BIENVENUE

TECHNIFUTUR®
CENTRE DE COMPETENCES

www.enmieux.be

FEDER-FSE
UNION EUROPEENNE

Wallonie

L'UNION EUROPÉENNE ET LA WALLONIE
INVESTISSENT DANS VOTRE AVENIR

Api Collection

INTRODUTION

- Le framework collection est une architecture unifiée pour représenter et manipuler des collections d'objets.
  - Des interfaces
    - Pour manipuler les collections indépendamment de leurs implémentations.
    - Les interfaces sont hiérarchisées pour regrouper les fonctionnalités communes
  - Des implémentations
    - Utilisations des techniques de stockage classique
    - Implémentions réutilisables dans un maximum de citrconstance
  - Des algorithmes
    - Ex recherche tri …
    - Réutilisable pour différentes implèmentations
- Bénéfices attendus
  - Réduction de l'effort de programmation
  - Amélioration de la rapidité et de la qualité des programmes
  - Interopérabilité des API
  - Réduction de l'effort d'apprentissage
  - …

DIFFÉRENCES ENTRE LES COLLECTIONS

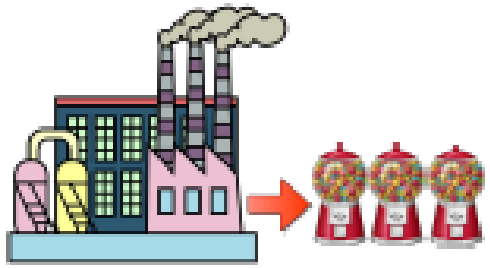
TECHNIFUTUR
CENTRE DE COMPETENCES

- Qualités différenciant les Collections
  - Taille limité ou non
  - Immuable ou non
  - Accepte les doublons ou non
  - Accepte la valeurs « null »
  - Synchronisée ou non
  - Ordonnée ou non
  - Triée ou non
- Implémentation
  - Centaines méthodes sont documentées *optional*
    - *Ces méthodes soulève alors une « UnsupportedOperationException »*

TECHNIFUTUR
CENTRE DE COMPETENCES

Toute les collections sont itérables

java.lang.Iterable<E>
Iterator<E> iterator() default void forEach(Consumer<? super T> action) default Spliterator<T> spliterator()

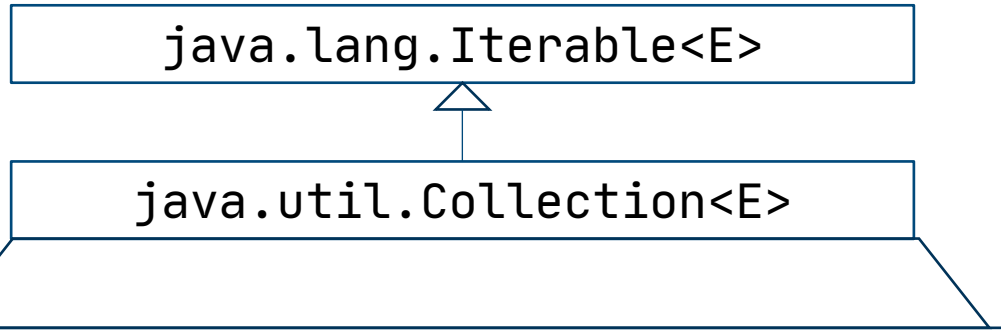


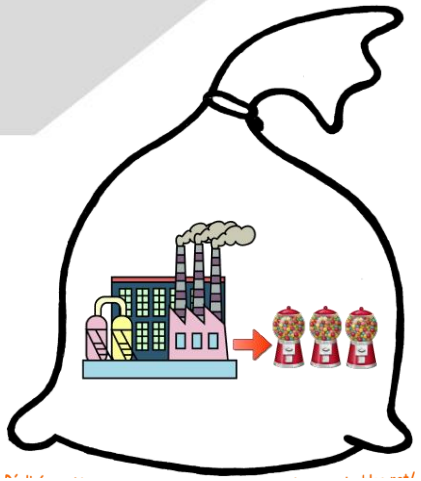
java.util.Iterator<E>
boolean hasNext() E next() default void remove() default void forEachRemaining(Consumer<? super E> action)

