16.10. | 14

S01 – Collections in Java

Mathias Ritter 4AHIT

Dieses Dokument enthält eine Auflistung der Collections in Java (inkl. Maps) und deren Eigenschaften sowie Einsatzgebiete

Inhaltsverzeichnis

Collection

Eine Collection ist ein Objekt, das andere Objekte zu einer Einheit zusammenfasst. In einer Collection (die auch manchmal als "Container" bezeichnet wird) werden Daten gespeichert. Maps gehörten eigentlich nicht zu den Collections, da diese nicht vom Collection Interface erben. Trotzdem sind sie in dieser Auflistung enthalten, da sie ähnlich wie alle anderen Collections Daten speichern. [1]

Einsatzgebiet
Speicherung von Daten

Set

Ein Set ist eine Collection, welche keine doppelten Elemente enthält. Unter einem Set kann man sich eine aus der Mathematik bekannten Menge vorstellen. Damit auf ein Element eines Sets zugegriffen werden kann, muss es mit einem Iterator aus dem Set geholt werden. Der Vergleich, ob ein Element bereits vorhanden ist, wird mit euqals() durchgeführt, daher muss diese Methode eventuell überschrieben werden. [2]

Einsatzgebiet
Es sollen keine doppelten Elemente vorkommen

SortedSet

Ist ein Set, welches zusätzlich die Elemente nach ihrer natürlichen Reihenfolge oder mittels eines festgelegten Comparators sortiert. Die natürliche Reihenfolge wird durch die Methode compareTo des hinzugefügten Objekts bestimmt und sollte daher überschrieben werden. [3]

Einsatzgebiet
Die Elemente sollen sortiert vorliegen

Navigable Set

Ist ein SortedSet, welches über Methoden verfügt, die ein höheres oder niedrigeres als das angegebene Element aus dem Set zurückliefern. Existiert ein solches Element nicht, wird null zurückgeliefert. [4]

Einsatzgebiet
Ausgehend von einem Element soll schnell das nächstgrößere/nächstkleinere gefunden werden

HashSet

Ist ein Set, welches die Elemente in einer Hash-Tabelle speichert, wodurch ein schnellerer Zugriff ermöglicht wird. Voraussetzung ist eine korrekte Implementierung der hashCode() Methode, die überschrieben werden sollte. [5]

Einsatzgebiet
Auf die Elemente wird oft zugegriffen

LinkedHashSet

Ist ein Set, welches eine LinkedList in Kombination mit einem HashSet ist. Beim Einfügen der Elemente wird die Reihenfolge beibehalten und durch eine Hash-Tabelle wird ein schneller Zugriff ermöglicht. Details siehe HashSet bzw. LinkedList. [6]

Einsatzgebiet
Siehe HashSet und LinkedList

TreeSet

Ist ein Set, welches die Elemente wie ein SortedSet sortiert abspeichert. Die Elemente werden in einem Binärbaum (Baum mit Elementen, welche immer 2 Nachfolger besitzen) gespeichert. Implementiert das Interface SortedSet. [7]

Einsatzgebiet
Siehe SortedSet

EnumSet

Ist ein Set, welches ausschließlich Enum-Objekte beinhält. [8]

Einsatzgebiet
Enum-Objekte sollen in einem Set gespeichert werden

List

Eine List ist eine Collection, welche doppelte Elemente erlaubt. Es kann bestimmt werden, an welcher Stelle ein neues Element hinzugefügt werden soll. Auf die Elemente kann einfach mit der Angabe des Index (Position in der List) zugegriffen werden. Null-Elemente sind ebenfalls erlaubt. [9]

Einsatzgebiet
Es soll wahlfreier Zugriff auf die Elemente möglich sein

ArrayList

Ist eine List, welche ein in der Größe veränderbares Array darstellt. Im Hintergrund wird ein normales Array verwendet, welches bei Änderungen meist durch ein neues Array ersetzt werden muss. Es kann die Größe des im Hintergrund verwendeten Arrays auch manuell verändert werden. [10]

