names.add("Khanendy");
13
14 for (var name : names) {
15     System.out.println(name);
16 }
17 }
18 }
19
```



# HashTable Class

- HashTable adalah implementasi dari Map yang mirip dengan HashMap
- Sama seperti Vector, semua method di HashTable memiliki kata kunci synchronized, sehingga performanya lebih lambat dibandingkan HashMap
- Dan karena di versi Java baru sudah ada juga ConcurrentHashMap, sehingga penggunaan HashTable sudah jarang sekali ditemui





## Kode : Hashtable

```
10
11 Map<String, String> map = new Hashtable<>();
12 map.put("Eko", "Eko");
13 map.put("Kurniawan", "Kurniawan");
14 map.put("Khanendy", "Khannedy");
15
16 for (var entry : map.entrySet()) {
17     System.out.println(entry.getKey() + " : " + entry.getValue());
18 }
19 }
20 }
21 |
```



# Stack Class

- Stack adalah implementasi struktur data tumpukan LIFO (Last In First Out)
- Namun fitur yang lebih komplrit dan konsisten sudah disediakan di Deque, jadi tidak ada alasan lagi menggunakan class Stack



## Kode : Stack

```
Stack<String> names = new Stack<>();
names.push( item: "Eko");
names.push( item: "Kurniawan");
names.push( item: "Khanendy");

for (var name = names.pop(); name != null; name = names.pop()) {
    System.out.println(name);
}
}
```

---

# Sorting



# Sorting

- Sorting atau pengurutan adalah algoritma yang sudah biasa kita lakukan
- Di Java Collection juga sudah disediakan cara untuk melakukan pengurutan, jadi kita tidak perlu melakukan pengurutan secara manual
- Namun perlu diingat, yang bisa di sort hanyalah List, karena Set, Queue, Deque dan Map cara kerjanya sudah khusus, jadi pengurutan hanya bisa dilakukan di List



## Sorting di List

Method	Keterangan
<code>Collections.sort(list)</code>	Mengurutkan list dengan comparable bawaan element
<code>Collections.sort(list, comparator)</code>	Mengurutkan list dengan comparator

## Kode : Sorting List

```
11 List<String> names = new ArrayList<>();
12 names.addAll(List.of("Eko", "Budi", "Joko"));
13
14 Collections.sort(names);
15 System.out.println(names);
16
17 Collections.sort(names, new Comparator<String>() {
18     @Override
19     public int compare(String o1, String o2) {
20         return o2.compareTo(o1);
21     }
22 });
```

---

# Binary Search

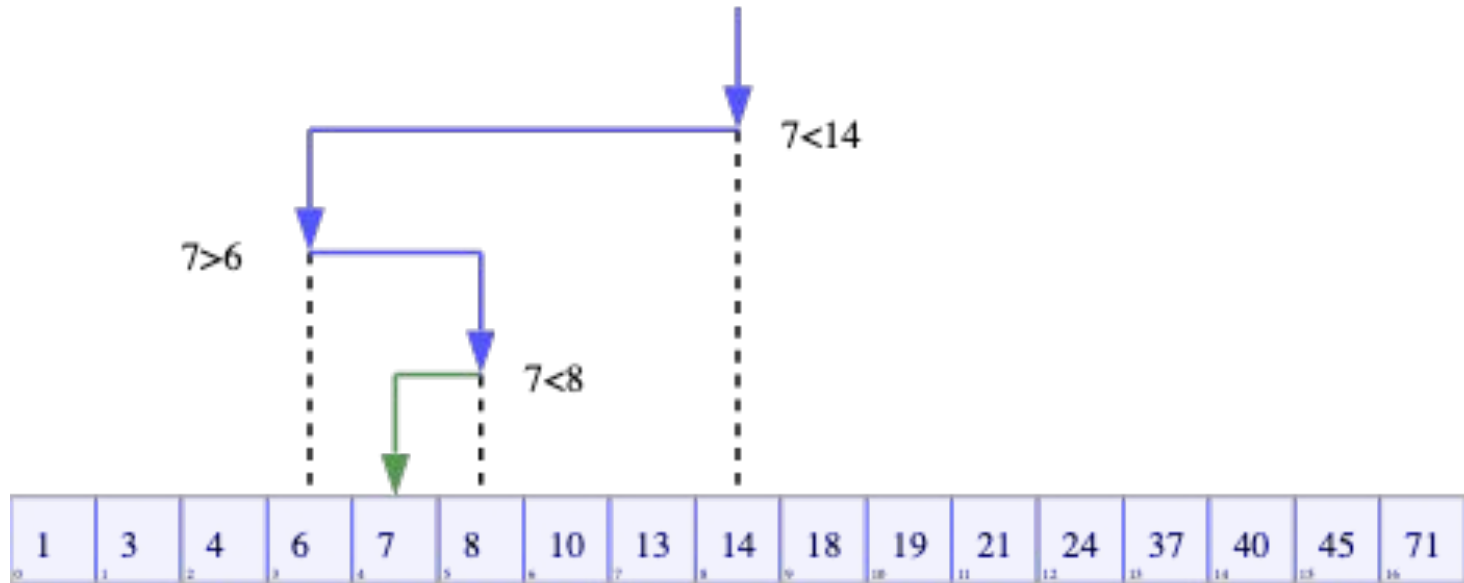




# Binary Search

- Secara default List di Java memiliki fitur search atau get data, namun implementasinya menggunakan sequential search, artinya data akan di cek satu per satu dari awal
- Salah satu algoritma pencarian yang populer adalah binary search, namun binary search hanya bisa dilakukan jika datanya telah berurutan
- Untungnya di Java Collection sudah ada implementasi binary search, sehingga kita tidak perlu membuatnya secara manual

## Binary Search Diagram





# Binary Search di List

Method	Keterangan
<code>Collections.binarySearch(list, value)</code>	Mencari menggunakan binary search
<code>Collections.binarySearch(list, value, comparator)</code>	Mencari menggunakan binary search dengan bantuan comparator



## Kode : Binary Search di List

