Folien zum Lehrmodul

Objektorientierter Entwurf

Lernziele:

- Differenzen zwischen objektorientierter Analyse und objektorientierten Entwurf kennen
- Entwurfsklassendiagramme in UML kennen und erstellen können (Übung!)

notwendige Vorkenntnisse:

- Grundlagen der objektorientierten Programmierung
- objektorientierte Analyse in UML

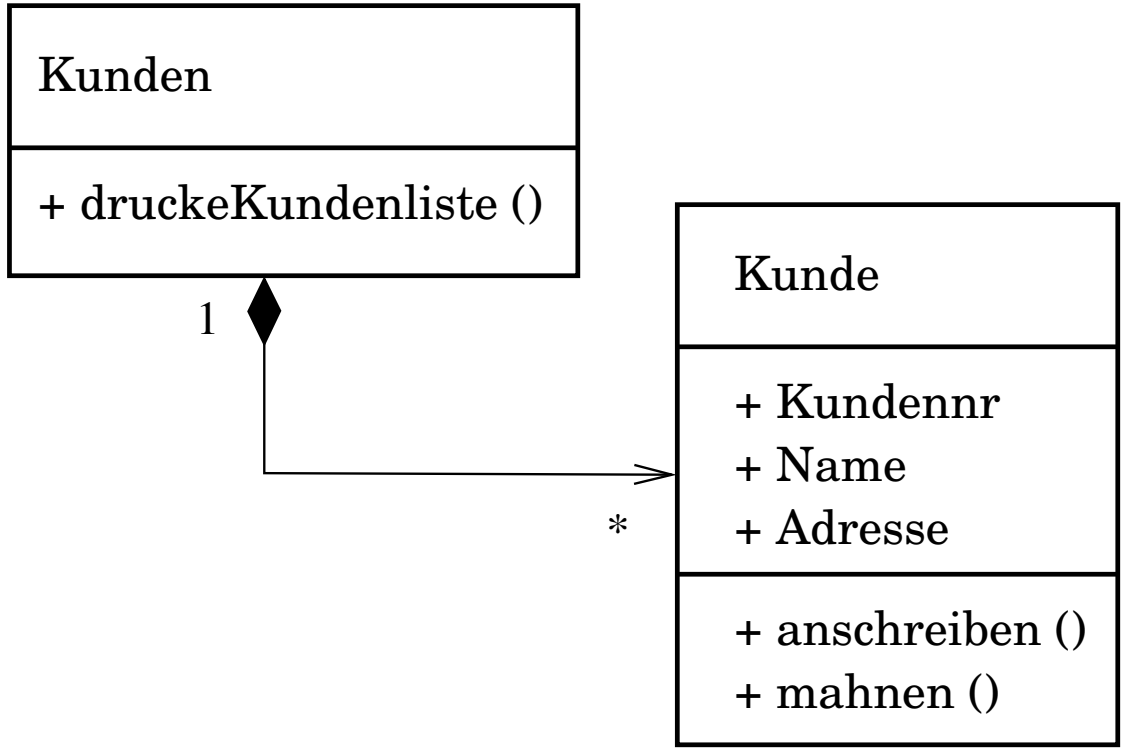
Inhaltsverzeichnis

1 Klassen

1 Entwurfsklasse repräsentiert i.d.R. 1 Programmklasse
(Ausnahmen: z.B. Framework-Komponenten)

graphische Darstellung:

wie Analyseklassen, mit Erweiterungen



explizite Verwaltung der Instanzen eines Typs durch **Container**-Objekte (keine implizite Objektverwaltung wie bei Analyseklassen)

hierzu: **Container-Klasse**; definiert Zugriffsstruktur, z.B. Array, Liste usw.

Operationen, die mit Mengen von Objekten arbeiten, der Container-Klasse zuordnen

z.T. komplexere Formen der Objektverwaltung

Resümee:

- strukturelle Differenzen zwischen Analysemodell und Entwurf
- Entwurf deutlich detail- und umfangreicher als das Analysemodell

2 Attribute

Attributspezifikation in der UML:

```
Sichtbarkeit Attributname: Typ = Anfangswert {
Merkmale }
```

- bei der Angabe von Attributnamen und Attributtyp die Gegebenheiten der Programmiersprache zu beachten
- [UML1.3], 3.25: “the details of the attribute type expressions are not specified by UML.”

