1 Collection Framework

Überblick: http://docs.oracle.com/javase/8/docs/technotes/guides/collections/
Tutorial: http://docs.oracle.com/javase/tutorial/collections/
API Docu: http://docs.oracle.com/javase/10/docs/api/

- Seit JDK 1.2
- Seit JDK 1.5 (Java 5) mit Generics
- Seit JDK 1.8 (Java 8) mit Support für Lambda-Ausdrücke

Definitionen

Collection

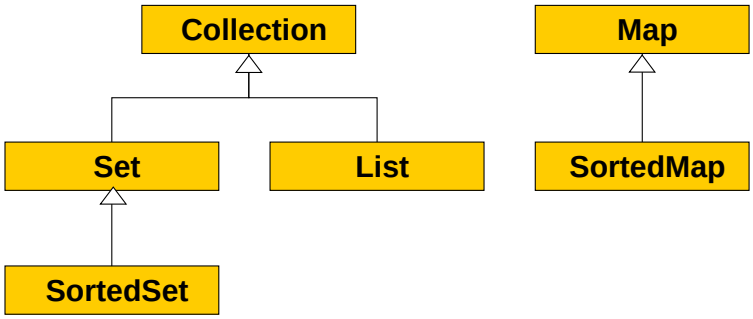
- *Sammlung*, repräsentiert durch ein Objekt, welches mehrere Elemente zu einer Einheit zusammenfasst und verwaltet.
- Collections werden verwendet um Daten zu speichern, abzufragen, zu manipulieren und von einer Methode zur anderen zu übertragen.

Framework

1. Menge von Schnittstellen — *Benutzersicht*
   - Set (Duplikate nicht erlaubt)
   - List (Zugriff über Index möglich)
   - Map (Zugriff über einen Schlüssel)
2. Menge von Klassen welche die Schnittstelle implementieren — *Implementierungssicht*
   - Array, lineare Listen, Baum
3. Algorithmen welche die Schnittstelle benutzen — *Generische Algorithmen*
   - Suchen, Sortieren

↳ kann mit neuen Klassen und Algorithmen erweitert werden.

Interfaces:



Die Menge der Interfaces wurde bewusst klein gehalten
Folgende Aspekte wurden *nicht* mit speziellen Interfaces modelliert:
- Unveränderbarkeit (no add/remove)
- nur erweiterbar (no remove)
- `null` als Argument erlaubt

Lösung:
Die Methoden dürfen Ausnahmen werfen (*RuntimeExceptions*):
- Optionale Methoden werfen eine *UnsupportedOperationException*
- Bei eingeschränkten Wertebereichen wird eine *IllegalArgumentException* geworfen

Collection

```
interface Collection<E> extends Iterable<E> {
   int       size();
   boolean   isEmpty();

   boolean   contains(Object x);
   boolean   containsAll(Collection<?> c);

   boolean   add(E x);
   boolean   addAll(Collection<? extends E> c);

   boolean   remove(Object x);
   boolean   removeAll(Collection<?> c);
   boolean   retainAll(Collection<?> c);
   void      clear();

   Object[]  toArray();
   <T> T[]   toArray(T[] a);

   Iterator<E>  iterator();
    // default methods: forEach, spliterator, parallelStream, removeIf, stream
}
```

Iterator

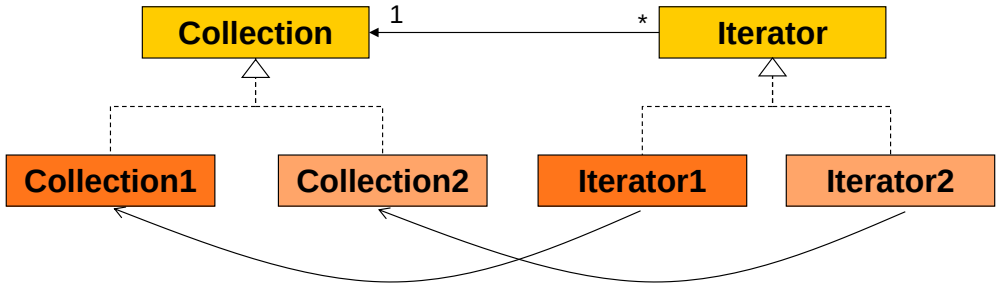
```
interface Iterator<E> {
   boolean   hasNext();
   E         next();
   void      remove();    // default: throws UnsupportedOperationException
}
```

Vorteile:

- Mehrere „Zugriffspfade“ auf eine Sammlung
- Iteratoren können leicht spezialisiert werden, z.B. nur ungerade Zahlen

Nachteil:

- Für jede konkrete Collection-Implementation muss ein spezieller Iterator geschrieben werden.