Einsatzgebiet:
Auf ein bestimmtes Element soll (über den Index) schnell zugegriffen werden

LinkedList

Ist eine List, welche die Elemente in einer verketteten Liste speichert. In einer verketteten Liste wird immer das aktuelle Element und ein Verweis auf das nächste Element gespeichert. Bei Änderungen muss der Verweis nur geändert werden. Bei der Suche nach einem bestimmten Element muss die LinkedList allerdings aufwändiger durchsucht werden. [11]

Einsatzgebiet:
Elemente werden oft hinzugefügt/entfernt

Stack

Ein Stack ist eine Collection, in der Elemente nach dem LIFO-Prinzip gespeichert werden (Last in first out). Man kann sich einen Stack wie einen Stapel vorstellen, auf den man immer nur oben ein Element legen kann bzw. entfernen kann. [12]

Einsatzgebiet:
Es soll immer das zuletzt hinzugefügte Element zuerst abgearbeitet werden

Queue

Eine Queue ist eine Collection, in der Elemente nach dem FIFO-Prinzip gespeichert werden (First in first out). Man kann sich eine Queue daher wie eine Warteschlange vorstellen, in der sich neue Elemente immer hinten einreihen und danach vom ersten Element ausgehend nach hinten abgearbeitet werden. Null-Elemente sollten nicht hinzugefügt werden, da diese meist bedeuten, dass die Queue leer ist.[13]

Einsatzgebiet
Es sollen Elemente in der selben Reihenfolge abgearbeitet werden, wie diese hinzugefügt wurden.

PriorityQueue

Ist eine Queue, in der die Elemente nach ihrer natürlichen Reihenfolge sortiert werden (Im Gegensatz zur Queue kein FIFO-Prinzip). Die natürliche Reihenfolge wird durch die Methode compareTo des hinzugefügten Objekts bestimmt und sollte daher überschrieben werden.[14]

Einsatzgebiet
Die hinzugefügten Elemente sollen nach ihrer Wichtigkeit abgearbeitet werden.

Deque

Eine Deque ist eine Collection, die der Queue ähnelt. Im Unterschied zur Queue können Elemente aber auch am vorderen Ende eingefügt werden und am hinteren Ende entfernt werden. Daher wird die Deque auch als „double ended queue“ bezeichnet. Die Größe einer Deque wird von manchen Deque-Implementierungen beschränkt. Null-Elemente sollten nicht verwendet werden, da diese meist bedeuten, dass die Deque leer ist.[15]

Einsatzgebiet
Von beiden Enden einer Schlange von Elementen sollen diese hinzugefügt/entfernt werden.

ArrayDeque

Ist eine Deque, die im Hintergrund ein Array verwendet. Diese Deque ist in ihrer Größe nicht beschränkt. Null-Elemente sind nicht erlaubt. Die ArrayDeuque ist eher schneller als ein Stack, wenn diese wie einer verwendet wird und schneller als eine LinkedList, wenn diese wie eine Queue verwendet wird.[16]

Einsatzgebiet
Siehe Deque

Map

Die Map gehört strenggenommen nicht zu den Collections, da sie ein eigenes Interface darstellt. In einer Map wird zu jedem Key eine Value gespeichert. Die Keys können nicht doppelt vorkommen und jedem Key wird maximal eine Value zugewiesen. Vorsichtig sollte man bei Key-Objekten sein, die sich nach dem Hinzufügen verändern können. Null Keys/Values werden von den Map-Implementierungen unterschiedlich gehandhabt. Zum Vergleich zweier Keys/Values wird euqals() und hashCode() verwendet.[17]

Einsatzgebiet
Zu einem bestimmten Schlüssel (Key) soll ein Wert (Value) gespeichert werden.

