Collections en Java

Table of Contents

Array

```
digraph G {
  graph[bgcolor=transparent]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  a [label=" 0 | 1 | 2 | 3 | 4 "]
}
```

Simple declaration

```
class Application{
  public static void main(String[] args){
    int nb = 3;
    String[] boyNames = new String[nb];
    names[0] = "Camille";
    names[1] = "Lucas";
    names[2] = "Enzo";

    double[][] matrix = new double[4][4];
    int[][]  dimensions = new int[2][];
    dimensions[0] = new int[4];
    dimensions[1] = new int[9];
  }
}
```

Advanced declaration

```
class Application{
  public static void main(String[] args){
    String[] girlNames = {"Camille","Léa","Chloé"};
    int[] indices = {10, 20, 30, 40, 50};
    int[][] valueSet = {{}, {1}, {2, 3}};

    Object[] objects = {
      girlNames[0],
      girlNames[0].length(),
      indices,
      valueSet.length};
  }
}
```

Utilitaires

```
class Application{
  public static void main(String[] args){
    String[] boyNames = {"Camille","Lucas","Enzo"};
    String[] girlNames = {"Camille","Léa","Chloé"};

    Arrays.equals(boyNames, girlNames);
    Arrays.hashCode(girlNames);
    Arrays.toString(boyNames);

    double[][] polygonA = {{0.0, 0.0}, {0.0, 1.0}, {2.0, 0.0}};
    double[][] polygonB = {{0.0, 0.0}, {0.0, 1.0}, {2.0, 0.0}};
    Arrays.deepEquals(polygonA, polygonB);
    Arrays.deepHashCode(polygonB);
    Arrays.deepToString(polygonA);
  }
}
```

Utilitaires 1/2

```
class Application{
  public static void main(String[] args){
    double[][] polygonA = {{0.0, 0.0}, {0.0, 1.0}, {2.0, 0.0}};
    double[][] polygonB = Arrays.copyOf(polygonA);

    System.arraycopy(polygonA, 0, polygonB, 0, polygonA.length);

    String[] boyNames = {"Camille","Lucas", /*...,*/ "Enzo"};
    String[] range = Arrays.copyOfRange(boyNames, boyNames.length - 2,
boyNames.length);

    char[] password = {'b', 'a', 'd', 'p', 'w', 'd', '0', '!'};
    Arrays.fill(password, '\0');
  }
}
```

Utilitaires 2/2

```
class Application{
  public static void main(String[] args){
    String[] boyNames = {"Camille","Lucas","Enzo"};
    String[] girlNames = {"Camille","Léa","Chloé"};

    Arrays.equals(boyNames, girlNames);
    Arrays.hashCode(girlNames);
    Arrays.toString(boyNames);

    double[][] polygonA = {{0.0, 0.0}, {0.0, 1.0}, {2.0, 0.0}};
    double[][] polygonB = {{0.0, 0.0}, {0.0, 1.0}, {2.0, 0.0}};
    Arrays.deepEquals(polygonA, polygonB);
    Arrays.deepHashCode(polygonB);
    Arrays.deepToString(polygonA);
  }
}
```

Reflection

```
class Application{
  public static void main(String[] args){
    Object object = "Object typed from an API";
    Object anotherObject = "Another object typed from the same API";

    Object array = Array.newInstance(object.getClass(), 2);
    Array.set(array, 0, object);
    Array.set(array, 1, anotherObject);

    Object o = Array.get(array, 0);
  }
}
```

Legacy

```
digraph G {
  graph[bgcolor=transparent]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  Stack -> Vector
  Vector -> Enumeration [label="elements()", style="dotted"]

  Hashtable -> Dictionary
  Dictionary -> Enumeration [label="elements()", style="dotted"]
  Dictionary -> Enumeration [label="keys()", style="dotted"]

  Properties -> Hashtable

  StringTokenizer -> Enumeration

  BitSet
}
```

Enumeration

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

interface Enumeration {
    boolean hasMoreElements()
    Object nextElement()
}
@enduml
```

Vector

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

interface Enumeration {
    boolean hasMoreElements()
    Object nextElement()
}

class Vector {
    # Object[] elementData;
    # int elementCount;
    + int size()
    + Object elementAt(int)
    + void addElement(Object)
    + void insertElementAt(Object, int)
    + void setElementAt(Object, int)
    + void removeElementAt(int)
    + void removeAllElements()
    + Object firstElement()
    + Object lastElement()
}

Vector --> Enumeration : elements()
@enduml
```