UTILISATION D'UN ITÉRABLE

```
public static void main(String[] args) {
    //Iterable<Personne> personnes = Arrays.asList(Personne.dataTest());
    Iterable<Personne> personnes = new ArrayList<>(Arrays.asList(Personne.dataTest()));

    Iterator<Personne> iterator = personnes.iterator();
    while (iterator.hasNext()){
        Personne p = iterator.next();
        System.out.println(p.getName());
    }

    iterator = personnes.iterator();
    while (iterator.hasNext()){
        Personne p = iterator.next();
        if(p.getName().endsWith("d"))
            iterator.remove();
     }

    for (Personne p : personnes){
        System.out.println(p.getPrenom());
    }
}
```

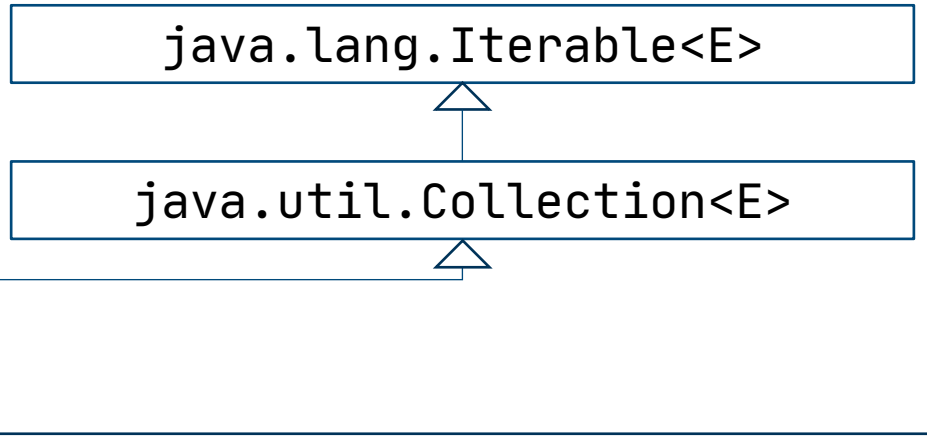
JAVA.UTIL.COLLECTION<E>

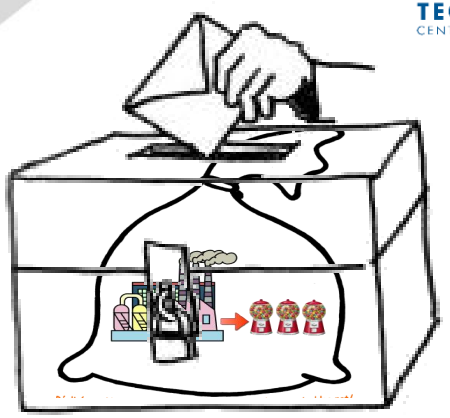




- Query operations
  - + size() :int
  - + isEmpty():boolean
  - + contains(o:Object):boolean
  - + iterator():Iterator
  - + toArray():Object[]
  - + toArray(a:T[]):T[]
- Modification Operations
  - + add(e:E):boolean
  - + remove(o:Object):boolean
- Bulk operations
  - + containsAll(c:Collection):boolean
  - + addAll(c:Collection):boolean
  - + removeAll(c:Collection):boolean
  - + retainsAll(c:Collection):boolean
  - + Clear()
- Comparison and hashing
  - + equals(o:Object):boolean
  - + hashCode():int

