Technische Hochschule Rosenheim

Modul - Objektorientierte Programmierung

Bachelor Wirtschaftsinformatik

06 - List, Set und Map

Prof. Dr. Marcel Tilly

Fakultät für Informatik, Cloud Computing

Datenstrukturen

Technische Hochschule Rosenheim

bisher

- **List** als sequenziellen

```
interface List<T> extends Iterable<T> {
    void add(T o);
    T get(int i);
    int length();
    T remove(int i);
}
```

- **Set** (bzw. Binärbaum) als duplikatfreien Container

```
interface Set<T> extends Iterable<T> {
    void add(T c);
    boolean contains(T c);
    int size();
}
```

Wann wird was benutzt?

Assoziatives Datenfeld: Map

Speicher zu einem Schlüsselobjekt *K* genau ein Wertobjekt *V (K -> V)*

```
interface Map<K, V> {
    void put(K key, V value);
    V get(K key);
    boolean containsKey(K key);
}
```

Auffallend ist dabei, dass die Map über zwei Typvariablen verfügt:

1. **K** für den Schlüsseltyp (key) (= wie *Set*, speichert Wert genau einmal)
2. **V** für den Wertetyp (value) (mehrfach)

Objektgleichheit

Technische Hochschule Rosenheim

Gleichheit prüfen mittels `equals`

In Java können zwei Objekte mit `equals` auf *inhaltliche Gleichheit* verglichen werden:

```
String s1 = "Hans";
String s2 = "Dampf"

System.out.println(s1.equals(s2));  // "false"
```

'equals' -Methode

Technische Hochschule Rosenheim

Daher: Bei eigenen Klassen `equals` überschreiben:

```
class MeineKlasse {
    int attribut;
    public boolean equals(Object o) {
        // 1. Das_selbe_ Objekt?
        if (o == this)
            return true;
        // 2. Passt die Klasse?
        if (!(o instanceof MeineKlasse))
            return false;
        // umwandeln...
        MeineKlasse other = (MeineKlasse) o;
        // 3. Attribute vergleichen
        if (this.attribut != other.attribut)
            return false;
        return true;
    }
}
```

Objektvergleich

Technische Hochschule Rosenheim

Objekte vergleichen mittels `compareTo`

```
public interface Comparable<T> {
    /**
     * @return 0 bei Gleichheit,
     *         negativ wenn o groesser als this,
     *         positiv wenn o kleier als this ist.
     */
    int compareTo(T o);
}
```

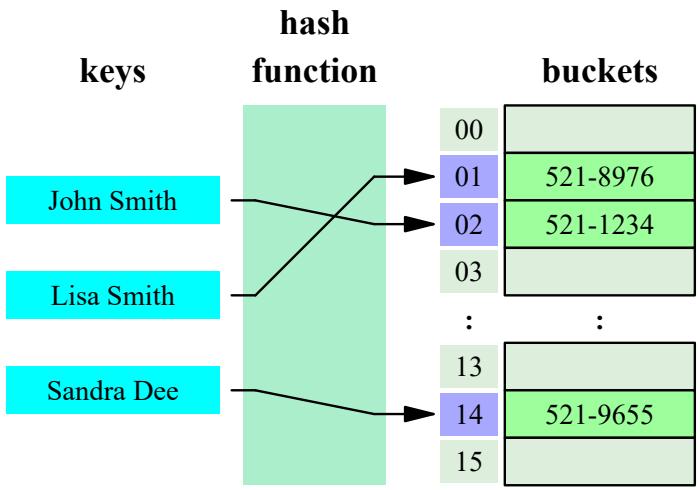
Comparable in eigener Klasse

```
class MeineKlasse implements Comparable<MeineKlasse> {
    int attribut;
    public int compareTo(MeineKlasse other) {
        // 0 bei Gleichheit, negativ wenn kleiner als other
        if (this.attribut == other.attribut)
            return 0;
        else if (this.attribut < other.attribut)
            return -1;
        else
            return 1;

        // alternativ sehr viel kuerzer:
        return this.attribut - other.attribut;
    }
}
```

Exkurs: Effizienz durch Hashing

- Verwende Array wegen schnellem Direktzugriff
- Hashfunktion um von Schlüssel zu "Schublade" zu gelanden



