Collections

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

INHOUD

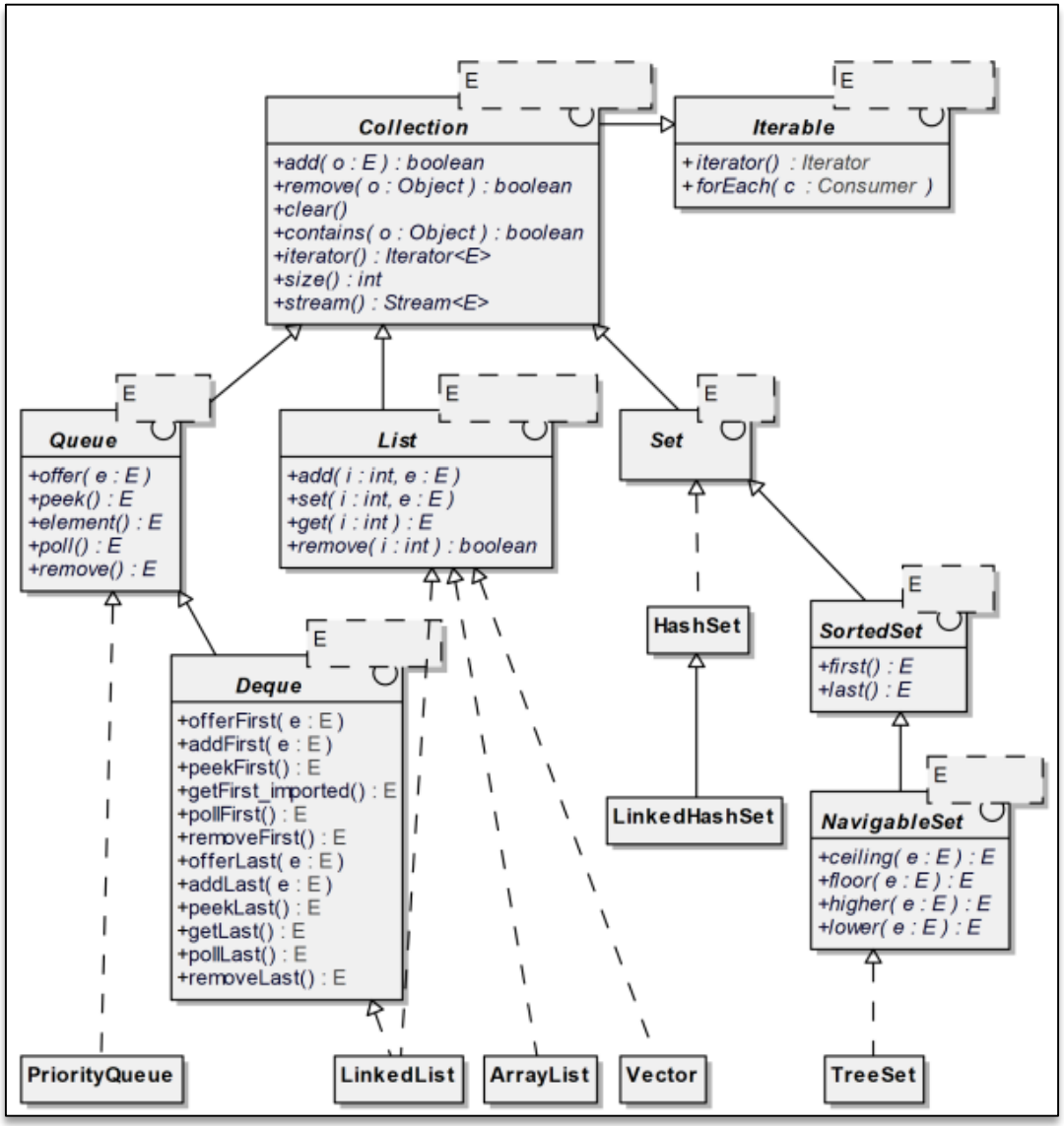
- Collection interface
- List
- Set
- Queue
- Map
- Verzamelingen sorteren
- Oefeningen

Collection interface

- Verzameling van objecten (elementen)
- Basis methoden gedefinieerd
  - Meer specifiek gedrag => afgeleide interface
- Generiek type

Methode	***Omschrijving***
`boolean add(E e)`	Voegt een element toe aan de verzameling.
`boolean remove(Object o)`	Verwijdert een element uit de verzameling.
`void clear()`	Verwijdert alle elementen uit de verzameling.
`boolean contains(Object o)`	Gaat na of een bepaald element aanwezig is in de verzameling.
`Iterator<E> iterator()`	Geeft een *iterator* om de elementen te overlopen.
`int size()`	Geeft het aantal elementen in de verzameling.
`forEach()`	Overloopt de verschillende elementen en past er een functionele methode op toe.
`stream()`	Geeft een *stream* met alle elementen van deze verzameling.

