03-05-H
-709.53

Informatik für Nichtinformatiker (4)

Prof. Dr. Udo Frese
Tobias Hammer

UML Klassendiagramme

Container, Blöcke und Iteratoren

FIfF

- **Forum Informatik für Frieden und gesellschaftliche Verantwortung**
- **13.-15.11.09**
- **Motto: Verantwortung 2.0**
- **Veranstaltungen**
  - 13.11.: Auftakt (19:00), Haus der Wissenschaft (Sandstr. 4-5) für die breite Öffentlichkeit
  - 14.11.: Arbeitsgruppen, GW2 B3009
  - 15.11.:Villa Ichon: Mitgliederversammlung FifF
- **Veranstaltungen öffentlich, Eintritt frei**
- **http://www.fiff.de/2009**

Was bisher geschah

- **`class` Klassenname < Oberklasse Definitionen end**
- **`initialize` legt Instanzvariable (`@name`) an und weist einen Anfangswert zu (meist als Parameter übergeben)**
- **wird beim Erstellen eines Objektes mit `new` aufgerufen**
- **Methoden von Oberklasse werden geerbt**
  - können neu definiert werden
  - werden mit super aufgerufen
- **Instanzvariablen (z.B. `@name`) nicht von außen zugreifbar**
  - Methode `name` zum lesen (kurz `attr_reader`)
  - Methode `name=` zum schreiben (kurz `attr_writer`)
  - für Wertekontrollen, Aktualisierung, Invalidierung
- **Klassenmethoden Class.method**
- **Zugriffskontrolle für Methoden: `public`, `protected`, `private`**
- **Variablen sind Platzhalter für *Referenzen* auf Objekte**

UML Klassendiagramme

UML Klassendiagramme

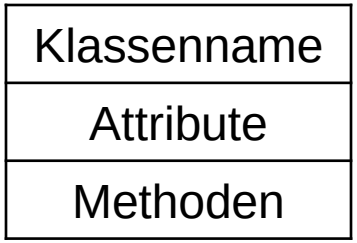
Was ist UML?

- **Unified Modeling Language**
- **ISO/IEC 19501 standardisierte Notationsform für Diagramme für objektorientierte Analyse und Entwurf**
- **Ziel: Gedanken zum Überblick des Entwurfs aufschreiben**
- **Ziel CRC-Karten: Gedanken zum Überblick des Entwurfs entwickeln**
- **Hier nur Klassendiagramm (Quelle: wikipedia / Klassendiagramm)**

UML Klassendiagramme

Grafische Elemente des Klassendiagramms

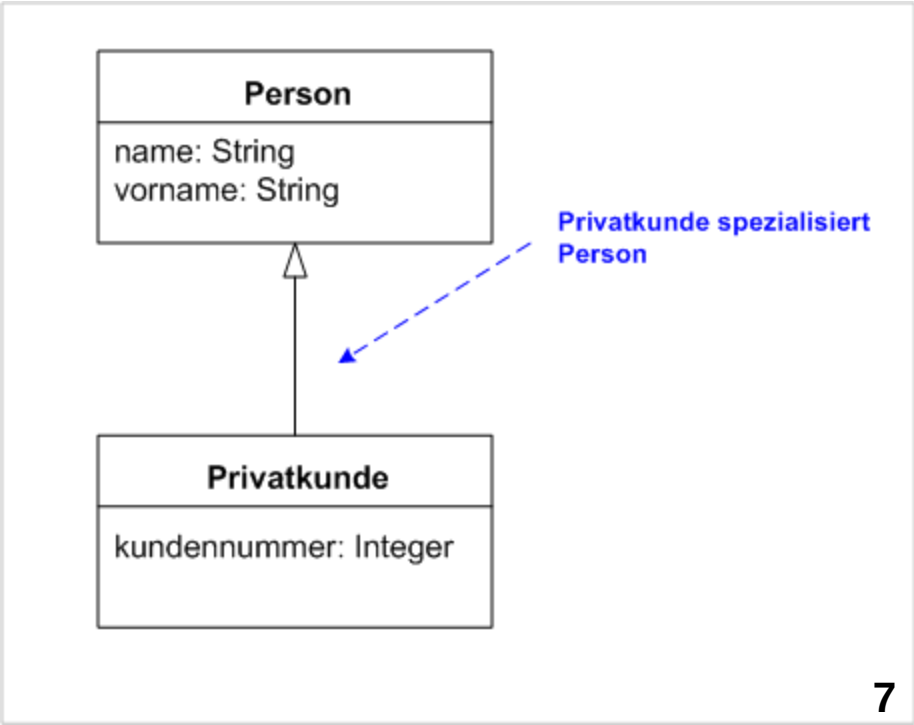
- **Klassen**
  - Kästchen (analog zu CRC Karten)
  - Klassenname
  - Attribute, die den Zustand der Klasse nach außen darstellen
  - Methoden, die die Funktionalität bilden
- **Verfeinerung der „Verantwortlichkeit“ in CRC-Karten**



