Auszüge aus der Java Klassenbibliothek

Marcel Lüthi
Departement Mathematik und Informatik

Agenda

- Collections
- Streams
- Weitere nützliche Pakete.

Collections

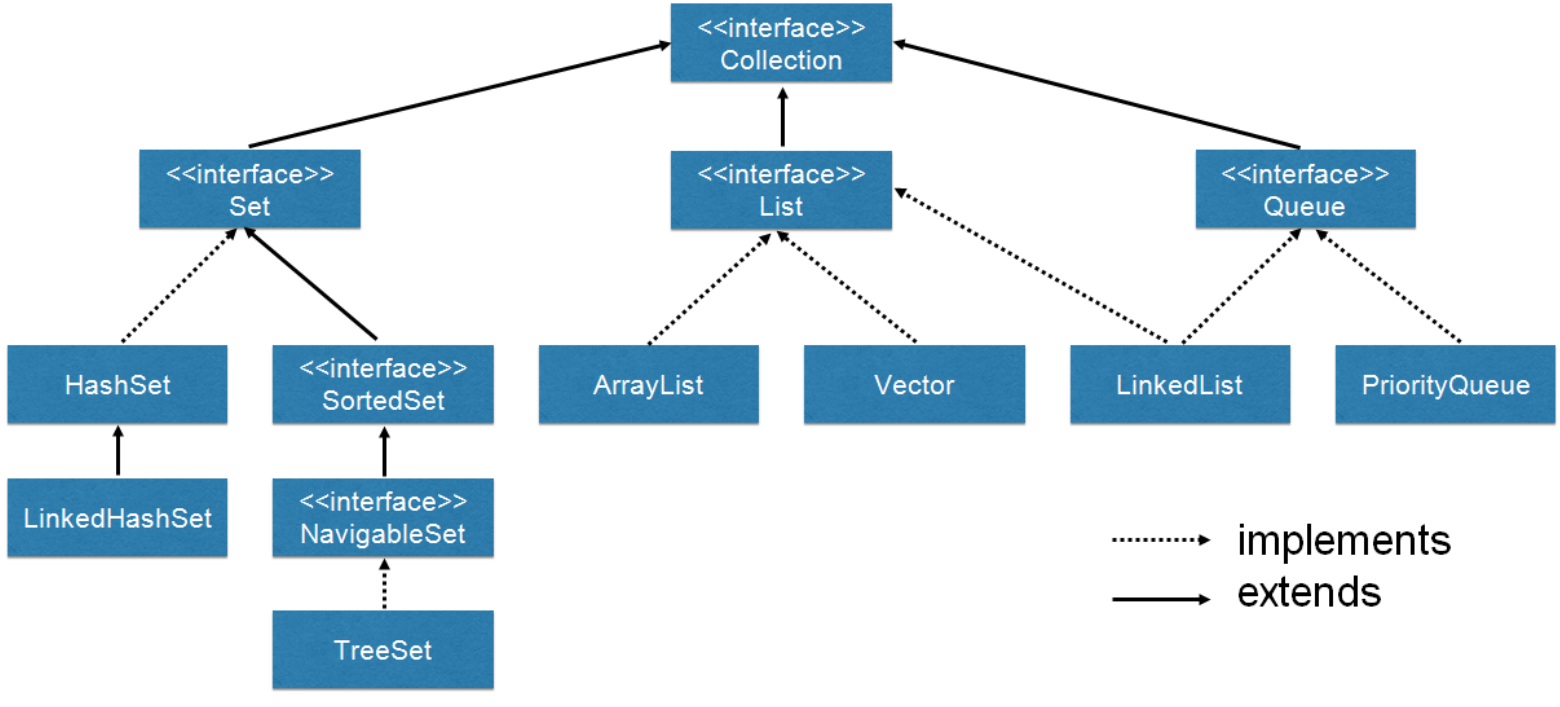
Collections

Collections fassen Objekte (Elements) zusammen

- Beispiele:
  - Schulklasse (Gruppe von Schülern)
  - Einkaufsliste (Sammlung von Lebensmitteln)

Auf Instanzen beliebiger Klassen anwendbar (Generic)

Wichtigste Collections



Methoden von Collections

Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	**clear**() Removes all of the elements from this collection (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	**equals**(Object o) Compares the specified object with this collection for equality.
int	**hashCode**() Returns the hash code value for this collection.
boolean	**isEmpty**() Returns true if this collection contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.
default Stream<E>	**parallelStream**() Returns a possibly parallel Stream with this collection as its source.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
default boolean	**removeIf**(Predicate<? super E> filter) Removes all of the elements of this collection that satisfy the given predicate.
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this collection.
default Spliterator<E>	**spliterator**() Creates a Spliterator over the elements in this collection.
default Stream<E>	**stream**() Returns a sequential Stream with this collection as its source.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