SortedMap

Ist eine Map, in der die Keys nach ihrer natürlichen Reihenfolge sortiert werden. Die natürliche Reihenfolge wird durch die Methode compareTo des hinzugefügten Objekts bestimmt und sollte daher überschrieben werden. Die Reihenfolge der Elemente wird auch beibehalten, wenn auf diese mit einem Iterator zugegriffen wird.[18]

Einsatzgebiet
Die Keys sollen in sortierter Reihenfolge vorliegen

HashMap

Ist eine Map, welches die Elemente in einer Hash-Tabelle speichert, wodurch ein schnellerer Zugriff ermöglicht wird. Die Reihenfolge der Elemente kann sich jederzeit ändern. Voraussetzung ist eine Korrekte Implementierung der hashCode() Methode, die überschrieben werden sollte.[19]

Einsatzgebiet
Auf die Elemente soll schnell zugegriffen werden

LinkedHashMap

Kombiniert eine HashMap mit einer LinkedList. Der Vorteil gegenüber einer HashMap ist, dass die Reihenfolge der Elemente konstant bleibt. Die Reihenfolge richtet sich das der Einfügereihenfolge der Elemente. Der Zugriff auf die Elemente ist allerdings etwas langsamer.[20]

Einsatzgebiet
Die Reihenfolge der Elemente soll beibehalten werden

IndentityHashMap

Ist eine HashMap, welche allerdings im Gegensatz zur normalen HashMap den „==“-Operator zum Vergleich zweier Keys oder Values verwendet. Das heißt diese Map kann auch Keys enthalten, die zwar laut equals() gleich sind, aber nicht identisch sind.[21]

Einsatzgebiet
Es sollen Keys gespeichert werden, die gleich, aber nicht ident sind.

WeakHashMap

Ist eine HashMap, die sogenannte „weak keys“ enthält. Das heißt, dass die Keys schwache Referenzen auf andere Objekte sind. Wird das Objekt gelöscht, wird die schwache Referenz und somit der Key ebenfalls vom Garbage Collector gelöscht.[22]

Einsatzgebiet
Objekte, die nicht mehr existieren, sollen automatisch auch aus der Map entfernt werden

EnumMap

Ist eine Map, deren Keys Enum-Objekte sind. Die Reihenfolge der Keys ist die Reihenfolge, in der die Enum-Constants deklariert wurden.[23]

Einsatzgebiet
Zu Enum-Keys soll ein bestimmter Wert gespeichert werden

NavigableMap

Ist eine Map, die wie eine SortedMap die Keys in sortierter Reihenfolge beinhält. Zusätzlich kann einfach das höchste/nächsthöhere/niedrigere/niedrigste Element gefunden werden, da entsprechende Methode verfügbar sind.[24]

Einsatzgebiet
Ausgehend von einem Key soll schnell ein höherer/niedrigerer Key gefunden werden

TreeMap

Ist eine Map, die die NavigableMap erweitert. Die Elemente werden in einem Binärbaum (Baum mit Elementen, welche immer 2 Nachfolger besitzen) gespeichert. Implementiert das Interface SortedMap.[25]

Einsatzgebiet
siehe NavigableMap und SortedMap

Concurrent Collections

Die Concurrent Collections sind darauf ausgelegt, dass mehrere Threads auf eine Collection zugreifen können, ohne das dies zu Thread-Interferenzen oder Speicher-Inkonsistenzen führt.[48]
Dieser Vorteil gilt, obwohl nicht jedes Mal erwähnt, für alle unten beschriebenen Collections.

Set

CopyOnWriteArraySet

Ist ein Set, das im Hintergrund eine CopyOnWriteArrayList verwendet. Bei jeder Veränderung wird das komplette im Hintergrund verwendete Array kopiert. Daher sind Lesezugriffe schnell, während alle Schreibzugriffe langsam sind. Ein Iterator kann das Set nicht modifizieren.[26]

Einsatzgebiet
Es finden viele lesende Zugriffe statt

ConcurrentSkipListSet

Ist ein NavigableSet, welches auf einer ConcurrentSkipListMap basiert. Die Elemente werden nach ihrer natürlichen Reihenfolge oder mittels eines festgelegten Comparators sortiert. Daher sollte die Methode compareTo der hinzugefügten Objekte überschrieben werden. Die size-Methode sollte nicht nebenläufig aufgerufen werden, da diese bei nebenläufigen Modifikationen falsche Ergebnisse zurückgibt.[27]