UML Klassendiagramme

Grafische Elemente des Klassendiagramms

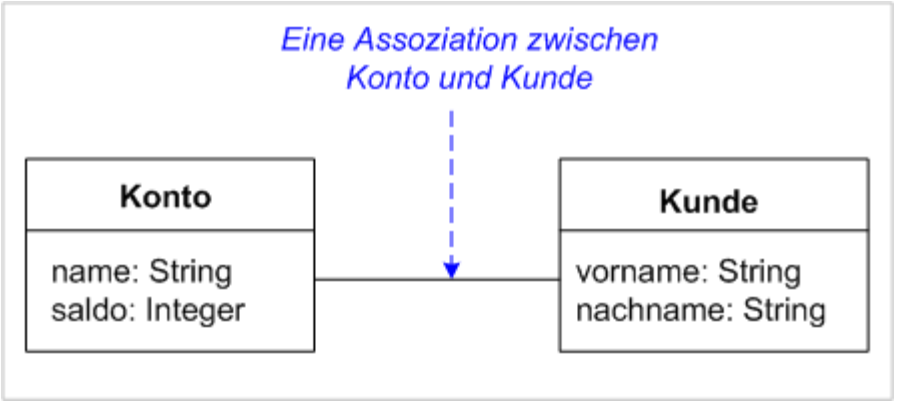
- **Vererbung („Generalisierung“)**
  - Dreieckspfeil auf Oberklasse
  - Unterklasse erbt Attribute und Methoden der Oberklasse
  - `is_a` Beziehung



UML Klassendiagramme

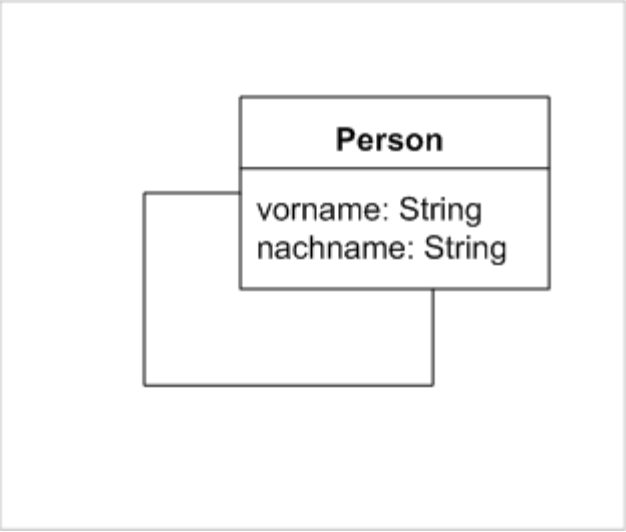
Grafische Elemente des Klassendiagramms

- **Assoziationen**
  - allgemeine Beziehung zwischen Objekten zweier Klassen
  - durch Text näher spezifiziert
  - Teil des Zustandes
  - für jedes Paar: sind sie assoziert?
  - für jedes Objekt: alle assozierten Objekte.
  - Kardinalitäten durch min…max; 0,1,…,* am Linienende näher spezifiziert
- **Assoziation kann eigenes Objekt werden**
- **Verfeinerung von Kollaboration in CRC Karten**



