Datenstrukturen in Java

Datenstrukturen und Algorithmen im JDK

Andreas Klipp,
Stephan Prätsch

26. Februar 2016

Inhaltsverzeichnis

1. Einleitung
2. java.util.List
3. java.util.Map
4. java.util.Queue
5. java.util.Set
6. Hilfsfunktionen
7. Abschluss

mercateo

Einleitung

Einleitung

Warum das alles?

Einleitung

Warum das alles?

- Welche Liste soll ich nehmen?

mercateo

Einleitung

Warum das alles?

- Welche Liste soll ich nehmen?
- Ich brauche eine Map, die thread-safe ist. Welche nehme ich?

mercateo

Einleitung

Warum das alles?

- Welche Liste soll ich nehmen?
- Ich brauche eine Map, die thread-safe ist. Welche nehme ich?
- Ich brauche eine Queue. Welche gibt's überhaupt?

mercateo

Einleitung

Warum das alles?

- Welche Liste soll ich nehmen?
- Ich brauche eine Map, die thread-safe ist. Welche nehme ich?
- Ich brauche eine Queue. Welche gibt's überhaupt?
- Mein Set soll sortiert sein. Gibt's ein SortedHashSet?

mercateo

Einleitung

Warum das alles?

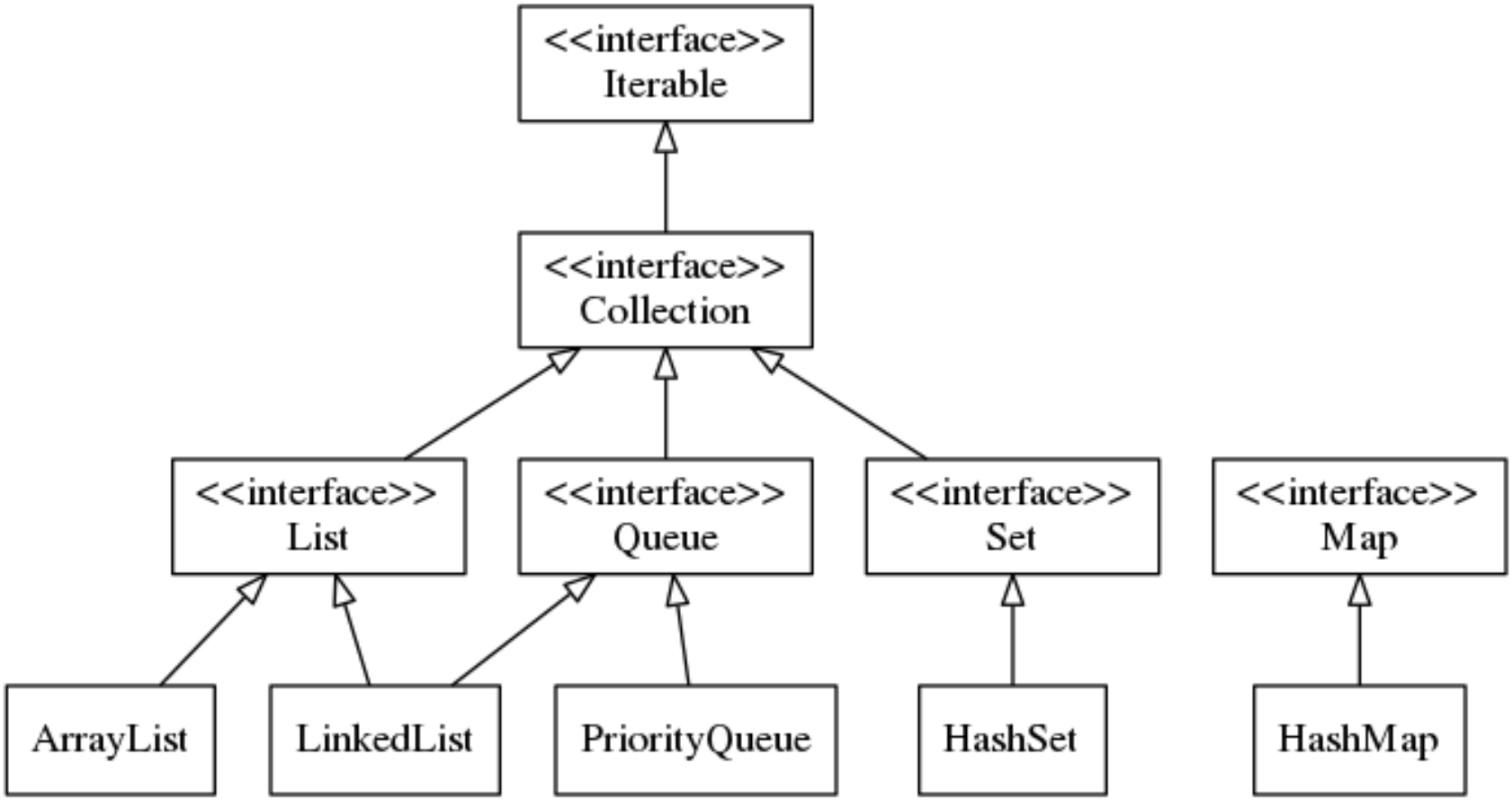
- Welche Liste soll ich nehmen?
- Ich brauche eine Map, die thread-safe ist. Welche nehme ich?
- Ich brauche eine Queue. Welche gibt's überhaupt?
- Mein Set soll sortiert sein. Gibt's ein SortedHashSet?