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

participant User
create Vector
User -> Vector: new Vector()
create array
Vector -> array: new Object[10]
array --> Vector
Vector --> User

loop 10times
    User -> Vector: add(Object)
    Vector -> array: set(elementCount)
    array --> Vector
    Vector --> User
end

User -> Vector: add(Object)
note over Vector, array
    copy the  content of array into
    another array twice the size
end note

Vector -> array: set(elementCount)
array --> Vector
Vector --> User
@enduml
```

Stack

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

class Vector {
    + int size()
    + Object elementAt(int)
    + Object firstElement()
    + Object lastElement()
}

class Stack {
    + Object push(Object)
    + Object pop()
    + Object peek()
    + boolean empty()
    + int search(Object)
}
Stack  -|> Vector
@enduml
```

```
digraph G{
    graph[bgcolor=transparent]
    node[shape=record, color=white,
fontcolor=white, fontname="Consolas"]
    edge[color=white, fontcolor="white",
fontname="Consolas"]

    push[label="push", shape=none,
contraint=false]

stack[label="item1|item2|<2>item3|____"]
    pop[label="pop/peek", shape=none,
contraint=false]

    push -> stack:2 -> pop;

    {rank=same; push; pop;}
}
```

Dictionary, Hashtable, Properties

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

class Properties{
    + void load(Reader)
    + void load(InputStream)
    + void loadFromXML(InputStream)
    + void store(Writer)
    + void store(OutputStream)
    + void storeToXML(OutputStream, String)
}

class Hashtable{
    + boolean replace(Object, Object, Object)
    + Object replace(Object, Object)
}

abstract Dictionary{
    + int size()
    + boolean isEmpty()
    + Object get(Object)
    + Object put(Object, Object)
    + Object remove(Object)
}

interface Enumeration {
    boolean hasMoreElements()
    Object nextElement()
}

Dictionary --> Enumeration : elements()
Dictionary --> Enumeration : keys()

Properties -|> Hashtable
Hashtable -|> Dictionary

@enduml
```

BitSet

```
boolean[] bits = new boolean[1024];
System.out.println(ClassLayout.parseInstance(bits).toPrintable());
```

```
[Z object internals:
 OFFSET  SIZE      TYPE DESCRIPTION            VALUE
      0     4           (object header)        01 00 00 00 (00000001 00000000 00000000
00000000) (1)
      4     4           (object header)        00 00 00 00 (00000000 00000000 00000000
00000000) (0)
      8     4           (object header)        7b 12 07 00 (01111011 00010010 00000111
00000000) (463483)
     12     4           (object header)        00 04 00 00 (00000000 00000100 00000000
00000000) (1024)
     16  1024   boolean [Z.                     N/A
Instance size: 1040 bytes
```

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

class BitSet{
    + {static} valueOf(Long[]/LongBuffer)
    + {static} valueOf(Byte[]/ByteBuffer)
    ---
    + void set(int, int)
    + void clear(int, int)
    + void flip(int, int)
    + boolean get(int, int)
    + BitSet get(int, int)
    ----
    + void and(BitSet)
    + void andNot(BitSet)
    + void xor(BitSet)
    + void or(BitSet)
    ---
    ...
}

@enduml
```

Collections Framework

```
digraph G {
  graph[bgcolor=transparent, color=grey]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  Iterator
  Iterable
  Collection [color=green]
  List [color=green]
  Queue [color=green]
  Deque
  Set [color=green]
  SortedSet
  Map [color=green]
  SortedMap

  Comparable
  Comparator

  {Comparable, Comparator} -> SortedSet  [style=invis]
  {Comparable, Comparator} -> SortedMap  [style=invis]

  Collection -> Iterable
  {List, Queue, Set} -> Collection
  Deque -> Queue
  SortedSet -> Set
  SortedMap -> Map
  Iterator -> Iterable [label="iterator()", dir=back, style=dotted, color=grey,
fontcolor=grey]

  Set -> Map [label="keys()", dir=back, style=dotted, color=grey, fontcolor=grey]
  Set -> Map [label="entrySet()", dir=back, style=dotted, color=grey, fontcolor=grey]
  Collection -> Map [label="values()", dir=back, style=dotted, color=grey,
fontcolor=grey]
}
```