UML Klassendiagramme

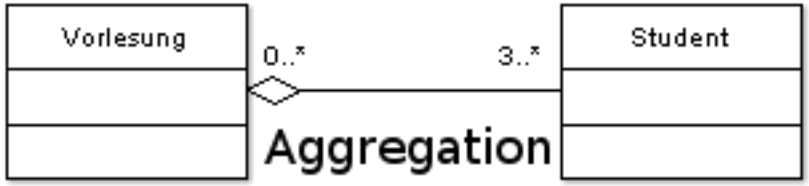
<u>Frage an das Auditorium:</u> Erklärt dies Diagramm! Was für eine Assoziation könnte es sein?



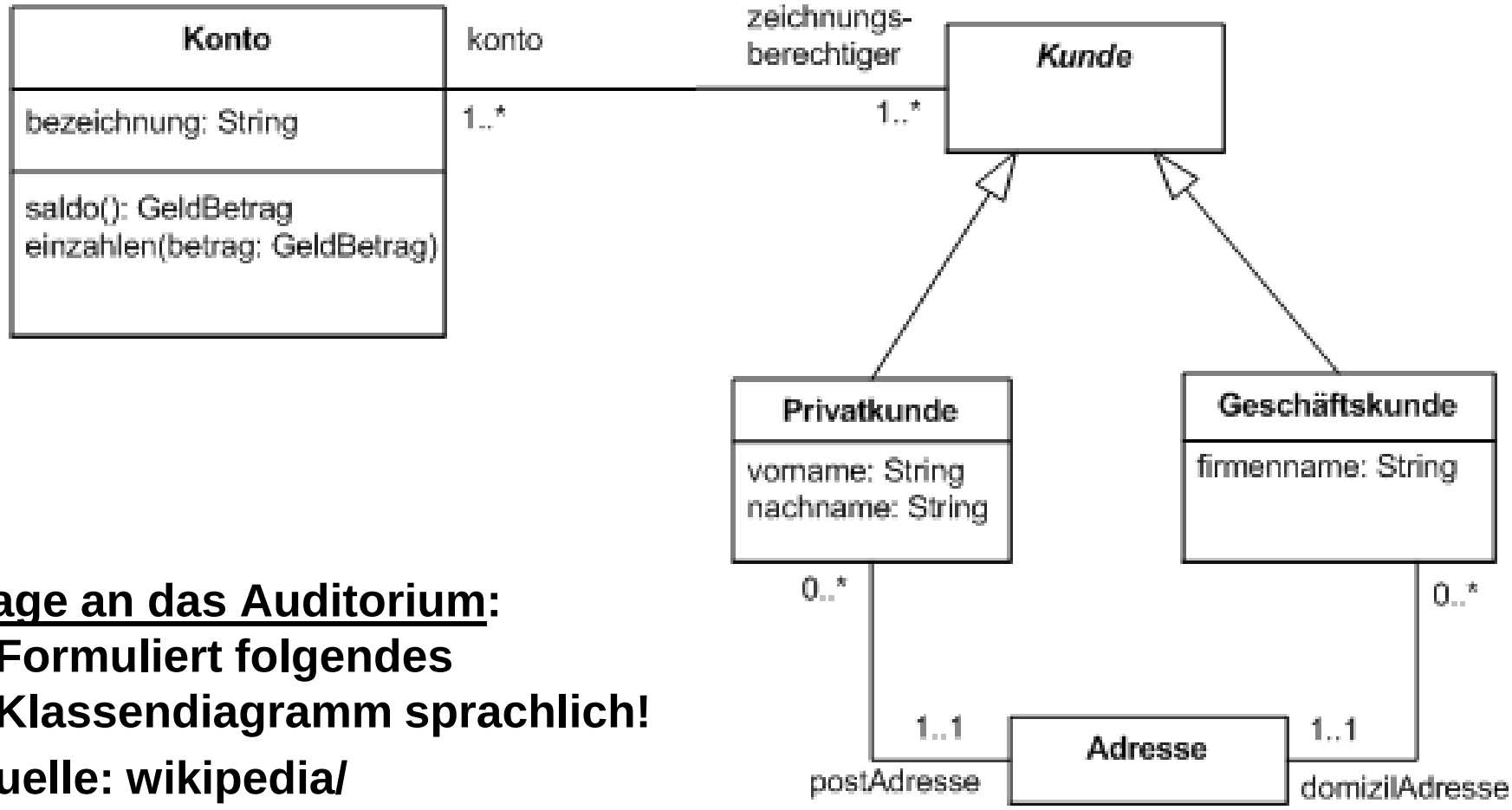
UML Klassendiagramme

Grafische Elemente des Klassendiagramms

- **Komposition**
  - Beziehung zwischen dem Ganzen und seinen Teilen
  - has_a Beziehung
  - Teil-Objekt nicht ohne Ganzes-Objekt sinnvoll
  - Kardinalität 1 beim Ganzen-Objekt
- **Aggregation**
  - wie Komposition, aber Teil-Objekt auch ohne Ganzes-Objekt sinnvoll



UML Klassendiagramme



<u>Frage an das Auditorium</u>:
Formuliert folgendes Klassendiagramm sprachlich!

(Quelle: wikipedia/ Klassendiagramm)

UML Klassendiagramme

Frage an das Auditorium: Welche Beziehungen gibt es?

Server-Raum

Server

Gerät

Raum

Festplatte

Computer

Büro

Desktop-PC

PDA

Container, Blöcke und Iteratoren

- **Kapitel 4 aus D. Thomas et al., „Programming Ruby“**
- **Rubyprogramm in inifrese0904_container.rb**

Container, Blöcke und Iteratoren

- **Was sind Container?**
- **Engl. Behälter**
- **Objekte, deren Aufgabe es ist, mehrere (viele) andere Objekte zu halten und systematisch zugreifbar zu machen**
- **Container betrachtet seine Einträge als `Object`**