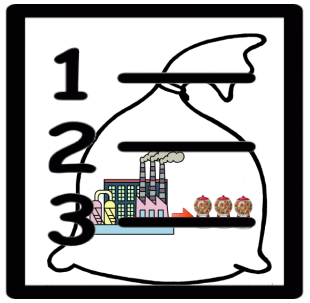
JAVA.UTIL.SET<E>





- Pas de nouvelles opérations
  Mais
- Les « Set » garantissent qu'il n'y a pas de doublons

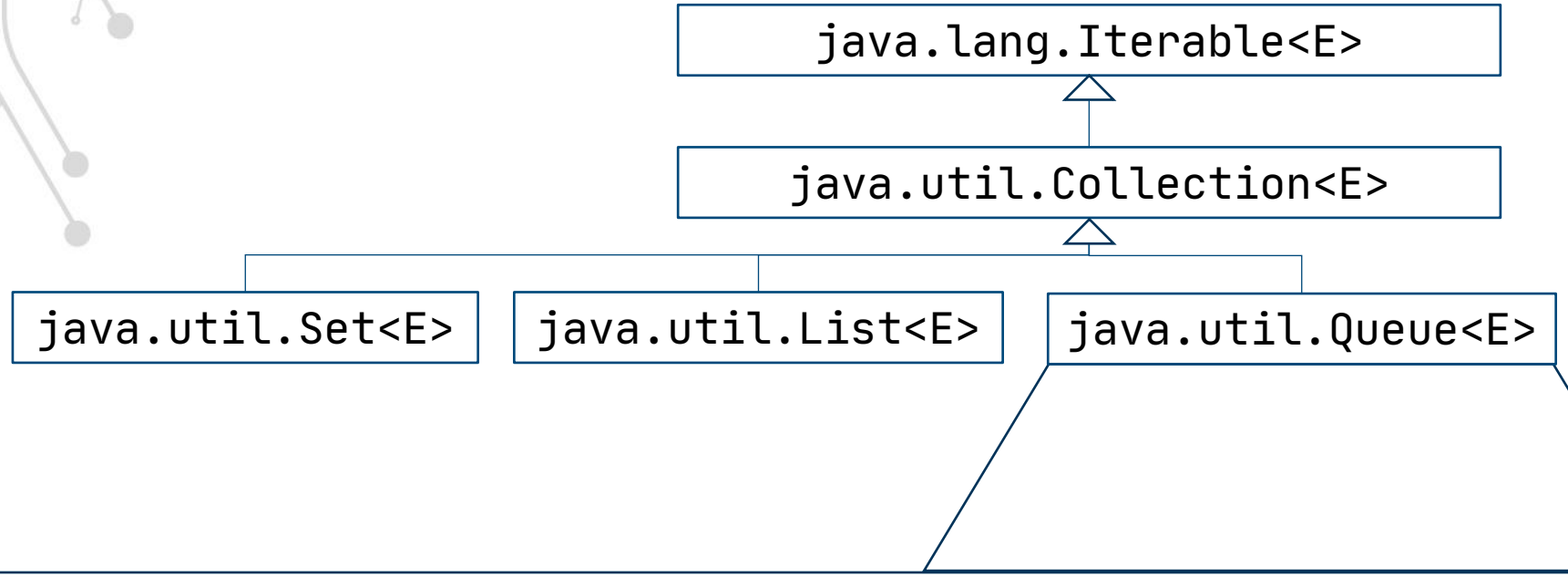
JAVA.UTIL.LIST<E>

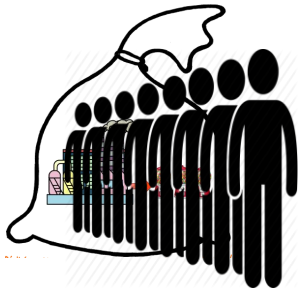


Les éléments d'une liste sont ordonnés

- Positional access operations
  - + get(index:int):E
  - + set(index:int,element:E):boolean
  - + add(index:int,element:E):boolean
  - + remove(index:int):E
  - + addAll(index:int,c:Collection) :boolean
- Search operations
  - + indexOf(o:Object):int
  - + lastindexOf(o:Objetc):int
  - + listIterator():ListIterator<E>
  - + listIterator(index:int):Listerator<E>
- View
  - + subLis(fromIndex:int, toIndex:int):List<E>

