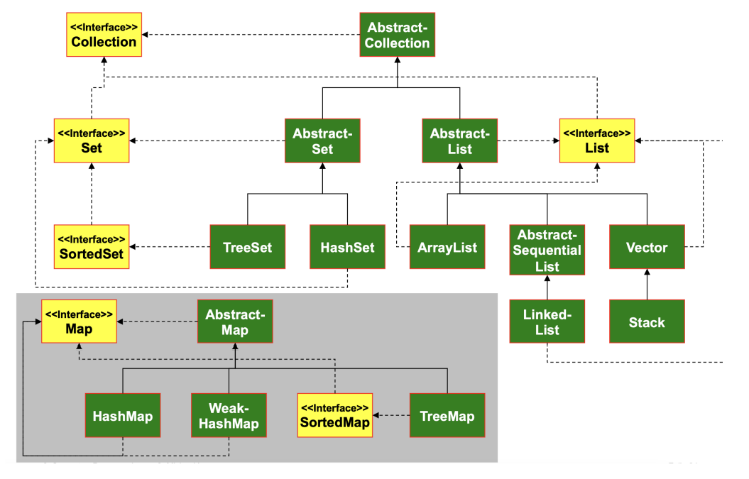
**Collection Framework**

- bietet generische Container  
- können verschiedenste Objekte enthalten   
- können beliebig viele Objekte aufnehmen   
- können auf bestimmte Objekte typisiert werden

Die drei Arten von Containern

Listen (List)   
- Zugriff sequenziell oder wahlfrei  
- Duplikate erlaubt  
- Reihenfolge des Einfügens bleibt erhalten

Mengen (Set)  
- Zugriff erfolgt über Iteratoren  
- keine Duplikate   
- Reihenfolge des Einfügens bleibt nicht erhalten

Schlüssel-Werte-Paare (Map)   
- zusammengehörige Objektpaare  
- Schlüssel sind immer eindeutig  
- Zugriff über Schlüssel

**Das Interface List**

- befindet sich in java.util  
- Zugriff auf die Container erfolgt sequentiell oder über Indexzugriff  
- sequentieller Zugriff: erfolgt über Iteratoren   
- Index: 0 bis n-1 bei Größe n  
- Länge ändert sich dynamisch bei Bearbeitung  
- Duplikate sind erlaubt   
- die Reihenfolge der Elemente bleibt erhalten  
- meist genutzte Implementierung: ArrayList & Vector  
 - intern als Arrays realisiert   
 - Hauptunterschied: Zugriffsmethoden auf Vector sind synchronisiert (wichtig bei Threads)

Wesentliche Methoden im Umgang mit Listen

add(int i, Object o) / add(Object o) fügt neue Objekte in die Liste ein

set(int i, Object o) überschreibt das Objekt an der Stelle i mit dem  
 Objekt o

get(int i) liefert das Objekt an der Stelle i zurück

contains(Object o) überprüft, ob das Objekt o in der Liste enthalten ist

indexOf(Object o) liefert den Index des Objekt o (-1 = nicht enhalten)

remove(int i) oder remove(Object o) löscht das Objekt aus der Liste

clear() initialisiert die Liste

size() liefert die Länge der Liste zurück

Der Umgang mit Iteratoren

Merkmale von Iteratoren

- einheitlicher Standard zum Durchlaufen von Datencontainern  
 - Container wird sequenziell durchlaufen  
 - es können keine Elemente übersprungen werden  
 - der Container kann sowohl vorwärts als auch rückwärts durchlaufen werden  
 - bei Änderung des Containerinhalts muss der Iterator neu erzeugt werden

Wichtige Iterator-Methoden

hasNext() hat Nachfolger?

next() greift auf nächstea Element zu

remove() löscht zuletzt gelesenes Element aus dem Container

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Die Klasse TreeSet**

- befindet sich in java.util  
- Zugriff auf die Container erfolgt sequentiell oder über Indexzugriff  
- sequentieller Zugriff: erfolgt über Iteratoren   
- Index: 0 bis n-1 bei Größe n  
- Länge ändert sich dynamisch bei Bearbeitung  
- Duplikate sind nicht erlaubt   
- Reihenfolge der Elemente bleibt nicht erhalten   
- Sortieren nach der natürlichen Ordnung durch Implementierung des Comparable-Interface  
- Comparable-Interface muss immer implementiert werden, wenn Objekte eingefügt werden  
- Beliebige Sortierung durch Implementierung des Comparator- Interface

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Das Interface comparable

- sortiert Elemente beim Einfügen in Sets oder Maps  
- Sortierung erfolgt nach der natürlichen Ordnung  
- muss für alle Klassen implementiert werden, deren Instanzen in Sets oder Maps gespeichert werden  
- beinhaltet genau eine Methode: public int compareTo(Object o)  
- Bedeutung der Rückgabewert:  
 Wert < 0: das einzufügende Element liegt vor dem Vergleichsobjekt  
 Wert = 0: das einzufügende Element und das Vergleichsobjekt sind gleich  
 Wert > 0: das einzufügende Element liegt hinter dem Vergleichsobjekt

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Sortieren von Listen

- Listen (Vector, ArrayList, ...) sind normalerweise unsortiert

- die Klasse Collections bietet eine überladene Sortiermethode zum Sortieren von List-Objekten an

static void sort(List liste) sortiert die Liste nach der natürlichen Ordnung dazu müssen die  
Klassen das Interface Comparable implementieren, deren Instanzen in der Liste gespeichert sind

static void sort(List liste, Comparator c) übersteuert die natürliche Ordnung und sortiert die Objekte der Liste über den entsprechenden Comparator c

Umgang mit Wrapper-Klassen

- statt elementarer Datentypen werden Objekte erwartet (z.B. in Datencontainern)  
- um elementare Datentypen in Objekten zu kapseln, gibt es die Wrapper-Klassen  
 - stellen Methoden zur Ein- und Ausgabe sowie zur Manipulation zur Verfügung  
 - stellen Methoden zur Umwandlung von Datentypen zur Verfügung  
- Wrapper-Klassen existieren für boolean, byte, char, double, float, int, long, short,  
- Auto-Boxing / Auto-Unboxing  
 - Java erstellt automatisch ein Objekt der Wrapper-Klasse wenn Objekt erwartet, aber  
 einfacher Datentyp bereitgestellt   
 - umgekehrt wird der Wert als einfacher Datentyp bereitgestellt, wenn ein Objekt der  
 Wrapper-Klasse zurückgegeben wird (Auto-Unboxing)