- **Ruby**
  - Array: Zugriff über natürliche Zahlen, mit Reihenfolge
  - Hash: Zugriff über beliebige Schlüsselobjekte (z.B. Zeichenketten)

Container, Blöcke und Iteratoren

Array

- fortlaufende endliche Folge von Objekten, indiziert durch natürliche Zahlen, startend mit 0
- hinschreiben durch Objekte mit `,` getrennt und `[ ]` eingefasst
- sei beispielsweise `a=[1,3,7]` ein array
- Länge `a.length`
- Zugriff mit `a[index]`, wobei `a[0]` erstes Element, `a[1]` zweites, …
- negativer Index von hinten: `a[-1]` letztes Element, `a[-2]` vorletztes,
- `a[start, count]` ist Unterarray von `count` Elementen beginnend bei `start`, also `a[start], …, a[start+count-1]`
- `a[start..end]` ist Unterarray von Element `start` bis `end` (inkl.), also `a[start],…,a[end]`
- `a[start…end]` ist Unterarray von Element `start` bis `end` (exkl.), also `a[start],…,a[end-1]`

Container, Blöcke und Iteratoren

String als Array von Zeichen

- String ist im wesentlichen ein Array von Zeichen
- `s[start, count], s[start..end], s[start…end]` liefern Unterzeichenketten analog zum Array
- `s[index]` liefert den Code des Zeichens an Stelle `index` als Zahl
- mehr historisch motiviert
- für einzelnes Zeichen verwende: `s[index,1]`