Abstract implementations

```
digraph G {
  graph[bgcolor=transparent, color=grey]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  Collection [color=gray, fontcolor=grey]
  List [color=gray, fontcolor=grey]
  Set [color=gray, fontcolor=grey]
  Queue [color=gray, fontcolor=grey]
  Map [color=gray, fontcolor=grey]

  List -> Collection [color=gray]
  Set -> Collection [color=gray]
  Queue -> Collection [color=gray]

  AbstractCollection -> Collection
  AbstractList -> {AbstractCollection, List}
  AbstractSequentialList -> AbstractList
  AbstractSet -> {AbstractCollection, Set}
  AbstractQueue -> {AbstractCollection, Queue}
  AbstractMap -> Map
}
```

Collection Interface

constructor	traverse

`Collection()`
`Collection(Collection<? extends E> c)`
	`boolean equals(Object o)`
	`int hashCode()`
	`Object[] toArray()`
	`<T> T[] toArray(T[] a)`

presence	traverse

`boolean isEmpty()`
`boolean contains(Object o)`
`boolean containsAll(Collection<?> c)`
	`Iterator<E> iterator()`

add [Optional, may throw UnsupportedOperationException]	remove [Optional, may throw UnsupportedOperationException]

```
boolean add(E e)
boolean addAll(Collection<? extends E>
c)
```

```
void clear()
boolean remove(Object o)
boolean removeAll(Collection<? extends
E> c)
boolean retainAll(Collection<? extends
E> c)
```

Iterator

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old

interface Enumeration<E> {
    boolean hasMoreElements()
    E nextElement()
}

interface Iterator<E>  {
    boolean hasNext()
    E next()
    void remove()
}
@enduml
```

List

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface List<E>
interface RandomAccess
abstract AbstractList<E>
abstract AbstractSequentialList<E>
class LinkedList<E> #grey
class ArrayList<E> #grey
class Arrays.asList<E> #grey
class CopyOnWriteArrayList<E> #grey
class SubList<E>
class RandomAccessSubList<E>

AbstractSequentialList --|> AbstractList
LinkedList --|> AbstractSequentialList
ArrayList --|> AbstractList
CopyOnWriteArrayList ..|> List
CopyOnWriteArrayList ..|> RandomAccess
ArrayList ..|> RandomAccess
SubList --|> AbstractList
RandomAccessSubList ..|> RandomAccess
RandomAccessSubList --|> SubList
Arrays.asList ..|> RandomAccess
Arrays.asList ..|> AbstractList
@enduml
```

List Interface

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface List<E>{
  void add(int index, E e)
  boolean addAll(int index, Collection<? extends E> c)
  --
  int indexOf(Object o)
  int lastIndexOf(Object o)
  E get(int index)
  E set(int index, E e)
  E remove(int index)
  --
  ListIterator<E> listIterator()
  ListIterator<E> listIterator(int indx)
  --
  List<E> subList(int fromIndex, int toIndex)
}

interface ListIterator<E> {
  boolean hasNext() // from Iterator<E>
  E next() // from Iterator<E>
  int nextIndex()
  --
  void set(E e)
  void add(E e)
  void remove() // from Iterator<E>
  --
  boolean hasPrevious()
  E previous()
  int previousIndex()
}
@enduml
```

ArrayList

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent

participant User
create ArrayList
User -> ArrayList: new ArrayList()
create array
ArrayList -> array: new Object[10]
array --> ArrayList
ArrayList --> User

loop 10times
    User -> ArrayList: add(Object)
    ArrayList -> array: set(elementCount)
    array --> ArrayList
    ArrayList --> User
end

User -> ArrayList: add(Object)
note over ArrayList, array
    copy the  content of array into
    another array twice the size
end note

ArrayList -> array: set(elementCount)
array --> ArrayList
ArrayList --> User
@enduml
```

LinkedList

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface LinkedList<E>{
  void addFirst(E e)
  E getFirst()
  E removeFirst()
  --
  void addLast(E e)
  E getLast()
  E removeLast()
}

LinkedList ..|> List
LinkedList ..|> Deque
@enduml
```

```
digraph G{
    graph[bgcolor=transparent,
rankdir=LR]
    node[shape=record, color=white,
fontcolor=white, fontname="Consolas"]
    edge[color=white, fontcolor="white",
fontname="Consolas", tailclip=false]

    item0 [label="<v>0|<p>"]
    item1 [label="<v>1|<p>"]
    item2 [label="<v>2|<p>"]
    null [shape=none]

    item0:p:c -> item1:v
    item1:p:c -> item2:v
    item2:p:c -> null
}
```

CopyOnWriteArrayList

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

class CopyOnWriteArrayList<E>{
  boolean addIfAbsent(int index, E e)
  int addAllAbsent(Collection< ? extends E>)
}
@enduml
```

