Mosbach – Java 2

VL 06

Wiederholung

- todo

Object.*method()*

Typische Methoden

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

toString()

- Ausgabe
- im Logging/Debugging oft gebraucht

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

b) Welche Attribute sind bei einem Objekt charakteristisch?

equals

- bei (immutable) Objekten: welches Attribut entscheidet

Personal

- Vorname
- Nachname
- Alter
- Pid
- Mid

equals: Anforderungen

3
- Äquivalenzrelation
  - reflexiv : a R a
  - symmetrisch : a R b → b R a
  - transitiv : a R b ∧ b R c → 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

hashCode

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

wer equals impl. → sollte auch hashCode impl.

Warum? A equals D → A.hashCode = D.hashCode

Aufgaben

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

Aufgaben

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

compareTo()

- A.compareTo(B)
  - -x: A < B
  - 0: A == B
  - x: A > B

- String.compareToIgnoreCase()

Lambda

Nutzung von Lambda-Ausdrücken

- Vorraussetzung:
  - interface mit nur einer Methode

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

Nutzung von Lambda-Ausdrücken, II

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

Nutzung von Lambda-Ausdrücken, III

- Alles ausschreiben in Dateien

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

- Annonyme Klasse

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

- Lambda-Ausdruck

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

Nutzen von compareTo(), Aufgaben

- 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));

Widerholung

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

- Arrays.asList()
- modernes Nutzen von Methoden

Widerholung

- slides ansehen