TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Klassenattribute und -methoden

Beispiel: Klassenattribute I

```
class Uebungsgruppe {
  static int maxTeilnehmer = 30;
  private String gruppenname;
  private Set<String> teilnehmer;

  Uebungsgruppe(String gruppenname){
    this.gruppenname = gruppenname;
    this.teilnehmer = new HashSet<String>();
  }  ...

  int freiePlaetze() {
    return maxTeilnehmer - teilnehmer.size();
  }
}
```

Beispiel: Klassenattribute II

```
public static void main (String[] args) {
    // Verwendung ohne Erzeugung von Objekten
    int i = Uebungsgruppe.maxTeilnehmer;

    Uebungsgruppe u = new Uebungsgruppe("Montagsgruppe");
    Uebungsgruppe u2 = new Uebungsgruppe("Dienstagsgruppe");
    System.out.println("Freie Plaetze montags: " + u.
    freiePlaetze());
    System.out.println("Freie Plaetze dienstags: " + u2.
    freiePlaetze());

    // Veraendern des Klassenattributs
    Uebungsgruppe.maxTeilnehmer = 32;
    System.out.println("Freie Plaetze montags: " + u.
    freiePlaetze());
    System.out.println("Freie Plaetze dienstags: " + u2.
    freiePlaetze());
}
```

Frage

Was machen wir, wenn Übungsgruppen unterschiedliche Größen haben sollen?

Klassenmethoden

Klassenmethoden (statische Methoden) besitzen keinen impliziten Parameter (`this`).

Sie können nur auf Klassenattribute, ihre Parameter und ihre eigenen lokalen Variable zugreifen.

Beispiel: Klassenmethoden

Deklaration:

```
class String {
    ...
    static String valueOf( long l )  { ... }
    static String valueOf( float f ) { ... }
    ...
}
```

Anwendung/Aufruf:

```
String.valueOf( (float)(7.0/9.0) )
```

liefert die Zeichenreihe: "0.7777778"

Beispiele: Klassenattribute, -methoden

Charakteristische Beispiele für Klassenattribute und -methoden liefert die Klasse `System`, die eine Schnittstelle von Programmen zur Umgebung bereitstellt:

```
class System {
  final static InputStream in  = ...;
  final static PrintStream out = ...;
  static void exit(int status)     { ... }
  static long currentTimeMillis() { ... }
}
```

Die Klasse `PrintStream` besitzt Methoden `print` und `println`:

```
System.out.print("Das erklaert die Syntax");
System.out.println(" von Printaufrufen");
```

Geschachtelte Klassen

Begriffsklärung: Geschachtelte Klassen

Eine Klasse heißt in Java **geschachtelt** (engl. *nested*), wenn sie innerhalb der Deklaration einer anderen Klasse deklariert ist:

- als Komponente (ähnlich einem Attribut),
- als **lokale** Klassen (ähnlich einer lokalen Variablen)
- als **anonyme** Klassen bei der Objekterzeugung.

Ist eine Klasse nicht geschachtelt, nennen wir sie **global** (engl. *top-level*).

Beispiel: Statische Klassen

```
public class LinkedList {
    private Node entries = null;
    private int size = 0;

    private static class Node {
        int elem;
        Node next;
    }
    int getFirst() { ... }
    ...
}
```

Der Typ `Node` ist nur innerhalb von `LinkedList` sichtbar und zugreifbar. $\Rightarrow$ Information Hiding

Statische geschachtelte Klassen

- Können erben und Interfaces implementieren und generisch sein.
  Achtung: Eine geschachtelte Klasse übernimmt **nicht** die Superklassen und Interfaces der umschließenden Klasse!
- Zugreifbar (falls sichtbar)über zusammengesetzte Namen von außerhalb (z.B. `Map.Entry` aus Java Collections).
- Zugriff auf private Attribute und Methoden der umfassenden Klasse (über Objektreferenz)

Organisation von Klassen- und Interface-Deklaration

Strukturierung von Typdeklarationen

- Schachtelung von Klassen
- Gruppierung zu Übersetzungseinheiten
  → .java-Datei kann mehrere Typdeklarationen enthalten; max. eine `public`
- Gruppierung von Übersetzungseinheiten zu Paketen

Warum Strukturierung?