Ziel:

Übersicht der **vorhandenen** Datenstrukturen im JDK und deren **Besonderheiten**.

mercateo

Stark gekürzte Übersicht

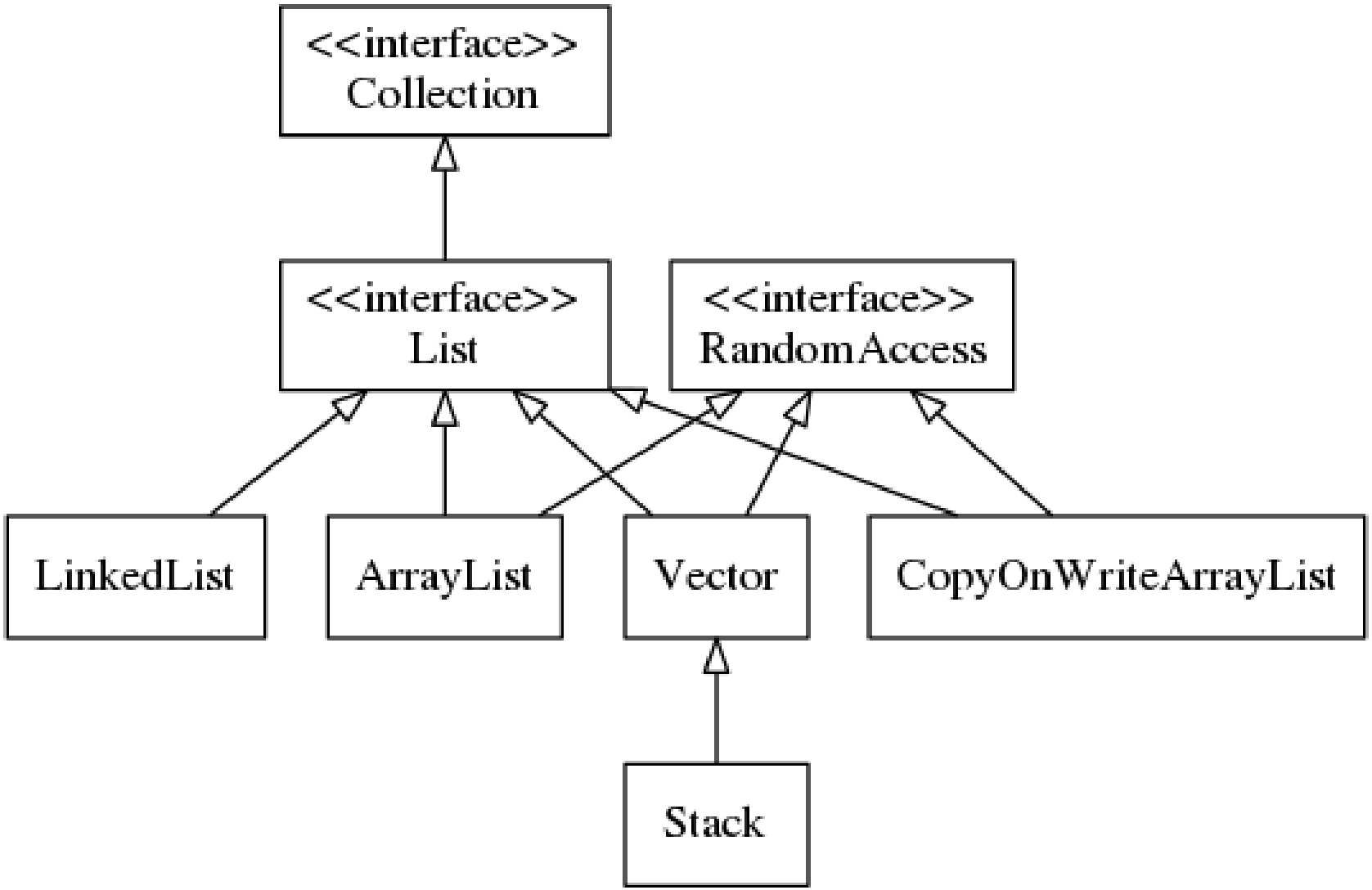


mercateo

java.util.List

mercateo

java.util.List



mercateo

ArrayList

- Array beinhaltet Elemente

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector
- null Element erlaubt

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector
- null Element erlaubt
- remove Operationen verkleinern das Array nicht (trimToSize() verkleinert bis auf die aktuelle Größe)

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector
- null Element erlaubt
- remove Operationen verkleinern das Array nicht (trimToSize() verkleinert bis auf die aktuelle Größe)
- Re-size Operationen mittels System.arrycopy (nativ, schnell)

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector
- null Element erlaubt
- remove Operationen verkleinern das Array nicht (trimToSize() verkleinert bis auf die aktuelle Größe)
- Re-size Operationen mittels System.arrycopy (nativ, schnell)
- remove und add sollten nur am Ende der Liste geschehen

mercateo

ArrayList

- Array beinhaltet Elemente
- Kapazität kann explizit durch ensureCapacity erhöht werden
- fail fast Iterator
- quasi ein nicht synchronisierter Vector
- null Element erlaubt
- remove Operationen verkleinern das Array nicht (trimToSize() verkleinert bis auf die aktuelle Größe)
- Re-size Operationen mittels System.arrycopy (nativ, schnell)
- remove und add sollten nur am Ende der Liste geschehen
- nicht thread-safe

ArrayList - Zugriffszeiten

Operation	Laufzeit
add	O(1) / O(n) mit re-size
add(int, Object)	je kleiner die Posistion, desto länger
remove(Object)	O(n) TODO so viel wegen shifting? was ist shifting?
remove(int)	O(n) wegen shifting
get	O(1)