(https://docs.oracle.com/javase/8/docs/api/index.html?java/util/Collection.html)

Interface List

- Geordnet
- Erlaubt Duplikate

Beispiele

- ArrayList
- LinkedList

Interface Set

- Ungeordnet
- Enthält jedes Element nur einmal

Beispiele

- HashSet
- TreeSet

Beispiel: Unterschied List/Set

In [55]:
```
String[] fruits = {"Banana", "Apple", "Mango", "Apple"};

List<String> fruitList = new ArrayList<>();
Set<String> fruitSet = new HashSet<>();

for (String fruit : fruits) {
    fruitList.add(fruit);
    fruitSet.add(fruit);
}
System.out.println("fruitList: "+ fruitList);
System.out.println("fruitSet: "+ fruitSet);
```

```
fruitList: [Banana, Apple, Mango, Apple]
fruitSet: [Apple, Mango, Banana]
```

Iterable

- Alle Java collections implementieren das *Iterable* interface
- Wichtigste Methode: Gibt Iterator zurück

```
public interface Iterable<T> {

    Iterator<T> iterator();

    ...
}
```

Iterator interface

- Sequentielles Durchlaufen einer Kollektion
- Wichtigste Methoden:
  - `boolean hasNext()`
  - `T next()`
- Kreieren eines neuen Itearators: `collection.iterator()`

Beispiel

In [56]:
```
Iterator<String> it = fruitList.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

```
Banana
Apple
Mango
Apple
```

Iterator form des For loops

Implementation von `Iterable` erlaubt die nutzen der Iterator form des For loops

In [59]:
```
for (String s : fruitList) {
    System.out.println(s);
}
```

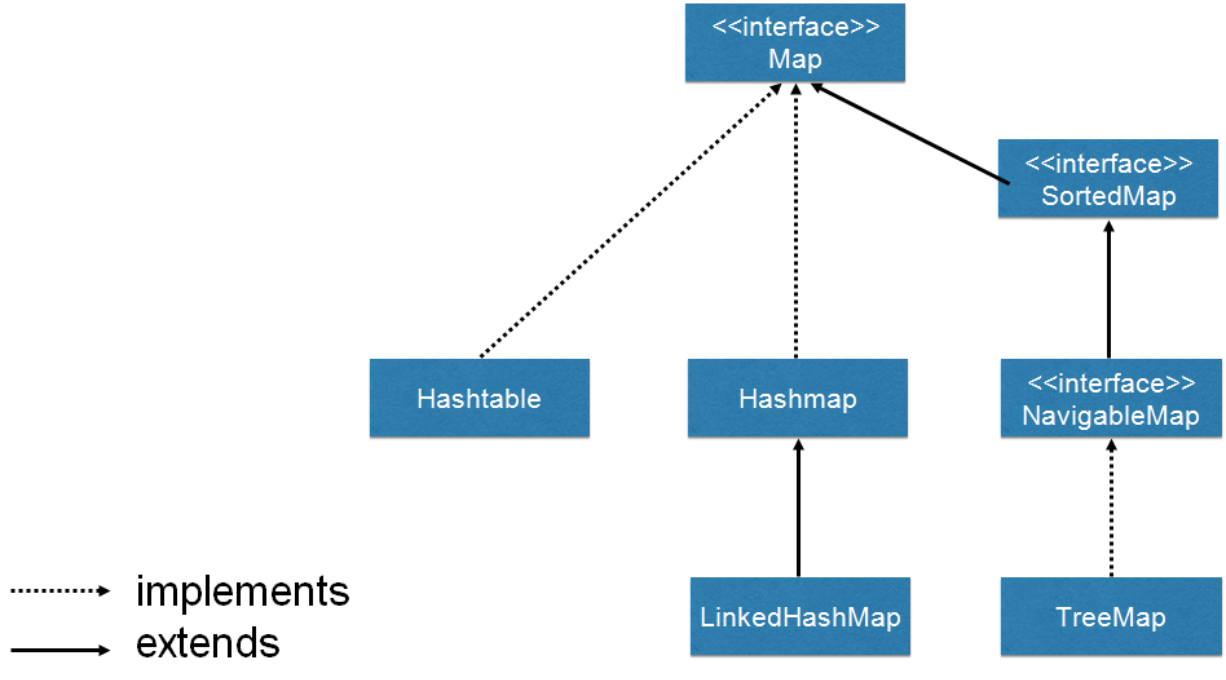
```
Banana
Apple
Mango
Apple
```

Maps

Map ist ähnlich wie `Set`, aber:

- Verlinkt zwei Objekte: *Key* und *Value* (z.B. Produkte mit ihren Preisen)
- Stammt nicht vom Collection Interface ab (z.B. put statt add)
- `get` eines Elements via *Key*

Maps - Hierarchie



Anwendungsbeispiel Map

In [60]:
```
Map<String, Double> prices = new HashMap<>();

