Java Collections Framework

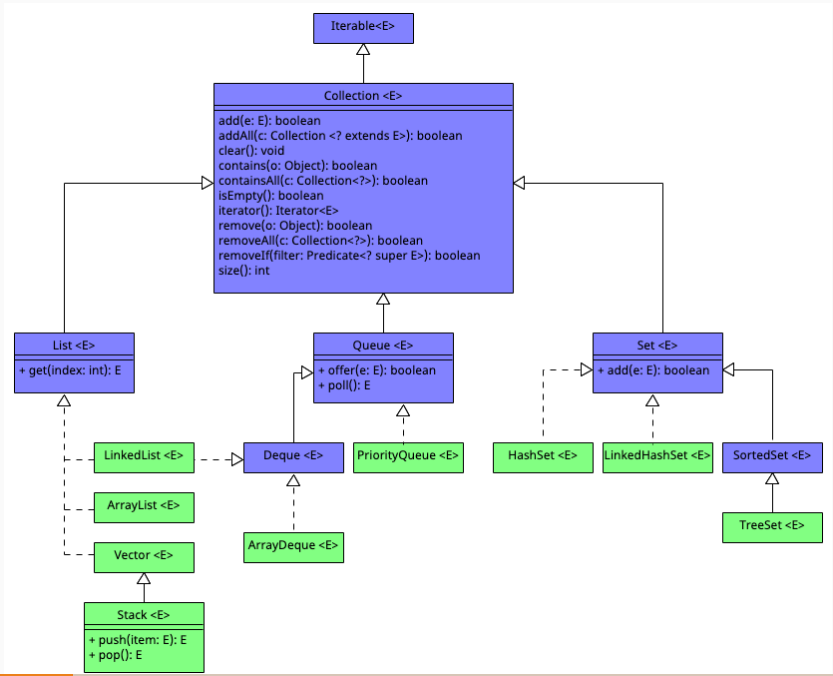
André Matutat & Carsten Gips (FH Bielefeld)

Motivation: Snippet aus einer Klasse im PM-Dungeon

```java
private List<Entity> entities = new ArrayList<>();

public void add(Entity e){
    if (!entities.contains(e)) entities.add(e);
}
```

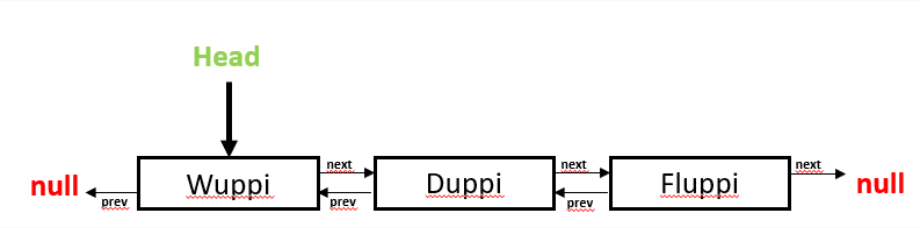
Collection-API in Java



Listen: *ArrayList*

```
private List<Entity> entities = new ArrayList<>();
```

Listen: *LinkedList*



Vector und *Stack*

- `Vector<T>`:
  - Ein `Vector<T>` ähnelt einer `ArrayList<T>`
  - Das Array eines Vector wird jedoch verdoppelt, wenn es vergrößert wird
  - Die Methoden von `Vector<T>` sind `synchronized`

- `Stack<T>`:
  - Schnittstelle: "last in first out"-Prinzip
    - `push(T)`: Pushe Element oben auf den Stack
    - `pop(): T`: Hole oberstes Element vom Stack
  - Tatsächlich aber: `class Stack<E> extends Vector<E>`

Iterierbarkeit: *Iterable* und *Iterator*

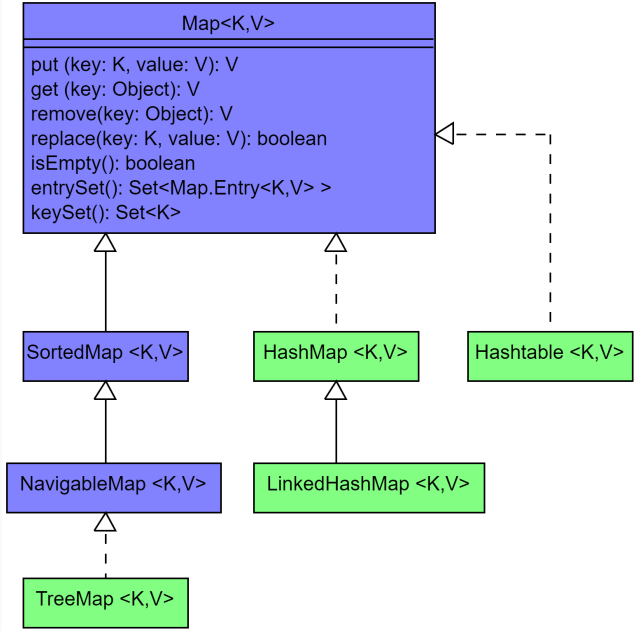
```java
private List <Entity> entities = new ArrayList<>();

for (Entity e : entities) { ... }
entities.forEach(x -> ...);
```

