Praktische Datenstrukturen in Java
Eine Einführung

Algorithmen und Datenstrukturen 1
VU 186.813, 4h, 6 ECTS, SS 2016
Letzte Änderung: 4. Mai 2016

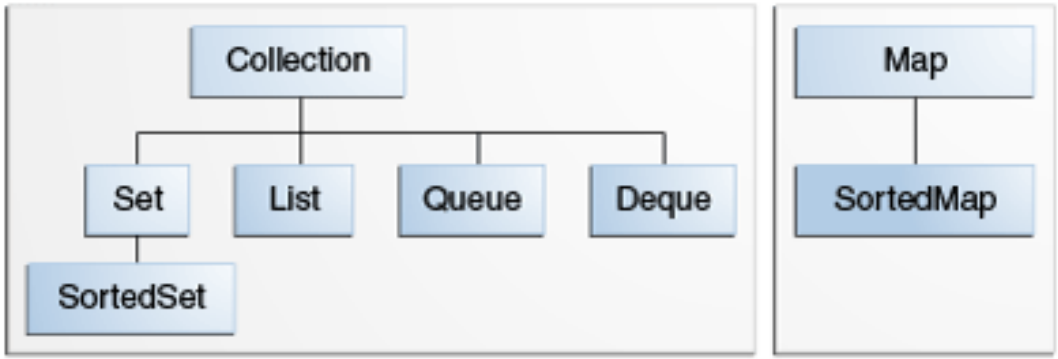
Überblick

Themen:

- Collections-Framework – Überblick
- Ausgewählte Klassen
- Algorithmen im Collections-Framework
- Weitere Beispiele für Bibliotheken

Java Collections-Framework - Überblick

Grundlagen



Interfaces:

- Für unterschiedliche Datenstrukturen stehen unterschiedliche Interfaces zur Verfügung.
- Geben die Schnittstellen für den Zugriff auf die Datenstrukturen an.
- Konkrete Implementierungen realisieren diese Interfaces.
- Seit Java 5 generisch implementiert.

Beschreibung: `https://docs.oracle.com/javase/8/docs/technotes/guides/collections/`

Interfaces

Collection:

- Eine Collection verwaltet Objekte (Elemente).
- Interface enthält generelle Methoden (für alle Collections-Typen).
- Es gibt keine direkte Implementierung dieses Interfaces.

Set:

- Eine Collection die keine duplizierten Elemente enthält.

List:

- Eine geordnete Collection (mit duplizierte Elementen).
- Elemente können über einen Index angesprochen werden (nicht immer effizient).

Queue / Deque:

- Verwalten von Warteschlangen.
- FIFO, Prioritätswarteschlangen.
- Einfügen und Löschen an beiden Enden bei Deque.

Interfaces

Map:

- Maps verwalten Schlüssel mit dazugehörigen Werten.

SortedSet und SortedMap:

- Sind spezielle Versionen von Set und Map, bei denen die Elemente (Schlüssel) in aufsteigender Reihenfolge verwaltet werden.

Generelle Implementierungen (Beispiele)

Tabelle: Implementierungen.

Interface	**Hashtabellen**	**Dynamische Arrays**	**Bäume**	**Verkettete Listen**	**Hashtabellen + verkettete Listen**
Set	HashSet		TreeSet		Linked-HashSet
List		ArrayList		LinkedList	
Queue		ArrayDeque		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		Linked-HashMap

Anmerkung: Interfaces SortedSet und SortedMap werden von TreeSet bzw. TreeMap zusätzlich implementiert.

Ausgewählte Klassen

Listenimplementierungen

ArrayList:

- Indexzugriff auf Elemente ist überall gleich schnell ($O(1)$).
- Einfügen und Löschen ist am Listenende schnell und wird mit wachsender Entfernung vom Listenende langsamer ($O(n)$).