JAVA.UTIL.QUEUE<E>



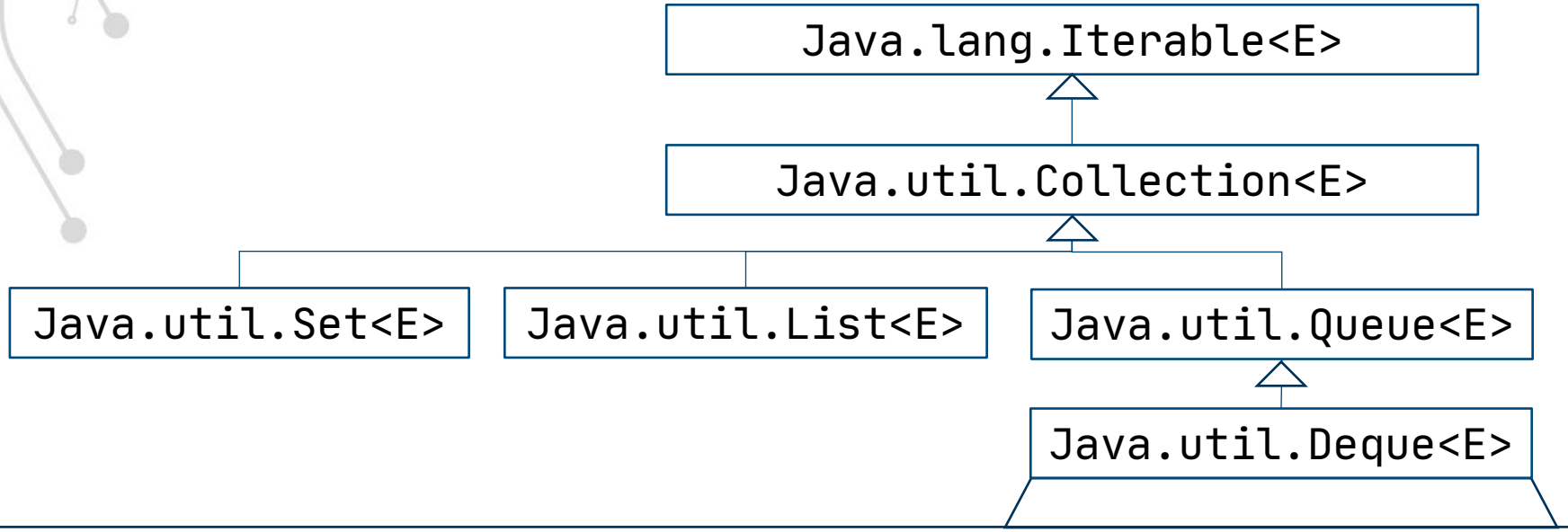


Collection permettant l'accès aux éléments suivant un ordre de priorité.
Par exemple: FOFO, LIFO

	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

```
+ element():E
+ offer(e:E):boolean
+ peek():E
+ poll():E
+ remove():E
```