```
List<Integer> numbers = new ArrayList<>();  
for (int i = 0; i < 1000; i++) {  
    numbers.add(i);  
}  
  
int index = Collections.binarySearch(numbers, key: 500);  
System.out.println(index);  
}  
}
```

---

# Collections Class



# Collections Class

- Collections adalah class yang berisikan utility static method untuk membantu kita menggunakan Java Collection
- Di materi-materi sebelumnya kita sudah bahas beberapa, seperti membuat immutable collection misalnya
- Namun sebenarnya masih ada banyak static method yang bisa kita gunakan di class Collections
- <https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Collections.html>



## Static Method di Collections Class (1)

Method	Keterangan
<code>void copy(listTo, listFrom)</code>	Copy semua data dari listFrom ke listTo
<code>int frequency(collection, object)</code>	Mengambil berapa banyak element yang sama dengan object
<code>max(collection)</code> dan <code>max(collection, comparable)</code>	Mengambil element paling tinggi di list
<code>min(collection)</code> dan <code>min(collection, comparable)</code>	Mengambil element paling kecil di list



## Static Method di Collections Class (2)

Method	Keterangan
void reverse(list)	Membalikkan seluruh element di list
void shuffle(list)	Mengacak posisi element di list
void swap(list, from, to)	Menukar posisi from ke to di list
... dan masih banyak	





## Kode : Collections Class

```
1 List<String> names = new ArrayList<>();
2 names.addAll(List.of("Eko", "Kurniawan", "Khannedy", "Programmer", "Zaman", "Now"));
3 System.out.println(names);
4
5 Collections.reverse(names);
6 System.out.println(names);
7
8 Collections.shuffle(names);
9 System.out.println(names);
10
11 }
```

---

# Abstract Collection



# Abstract Collection

- Struktur data collection di Java selalu berkembang, namun biasanya algoritma dasarnya selamu sama antar jenis collection
- Hampir semua interface collection di Java sudah disediakan abstract class nya sebagai dasar algoritma nya, dan hampir semua implementasi class konkrit nya selalu extends abstract class collection
- Hal ini mempermudah kita, jika ingin membuat collection manual, kita tidak perlu membuat dari awal, kita bisa menggunakan abstract class yang sudah disediakan



# Abstract Class

Abstract Class	Untuk
AbstractCollection	Collection
AbstractList	List
AbstractMap	Map
AbstractQueue	Queue
AbstractSet	Set

## Kode : Membuat Single Queue