[Quelle: Wikimedia Commons]

Hashfunktion

Technische Hochschule Rosenheim

Schlüsselobjekte

- `Object.hashCode` (Doku)
- Definiert für alle API Klassen (String, Double, etc.)
- Für eigene Klassen: `hashCode` implementieren:

```
class MeineKlasse {
    int a;
    String s;
    public int hashCode() {
        return s.hashCode() + a;   // zum Beispiel...
    }
}
```

Hashfunktion

Technische Hochschule Rosenheim

Schlüsselobjekte

- In der Praxis: Verwendung von Hilfsbibliothek
- `org.apache.commons.lang3.builder.HashCodeBuilder`

```
class MeineKlasse {
    int a;
    String s;
    public int hashCode() {
        // wähle zwei beliebige ungerade Zahlen
        HashCodeBuilder b = new HashCodeBuilder(17, 19);

        // füge alle wichtigen Elemente an
        b.append(a).append(s);

        return b.hashCode();
    }
}
```

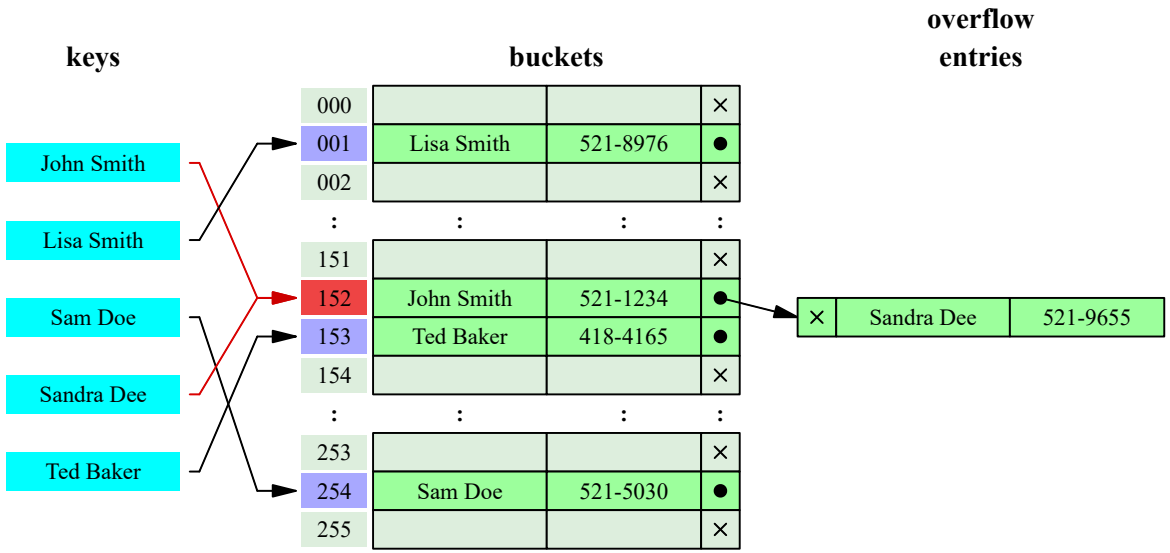
Hashfunktion

Abbildung auf Array Index

- Hash des Schlüssels berechnen (mit `hashCode`)
- Rechenvorschrift Hash zu Arrayindex (`0..bins.length-1`)
- Verwende Zweierpotenzen als Arraygrößen, Bitmasking (`&`) für Index

```
class HashMap<K, V> implements Map<K, V> {
    Map.Entry<K, V>[] bins = new Map.Entry<> [32];  // Multiple von 2
    public V get(K key) {
        // & bit-weises UND, daher bleiben genau 0..bins.length-1
        int index = (bins.length - 1) & key.hashCode();
        return bins[index];
    }
    public V put(K key, V value) {
        int index = (bins.length - 1) & key.hashCode();
        return bins[index] = value;
    }
}
```

Kollisionen



[Quelle: Wikipedia]

- Indizes aus Hash können *kollidieren*
- Verwende Liste statt einzelne Elemente

Hashes in Java

Technische Hochschule Rosenheim

- Warum?
  - Nicht immer kann sinnvoll verglichen werden
  - `hashCode` oft einfacher zu implementieren als Vergleich
- Java API: `HashMap` und `HashSet`