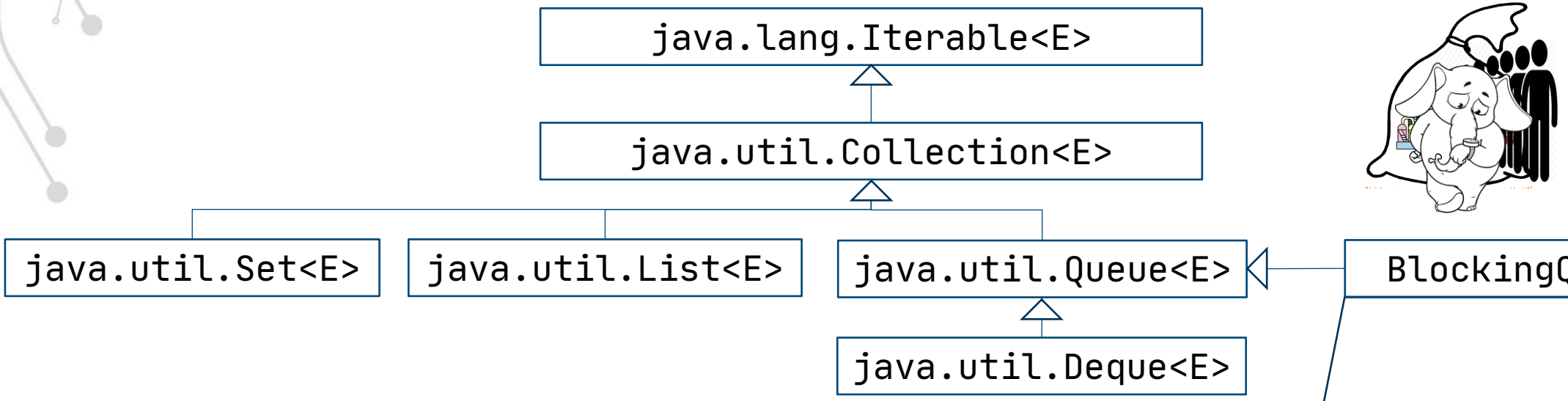
JAVA.UTIL.DEQUE<E>



Queue permettant l'accès, suivant un ordre de priorité, au premier et au dernier éléments.

	First Element (Head)		Last Element (Tail)	
	Throws exception	Special value	Throws exception	Special value
Insert	addFirst(e)	offerFirst(e)	addLast(e)	offerLast(e)
Remove	removeFirst()	pollFirst()	removeLast()	pollLast()
Examine	getFirst()	peekFirst()	getLast()	peekLast()

JAVA.UTIL.BLOCKINGQUEUE<E>

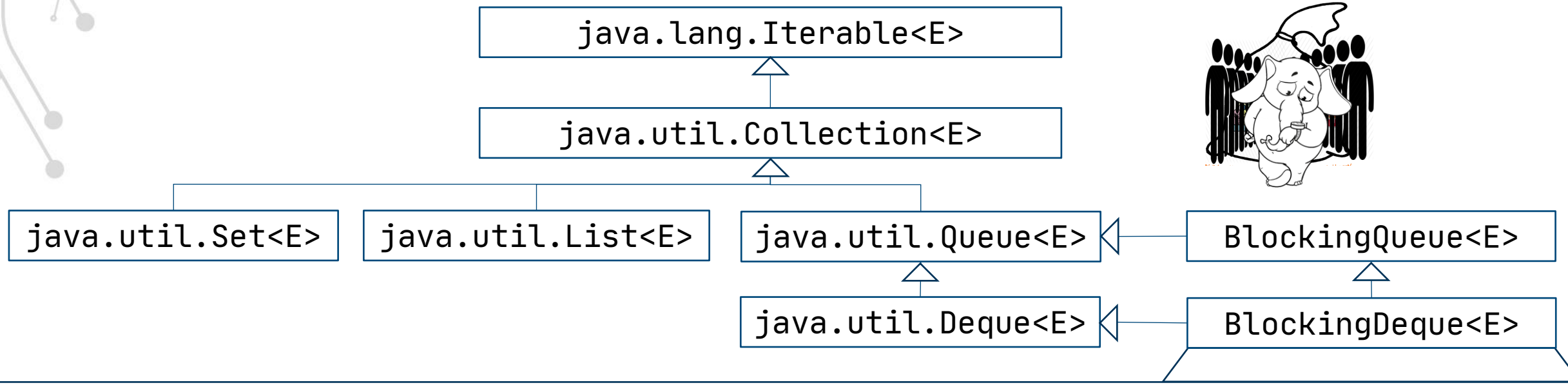


Queue permettant de mettre le thread utilisateur en attente :

- S'il n'y a pas d'éléments.
- Si la collection est remplie.