Arrays.asList

```
class Application {
  public static void main(String[] args){
    Arrays.asList(1, 2, 3 ,4);
    List.of(1,2,3,4);
  }
}
```

Set

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface Set<E>
interface SortedSet<E>
interface NavigableSet<E>

abstract AbstractSet<E>
abstract EnumSet<E>

class HashSet<E> #grey
class LinkedHashSet<E> #grey
class TreeSet<E> #grey
class CopyOnWriteArraySet<E> #grey
class ConcurrentSkipListSet<E>

SortedSet --|> Set
NavigableSet --|> SortedSet

AbstractSet ..|> Set
HashSet ..|> Set
LinkedHashSet ..|> Set
TreeSet ..|> NavigableSet
ConcurrentSkipListSet ..|> NavigableSet

EnumSet --|> AbstractSet
HashSet --|> AbstractSet
CopyOnWriteArraySet --|> AbstractSet
TreeSet --|> AbstractSet
ConcurrentSkipListSet --|> AbstractSet

LinkedHashSet --|> HashSet
@enduml
```

Set Interfaces

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none
```

```
interface Collection<E>{
  boolean isEmpty()
  boolean contains(Object o)
  boolean containsAll(Collection<?> c)

  boolean add(E e)
  boolean addAll(Collection<? extends E> c)

  void clear()
  boolean remove(Object o)
  boolean removeAll(Collection<? extends E> c)
  boolean retainAll(Collection<? extends E> c)
}

interface Set<E>

interface SortedSet<E>{
  E first()
  E last()
  SortedSet headSet(E toElement)
  SortedSet tailSet(E fromElement)
  SortedSet subSet(E fromElement, E toElement)
  Comparator<? super E> comparator()
}

interface NavigableSet<E>{
  E ceiling(E e)
  E floor(E e)
  E higher(E e)
  E lower(E e)

  E pollFirst()
  E pollLast()

  NavigableSet<E> headSet(E toElement, boolean inclusive)
  NavigableSet<E> tailSet(E fromElement, boolean inclusive)

  NavigableSet<E> subSet(E from, boolean inclusive, E to, boolean inclusive)

  Iterator<E> descendingIterator()
  NavigableSet<E> descendingSet()
}

Collection <|- Set
Set <|- SortedSet
SortedSet <|- NavigableSet
@enduml
```

HashSet

LinkedHashSet

EnumSet

TreeSet

ConcurrentSkipListSet

CopyOnWriteArraySet

Map

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface Map<K, V>
interface SortedMap<K, V>
interface NavigableMap<K, V>
abstract AbstractMap<K, V>

interface ConcurrentMap<K, V>
interface ConcurrentNavigableMap<K, V>

class ConcurrentHashMap<K, V>
class ConcurrentSkipListMap<K, V>

class EnumMap<K, V>
class HashMap<K, V>
class LinkedHashMap<K, V>
class IdentityHashMap<K, V>
class WeakHashMap<E>
class TreeMap<E>

SortedMap --|> Map
NavigableMap --|> SortedMap
ConcurrentMap --|> Map
ConcurrentNavigableMap --|> NavigableMap
ConcurrentNavigableMap --|> ConcurrentMap

AbstractMap ..|> Map
ConcurrentHashMap ..|> ConcurrentMap
ConcurrentSkipListMap ..|> NavigableMap

AbstractMap <|-- EnumMap
AbstractMap <|-- HashMap
AbstractMap <|-- IdentityHashMap
WeakHashMap --|> AbstractMap
TreeMap --|> AbstractMap
ConcurrentHashMap --|> AbstractMap
ConcurrentSkipListMap --|> AbstractMap

LinkedHashMap --|> HashMap
@enduml
```

Map Interface

HashMap

LinkedHashMap

ConcurrentHashMap

TreeMap

EnumMap

WeakHashMap

IdentityHashMap

Queue

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface Queue<E>
interface Deque<E>
interface BlockingQueue<E>
interface BlockingDeque<E>
interface TransferQueue<E>

abstract AbstractQueue<E>

class LinkedList<E>
class ArrayDeque<E>
class ConcurrentLinkedQueue<E>
class PriorityBlockingQueue<E>
class PriorityQueue<E>
class ArrayBlockingQueue<E>
class LinkedTransferQueue<E>
class DelayQueue<E>
class LinkedBlockingQueue<E>
class SynchronousQueue<E>
@enduml
```