- Auffinden der Deklarationen und Verstehen des Zusammenhangs
- Übersetzungs-, Installationsorganisation
- Zugriffsrechte, Namensräume
- Pflege und Wartung

Warum Strukturierung?

- Auffinden der Deklarationen und Verstehen des Zusammenhangs
- Übersetzungs-, Installationsorganisation
- Zugriffsrechte, Namensräume
- Pflege und Wartung

Konzept **Modul**

- abgeschlossene funktionale Einheit
- getrennte Übersetzung
- Interaktion über Schnittstellen
- Komposition zu einem größeren Ganzen

Pakete in Java I

- Ein Paket ist eine endliche Menge von Übersetzungseinheiten (Dateien)
- Ein Paket hat einen Namen `p` (z.B. `java.util.concurrency`).
  - typischerweise klein geschrieben
  - besteht ggf. aus mehreren Teilen
- Paketname erscheint am Anfang dieser Übersetzungseinheiten.
  Syntax:

```
package  Paketname;
```

- Ein Paket `p` ist üblicherweise in einem Dateiverzeichnis mit Namen `p` abgelegt.
  Beispiel:

  Das Paket `java.lang` ist abgelegt in einem Verzeichnis `.../java/lang`

Pakete in Java II

- Namensräume: Ist `p` ein Paket, dann können alle Klassen `K` in `p` mittels `p.K` angesprochen werden (**vollständiger Name**) Innerhalb eines Pakets darf ein Klassenname max. einmal vorkommen.
- Es gibt **keine** Paket-Hierarchie/Unterpakete (z.B. `java.awt.event` kein Unterpaket von `java.awt`).
- Schnittstelle eines Java-Pakets: öffentlich sichtbare Elemente (Modifikator `public`).

Beispiel I

- Pakete `se1.list` und `se1.tree`
- Packetstruktur $\Rightarrow$ Verzeichnisstruktur

```
/home/karlheinz/uni/se1/list/LinkedList.java
/home/karlheinz/uni/se1/list/ListIterator.java
/home/karlheinz/uni/se1/list/Node.java
/home/karlheinz/uni/se1/tree/SearchTree.java
/home/karlheinz/uni/se1/tree/Node.java
```

Beispiel II

Die Java-Typen der Schnittstelle sind folgendermaßen definiert:

```
package se1.list;
public class LinkedList {...}
```

bzw.

```
package se1.tree;
public class SearchTree {...}
```

Die Klasse `Node` ist nur innerhalb des Pakets sichtbar, daher ist sie nicht als `public` deklariert (kein Modifikator):

```
package se1.list;
class Node {...}
```

Beispiel III

Adressierung über vollständigen Namen:

```
import se1.tree.SearchTree;

class LinkedListTest {
  public static void main(String[] args) {
    se1.list.LinkedList ll = new se1.list.LinkedList();
    java.util.LinkedList<String> ls =
      new java.util.LinkedList<String>();

    SearchTree t = new SearchTree();
    ....
  }
}
```

Überblick: Zugriffsmodifikatoren

Siehe Beispiele im Skript!

Hat Zugriff auf:	`public`	`private`	`protected`	ohne
Klasse selbst				
(nicht Sub-)Klasse, gleiches Paket				
Subklasse, gleiches Paket				
Subklasse, anderes Paket				
(nicht Sub-)Klasse, anderes Paket				

Lösung

Hat Zugriff auf:	`public`	`private`	`protected`	ohne
Klasse selbst	J	J	J	J
Klasse aus gleichem Paket	J	N	J	J
Subklasse, gleiches Paket	J	N	J	J
Subklasse, anderes Paket	J	N	J/N	N
(nicht Sub-)Klasse, anderes Paket	J	N	N	N

- Unterscheide: Anwendungsbeziehung vs. Vererbungsbeziehung!
- Subklasse anderes Paket: Wird Objekt der Superklasse erzeugt, ist `protected` nicht zugreifbar

Frage: Warum ist Zugriff `other.a` erlaubt?

```
1   public class C {
2     private int a;
3     ...
4     public boolean equals(Object o) {
5       if (o == null) {
6         return false;
7       }
8       if (o instanceof C) {
9         C other = (C)o;
10        return other.a == this.a;
11      } else {
12        return false;
13      }
14    }
15  }
16
```