Einsatzgebiet
Siehe ConcurrentSkipListMap
Es finden viele schreibende Zugriffe statt

List

Vector

Entspricht einer synchronisierten ArrayList. Der Itaretor wirft eine ConcurrentModificationException wenn nach dem Initialisieren des Iterators die Collection modifiziert wird.[28]

Einsatzgebiet
Siehe ArrayList

CopyOnWriteArrayList

Ist eine ArrayList. Bei jeder Veränderung wird das im Hintergrund verwendete Array komplett kopiert. Daher sind Lesezugriffe schnell, während alle Schreibzugriffe langsam sind. Bei Lesezugriffen wird sichergestellt, dass vorher nebenläufige Schreibzugriffe abgeschlossen sind.[29]

Einsatzgebiet
Siehe ArrayList
Es finden viele lesende Zugriffe statt

Queue

BlockingQueue

Ist eine Queue, die blockiert (wartet), wenn beim Entfernen keine Elemente vorhanden sind oder wenn beim Hinzufügen die Queue voll ist. Es kann auch eine maximale Zeit angegeben werden, die die Queue maximal blockieren(warten) soll. Alle Operationen werden atomar ausgeführt, da sie für alle Operationen interne Locks verwenden. Die Methoden addAll, containsAll, retainAll und removeAll sind davon ausgenommen.[30]

Einsatzgebiet
Es sollen ein oder mehrere Konsumenten/Produzenten einfach auf die Queue zugreifen sollen

ArrayBlockingQueue

Ist eine BlockingQueue, die im Hintergrund mit einem Array arbeitet. Daher gibt es eine maximale Anzahl an Elementen, die vorher festgelegt werden muss.[31]

Einsatzgebiet
Die BlockingQueue soll auf eine maximale Anzahl an Elementen eingeschränkt werden

LinkedBlockingQueue

Ist eine BlockingQueue, die im Hintergrund verketteten Elementen arbeitet. Die Anzahl der maximalen Elemente ist daher nicht beschränkt (eigentlich Integer.MAX_VALUE).[32]

Einsatzgebiet
Die BlockingQueue soll nicht auf eine maximale Anzahl an Elementen eingeschränkt werden.

PriorityBlockingQueue

Kombiniert eine PriorityQueue mit einer BlockingQueue.[33]

Einsatzgebiet
Siehe BlockingQueue und PriorityQueue

TransferQueue

Ist eine BlockingQueue. Zusätzlich zur BlockingQueue kann eine Queue auch so lange blockieren, bis ein hinzugefügtes Element wieder aus der Queue herausgenommen wird.[34]

Einsatzgebiet
Es soll sichergestellt werden, dass ein Element vom Produzent auch wirklich beim Konsument angekommen ist. Bis zu diesem Zeitpunkt soll gewartet werden.

LinkedTransferQueue

Implementiert die TransferQueue und arbeitet im Hintergrund mit einer LinkedList.[35]

Einsatzgebiet
Siehe TransferQueue

DelayQueue

Ist eine BlockingQueue, deren Elemente nur nach einer bestimmten Zeitverzögerung (Delay) entnommen werden können. Ganz vorne in der Queue befindet sich das Element, dessen Zeitverzögerung bereits am längsten abgelaufen ist. Falls kein Element vorliegt, dessen Zeitverzögerung abgelaufen ist, so enthält die Queue keine Elemente. Null-Elemente sind nicht erlaubt, da sie eine leere Queue bedeuten.[36]

Einsatzgebiet
Ein Konsument soll ein Element erst nach einer bestimmten Zeitverzögerung erhalten und nicht sofort, wenn es verfügbar ist.