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays
- "snaphot style iterator"

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays
- "snaphot style iterator"
- diese Iteratoren unterstützen keine manipulativen Operationen (UnsupportedOperationException)

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays
- "snaphot style iterator"
- diese Iteratoren unterstützen keine manipulativen Operationen (UnsupportedOperationException)
- null Elemente erlaubt

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays
- "snaphot style iterator"
- diese Iteratoren unterstützen keine manipulativen Operationen (UnsupportedOperationException)
- null Elemente erlaubt
- seit Java 1.5

mercateo

CopyOnWriteArrayList

- thread-safe Variante von ArrayList
- schreibende Operationen (add, set, ...) erstellen eine neue Kopie des Arrays
- "snaphot style iterator"
- diese Iteratoren unterstützen keine manipulativen Operationen (UnsupportedOperationException)
- null Elemente erlaubt
- seit Java 1.5
- lesen so teuer wie ArrayList, schreiben teurer wegen der Kopie

mercateo

CopyOnWriteArrayList vs Collections.synchronizedList(new ArrayList()

- synchronizedList synchronisiert immer, auch lesende Zugriffe

mercateo

CopyOnWriteArrayList vs Collections.synchronizedList(new ArrayList()

- synchronizedList synchronisiert immer, auch lesende Zugriffe
- Iterator der synchronizedList muss eigenständig synchronisiert werden (fail fast), CopyOneWrite hat fail save

mercateo

CopyOnWriteArrayList vs Collections.synchronizedList(new ArrayList()

- synchronizedList synchronisiert immer, auch lesende Zugriffe
- Iterator der synchronizedList muss eigenständig synchronisiert werden (fail fast), CopyOneWrite hat fail save

$\Rightarrow$ CopyOnWriteArrayList

mercateo

CopyOnWriteArrayList - Wann nehmen?