Container, Blöcke und Iteratoren

- **Frage an das Auditorium: Wir wollen eine Laufschrift programmieren. Gebt eine Methode an, die eine Zeichenkette „rollt“, d.h. ein Zeichen nach links schiebt und den Anfang ans Ende bewegt.**

Container, Blöcke und Iteratoren

- **Frage an das Auditorium: Wir wollen eine Laufschrift programmieren. Gebt eine Methode an, die eine Zeichenkette „rollt“, d.h. ein Zeichen nach links schiebt und den Anfang ans Ende bewegt. Und anders herum?**

```
def roll_left (s)
  s[1..-1]+s[0,1]
end
```

Container, Blöcke und Iteratoren

- **Frage an das Auditorium: Wir wollen eine Laufschrift programmieren. Gebt eine Methode an, die eine Zeichenkette „rollt“, d.h. ein Zeichen nach links schiebt und den Anfang ans Ende bewegt. Und anders herum?**

```
def roll_right (s)
  s[-1,1]+s[0..-2]
end
```

Container, Blöcke und Iteratoren

Zuweisung von Array Elementen

- **sei a ein Array**
- **einzelnes Element mit `a[index]=` Zuweisen**
- **Zuweisung von Unterarrays: `a[start,count] = [el1, …]`**
  - löscht das alte Unterarray
  - fügt die rechte Seite als neues Unterarray ein

Container, Blöcke und Iteratoren

- **Frage an das Auditorium: Für eine Liste (Array) von Vorträgen (Strings) soll alle 3 Vorträge der Programmpunkt Kaffeepause eingefügt werden.**

Container, Blöcke und Iteratoren

- **Frage an das Auditorium: Für eine Liste (Array) von Vorträgen (Strings) soll alle 3 Vorträge der Programmpunkt Kaffeepause eingefügt werden.**

```
def insert_coffeebreak(talks)
   talks = talks.dup
   index = 3
   while index<talks.length
      talks[index,0] = „Coffeebreak“
      index += 4
   end
   talks
end

my_talks = [„Ruby“, „Python“, „Smalltalk“, „Lisp“,
   „C“, „C++“, „Pascal“, „Haskel“, „Smalltalk“,
   „Ada“, „C#“, „Java“]
```

Container, Blöcke und Iteratoren

Hash

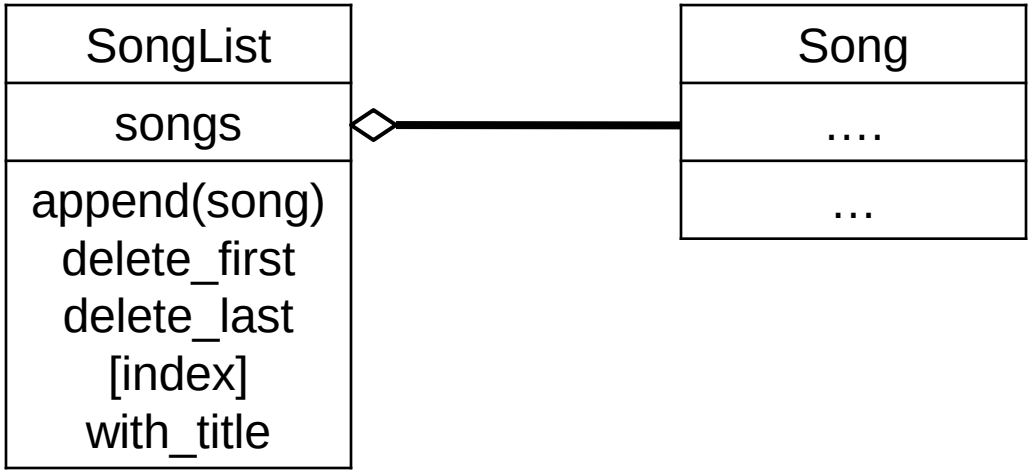
- alias assoziative arrays, maps, dictionaries
- Zuordnung von Objekten zu Schlüsseln
- beliebige Schlüsselobjekt (nicht nur Zahlen)
- keine Ordnung
- hinschreiben durch Schlüssel => Objekt, getrennt durch `,` umrahmt von `{}`
- Sei `h = {„Mensch“=>2, „Hund“=>4, „Spinne“=>8, „Flamingo“=>1}`
- Zugriff durch `h[schluessel]`; liefert Objekt, falls vorhanden, sonst `nil`
- Zuweisung über `h[schlussel]=`; überschreibt, falls vorhanden und fügt sonst hinzu
- statt `nil` kann Nullobjekt beim initialisieren definierten werden