SynchronousQueue

Ist eine BlockingQueue, deren Elemente immer nur entnommen werden können, wenn ein anderes hinzugefügt wird und umgekehrt. Der „Head“ der Queue ist daher das Element, das als erstes versucht wird, hinzuzufügen. Diese Queue verhält sich wie eine leere Queue, da die Elemente immer direkt nachdem sie platziert wurden konsumiert werden.[37]

Einsatzgebiet
Es sollen keine Elemente in der Queue stehen bleiben, sondern diese sollen immer auch sofort wieder herausgenommen werden

ConcurrentLinkedQueue

Ist eine Queue, die im Hintergrund mit verlinkten Elementen (ähnlich wie eine LinkedList) arbeitet. Im Gegensatz zur BlockingQueue blockiert diese Queue nicht, daher kann es zu einer Exception kommen, wenn z.B. aus einer leeren Queue ein Element entnommen werden soll. Daher sollte vor dem Entnehmen von Elementen geprüft werden, ob die Queue nicht leer ist.[38]

Einsatzgebiet
Wie Queue, nur zusätzlich thread-sicher

Deque

BlockingDeque

Ist eine Deque, die ähnlich wie eine BlockingQueue funktioniert (siehe BlockingQueue). Im Gegensatz zur BlockingQueue können Elemente von beiden Enden entfernt/hinzugefügt werden.[39]

Einsatzgebiet
Siehe Deque und BlockingQueue

LinkedBlockingDeque

Implementiert eine BlockingDequel, die ähnlich wie eine LinkedBlockingQueue funktioniert (siehe LinkedBlockingQueue).[40]

Einsatzgebiet
Siehe Deque und LinkedBlockingQueue

ConcurrentLinkedDeque

Ist eine Deque, die ähnlich wie eine ConcurrentLinkedQueue funktioniert (siehe ConcurrentLinkedQueue). Im Gegensatz zur ConcurrentLinkedQueue können Elemente von beiden Enden entfernt/hinzugefügt werden.[41]

Einsatzgebiet
Siehe Deque und ConcurrentLinkedQueue

Concurrent Maps

Die Concurrent Maps sind darauf ausgelegt, dass mehrere Threads auf eine Map zugreifen können, ohne das dies zu Thread-Interferenzen oder Speicher-Inkonsistenzen führt.[48]
Dieser Vorteil gilt, obwohl nicht jedes Mal erwähnt, für alle unten beschriebenen Maps.

HashTable

Ist eine Map, die ähnlich wie eine HashMap funktioniert (siehe HashMap oben). Die HashTable ist zwar synchronized, allerdings sollte bei vielen nebenläufigen Zugriffen aus Performancegründen die ConcurrentHashMap verwenden.[42]

Einsatzgebiet
Siehe HashMap

ConcurrentMap

Siehe Map. Bei Lesezugriffen wird sichergestellt, dass vorher nebenläufige Schreibzugriffe abgeschlossen sind.[43]

Einsatzgebiet
Siehe Map

ConcurrentNavigableMap

Siehe NavigableMap.[44]

Einsatzgebiet
Siehe NavigableMap

ConcurrentSkipListMap

Siehe SkipListMap. Die size-Methode kann bei einem nebenläufigen Aufruf einen falschen Wert zurückgeben, wenn die Map gerade modifiziert wird.[45]

Einsatzgebiet
Siehe SkipListMap

ConcurrentHashMap

Siehe HashMap. Iterator werfen zwar keine ConcurrentModificationException, sollten aber trotzdem nicht nebenläufig verwendet werden.[46]

Einsatzgebiet
Siehe HashMap

Synchronization Wrappers

Mit Hilfe von Synchronization Wrappers können thread-sichere Collections (und Maps) initialisiert werden. Der Synchronization Wrapper fügt zu der angegebenen Collection (oder Map) die Synchronisierung automatisch hinzu.