LinkedList:

- Indexzugriff auf Elemente ist an den Enden schnell und wird mit der Entfernung von den Enden langsamer ($O(n)$).
- Einfügen und Löschen ohne Indexzugriff ist überall gleich schnell ($O(1)$).
- Ansonsten abhängig vom Indexzugriff.

Set-Implementierungen

HashSet:

- null-Elemente sind zulässig.
- Einfügen, Suchen und Löschen sind in konstanter Zeit möglich (abhängig von der Verteilung der Einträge in der Hashtabelle).
- Aber: Rehashing kann zu Performance-Problemen führen (siehe auch API-Beschreibung).

TreeSet:

- null-Elemente sind nicht erlaubt.
- Die Laufzeit von Einfügen, Suchen und Löschen liegt bei n Elementen in $O(\log n)$.
- Auf die Elemente eines TreeSets muss eine Ordnung definiert sein (müssen vergleichbar sein, d.h. das Interface Comparable implementieren).

Beispiel für Anwendung

Demo: ArrayList, Linkedlist, HashSet

ArrayList: `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

LinkedList: `https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html`

HashSet: `https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html`

Maps

Maps:

- Maps sind eine Verallgemeinerung von Arrays mit einem beliebigen Indextyp (nicht nur int).
- Eine Map ist eine Menge von Schlüssel-Werte Paaren.
  - Schlüssel müssen innerhalb einer Map eindeutig sein.
  - Werte müssen nicht eindeutig sein.

Arten von Maps: Wie bei Sets gibt es grundsätzlich zwei Versionen.

- HashMap: Ungeordnet, basiert auf Hashing.
- TreeMap: Geordnet, basiert auf Rot-Schwarz-Bäumen (balancierte Bäume).

Operationen auf Maps: Bestimmte Methoden wie z.B. put, get, containsKey, containsValue, remove etc. werden angeboten.

Beispiel für Anwendung

Demo: HashMap

HashMap: `https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html`

TreeMap: `https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html`

Iteratoren

Iteratoren:

- Iteratoren sind eine Verallgemeinerung von Indexwerten.
- Werden ähnlich wie Indexwerte verwendet, ihr innerer Aufbau bleibt aber verborgen.
- Iteratoren haben den gleichen Elementtyp wie die zugrunde liegende Collection.

Verwendung:

- Iterator wird durch eine Methode (z.B. iterator) erzeugt.
- Ein Iterator bietet folgende Methoden an:
  - `hasNext` – testet, ob es weitere Elemente gibt (true) oder nicht (false).
  - `next` – liefert das nächste Element und rückt den Iterator gleichzeitig um ein Element weiter, d.h. aufeinanderfolgende Aufrufe von next liefern immer neue Elemente.
  - `remove` (optional) – entfernt das zuletzt zurückgegebene Element.

Beispiel für Iteratoren

Demo: Beispiele für Iteratoren.

Iterator: `https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html`

Abstrakte Klassen

Abstrakte Klassen:

- Unterstützen die Entwicklung neuer Klassen im Framework.
- Implementieren schon einen großen Teil eines Interfaces und lassen bestimmte Teile noch offen.

Neue Klasse implementieren:

- Auswählen einer geeigneten abstrakten Klasse von der geerbt wird.
- Implementierung aller abstrakten Methoden.
- Sollte die Collection modifizierbar sein, dann müssen auch einige konkrete Methoden überschrieben werden (siehe API).
- Testen der neuen Klasse (inklusive Performance).

Abstrakte Klassen

Beispiele:

- AbstractCollection
- AbstractSet
- AbstractList (basierend auf Array)
- AbstractSequentialList (basierend auf verketteter Liste)
- AbstractQueue
- AbstractMap

API: API-Dokumentation jeder abstrakten Klasse beschreibt genau, wie man eine Klasse ableiten muss:

- Grundlegende Implementierung.
- Welche Methoden müssen implementiert werden.
- Welche Methoden müssen überschrieben werden, wenn man Modifikationen zulassen möchte.