	Throws exception	Special value	Blocks	Times out
Insert	add(e)	offer(e)	put(e)	offer(e, time, unit)
Remove	remove()	poll()	take()	poll(time, unit)
Examine	element()	peek()	not applicable	not applicable

JAVA.UTIL.BLOCKINGDEQUE<E>

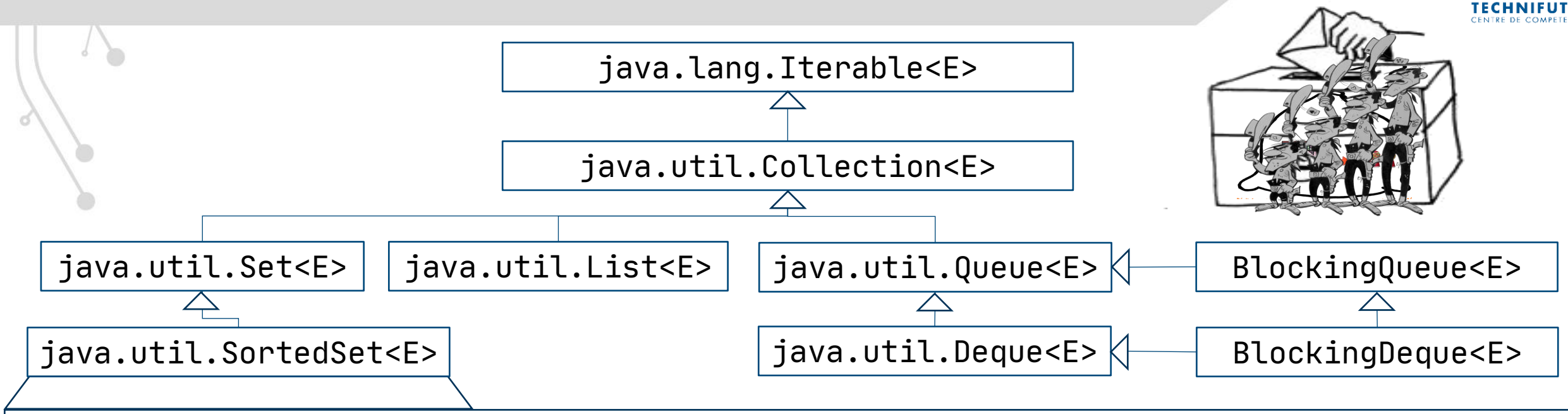


Deque permettant de mettre le thread utilisateur en attente :

- S'il n'y a pas d'éléments.
- Si la collection est remplie.

First Element (Head)				
	Throws exception	Special value	Blocks	Times out
Insert	addFirst(e)	offerFirst(e)	putFirst(e)	offerFirst(e, time, unit)
Remove	removeFirst()	pollFirst()	takeFirst()	pollFirst(time, unit)
Examine	getFirst()	peekFirst()	not applicable	not applicable
Last Element (Tail)				
Insert	addLast(e)	offerLast(e)	putLast(e)	offerLast(e, time, unit)
Remove	removeLast()	pollLast()	takeLast()	pollLast(time, unit)
Examine	getLast()	peekLast()	not applicable	not applicable

JAVA.UTIL.SORTEDSET<E>



- Range-view
  - `+ subSet(fromElement:E, toElement:E):SortedSet<E>`
  - `+ headSet(toElement:E):SortedSet<E>`
  - `+ tailSet(toElement:E):SortedSet<E>`
- Comparator access
  - `+ comparator():Comparator<? super E>`
- Endpoints
  - `+ first():E`
  - `+ last():E`

