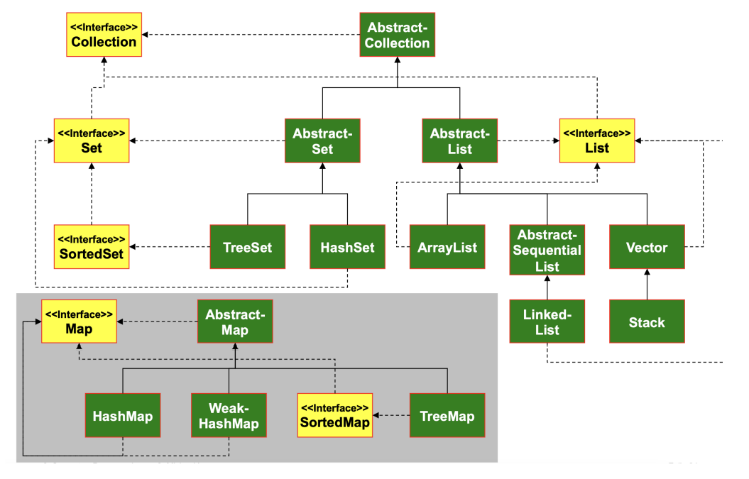
**Collection Framework**

- 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

Der Vergleich von Objekten

- Vergleich mit dem ==-Operator prüft, ob es sich um die identische Speicherreferenz handelt  
- inhaltliche Vergleiche erfolgen über die equals()-Methode  
- der equals-Contract aus der Dokumentation:

- reflexiv Selbstvergleich ist true  
 - Symmetrisch x.equals(y) = y.equals(x)  
 - transitiv x.equals(y) und y.equals(z) heißt x.equals(z)  
 - konsistent Wenn Objekte gleich dann Ergebnis gleich  
 - Objekte dürfen nicht „null“ sein

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)

Das Überschreiben der equals()-Methode

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Zusammenhang hashCode() und equals()

- Verwendung für die Verwaltung der Einträge in hash-basierten Datencontainern  
- korrekte Verwaltung der Einträge basiert auf folgender Bedingung

Wenn: o1.equals(o2) = true, dann o1.hashCode = o2.hashCode()

- Vorschlag zur Implementierung  
 - Verwendung der Attribute, die bei Implementierung von equals()-Methode verwenden  
 - Ermittlung der Hash-Codes der ausgewählten Attribute einer Klasse  
 - Addition oder bitweise Verknüpfung mit exklusivem Oder der einzelnen Hash-Codes

Was hat das mit Comparable zu tun?

- comapreTo() sortiert Objekte nach einer "natürlichen" Ordnung (Rückgabe Wert 0: die Objekte sind gleich)

- Rückgabewert 0 für zwei Objekte = Rückgabewert von true beim Vergleich mit equals()

- Zusammengefasst: equals(), hashCode() und compareTo() sollten für ein Objekt immer auf den gleichen Attributen basieren

Maps

- in java.util  
- kein Sub-Interface von Collections  
- immer Schlüssel-Wert-Paare  
- jeder Schlüssel ist eindeutig   
- wird mit dem gleichen Schlüssel ein weiterer Wert eingefügt, so wird der erste Wert überschrieben  
- Zugriff auf die Werte-Objekte erfolgt über die Schlüssel

zwei wesentliche Vertreter:

- TreeMap: Einträge werden nach Schlüsseln sortiert -> Schlüssel- Klasse muss das Interface  
 Comparable implementieren

- HashMap: auf Basis der hashCode()-Methode der Schlüsselklasse wird eine interne Position  
 (Bucket) berechnet, an der das Schlüssel- Werte-Paar in die Map aufgenommen wird

Wesentliche Methoden im Umgang mit Maps

keySet() liefert ein Set der Schlüssel einer Map ohne Duplikate zurück

values() liefert eine Collection der Werte einer Map zurück (Duplikate erlaubt)

put(Object k, Object v) nimmt ein Schlüssel-Werte-Paar in die Map auf

get(Object k) liefert den Wert zum Schlüssel-Objekt k zurück

containsKey(Object k) liefert true zurück, wenn zu dem Schlüssel k ein Eintrag in der Map ist

containsValue(Object v) liefert true zurück, wenn zu dem Wert v ein Eintrag in der Map ist

remove(Object k) löscht den Eintrag zum Schlüssel k aus der Map

size() liefert die Länge der Map zurück

clear() initialisiert die Map