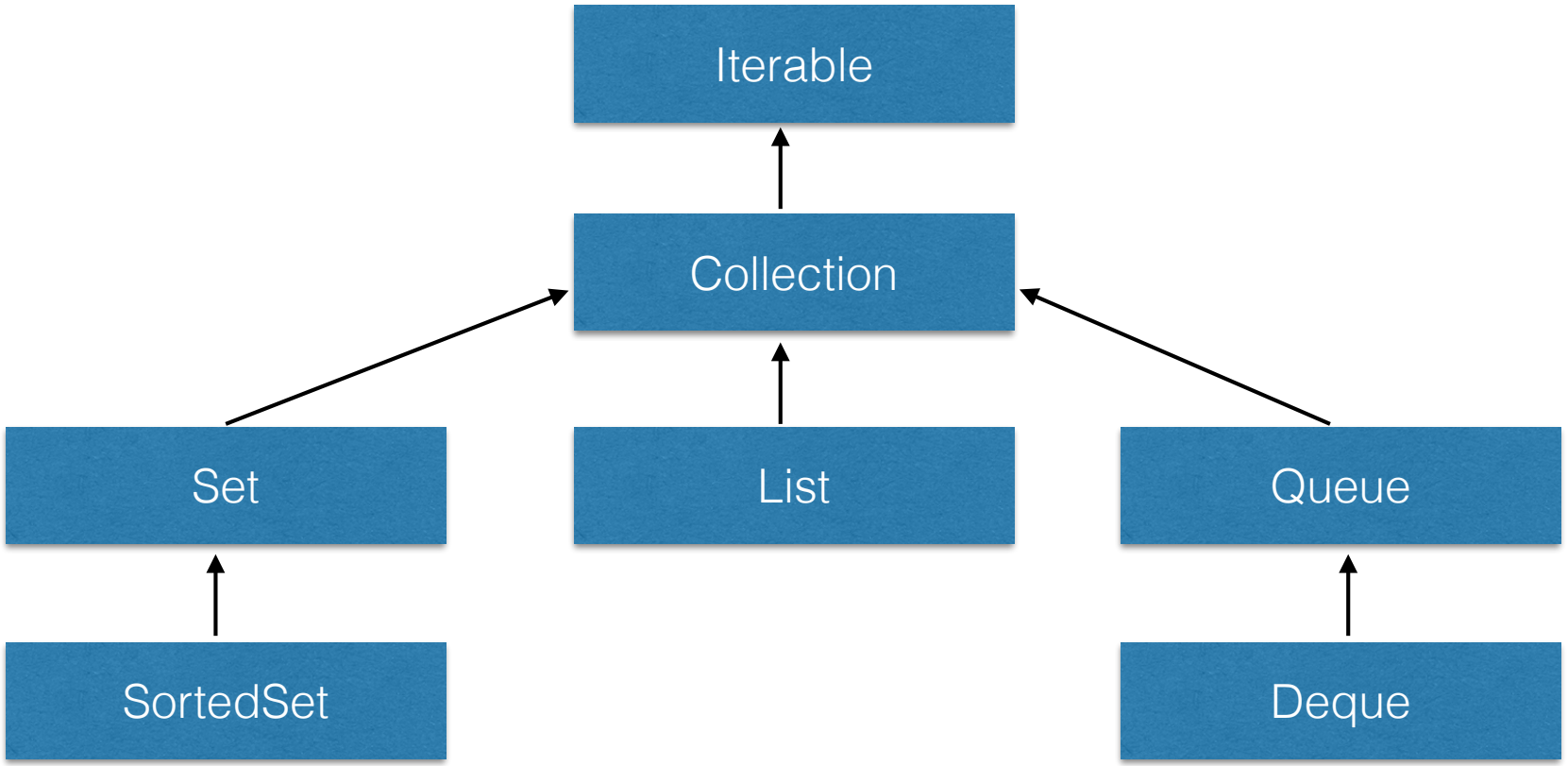
Datenstrukturen

Collections

Collections (Interfaces)



Collection

- List
  - erlaubt Duplikate und null
  - behält die Reihenfolge
- Set
  - erlaubt keine Duplikate
- Queue
  - FIFO
- Deque
  - Erlaubt das Hinzufügen von Elementen an beiden Seiten