```
public class SingleQueue<E> extends AbstractQueue<E> {  
  
    private E data;  
  
    @Override  
    public Iterator<E> iterator() {  
        return Collections.singleton(data).iterator();  
    }  
  
    @Override  
    public int size() {  
        return data == null ? 0 : 1;  
    }  
}
```



## Kode : Menggunakan Single Queue

```
Queue<String> queue = new SingleQueue<>();  
queue.offer( e: "Eko");  
queue.offer( e: "Kurniawan");  
queue.offer( e: "Khannedy");  
  
System.out.println(queue.size());  
System.out.println(queue.peek());  
System.out.println(queue.poll());  
System.out.println(queue.poll());  
System.out.println(queue.size());  
}  
}
```

---

# Default Method



# Default Method

- Di Java 8 ada fitur bernama Default Method, dimana kita bisa menambahkan konkrit method di interface
- Fitur ini banyak sekali digunakan di Java Collection, karena kita tahu semua collection di Java memiliki kontrak interface, sehingga dengan mudah di Java bisa meng-improve kemampuan semua collection hanya dengan menambahkan default method di interface collection nya





## Default Method di Collection

Default Method	Keterangan
<code>Iterable.forEach(consumer)</code>	Melakukan iterasi seluruh data collection
<code>List.removeIf(predicate)</code>	Menghapus data di collection menggunakan predicate
<code>List.replaceAll(operator)</code>	Mengubah seluruh data di collection

## Kode : Default Method Collection

```
5 numbers.replaceAll(new UnaryOperator<Integer>() {  
6     @Override  
7     public Integer apply(Integer integer) {  
8         return integer * 10;  
9     }  
10 });  
11 numbers.forEach(new Consumer<Integer>() {  
12     @Override  
13     public void accept(Integer integer) {  
14         System.out.println(integer);  
15     }  
16 });
```



## Default Method di Map (1)

Default Method	Keterangan
<code>getOrDefault(key, defaultValue)</code>	Mengambil data berdasarkan key, jika tidak ada, return defaultValue
<code>forEach(consumer)</code>	Melakukan iterasi seluruh data key-value
<code>replaceAll(function)</code>	Mengubah seluruh data value
<code>putIfAbsent(key, value)</code>	Simpan data ke map jika belum ada
<code>remove(key, value)</code>	Hapus jika key-value nya sama



## Default Method di Map (2)

Default Method	Keterangan
<code>replace(key, oldValue, newValue)</code>	Ubah key jika value sekarang sama dengan oldValue
<code>computeIfAbsent(key, function)</code>	Ubah key dengan value hasil function jika belum ada
<code>computeIfPresent(key, function)</code>	Ubah key dengan value hasil function jika sudah ada
... dan masih banyak	

## Kode : Default Method Map