Wenn man eine ArrayList braucht,

mercateo

CopyOnWriteArrayList - Wann nehmen?

Wenn man eine ArrayList braucht,
die thread-safe sein soll,

mercateo

CopyOnWriteArrayList - Wann nehmen?

Wenn man eine ArrayList braucht,
die thread-safe sein soll,
mit wenig schreibenden, aber vielen lesenden Zugriffen.

mercateo

LinkedList

- double linked

LinkedList

- double linked
- null Elemente erlaubt

mercateo

LinkedList

- double linked
- null Elemente erlaubt
- Operationen mit Index O(n): Traversierung durch gesamte Liste, vorn oder hinten beginnend

mercateo

LinkedList

- double linked
- null Elemente erlaubt
- Operationen mit Index O(n): Traversierung durch gesamte Liste, vorn oder hinten beginnend
- nicht thread-safe

mercateo

LinkedList

- double linked
- null Elemente erlaubt
- Operationen mit Index O(n): Traversierung durch gesamte Liste, vorn oder hinten beginnend
- nicht thread-safe
- fail fast Iterator

mercateo

LinkedList

- double linked
- null Elemente erlaubt
- Operationen mit Index O(n): Traversierung durch gesamte Liste, vorn oder hinten beginnend
- nicht thread-safe
- fail fast Iterator
- Deque Eigenschaften sind herausstechend: addFirst, getFirst, removeFirst, addLast, getLast und removeLast

mercateo

LinkedList - Zugriffszeiten

Operation	Laufzeit
add	O(1)
remove(Object)	O(1)
remove(int)	O(n)
get	O(n)

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)
- System.arraycopy muss sehr effizient sein, so dass es ein Vorteil von ArrayList ist

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)
- System.arraycopy muss sehr effizient sein, so dass es ein Vorteil von ArrayList ist
- ArrayList hat den Performance-Parameter *initial capacity*

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)
- System.arraycopy muss sehr effizient sein, so dass es ein Vorteil von ArrayList ist
- ArrayList hat den Performance-Parameter *initial capacity*
- LinkedList ist schneller beim Einfügen vorne und Löschen in der Mitte

mercateo

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)
- System.arraycopy muss sehr effizient sein, so dass es ein Vorteil von ArrayList ist
- ArrayList hat den Performance-Parameter *initial capacity*
- LinkedList ist schneller beim Einfügen vorne und Löschen in der Mitte
- Collectors.toList() erstellt eine neue ArrayList

LinkedList vs ArrayList

- Laut Oracle Doku: ArrayList ist besser
- ArrayList mit konstantem Zugriff auf Positionen / Index
- ArrayList ist meist schneller: Performance testen, bevor man sich für LinkedList entscheidet
- LinkedList erzeugt pro Eintrag ein Node-Object (Overhead)
- System.arraycopy muss sehr effizient sein, so dass es ein Vorteil von ArrayList ist
- ArrayList hat den Performance-Parameter *initial capacity*
- LinkedList ist schneller beim Einfügen vorne und Löschen in der Mitte
- Collectors.toList() erstellt eine neue ArrayList

$\Rightarrow$ ArrayList

mercateo

LinkedList als Deque

Folie zuvor:

> *Deque Eigenschaften sind herausstechend: addFirst, getFirst, removeFirst, addLast, getLast und removeLast*

mercateo

LinkedList als Deque

Folie zuvor:

> *Deque Eigenschaften sind herausstechend: addFirst, getFirst, removeFirst, addLast, getLast und removeLast*

Aber Java Performance Tuning Guide sagt:

- Wenn man schnellen LinkedList Code schreiben möchte, muss man ListIterators verwenden

mercateo

LinkedList als Deque

Folie zuvor:

> *Deque Eigenschaften sind herausstechend: addFirst, getFirst, removeFirst, addLast, getLast und removeLast*

Aber Java Performance Tuning Guide sagt:

- Wenn man schnellen LinkedList Code schreiben möchte, muss man ListIterators verwenden
- Wenn Queue / Deque benötigt wird, lieber ArrayDeque als LinkedList

mercateo

LinkedList - Wann nehmen?