JAVA.UTIL.NAVIGABLESET<E>

- Range-view
  - + subSet(from:E,fromInclusive:boolean,to:E,toInclusive:boolean):NagigableSet<E>
  - + headSet(toElement:E,inclusive:boolean):SortedSet<E>
  - + tailSet(fromElement:E,inclusive:boolean):SortedSet<E>
- Query operations
  - + ceilling(e:E):E
  - + floor(e:E):E
  - + higer(e:E):E
  - + lower(e:E):E

JAVA.UTIL.MAP<K,V>

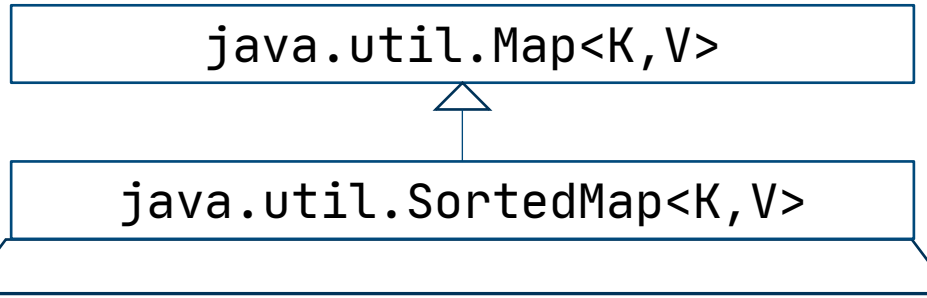
java.util.Map<K,V>

- Query Operations
  - + size():int
  - + isEmpty():boolean
  - + containsKey(key:Object):boolean
  - + containsValue(value:Object):boolean
  - + get(key:Object):V
- Modification operations
  - + put(key:K, value:V):V
  - + remove(key:Object):V
- Bulk Operations
  - + putAll(m:Map)
  - + clear()
- Views
  - + keySet():Set<K>
  - + values():Collection<V>
  - + entrySet():Set<Map.entry<K,V>>

java.util.Map.Entry<K,V>

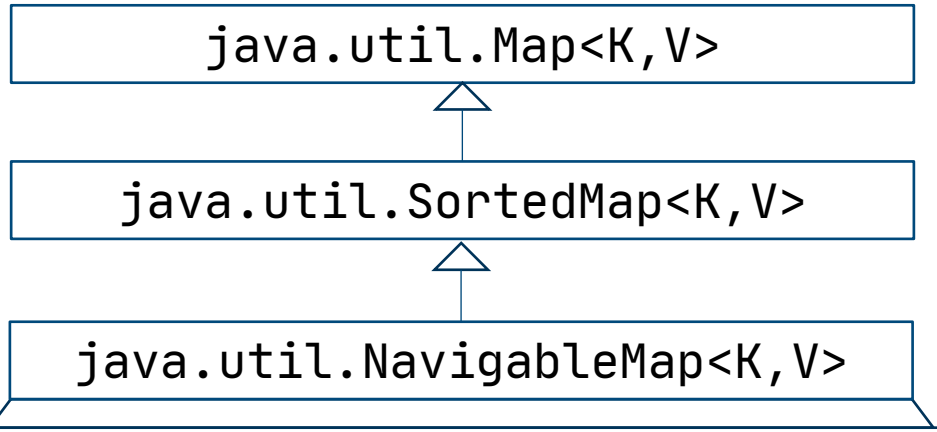
- + getKey():K
- + getValue():V
- + setValue(Value:V)

JAVA.UTIL.SORTEDMAP<K,V>



- Range-view
  - + subMap(fromKey:K, toKey:K):SortedMap<K,V>
  - + headMap(toKey:K):SortedMap<K,V>
  - + tailMap(fromKey:K):SortedMap<K,V>
- Comparator access
  - comparator():Comparator<? Super K>
- Endpoints
  - firstKey():K
  - lastKey():K