```
6 map.replaceAll(new BiFunction<String, Integer, Integer>() {  
7     @Override  
8     public Integer apply(String s, Integer integer) {  
9         return integer * 10;  
10    }  
11 });  
12 map.forEach(new BiConsumer<String, Integer>() {  
13     @Override  
14     public void accept(String s, Integer integer) {  
15         System.out.println(s + " : " + integer);  
16     }  
17 });
```

---

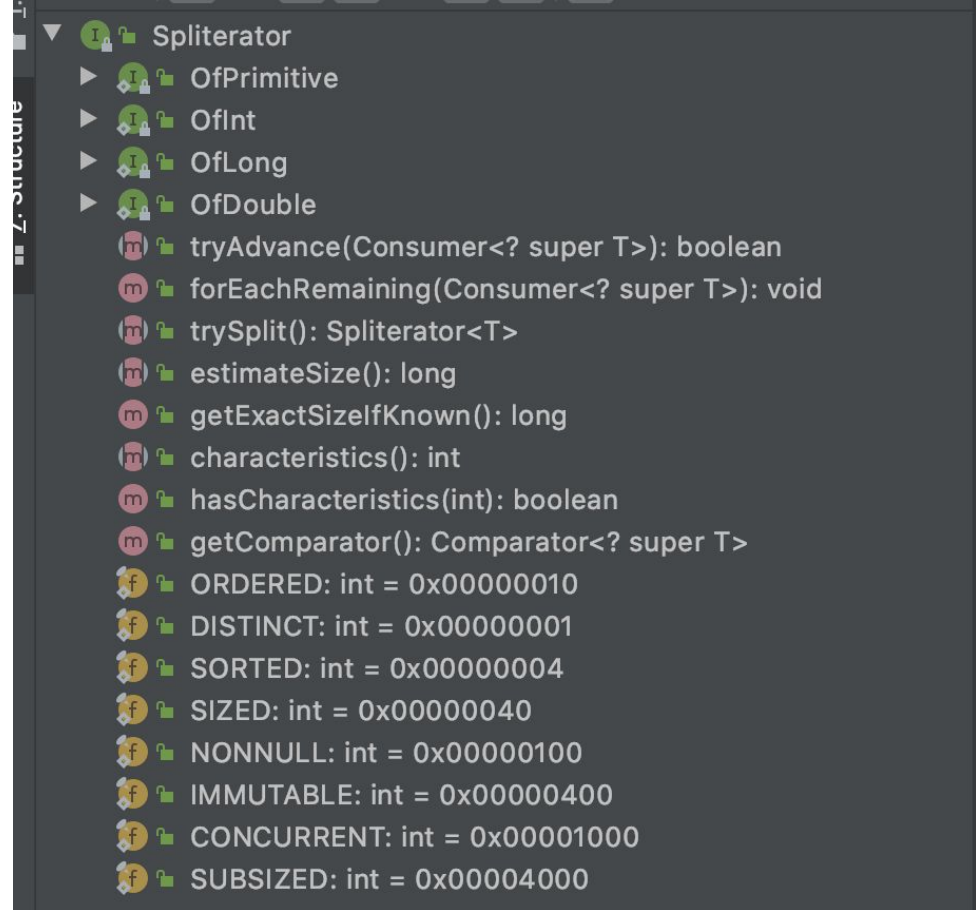
# Splititerator Interface



# Spliterator Interface

- Spliterator adalah interface yang bisa digunakan untuk melakukan partisi data collection
- Biasanya ini digunakan ketika kita akan memproses collection dalam jumlah besar, lalu agar lebih cepat di split menjadi beberapa dan diproses secara paralel agar lebih cepat
- Penggunaan Spliterator biasanya erat kaitannya dengan Java Thread atau Java Concurrency, namun di materi ini kita tidak akan membahas tentang itu, lebih fokus ke Spliterator

# Method di Spliterator







## Kode : Spliterator

```
List<String> names = List.of("Eko", "Kurniawan", "Khannedy", "Programmer", "Zaman", "Now");
Spliterator<String> spliterator1 = names.spliterator();
Spliterator<String> spliterator2 = spliterator1.trySplit();

System.out.println(spliterator1.estimateSize());
System.out.println(spliterator2.estimateSize());

spliterator1.forEachRemaining(new Consumer<String>() {
    @Override
    public void accept(String s) {
        System.out.println(s);
    }
})
```

---

# Konversi ke Array



# Collection Interface

- Interface Collection memiliki method `toArray()` untuk melakukan konversi collection ke Array
- Ini sangat cocok jika kita ingin mengubah collection ke Array, misal saja karena mau memanggil method yang memang parameternya tipenya array, bukan collection



## toArray Method di Collection

Method	Keterangan
Object[] toArray()	Mengubah collection menjadi array
T[] toArray(T[])	Mengubah collection menjadi array T



## Kode : Konversi ke Array

```
8
9      List<String> names = List.of("Eko", "Kurniawan", "Khannedy");
10     Object[] objects = names.toArray();
11     String[] strings = names.toArray(new String[]{});
12
13     System.out.println(Arrays.toString(objects));
14     System.out.println(Arrays.toString(strings));
15 }
16 }
17
```

---

# Materi Selanjutnya



# Materi Selanjutnya

- Apache Maven
- Java Lambda Expression
- Java Unit Testing
- Java Stream