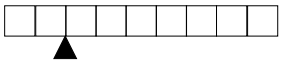


Verwendung: Ausgeben einer Collection

```
public static void print(Collection<?> c) {
   Iterator<?> it = c.iterator();
   System.out.print("[");
   while (it.hasNext()) {
      System.out.print(it.next());
      if(it.hasNext()) System.out.print(",");
   }
   System.out.println("]"),
}
```

⇨ Generischer Algorithmus, funktioniert mit allen Collections

Iterator



Konzeptionell zeigt ein Iterator zwischen zwei Elemente.

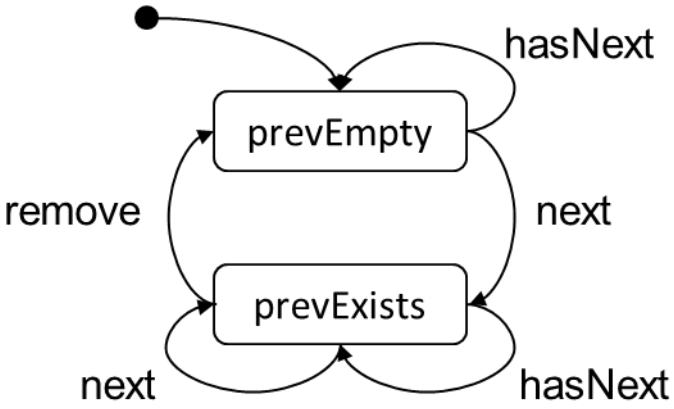
Bei Collections mit *n* Elementen gibt es also *n+1* mögliche Positionen für den Iterator.

Methoden:

`hasNext` = es gibt ein weiteres Element welches übersprungen werden kann.

`next` = übersprungenes Element wird zurückgegeben.
(wichtig bei `ListIterator`, der die zusätzlichen Methoden `hasPrevious()` und `previous()` enthält).

`remove` = Entfernt jenes Objekt, welches zuletzt von `next` (bzw. `previous`) zurückgeliefert wurde, d.h. dieses Objekt muss nicht erneut gesucht werden.
=> vor einem Aufruf von `remove` muss `next` (oder `previous`) aufgerufen werden



Wenn der Aufruf gemäss dieser Zustandsmaschine nicht zulässig ist, wird eine `IllegalStateException` geworfen.

Implementierung von Collections

Um ein Interface zu implementieren, müssen alle Operationen ausprogrammiert werden, denn ein Interface darf keinen Code enthalten.

Eine abstrakte Basisklasse kann hilfreich sein, da in dieser jene Operationen implementiert werden können, welche sich mit Hilfe anderer Operationen ausdrücken lassen. Konkrete Implementierungen werden dann von diesen abstrakten Basisklassen abgeleitet. Solche Default-Implementierungen könnten auch als Default-Methoden in der Schnittstelle bereitgestellt werden. Diese Methoden machen keine Annahmen über die Repräsentation der Sammlung selbst.

Aufgabe:
Welche Operationen aus dem Collection-Interface können in einer abstrakten Klasse `AbstractCollection` (oder als Java 8 Default-Methoden) mit Hilfe anderer Methoden implementiert werden?

```
abstract class AbstractCollection implements Collection {
   ...
}
```

Konkrete Implementationen:
Um eine konkrete Implementation zu erhalten sind
- die interne Repräsentation (Datenstruktur) festzugelegen
- die Klasse `AbstractCollection` und das Interface `Iterator` zu erweitern bzw. zu implementieren
- unter Umständen die Default-Methoden durch effizientere zu ersetzen