JAVA.UTIL.NAVIGABLEMAP<K,V>



- Access operations
  - + lowerEntry(K key):Map.Entry<K,V>
  - + lowerKey(K key):K
  - + floorEntry(K key):Map.Entry<K,V>
  - + floorKey(K key):K
  - + ceilingEntry(K key)Map.Entry<K,V>
  - + ceilingKey(K key):K
  - + higherEntry(K key)Map.Entry<K,V>
  - + higherKey(K key):K
  - + firstEntry():Map.Entry<K,V>
  - + lastEntry():Map.Entry<K,V>
- Modification operations
  - + pollFirstEntry()Map.Entry<K,V>
  - + pollLastEntry():Map.Entry<K,V>
- Range-view
  - + descendingMap():NavigableMap<K,V>
  - + navigableKeySet():NavigableSet<K>
  - + descendingKeySet()NavigableSet<K>
  - + subMap(from:K, inclusive:boolean, to:K, inclusive:boolean): NavigableMap<K,V>
  - + tailMap(from:K, inclusive:boolean): NavigableMap<K,V>
  - + headMap(to:K, inclusive:boolean): NavigableMap<K,V>

IMPLEMENTATION GÉNÉRALE

Interfaces	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
SortedSet			TreeSet		
List		ArrayList		LinkedList	
Queue		ArrayDeque		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap
SortedMap			TreeMap		

- PriorityQueue est une implémentation sous forme de tas à priorité pour Queue.

IMPLÉMENTATIONS PARTICULIÈRES

- Set
  - EnumSet
  - CopyOnWriteArraySet
- List
  - Vector
  - CopyOnWriteArrayList
- Map
  - EnumMap
  - WeakHashMap
  - IdentyHashMap

WRAPPER IMPLEMENTATIONS

- Méthode static dans la classe utilitaire java.util.Collections
- Pour la Synchronisation

```
+ <T> synchronizedCollection(c:Collection<T>):Collection<T>
+ <T> synchronizedSet(s:Set<T>):Set<T>
+ <T> synchronizedList(list:List<T>):List<T>
+ <K,V> synchronizedMap(m:Map<K,V>):Map<K,V>
+ <T> synchronizedSortedSet(s:SortedSet<T>):SortedSet<T>
+ <K,V> synchronizedSortedMap(m:SortedMap<K,V>):SortedMap<K,V>
```

- Wrappers non modifiables

```
+ <T> unmodifiableCollection(c:Collection<? extends T>):Collection<T>
+ <T> unmodifiableSet(s:Set<? extends T>):Set<T>
+ <T> unmodifiableList(list:List<? extends T>):List<T>
+ <K,V> unmodifiableMap(m:Map<? extends K, ? extends V>):Map<K, V>
+ <T> unmodifiableSortedSet(s:SortedSet<? extends T>):SortedSet<T>
+ <K,V> unmodifiableSortedMap(m:SortedMap<K, ? extends V> ):SortedMap<K, V>
```

ALGORITHMES

- Fonctionnalités réutilisable pour les Collections
  - Accessible dans la classe java.util.Collections
- Tri
  - Permet de réordonner les éléments d'une liste
    - Rapide : n log(n)
    - Stable
    - Ordre « naturel » / via Comparator
- Mélange
- Manipulation des données
  - Inverser : reverse
  - Remplir : fill
  - Copier : copy
  - Inversion d'éléments : swap
  - Ajout des éléments d'une collection ou d'un tableur dans une autre: addAll
- Recherche
  - Recherche dichotomique : binarySearch
- Composition
  - Fréquence d'un élément dans une collection
  - Tester si 2 collections sont disjointes
- Min max

TECHNIFUTUR®
CENTRE DE COMPETENCES

MERCI DE VOTRE ATTENTION

WWW.TECHNIFUTUR.BE

VOS CONTACTS