Queue Interface

LinkedTransferQueue

PriorityQueue

ConcurrentLinkedQueue

Deque Interface

ArrayDeque

BlockingQueue interface

ArrayBlockingQueue

LinkedBlockingQueue

PriorityBlockingQueue

DelayQueue

SynchronousQueue

BlockingDeque interface

LinkedBlockingDeque

Sorting

Comparable

Comparator

Collections Algorithms

Collections Framework

```
digraph G {
  graph[bgcolor=transparent, color=grey]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  Iterator
  Iterable
  Collection [color=green]
  List [color=green]
  Queue [color=green]
  Deque
  Set [color=green]
  SortedSet
  Map [color=green]
  SortedMap

  Comparable
  Comparator

  {Comparable, Comparator} -> SortedSet  [style=invis]
  {Comparable, Comparator} -> SortedMap  [style=invis]

  Collection -> Iterable
  {List, Queue, Set} -> Collection
  Deque -> Queue
  SortedSet -> Set
  SortedMap -> Map
  Iterator -> Iterable [label="iterator()", dir=back, style=dotted, color=grey,
fontcolor=grey]

  Set -> Map [label="keys()", dir=back, style=dotted, color=grey, fontcolor=grey]
  Set -> Map [label="entrySet()", dir=back, style=dotted, color=grey, fontcolor=grey]
  Collection -> Map [label="values()", dir=back, style=dotted, color=grey,
fontcolor=grey]
}
```

Abstract implementations

```
digraph G {
  graph[bgcolor=transparent, color=grey]
  node[shape=record, color=white, fontcolor=white, fontname="Consolas"]
  edge[color=white, fontcolor="white", fontname="Consolas"]

  Collection [color=gray, fontcolor=grey]
  List [color=gray, fontcolor=grey]
  Set [color=gray, fontcolor=grey]
  Queue [color=gray, fontcolor=grey]
  Map [color=gray, fontcolor=grey]

  List -> Collection [color=gray]
  Set -> Collection [color=gray]
  Queue -> Collection [color=gray]

  AbstractCollection -> Collection
  AbstractList -> {AbstractCollection, List}
  AbstractSequentialList -> AbstractList
  AbstractSet -> {AbstractCollection, Set}
  AbstractQueue -> {AbstractCollection, Queue}
  AbstractMap -> Map
}
```

Collection Interface

constructor	traverse

`Collection()`
`Collection(Collection<? extends E> c)`
	`boolean equals(Object o)`
	`int hashCode()`
	`Object[] toArray()`
	`<T> T[] toArray(T[] a)`

presence	traverse

`boolean isEmpty()`
`boolean contains(Object o)`
`boolean containsAll(Collection<?> c)`
	`Iterator<E> iterator()`

add [Optional, may throw UnsupportedOperationException]	remove [Optional, may throw UnsupportedOperationException]

```
boolean add(E e)
boolean addAll(Collection<? extends E>
c)
```

```
void clear()
boolean remove(Object o)
boolean removeAll(Collection<? extends
E> c)
boolean retainAll(Collection<? extends
E> c)
```

Iterator

```
@startuml

skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old

interface Enumeration<E> {
    boolean hasMoreElements()
    E nextElement()
}

interface Iterator<E>  {
    boolean hasNext()
    E next()
    void remove()
}
@enduml
```

List

```
@startuml
skinparam monochrome reverse
skinparam handwritten true
skinparam BackgroundColor transparent
skinparam genericDisplay old
set namespaceSeparator none

interface List<E>
interface RandomAccess
abstract AbstractList<E>
abstract AbstractSequentialList<E>
class LinkedList<E> #grey
class ArrayList<E> #grey
class Arrays.asList<E> #grey
class CopyOnWriteArrayList<E> #grey
class SubList<E>
class RandomAccessSubList<E>

AbstractSequentialList --|> AbstractList
LinkedList --|> AbstractSequentialList
ArrayList --|> AbstractList
CopyOnWriteArrayList ..|> List
CopyOnWriteArrayList ..|> RandomAccess
ArrayList ..|> RandomAccess
SubList --|> AbstractList
RandomAccessSubList ..|> RandomAccess
RandomAccessSubList --|> SubList
Arrays.asList ..|> RandomAccess
Arrays.asList ..|> AbstractList
@enduml
```

Set

SortedSet

NavigableSet

Map

Sorted Map

Navigable Map

Queue

Deque

Streams