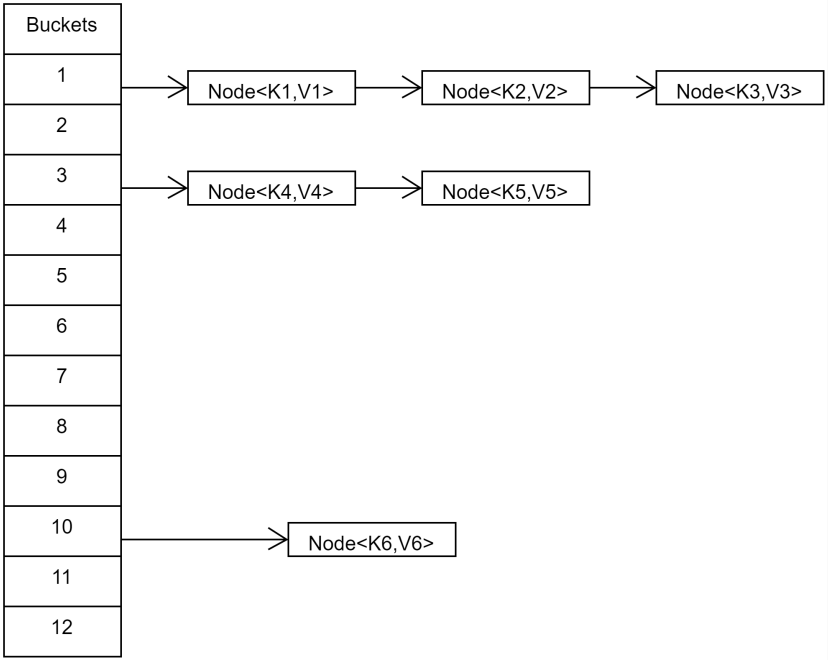
Hilfsklasse *Collections*

Collections
addAll(c: Collection<? super T>, elements: T...): boolean replaceAll(list: List<T>, oldVal: T, newVal: T): boolean rotate(list: List<?>, distance: int): void shuffle(list: List<?>): void sort(list: List<?>, c: Comparator<? super T>): void swap(list. List<?>, i: int, j: int): void frequency(c: Collection<?>, o: Object): int fill(list: List<? super T>, obj: T): void disjoint(c1: Collection<?>, c2: Collection<?>): boolean copy(dest: List<? super T>, src: List<? extends T>): void

Map


Map<K,V>
put (key: K, value: V): V
get (key: Object): V
remove(key: Object): V
replace(key: K, value: V): boolean
isEmpty(): boolean
entrySet(): Set<Map.Entry<K,V> >
keySet(): Set<K>
SortedMap <K,V>
HashMap <K,V>
Hashtable <K,V>
NavigableMap <K,V>
LinkedHashMap <K,V>
TreeMap <K,V>

HashMap


Buckets
1
2
3
4
5
6
7
8
9
10
11
12
Node<K1,V1>
Node<K2,V2>
Node<K3,V3>
Node<K4,V4>
Node<K5,V5>
Node<K6,V6>

Hashtable

- Nicht zu verwechseln mit der Datenstruktur: Hash-Tabellen (!)
- `Hashtable<K,V>` ist vergleichbar mit einer `HashMap<K,V>`
- `Hashtable<K,V>`-Methoden sind `synchronized`
- Kein Key oder Value darf `null` sein

Der *equals()-hashCode()-compareTo()*-Vertrag

Wird `equals()` überschrieben, sollte auch `hashCode()` (passend) überschrieben werden.

1. Wenn `x.equals(y) == true`, dann *muss* auch `x.hashCode() == y.hashCode()`
2. Wenn `x.equals(y) == false`, *sollte* `x.hashCode() != y.hashCode()` sein
3. `x.compareTo(y) == 0` gdw. `x.equals(y) == true`

Wrap-Up

- Interface `Collection<T>`: Schnittstelle für Datenstrukturen/Sammlungen
- Klasse `Collections`: Statische Hilfs-Methoden
- `Iterable<T>` liefert einen `Iterator<T>` zur Iteration über eine `Collection<T>`
- Interface `Map<K,V>`: Speichern von Key/Value-Paaren
- `equals()`-`hashCode()`-`compareTo()`-Vertrag beachten

LICENSE