TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Überblick

- Anwendung des Collection-Frameworks
  - Case Study: Todo-Listen
- Vergleichen von Objekten mittels `Comparator`

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Beispiel: Verwendung der Collection Methoden

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Typische Verwendung

- Hinzufügen und Entfernen von Elementen
- Ausgabe aller Elemente
- Zusammenführen / Vereinigen von Collections
- Filtern von Elementen nach bestimmten Kriterien
- Transformieren von einer Liste in eine andere Liste
- Aggregieren von Information

Beispiel: Eine Aufgabenliste

In einer Aufgabenliste können sich verschiedene Arten von Aufgaben befinden. Eine Aufgabe ist entweder ein Einkauf oder ein Telefonanruf, der erledigt werden muss.

- Ein Einkaufstask besteht aus einer Beschreibung und Preis des Produkts, das gekauft werden soll.
- Ein Telefontask besteht aus einer Telefonnummer und einem Namen sowie einem Datum.

Implementierung: Aufgabenliste

Siehe die Implementierung in `Aufgabenliste.zip`!

Entfernen von Elementen aus einer Collection I

Rumpf von `foreach` darf die unterliegende Collection **nicht** ändern:

```
Collection<Task> tasks = ...;

// throws ConcurrentModificationException !!!
for (Task t : tasks) {
  if (t instanceof PhoneTask) {
    tasks.remove(t);
  }
}
```


Entfernen von Elementen aus einer Collection II

Auch das Auflösen des `foreach` hilft **nicht**:

```
Collection<Task> tasks = ...;

// throws ConcurrentModificationException !!!
Iterator<Task> it = tasks.iterator();
while(it.hasNext()) {
  Task t = it.next();
  if (t instanceof PhoneTask) {
    tasks.remove(t);
  }
}
```

Korrekte Version

Während ein Iterator aktiv ist, dürfen Veränderungen an der Collection nur über den Iterator selbst vorgenommen werden.

```
Collection<Task> tasks = ...;

Iterator<Task> it = tasks.iterator();
while (it.hasNext()) {
  Task t = it.next();
  if (t instanceof PhoneTask) {
    it.remove();
  }
}
```

Verwende `remove` Methode aus dem `Iterator`-Interface!

Vergleichen von Objekten

Vergleich von Objekten

- Um Elemente zu sortieren, muss eine *Ordnung* auf ihnen definiert werden.
- Eine Anwendung benötigt manchmal auch eine andere als die "natürliche" Ordnung.
- Beispiel: Strings nicht lexikographisch, sondern der Länge nach vergleichen
- Java stellt dafür das `Comparator` Interface bereit.

Das `Comparator` Interface

```
interface Comparator<T> {
  public int compare(T o1, T o2);
}
```

- `compare (x,y)` liefert einen `int`-Wert
  - < 0, falls `x` kleiner als `y`
  - $= 0$, falls `x` gleich `y`
  - > 0, falls `x` größer als `y`
- Merkhilfe:
  `compare(x,y) < 0`, genau dann wenn $x < y$
  (Man verschiebt quasi $<$, analog für die anderen Vergleichsoperatoren)
- Forderung: Konsistenz mit `equals` bei Verwendung mit SortedSet or SortedMap
  - `compare (x,y) == 0` genau dann, wenn `x.equals(y)` den Wert `true` liefert

Beispiel: `Date`-Objekte vergleichen

```
import java.util.Comparator;
public class DateComparator implements Comparator<Date> {
  public int compare(Date d1, Date d2) {
    int result = d1.getYear() - d2.getYear();
    if (result == 0) {
      result = d1.getMonth() - d2.getMonth();
      if (result == 0) {
        result = d1.getDay() - d2.getDay();
      }
    }
    return result;
  }
}
```


Identisch vs. gleich

- Zwei Referenzen verweisen auf ein **identisches** Objekt, wenn sie beide die selbe Objektinstanz referenzieren.
  - Test mittels ==
- Zwei Referenzen verweisen auf **gleiche** Objekte, wenn sie denselben Inhalt repräsentieren.
  - Test mittels `equals()`
- Die `equals()` Methode wird von der Klasse `Object` geerbt und kann bzw. muss bisweilen überschrieben und angepasst werden.

Erwünschte Eigenschaften von `equals()`

- *reflexiv*: Für jede Referenz `x` mit x != null,sollte `x.equals(x)` den Wert `true` liefern.
- *symmetrisch*: Für jede Referenz `x` und `y` mit x,y != null, sollte `x.equals(y)` den Wert `true` liefern, genau dann wenn `y.equals(x)` `true` liefert.
- *transitive*: Für jede Referenz `x,y,z` mit x,y,z != null, sollte `x.equals(z)` den Wert `true` liefern, wenn `x.equals(y)` und `y.equals(z)` `true` liefert.
- *konsistent*: Mehrfache Aufrufe von `x.equals(y)` sollten `true` bzw. `false` liefern, solange die referenzierten Objekte zwischenzeitlich nicht verändert wurden.
- Falls `x != null`, sollte `x.equals(null)` immer `false` liefern.

Anpassung von `equals` in Klasse `Date`

```
public class Date {
    ...
    public boolean equals(Object obj) {
      if (this == obj) {
         return true;
      }
      if (obj == null) {
         return false;
      }
      if (!(obj instanceof Date )) {
         return false; // andere Klasse
      }
      Date other = (Date) obj;
      return this.year == other.year
        && this.month == other.month
        && this.day == other.day;
    }
}
```

Beispiel: Strings zuerst der Länge nach vergleichen

```
class SizeOrder implements Comparator<String> {
  public SizeOrder () {}
  public int compare (String x, String y) {
    if (x.length() < y.length())
      return -1;
    if (x.length() > y.length())
      return 1;
    //lexikalische Ordnung der String-Klasse
    return x.compareTo(y);
  }
}
```

Tests:

```
assertTrue(new SizeOrder().compare("two","three") < 0);
assertTrue("two".compareTo("three") > 0);
```


Zusammenfassung

- Anwendung des Collection Frameworks
  - Kommentiertes Beispiel in `Collections.zip` auf der Vorlesungsseite!
  - Entfernen von Elementen aus einer Collection mittels Iterator-`remove()`
- Vergleichen von Objekten und das `Comparator`-Interface
  - Verwendung im Zusammenhang mit sortierten Collections
  - Konsistenz mit `equals()`, Eigenschaften von `equals()`