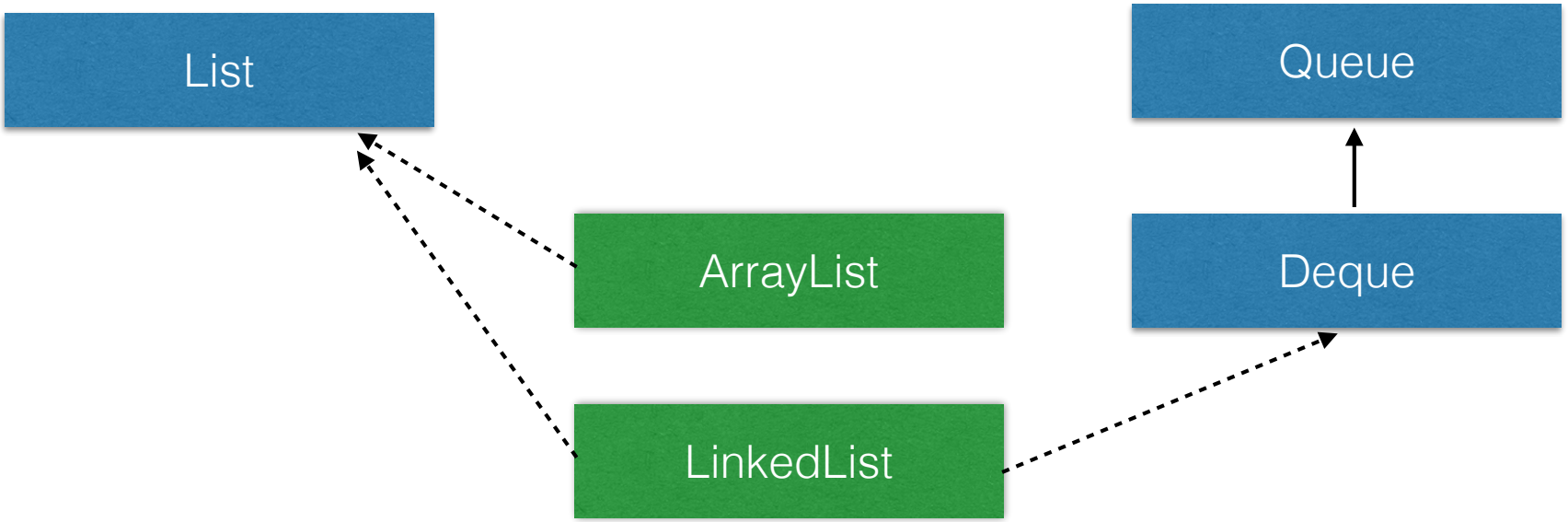
Collection Interface

- **Modifizieren**
  - Add
    - add()
    - addAll()
  - Remove
    - clear()
    - remove()
    - removeAll()
    - retainAll()
- **Anfragen**
  - contains()
  - equals()
  - isEmpty()
  - size()
- **Sonstiges**
  - iterator()
  - toArray()

List

- Elemente werden mit einem Index ausgestattet
- Methoden von Collection werden überladen um die Arbeit mit dem Index zu ermöglichen
- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Lists (Implementations)



ArrayList

- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- Verwendet intern ein Array
- bei add() an vorhandenen Positionen verschieben sich vorhandene Elemente nach rechts
- bei remove() verschieben sich vorhandene Elemente nach links
- Gleichheit wird über equals() sichergestellt

LinkedList

- https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html
- Listenelemente in Verbindung zum jeweiligen Vorgänger bzw. Nachfolger
- Elemente können schneller hinzugefügt und gelöscht werden
- Wahlfreier Zugriff langsamer
- Implementiert List und Deque

Set

- https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

HashSet

- schnelle hashbasierte Datenstruktur
- Elemente werden in einer Hash-Tabelle gespeichert
- keine Methoden, um auf ein Objekt innerhalb des Sets auf direktem Wege zuzugreifen