Container, Blöcke und Iteratoren

Beispiel SongList

- **Klasse SongList, für**
  - Liste der Songs, die die Jukebox kann
  - Liste der Songs, die zum Spielen eingereiht sind

Klasse: SongList	
Liste von Songs mit Reihenfolge halten und verwalten	Song



Container, Blöcke und Iteratoren

Beispiel SongList

- **Methoden `initialize`, `append`, `delete_first`, `delete_last`, `[]` durch Rückgriff auf `Array` Methoden (Delegation)**
- **Frage an das Auditorium: Macht Vorschläge für ein Rubyprogramm!**

Container, Blöcke und Iteratoren

- **Beispiel SongList**

```
class SongList
  attr_reader :songs
  def initialize
    @songs = Array.new
  end
  def append(song)
    @songs += [song]
    self
  end
  def delete_first
    first = @songs[0]
    @songs = @songs[1..-1]
    first
  end
  def delete_last
    last = @songs[-1]
    @songs = @songs[0..-2]
    last
  end
  def [](index)
    @songs[index]
  end
end
```

Container, Blöcke und Iteratoren

Iteratoren

- **Wie programmiert man with_title?**
- **Intuitiv: Laufe durch die Liste, bis man den passenden Eintrag findet**

```
class SongList
  def with_title (title)
    i = 0
    while i<@songs.length do
       if title==@songs[i].name
         return @songs[i]
       end
    end
    return nil
  end
end
```

Container, Blöcke und Iteratoren

Iteratoren

- **Greift unnötig stark in das Array ein**
- **besser: Array weiß, wie man es durchläuft, SongList weiß, was gesucht wird**

```
class SongList
  def with_title (title)
    @songs.each do |song|
       if title==song.name
         return song
       end
    end
    return nil
  end
end
```

Container, Blöcke und Iteratoren

Iteratoren

- **noch besser: Array weiß, wie man es durchsucht, SongList weiß, was gesucht wird**

```
class SongList
  def with_title (title)
    @songs.find do |song|
       title==song.name
    end
  end
end
```

Container, Blöcke und Iteratoren

Iteratoren

- **Array.collect {Block}**
- **Wendet eine Rechnung auf jedes Element des Arrays an und macht aus den Ergebnissen ein neues Array**

```
class SongList
  attr_reader break_duration
  def durations_with_breaks
    @songs.collect do |song|
       song.duration+break_duration
    end
  end
  def total_duration_with_breaks
    total = 0
    @songs.each do |song|
       total += song.duration
                +break_duration
    end
    total
  end
end
```

Zusammenfassung

- **Unified Modeling Language (UML) Klassendiagramme**
  - eine Stufe detaillierter, als CRC Karten
  - Klassen: Kasten mit Name, Attribute, Methoden
  - Pfeil für Assoziationen mit Kardinalitäten
  - Pfeil für „ist ein“ (Unterklasse)
  - Pfeil für „hat ein“ (Teile eines Ganzen: Aggregation und Komposition)
- **Container Array: `[e1, e2,…]`**
  - Zugriff: `[idx]`, `[start,count]`, `[start..end]`, `[start…excl]`
- **Container Hash: `{k1=>o1, k2=>o2}`**
  - Zugriff `[key]`, `nil`, falls noch nicht vorhanden
- **durchlaufen mit `.each`**
- **elementweise abbilden mit `.collect`**
- **durchsuchen mit `.find`**