Collection framework



Collection framework

Collections zijn:

- **Geordend** als elk element een vaste plaats heeft in verzameling

- **Gesorteerd** als de elementen gerangschikt zijn volgens een bepaald algoritme
    - Vaak via compareTo() methode, zie verder

List interface

- Interface, afgeleid van Collection
  - Erft methoden over

- Geordend, elk element heeft positie met index
- Methoden nodig hiervoor zijn extra in deze interface
  - Bv. add(index, element), get(index), …

- Implementatie van List: **ArrayList**
  - Achterliggend: array met dynamische grootte
  - Ideaal voor toegang op basis van indexwaarde
  - Minder geschikt voor toevoegen en verwijderen van elementen

ArrayList

```
List<String> list = new ArrayList<>();
list.add("First");
list.add("Second");
list.add("Third");
list.add("Fourth");
list.add("Fifth");

String s = list.get(3);

for(String el : list) {
    System.out.println(el);
}

int size = list.size();
list.clear();
```

Set interface

- Implementatie: (o.a.) ***HashSet***
    - Duplicaten bepaald op basis van *equals()* en *hashCode()*

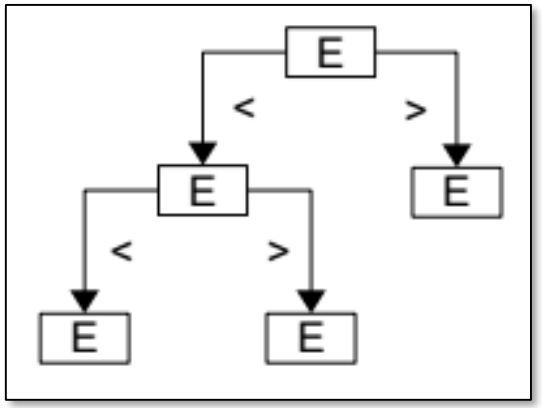
- Ongeordend en ongesorteerd
- Zeer efficiënt bij het opzoeken, toevoegen en verwijderen van een willekeurig element
- Opletten met objecten zonder implementatie van equals en hashCode

SortedSet interface

- Unieke elementen
    - Duplicaten bepaald op basis van *compareTo*
- Consistentie tussen equals en Comparator is belangrijk

SortedSet interface

- Implementatie: (o.a.) ***TreeSet***
- Geordend en gesorteerd
- Automatische sortering van elementen
    - Onmiddellijk bij toevoegen of verwijderen (*live*)
    - D.m.v. boomstructuur (=> *TreeSet*)
    - Niet super performant



TreeSet

```
NavigableSet<Integer> numbers = new TreeSet<>();
numbers.add(4);
numbers.add(8);
numbers.add(3);
for(int n: numbers) {
    System.out.println(n);       // 3 4 8
}

numbers.add(5);
for(int n: numbers) {
    System.out.println(n);       // 3 4 5 8
}

numbers.remove(4); // Elements must be unique, so delete by value is possible
for(int n: numbers) {
    System.out.println(n);       // 3 5 8
}
```

Queue interface

- Geordend
- Wachtrij => elementen één voor één afhandelen
    - Bv. enkel achteraan toevoegen en enkel vooraan afnemen

- Extra methoden

Methode	***Omschrijving***
`offer(E e)`	Voegt een element toe aan de *queue*.
`E peek()`	Geeft het eerstvolgende element maar verwijdert het niet. Geeft `null` als de *queue* leeg is.
`E element()`	Geeft het eerstvolgende element maar verwijdert het niet. Gooit een *exception* als de *queue* leeg is.
`E poll()`	Geeft het eerstvolgende element en verwijdert het. Geeft `null` als de *queue* leeg is.
`E remove()`	Geeft het eerstvolgende element en verwijdert het. Gooit een *exception* als de *queue* leeg is.