Beim Verwenden des Iterators muss dieser von einem synchronized-Block umschlossen werden. Damit werden Modifikationen während der Verwendung des Iterators ausgeschlossen. Verzichtet man auf die Synchronisierung, so kann eine ConcurrentModificationException geworfen werden.[47]

Folgende Synchronization Wrappers stehen zur Verfügung

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

Eine synchronisierte Collection kann z.B. folgendermaßen initialisiert werden:

```
List<Type> list = Collections.synchronizedList(new ArrayList<Type>());
```

Im Gegensatz zu den meisten Concurrent Collections (und Concurrent Maps) wird auf die gesamte Collection/Map synchronisiert, dadurch sind diese nicht so performant.[47]

Unmodifiable Wrappers

Im Gegensatz zu den Synchronization Wrappers, die eine Collection um die Synchronisierung erweitern, werden durch die Unmodifiable Wrappers Funktionen wieder entfernt. Genauer gesagt heißt das, dass die Collection dann nicht mehr modifizierbar ist. Versucht man trotzdem, die Collection zu modifizieren, wird eine UnsupportedOperationException geworfen.[47]

Folgende Unmodifiable Wrappers stehen zur Verfügung

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

Quellverzeichnis

[1] Oracle (1995, 2014): Introduction of Collections [Online]. Verfügbar unter: http://docs.oracle.com/javase/tutorial/collections/intro/index.html [abgerufen am 2.10.2014]

[2] Oracle (1993, 2014): Java Platform SE 8 API - Set [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Set.html [abgerufen am 11.10.2014]

[3] Oracle (1993, 2014): Java Platform SE 8 API - SortedSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html [abgerufen am 11.10.2014]

[4] Oracle (1993, 2014): Java Platform SE 8 API - NavigableSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/NavigableSet.html [abgerufen am 11.10.2014]

[5] Oracle (1993, 2014): Java Platform SE 8 API - HashSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html [abgerufen am 11.10.2014]

[6] Oracle (1993, 2014): Java Platform SE 8 API - LinkedHashSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html [abgerufen am 11.10.2014]

[7] Oracle (1993, 2014): Java Platform SE 8 API – TreeSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html [abgerufen am 11.10.2014]

[8] Oracle (1993, 2014): Java Platform SE 8 API - EnumSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html [abgerufen am 11.10.2014]

[9] Oracle (1993, 2014): Java Platform SE 8 API - List [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/List.html [abgerufen am 11.10.2014]

[10] Oracle (1993, 2014): Java Platform SE 8 API - ArrayList [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html [abgerufen am 11.10.2014]

[11] Oracle (1993, 2014): Java Platform SE 8 API - LinkedList [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html [abgerufen am 11.10.2014]

[12] Oracle (1993, 2014): Java Platform SE 8 API - Stack [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Stack.html [abgerufen am 11.10.2014]

[13] Oracle (1993, 2014): Java Platform SE 8 API - Queue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Queue.html [abgerufen am 11.10.2014]

[14] Oracle (1993, 2014): Java Platform SE 8 API – PriorityQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html [abgerufen am 11.10.2014]

[15] Oracle (1993, 2014): Java Platform SE 8 API – Deque [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Deque.html [abgerufen am 11.10.2014]

[16] Oracle (1993, 2014): Java Platform SE 8 API – ArrayDeque [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html [abgerufen am 11.10.2014]

[17] Oracle (1993, 2014): Java Platform SE 8 API – Map [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Map.html [abgerufen am 11.10.2014]

[18] Oracle (1993, 2014): Java Platform SE 8 API – SortedMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/SortedMap.html [abgerufen am 11.10.2014]

[19] Oracle (1993, 2014): Java Platform SE 8 API – HashMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html [abgerufen am 11.10.2014]

[20] Oracle (1993, 2014): Java Platform SE 8 API – LinkedHashMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html [abgerufen am 11.10.2014]

[21] Oracle (1993, 2014): Java Platform SE 8 API – IdentityHashMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/IdentityHashMap.html [abgerufen am 11.10.2014]

[22] Oracle (1993, 2014): Java Platform SE 8 API – WeakHashMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/WeakHashMap.html [abgerufen am 11.10.2014]