prices.put("Banana", 1.5);
prices.put("Apple", 1.0);
prices.put("Mango", 2.5);

System.out.println("Price of a Mango " +prices.get("Apple"));
```

```
Price of a Mango 1.0
```

Anwendungsbeispiel Map

In [60]:
```
Map<String, Double> prices = new HashMap<>();

prices.put("Banana", 1.5);
prices.put("Apple", 1.0);
prices.put("Mango", 2.5);

System.out.println("Price of a Mango " +prices.get("Apple"));
```

```
Price of a Mango 1.0
```

Übungen

- Was passiert wenn man einen Schlüssel zweimal (mit unterschiedlichem Wert) einfügt
- Schreiben Sie einen for-loop, der alle Preise ausgibt.
  - Tip: Die Schlüssel erhalten Sie via der Methode `keySet`
- Wie ändert sich die Ausgabe, wenn Sie eine TreeMap verwenden?

Kollektionen: Beispiel

- Mögliche Modellierung eines "Früchteladens"

```
import java.util.*;

class Fruit {
    String name;
}

public class FruitShop {

    Set<Fruit> products = new HashSet<Fruit>();

    Map<Fruit, Double> priceForFruit = new HashMap<Fruit, Double>();

    Queue<Person> customers = new LinkedList<Person>();
}
```

Streams

Streams

Funktionaler Ansatz um Elemente zu prozessieren

- Aus allen Collection kann mit Methode `stream` ein `Stream` Objekt erzeugt werden
- Wichtige Methoden
  - `map`
  - `filter`
  - `reduce`
  - …

map Methode

Signatur (in Interface `Stream<T>`)

```
<R> Stream<R>  map(Function<T,R> mapper)
```

- Führt Funktion auf jedem Element vom Stream aus
  - produziert neue Liste

Beispiel: map-Methode

In [62]:
```
import java.util.stream.Stream;

Stream<String>  newFruitStream = fruitList.stream().map(f -> f.toUpperCase());
newFruitStream.forEach(f -> System.out.println(f));
```

```
BANANA
APPLE
MANGO
APPLE
```

Beispiel: map-Methode

In [62]:
```
import java.util.stream.Stream;

Stream<String>  newFruitStream = fruitList.stream().map(f -> f.toUpperCase());
newFruitStream.forEach(f -> System.out.println(f));
```

```
BANANA
APPLE
MANGO
APPLE
```

Übung:

- Erzeugen Sie einen Stream von den Zahlen 1, 2 und 3, indem Sie die statische Methode of von Streams nutzen
- Nutzen Sie die Methode map um diese in einen String umzuwandlen.
- Geben Sie die Elemente des Streams aus.

In [64]:
```
import java.util.stream.IntStream;

Stream<String> s = Stream.of(1,2,3).map(n -> Integer.valueOf(n).toString());
s.forEach(f -> System.out.println(f));
```

```
1
2
3
```

filter-Methode

Signatur (in Interface `Stream<T>`)

``` Stream filter(Predicate filter) ````

- Gibt Stream mit allen Elementen e zurück für die gilt `filter(e) == true`

Beispiel: filter-Methode

In [65]:
```
Stream<String> newFruitStream = fruitList.stream().filter(f -> f.contains("n"));
newFruitStream.forEach(f -> System.out.println(f));
```

```
Banana
Mango
```
```

reduce-Methode

Signatur (in Interface `Stream<T>`)

```
<R> Stream<R>  Reduce(T identity, BinaryOperator<T> accumulator)
```

- Zieht Element zusammen, durch ausführen von `accumulator`
- BinaryOperator ist `FunctionalInterface` mit zwei Argumenten vom selben Typ

Beispiel: reduce

In [66]:
```
import java.util.function.BinaryOperator;

BinaryOperator<String> concat = (s, t) -> s + t;
fruitList.stream().reduce("", concat);
```

Out[66]:
```
BananaAppleMangoApple
```