Queue interface

- Implementatie: **LinkedList**
- FIFO queue (*first in first out*)

- peek() : *kijkt* alleen maar naar volgende element
- poll() : verwijdert het ook uit de rij

- Implementatie: **PriorityQueue**
- Queue met gesorteerde items (prioriteit)
    - compareTo()

Queue interface

- Implementatie: **Deque**
- Double ended queue
  - Elementen benaderen aan twee zijden

Methode	*Omschrijving*
`offerFirst(E e)` `addFirst(E e)`	Voegt een element toe aan de kop.
`E peekFirst()` `E getFirst()`	Geeft het eerstvolgende element aan de kop maar verwijdert het niet.
`E pollFirst()` `E removeFirst()`	Geeft het eerstvolgende element aan de kop en verwijdert het.
`offerLast(E e)` `addLast(E e)`	Voegt een element toe aan de staart.
`E peekLast()` `E getLast()`	Geeft het eerstvolgende element aan de staart maar verwijdert het niet.
`E pollLast()` `E removeLast()`	Geeft het eerstvolgende element aan de staart en verwijdert het.

LinkedList

```
Queue<String> queue = new LinkedList<>(); // FIFO
queue.offer("Sam");
queue.offer("Bart");
queue.offer("Nele");

String str = queue.peek();          // just 'look' at first element
while(str != null) {
    System.out.println("About to handle " + str);
    str = queue.poll();
    System.out.println("Handling " + str);
    str = queue.peek();
}
```

```
> About to handle Sam
> Handling Sam
> About to handle Bart
> Handling Bart
> About to handle Nele
> Handling Nele
```

Map interface

- Aparte hiërarchie (niet gekoppeld aan *Collection*)
- Key-value pairs
    - Objecten makkelijk terug te vinden a.d.h.v. sleutel / etiket

Methode	***Omschrijving***
`put(K k, V v)`	Voegt een sleutel-waardepaar toe.
`V get(K k)`	Geeft de waarde van een bepaalde sleutel.
`V remove(Object k)`	Verwijdert een sleutel-waardepaar.
`keySet()`	Geeft de verzameling van sleutels in de vorm van een `Set`.
`default forEach()`	Overloopt de sleutel-waardeparen en past er een functionele methode op toe.

Map interface

- Implementatie: **HashMap**
- Ongeordend, ongesorteerd
    - Geen vaste volgorde van sleutels

- Twee generieke types (key & value)
    - Bv. ID en Person object linken, later opzoeken via ID
    - Map<String, Person> = new HashMap<>();

HashMap

```
Map<String, Integer> ingredients = new HashMap<>();
ingredients.put("Potatoes", 5);
ingredients.put("Carrots", 4);
ingredients.put("Beans", 2);
ingredients.put("Chicken", 1);

System.out.println(ingredients.get("Chicken"));

for(String key : ingredients.keySet()) {
    System.out.println(key + ": " + ingredients.get(key));
}
```

Sorteren van verzamelingen

- Zie TreeSet, PriorityQueue, …
  - Sorteren automatisch elementen
  - Regels voor sorteren bepalen

- Optie 1: natuurlijke volgorde
  - Elementen in verzameling implementeren *Comparable<T>*
  - *Eén methode:* public int compareTo(T o);
  - Zelf te implementeren m.b.v. gegevens in de klasse
  - Return waarde:
    - Huidig element (*this*) **VOOR** parameter (*o*)*:* **< 0**
    - Huidig element (*this*) **NA** parameter (*o*): **> 0**

Comparable<T>

```
public class Box implements Comparable<Box> {
    private int width, length, height;

    public Box(int width, int length, int height) {
        this.width = width;
        this.length = length;
        this.height = height;
    }

    public int getVolume() {
        return width * length * height;
    }

    @Override
    public int compareTo(Box o) {
        return this.getVolume() - o.getVolume();
    }
}
```

Sorteren van verzamelingen

- Optie 2: ***comparator***
  - Indien natuurlijke volgorde niet voldoet
  - Comparator interface gebruiken in klasse
  - Comparator meegeven in constructor van sorterende lijst

```
public class BoxLengthComparator implements Comparator<Box> {
    @Override
    public int compare(Box box1, Box box2) {
        return box1.getLength() - box2.getLength();
    }
}

…

  Set<Box> boxes = new TreeSet<>(new BoxLengthComparator());
```

Overzicht

Verzameling	Uniek	Geordend	Gesorteerd	Random Access
HashSet	JA			
LinkedHashSet	JA	JA		
TreeSet	JA	JA	JA	
LinkedList		JA		
ArrayList		JA		JA
PriorityQueue		JA	JA	
Deque		JA		

Leerstof

- Handboek: Hoofdstuk 6
- Oefeningen: Zie Blackboard

- Extra: PluralSight - Collections