Hash

- Vergleiche werden nach dem Hashwert und der Gleichheit durchgeführt werden, nicht aber nach der Identität

equals und hashCode

- wenn equals(…) von zwei Objekten gleichen Typs true ergibt, muss der Hashwert auch gleich sein
- wenn zwei Hashwerte ungleich sind, darf equals(…) nicht wahr ergeben

clone und "flache" Kopie

- Die Kopie bezieht sich nur auf den Assoziativspeicher selbst; die Schlüssel- und Wert-Objekte teilen sich Original und Klon

TreeSet

- Alle Elemente die eingefügt werden sollen, müssen das Interface Comparable und dessen Methode compareTo implementieren
- Suche nach einem einzigen Element ist aber etwas langsamer als im HashSet
- Einfüge-/Löschzeiten schlechter als im HashSet (Reorganisation des Baums)
- TreeSet<String> set = new TreeSet<>(String.CASE_INSENSITIVE_ORDER);

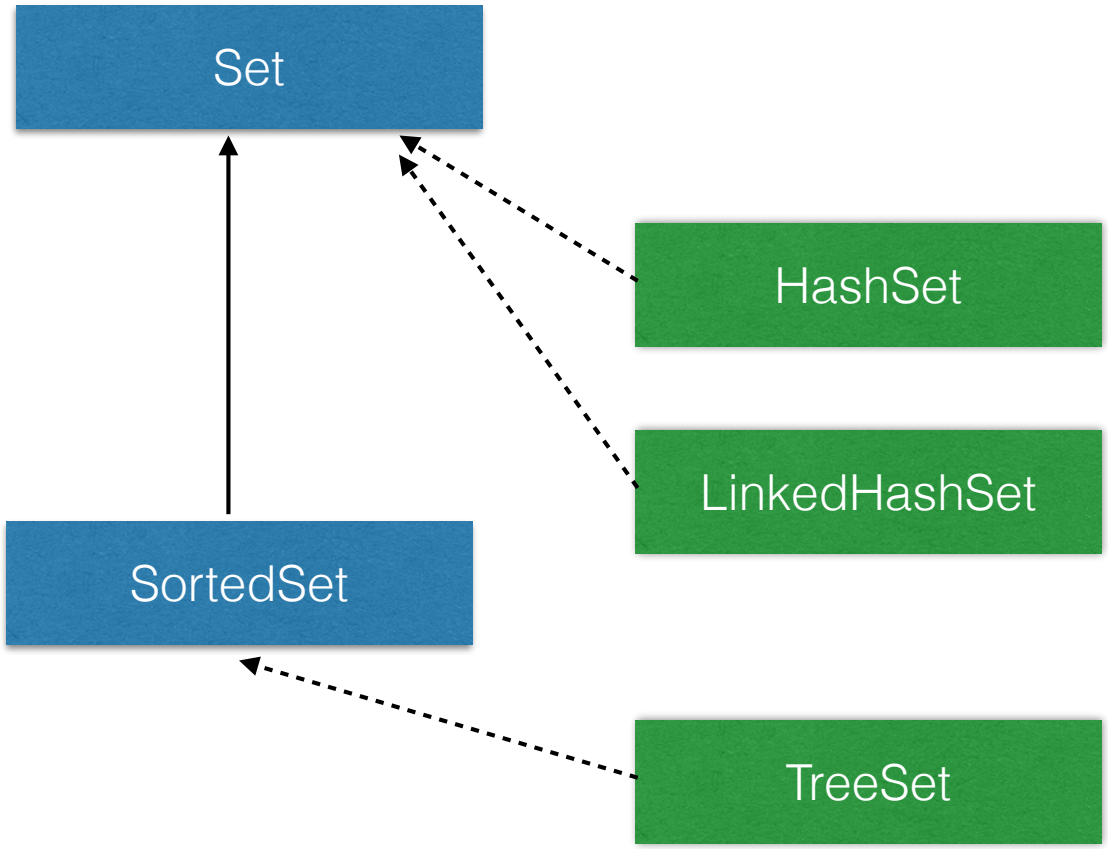
LinkedHashSet

- Reihenfolge wie bei einer Liste
- Hohe Performance für Mengenoperationen
- Iterator liefert die Elemente in der Einfügereihenfolge