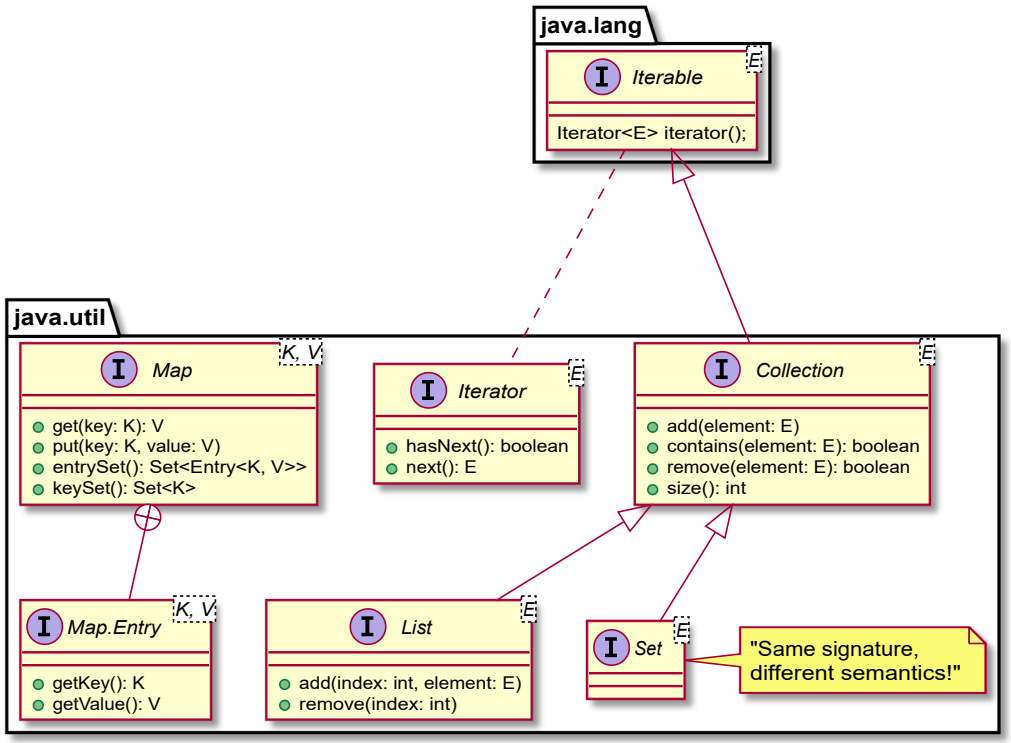
```
Set<String> set = new HashSet<>();

set.add("Hans");  // String implementiert hashCode()
```

```
Map<Integer, String> map = new HashMap<>();

map.put(31337, "Hans");  // ...Integer ebenso
```

Container in Java



Container in Java

Technische Hochschule Rosenheim

- `List` als sequenzielle Datenstruktur
- `Set` als duplikatfreie (ungeordnete) Datenstruktur
- `List` und `Set` erweitern `Collection`, was wiederum `Iterable` erweitert, d.h. alle sind iterierbar via `iterator`
- `Map` als assoziative Datenstruktur
- Realisierungen in der Java API:
  - `ArrayList` und `LinkedList`
  - `TreeSet` und `HashSet`
  - `TreeMap` und `HashMap`

Zusammenfassung

Technische Hochschule Rosenheim

- Liste, Set und Map sind die grundlegenden Datenstrukturen der Informatik
- Entwickeln Sie nach Möglichkeit immer gegen diese *Schnittstellen*
- Verwenden Sie dazu *Instanzen* von Klassen der Java API
- Beispiel: `Set<String> s = new TreeSet<String>()`
- Gute Praxis: Code bleibt unabhängig von der tatsächlichen Realisierung
- Überschreiben Sie bei eigenen Klassen mindestens die `equals` Methode, besser auch noch `hashCode`
- `Collections` sind `Iterable`, man kann diese also in `for-each` Schleifen verwenden, oder einen `Iterator` zur Traversierung erhalten.

Fragen?