[23] Oracle (1993, 2014): Java Platform SE 8 API – EnumMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/EnumMap.html [abgerufen am 11.10.2014]

[24] Oracle (1993, 2014): Java Platform SE 8 API – NavigableMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/NavigableMap.html [abgerufen am 11.10.2014]

[25] Oracle (1993, 2014): Java Platform SE 8 API – TreeMap [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html [abgerufen am 11.10.2014]

[26] Oracle (1993, 2014): Java Platform SE 8 API – CopyOnWriteArraySet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArraySet.html [abgerufen am 12.10.2014]

[27] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentSkipListSet [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentSkipListSet.html [abgerufen am 12.10.2014]

[28] Oracle (1993, 2014): Java Platform SE 8 API – Vector [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Vector.html [abgerufen am 12.10.2014]

[29] Oracle (1993, 2014): Java Platform SE 8 API – CopyOnWriteArrayList [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html [abgerufen am 12.10.2014]

[30] Oracle (1993, 2014): Java Platform SE 8 API – BlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html [abgerufen am 12.10.2014]

[31] Oracle (1993, 2014): Java Platform SE 8 API – ArrayBlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html [abgerufen am 12.10.2014]

[32] Oracle (1993, 2014): Java Platform SE 8 API – LinkedBlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html [abgerufen am 12.10.2014]

[33] Oracle (1993, 2014): Java Platform SE 8 API – PriorityBlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/PriorityBlockingQueue.html [abgerufen am 12.10.2014]

[34] Oracle (1993, 2014): Java Platform SE 8 API – TransferQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TransferQueue.html [abgerufen am 12.10.2014]

[35] Oracle (1993, 2014): Java Platform SE 8 API – LinkedTransferQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedTransferQueue.html [abgerufen am 12.10.2014]

[36] Oracle (1993, 2014): Java Platform SE 8 API – DelayQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/DelayQueue.html [abgerufen am 12.10.2014]

[37] Oracle (1993, 2014): Java Platform SE 8 API – SynchronousQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html [abgerufen am 12.10.2014]

[38] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentLinkedQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html [abgerufen am 12.10.2014]

[39] Oracle (1993, 2014): Java Platform SE 8 API – BlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingDeque.html [abgerufen am 12.10.2014]

[40] Oracle (1993, 2014): Java Platform SE 8 API – LinkedBlockingQueue [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingDeque.html [abgerufen am 12.10.2014]

[41] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentLinkedDeque [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedDeque.html [abgerufen am 12.10.2014]

[42] Oracle (1993, 2014): Java Platform SE 8 API – Hashtable [Online]. Verfügbar unter: http://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html [abgerufen am 16.10.2014]

[43] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentMap [Online]. Verfügbar unter: http://download.java.net/lambda/b78/docs/api/java/util/concurrent/ConcurrentMap.html [abgerufen am 16.10.2014]

[44] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentNavigableMap [Online]. Verfügbar unter: http://download.java.net/lambda/b78/docs/api/java/util/concurrent/ConcurrentNavigableMap.html [abgerufen am 16.10.2014]

[45] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentSkipListMap [Online]. Verfügbar unter: http://download.java.net/lambda/b78/docs/api/java/util/concurrent/ConcurrentSkipListMap.html [abgerufen am 16.10.2014]

[46] Oracle (1993, 2014): Java Platform SE 8 API – ConcurrentHashMap [Online]. Verfügbar unter: http://download.java.net/lambda/b78/docs/api/java/util/concurrent/ConcurrentHashMap.html [abgerufen am 16.10.2014]

[47] Oracle (1995, 2014): Wrapper Implementations [Online]. Verfügbar unter: http://docs.oracle.com/javase/tutorial/collections/implementations/wrapper.html [abgerufen am 16.10.2014]

[48] Oracle (1995, 2014): Concurrent Collections [Online]. Verfügbar unter: http://docs.oracle.com/javase/tutorial/essential/concurrency/collections.html [abgerufen am 16.10.2014]