NavigableSet und SortedSet (Schnittstellen)

- Kann für ein Element das nächsthöhere/-kleinere liefern
- first(), last(),
- headSet(E toElement), tailSet(E fromElement), subSet(E fromElement, E toElement)

Sets (Implementations)



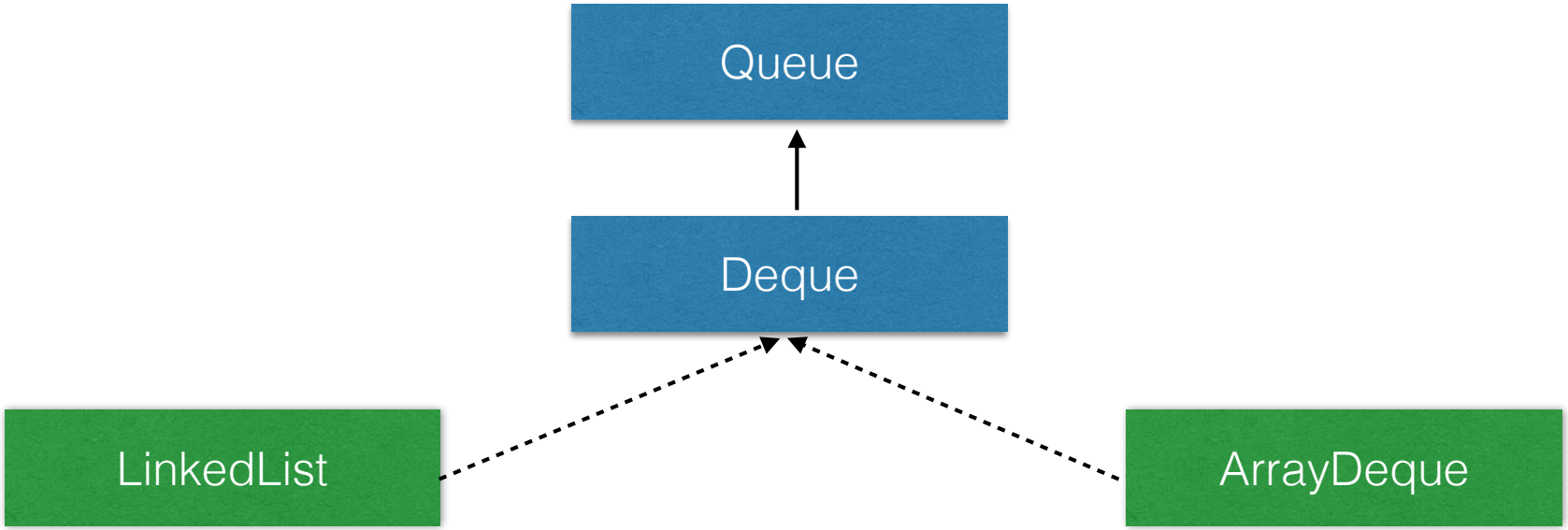
Queue

- Datenstrukturen, die nach dem FIFO-Prinzip

Queue

- Mit Exceptions
  - add(), element(), remove()
- Ohne Exception
  - offer(), peek(), poll()

Queues (Implementations)



PriorityQueue

- Prioritätswarteschlange
- Queue mit einem modifizierten FIFO-Prinzip
- Elemente werden sortiert
- Elemente müssen comparable sein (natürliche Sortierung) oder ein Comparator muss angegeben
- remove() entfernt das kleinste Element

Deque

- Erlaubt das Einfügen und Löschen an beiden Enden
  - Queue (FIFO)
    - addLast(), removeFirst()
  - Stack (LIFO)
    - push(), pop()

Deque

- Mit Exceptions
  - addFirst(), removeFirst(), getFirst()
  - addLast(), removeLast(), getLast()
- Ohne Exception
  - offerFirst(), pollFirst(), peekFirst()
  - offerLast(), pollLast(), peekLast()

Iterator

- hasNext()
- next()
- remove()
  - listIterator() (List)
  - descendingIterator() (NavigableSet)