Algorithmen im Collections-Framework

Algorithmen im Collections-Framework

Algorithmen:

- Das Collections-Framework bietet auch Algorithmen für die Verarbeitung von Container-Klassen an.
- Diese Algorithmen werden als statische Methoden (polymorphe Methoden) in der Hilfsklasse Collections gesammelt.

Beispiele:

- sort sortiert die Elemente einer generischen Liste nach aufsteigender Größe.
- binarySearch sucht ein Element in der sortierten Liste (Voraussetzung) und liefert einen Index zurück, wenn das Element gefunden wurde (ansonsten eine negative Zahl).
- max liefert das größte Element einer Collection.
- shuffle mischt die Elemente einer generischen Liste zufällig.

Algorithmen im Collections-Framework: Beispiel sort

Sortieren von Collections:

- Iterativer Merge-Sort mit Insertion-Sort kombiniert (TimSort).
- Laufzeit abhängig von der Inputsequenz
  - Best Case: $O(n)$ – wenn Liste schon sortiert.
  - Average- und Worst-Case: $O(n \log n)$.
- Zusätzlicher Speicher
  - Best Case: $O(1)$.
  - Average- und Worst-Case: $O(n)$.

Algorithmen im Collections-Framework: Beispiel sort

Sortieren in Java:

- TimSort wird nur auf Collections verwendet.
- Für das Sortieren von Arrays mit primitiven Datentypen (z.B. int) ist die Methode sort in der Hilfsklasse Arrays vorgesehen.
- Methode sort in der Klasse Arrays verwendet eine spezielle Quicksort-Variante.

Sortieren: Stabilität

Stabiles Sortierverfahren: Ein stabiles Sortierverfahren ist ein Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.

Beispiel:

- Liste von Mitarbeiterdaten mit Abteilungsnummer (innerhalb der Abteilung sortiert):
  3 - Daniel, 4 - Maria, 2 - Paul, 3 - Erika, 1 - Anton, 3 - Sarah
- Sortierung nach Abteilungsnummer mit stabilem Sortierverfahren:
  1 - Anton, 2 - Paul, 3 - Daniel, 3 - Erika, 3 - Sarah, 4 - Maria

Sortieren: Stabilität

Ausgangssequenz vereinfacht: 3 4 2 3 1 3

Beispiel Mergesort (verkürzt): Ist stabil

3 4 2 3 1 3
3 4 2 3 1 3
3 4 2 3 1 3
...
2 3 4 1 3 3
1 2 3 3 3 4

Beispiel Selectionsort (verkürzt): nicht stabil

3 4 2 3 1 3
1 4 2 3 3 3
Stabilität nicht mehr gegeben ...

Stabilität: Algorithmen in dieser Vorlesung

Beispiele für stabile Sortierverfahren:

- Insertionsort
- Mergesort

Hinweis: Korrekte Implementierung erforderlich!

Beispiele für instabile Sortierverfahren:

- Selectionsort
- Quicksort

Weitere Bibliotheken

Weitere Bibliotheken - Eclipse Collections

Erweiterungen gegenüber Java:

- Optimierte Sets und Maps
- Immutable Collections
- Collections für primitive Datentypen
- Multimaps, Bimaps
- Verschiedene Iterationsstile

Link: `https://github.com/eclipse/eclipse-collections`

Weitere Beispiele für Bibliotheken

Google Guava

- `https://github.com/google/guava`
- Unterstützt z.B. Multisets, Multimaps, Bimaps

Apache Commons Collections

- `https://commons.apache.org/proper/commons-collections/`

Brownies Collections

- `http://www.magicwerk.org/page-collections-overview.html`
- z.B. High Performance Listen (GapList, BigList)

JGraphT

- `https://github.com/jgrapht/jgrapht`
- Algorithmen und Datenstrukturen für Graphen