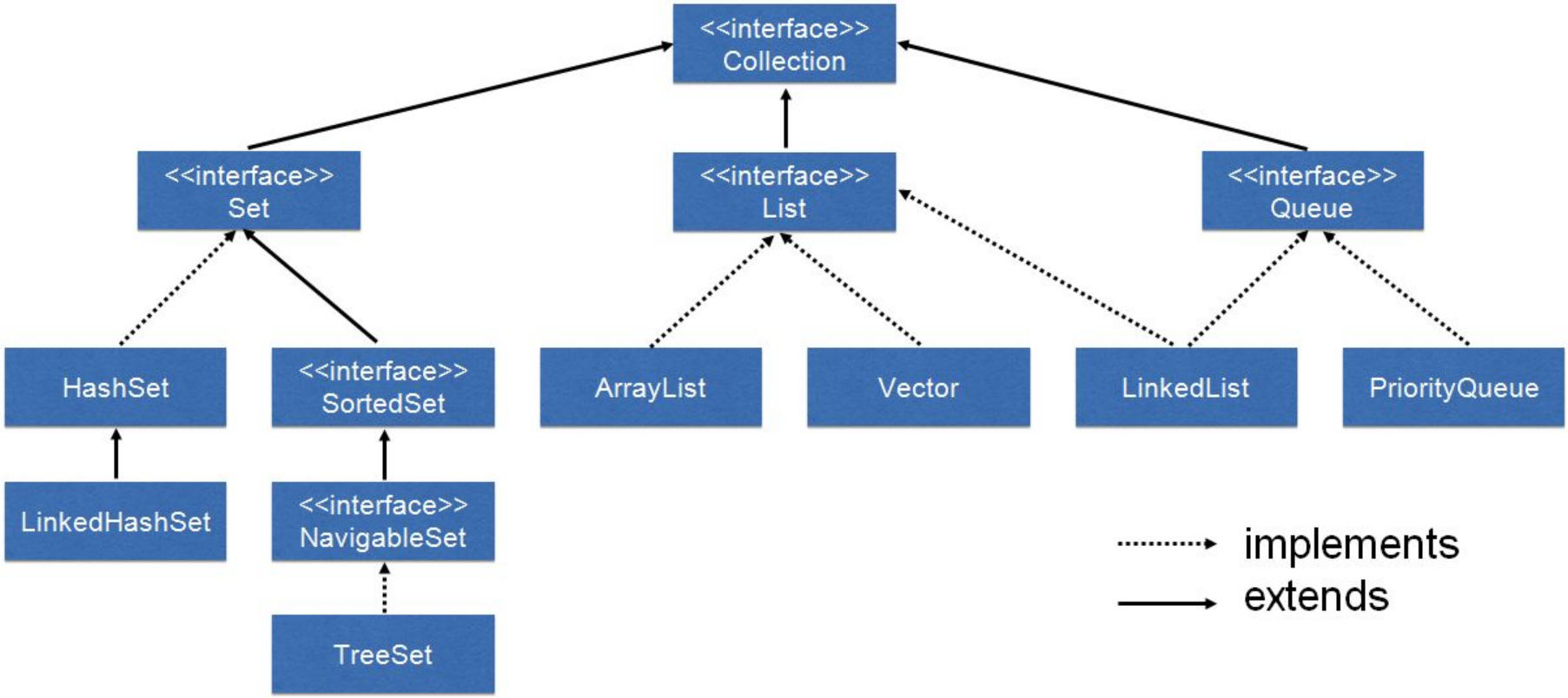
PGdP Woche #9 - Collections und Streams

Sebastian Oßner, ossner@in.tum.de

Ablauf:

- Überblick Collections & Enums
- P01: Datenstrukturen
- P02: Streams
- P03: Streams und FileIO

Collection Interface



Collections - Wichtige Unterschiede

Siehe GitHub Klasse ImportantCollections

Enums

Enums (Enumerators) sind spezielle “Klassen” die den Inhalt einer Gewissen Menge angeben. Damit kann man für alle X in Y etwas geltend machen.

Zum Beispiel alle Einheiten für Temperaturen: (siehe GitHub Temperatures)

```
public enum Temperatures {
    Celsius { ... },
    Fahrenheit { ... },
    Kelvin { ... },
    Rankine { ... },
    Reaumur { ... }
}
```

- Deklaration ähnlich wie Klassen
- Komma-getrennte Auflistung der Optionen
- In geschweiften Klammern können Methoden (z.B. toString() überschrieben werden. (siehe GitHub)
- Zugriff durch z.B. Temperatures.Kelvin

HashMap (implements Map)

HashMap<K, V>

- Besitzt 2 Generics, (K)ey und (V)alue (d.h. 2 Objekte, die aufeinander abbilden)
- Alle (individuellen) keys besitzen einen value
- Perfekt für 1:1 Mapping von Objekten

Key	Value
Januar	31
Februar	28

- Übersicht der verfügbaren Methoden hier

Queue (LinkedList)

LinkedList implementiert Queue interface (siehe slide 2), daher bietet sie einige Methoden an, die für Warteschlangen wichtig sind (Bei der nächsten Aufgabe allerdings nicht relevant):

poll()

element()

peek()

P01 - Schraubenladen

ScrewDrive: Enumerator über verschiedene Typen von Schrauben

Screw: Java-Darstellung einer Schraube (hat z.B. einen ScrewDrive als Kopf)

ScrewStore: Darstellung des Schraubenladens (mit Inventar HashMap und Bestellungs-Queue)

Order: Ein Bestellungs-Objekt

=> Implementiert jetzt die Methoden auf Artemis

Streams

+ tutorial

Streams sind Sequenzen von Elementen, auf denen verschiedene Operationen angewandt werden können.

Streams (z.B. auf collections) müssen zuerst erzeugt werden, d.h. Eine Methode wird aufgerufen, die einen Stream zurückgibt nämlich .stream()

Methoden auf Streams sind entweder intermediate (geben Stream zurück) oder terminal (void)

Daher kann man nach intermediate Methoden nochmal intermediate Methoden verwenden, so lange bis man eine terminale Methode verwendet

Streams II

Siehe GitHub Klasse StreamsExample

Die wichtigsten Methoden:

Intermediate Methoden:
- .map()
- .filter()
- .sorted()
- .mapToDouble()/Int/Long
- .average()

Terminal Methoden:
.collect()
.forEach()
.reduce()
.toArray()
.getAsDouble()

Streams: .collect(...)

Mit .collect(Collectors.to...) kann ein Stream in eine Collection überführt werden, diese wird dann zurückgegeben (=> terminal Methode)

Verfügbar:

Collectors.toSet()
Collectors.toList()
Collectors.toMap()

Streams Optional Klasse/Primitive vs. Generic

Optionals sind nützlich um Primitive Datentypen darzustellen, die es möglicherweise nicht gibt (minimum von {} ist nicht-existent)

Normalerweise verwendet man Streams auf Objekten (z.B. Integer), man kann sie aber auch auf primitive typen (z.B. int) anwenden.

Dafür muss man den Stream mit mapToDouble()/Int/Long zu den primitives ändern

Mit boxed() kann man dann zum Beispiel einen IntStream zu Integer machen (gilt auch für Double etc.)

P02 - Streams Grundlagen

Implementiert nun die Aufgaben auf Artemis, verwendet dafür Streams (duh)

P03 - Temperatur Streams

Wendet jetzt das Wissen der vorherigen Aufgabe an und bearbeitet das Template auf Artemis. Die ML verwendet übrigens weitere Stream Methoden:

- count()
- sorted()
- distinct()
- min() and max()
- sorted() mit comparing()