Comparator und Comparable

- Ordnung herzustellen
  - natürliche Ordnung
  - externes Vergleichsobjekt

Comparator und Comparable

- Datenstrukturen, die eine Sortierung verlangen, wie TreeSet oder TreeMap, nehmen entweder einen **Comparator** entgegen oder erwarten von den Elementen eine Implementierung von **Comparable**

Utility-Klassen

- java.util.Arrays
- java.util.Collections

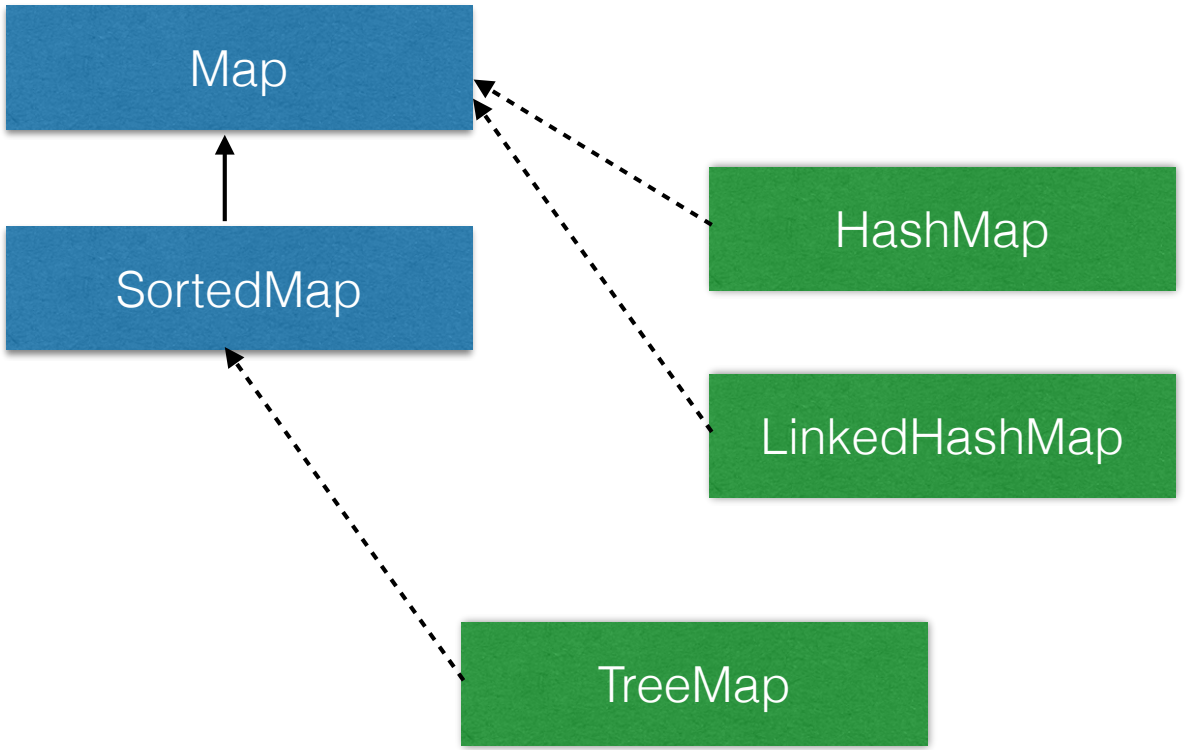
Collections

- Listen zu sortieren, mischen, umdrehen, kopieren und zu füllen
- Halbierungssuche / Binär Suche (BinarySearch)
- Elemente in einer Liste zu ersetzen
- etc.

Map

- Assoziativspeicher
- Schlüssel-Wert-Paare

Maps (Interfaces und Implementations)



HashMap

- Elemente werden unsortiert gespeichert
- Sortierung der Schlüssel ist nicht möglich
- Schlüsselobjekte müssen »hashbar« sein
  - equals(…) und hashCode() konkret implementieren

TreeMap

- Binärbaum
- Die Elemente müssen eine natürliche Ordnung besitzen, oder ein externer Comparator muss die Ordnung festlegen
- **NavigableMap** sortiert die Elemente eines Assoziativspeichers nach Schlüsseln und bietet Zugriff auf das kleinste oder größte Element