Wenn sehr oft remove(Object) genutzt wird oder

mercateo

LinkedList - Wann nehmen?

Wenn sehr oft remove(Object) genutzt wird oder
oft vorne Elemente eingefügt werden.

mercateo

LinkedList - Wann nehmen?

Wenn sehr oft remove(Object) genutzt wird oder
oft vorne Elemente eingefügt werden.
Kurz: **Eigentlich gar nicht. Besser ArrayList oder ArrayDeque**

mercateo

java.util.Map

mercateo

java.util.Map

Maps

java.util.Queue

java.util.Queuet

Queues

java.util.Set

mercateo

java.util.Set

Sets

Hilfsfunktionen

mercateo

Hilfsfunktionen in java.util.Collections

Hilfsfunktionen in Guava

Hilfsfunktionen in guava

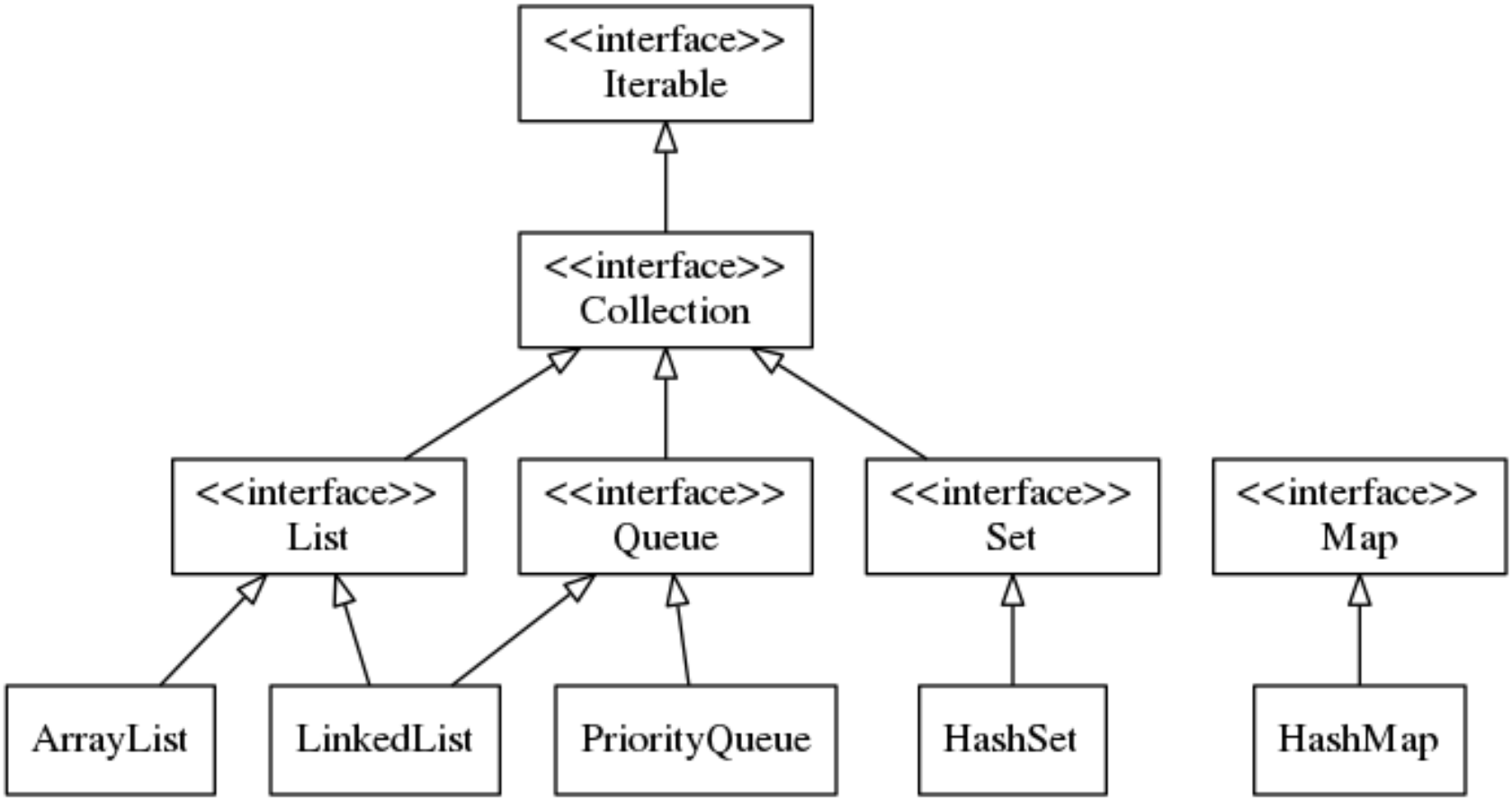
mercateo

Abschluss

Abschluss

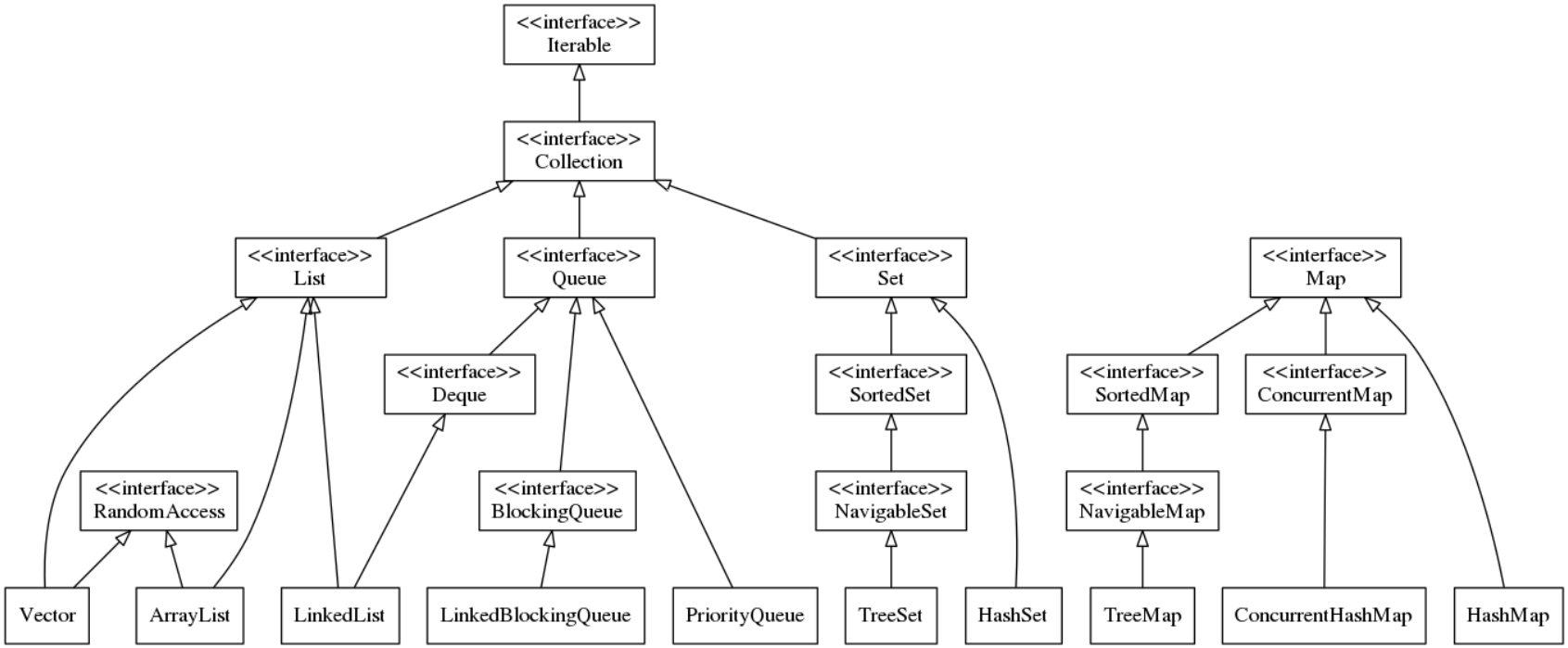
Warum das alles

Stark gekürzte Übersicht



mercateo

Leicht gekürzte Übersicht



mercateo