

# Mosbach – Java 2

VL 06

- todo

---

Object.*method()*

- `toString()`
- `equals()`
- `compareTo()`
- `hashCode()`

- Ausgabe
- im Logging/Debugging oft gebraucht

a) das erzeugte toString oft nutzlos (Zeiger, Typ...)

b) welche Attribute sind bei einem Objekt charakteristisch?

- bei (immutable) Objekten: welches Attribut entscheidet

*Personal*

- Vorname

- Nachname

- Alter

- Pid

- Mid

?

.

- ③ • Äquivalenzrelation
  - *reflexiv* :  $a R a$
  - *symmetrisch* :  $a R b \rightarrow b R a$
  - *transitiv* :  $a R b \wedge b R c \rightarrow a R c$
- ② • Keine Ausnahme werfen
- ① • Bei Vergleich mit Objekten anderer Klassen => ungleich

# equals: Gleichheitstest

---

- Irgendwas ist null => ungleich
- Verschiedene Objekte => ungleich
- Alles gleich casten (weil equals wird mit Object o deklariert)
- Jetzt selbst entscheiden, welche Attribute für Gleichheit entscheiden





- Idee hatten wir schon
- schnelles Finden, schneller Vergleich, evtl. Kollision
- bei Objekten wieder, welches Attribut entscheidet?

Wer equals impl.  $\rightarrow$  sollte auch hashCode

Warum? A equals B  $\rightarrow$  impl.  
A.hashCode = B.hashCode

- `Staff`
  - `toString()`
  - `equals()`, wenn gleicher Name, Geburtsdatum und Geburtsort (Date für Geburtstag nehmen)
  - `hashCode`

- Punkt
  - toString()
  - equals(), wenn ??

# compareTo()

Comparable Vergleich.  
Comparator Vergleich

## • A.compareTo(B)

- -x:      A < B
- 0:        A == B
- x:        A > B

It ist in der Lösung

a)  $\leq, \geq$  immer true

b) wenn DT true, eg. false

c)

7

## • String.compareToIgnoreCase()

3

---

Lambda

- Voraussetzung:
  - interface mit nur einer Methode

```
interface Comparator {  
    compare(A, B)  
}
```

- Irgendeine Klasse nutzt das Interface als Argument:
  - `SomeClass.sort(Comparator c)`

- Alles ausschreiben in Dateien

```
Comparator myComparator = ...  
    sort(A, B) {...}  
someClass.sort(myComparator)
```

- Anonyme Klasse

```
someClass.sort(new Comparator() {..  
    sort(A, B) {...}  
})
```

- Lambda-Ausdruck

```
someClass.sort((A, B) -> {...})
```



- BubbleSort anpassen
  - sort via Name, then Age
- ```
List<Staff> myStaff = Arrays.asList() ...  
myStaff.sort((m1, m2) -> {  
    if(m1.getName().equals .... return  
    } );
```

## Method Referencing

- ```
List<Staff> myStaff = Arrays.asList() ...  
myStaff.sort(Comparator.comparing(Staff::getName));
```



- `toString()`
- `equals()`
  - 
  - 
  - 
  -
- `hashCode()`
- `compareTo()`
- `Arrays.asList()`
- modernes Nutzen von Methoden

- slides ansehen