2.1 Sichtbarkeit

Sichtbarkeitsfestlegungen wie bei C++ und Java:

`+`	**public**	für alle Klassen sichtbar
`#`	**protected**	für diese Klasse und ihre Unterklassen sichtbar
`-`	**private**	für diese Klasse, aber nicht für ihre Unterklassen sichtbar

public-Attribute: schlecht, offener Typexport, möglichst vermeiden

protected-Attribute: bedenklich

möglichst private-Attribute und bei Bedarf Operationen `liesX` bzw. `setzeX`

`{frozen}` in Attributspezifikation: Attribut kann nach Initialisierung nicht mehr verändert werden

2.2 Abgeleitete Attribute

durch einen vorgestellten `/` gekennzeichnet

Umsetzung von abgeleiteten Attributen aus dem Analysemodell:

- Operation, die den Wert des Attributs berechnet
- Attribut, das mit Ausgangsgrößen konsistent gehalten wird (i.w. Puffer)

2.3 Klassenattribute

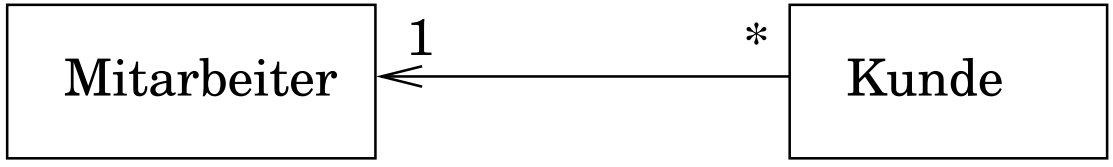
dargestellt mit Unterstreichung

können auch abgeleitet sein, s.o. (Operation vs. Attribut an der Container-Klasse)

Im Programm ggf. Klassenattribut verwenden (z.B. `static` -Attribut in Java)

3 Assoziationen

Entwurfsassoziationen sind *gerichtet*, Anzeige durch Pfeil



repräsentieren Referenzen / Zeiger im Programm; Sichtbarkeit wie bei Attributen, Angabe als Präfix des Rollennamens

bidirektionale Assoziation: gegenläufige Referenzen
Darstellung mit 2 oder 0 Pfeilspitzen
nur gemeinsam erzeugen / ändern / löschen

Kardinalitäten: wie bei Analyse-Assoziationen

attributierte Assoziationen: assoziative Klasse, wie bei Analyse

3.1 Umsetzung von Entwurfsassoziationen in Programme

Kardinalität 0:1 oder 1 : durch einen Zeiger in der Klasse realisierbar

feste Anzahl / Maximalzahl : ggf. Array von Zeigern verwenden

andere Kardinalitäten : dynamische Datenstruktur

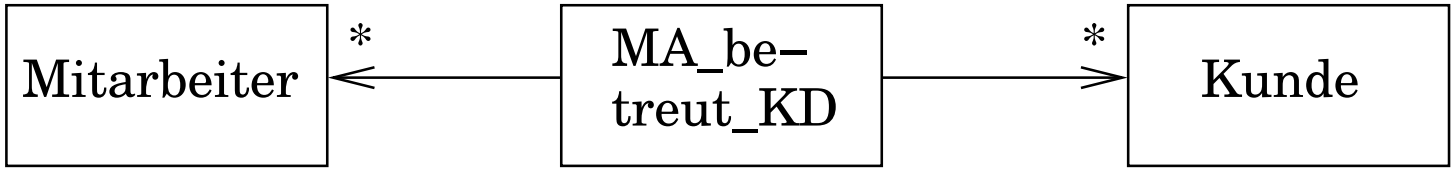
Kompositionen: Komponenten ggf. direkt in das Ganze einbetten (einfacheres Anlegen)

3.2 Umsetzung von Analyse-Assoziationen in Entwurfsassoziationen

hier nur binäre Assoziationen

1. 1:1-Umsetzung
2. Einsatz von Assoziationsobjekten: entspricht Verbindungstabelle

   weniger effizient, aber involvierte Klassen müssen nicht verändert werden



4 Operationen

Angaben zu jeder Operation:

1. die **Signatur**: Name der Operation, Folge der Parametertypen, Typ des Rückgabewerts
   Angaben zu jedem Parameter:
   - die Übergabeart `in` , `out` bzw. `inout`
   - der Name
   - ein Vorgabewert
2. Beschreibung der Wirkung der Operation
3. Sichtbarkeitsangabe wie bei Attributen
4. Kennzeichnung, ob die Operation abstrakt ist

überladene Operationsnamen möglich

Abstrakte Operationen: gemeinsame Schnittstelle für Unterklassen (keine Implementierung in der Klasse, in der sie stehen)

Name kursiv oder das Merkmal `abstract`

5 Schnittstellen

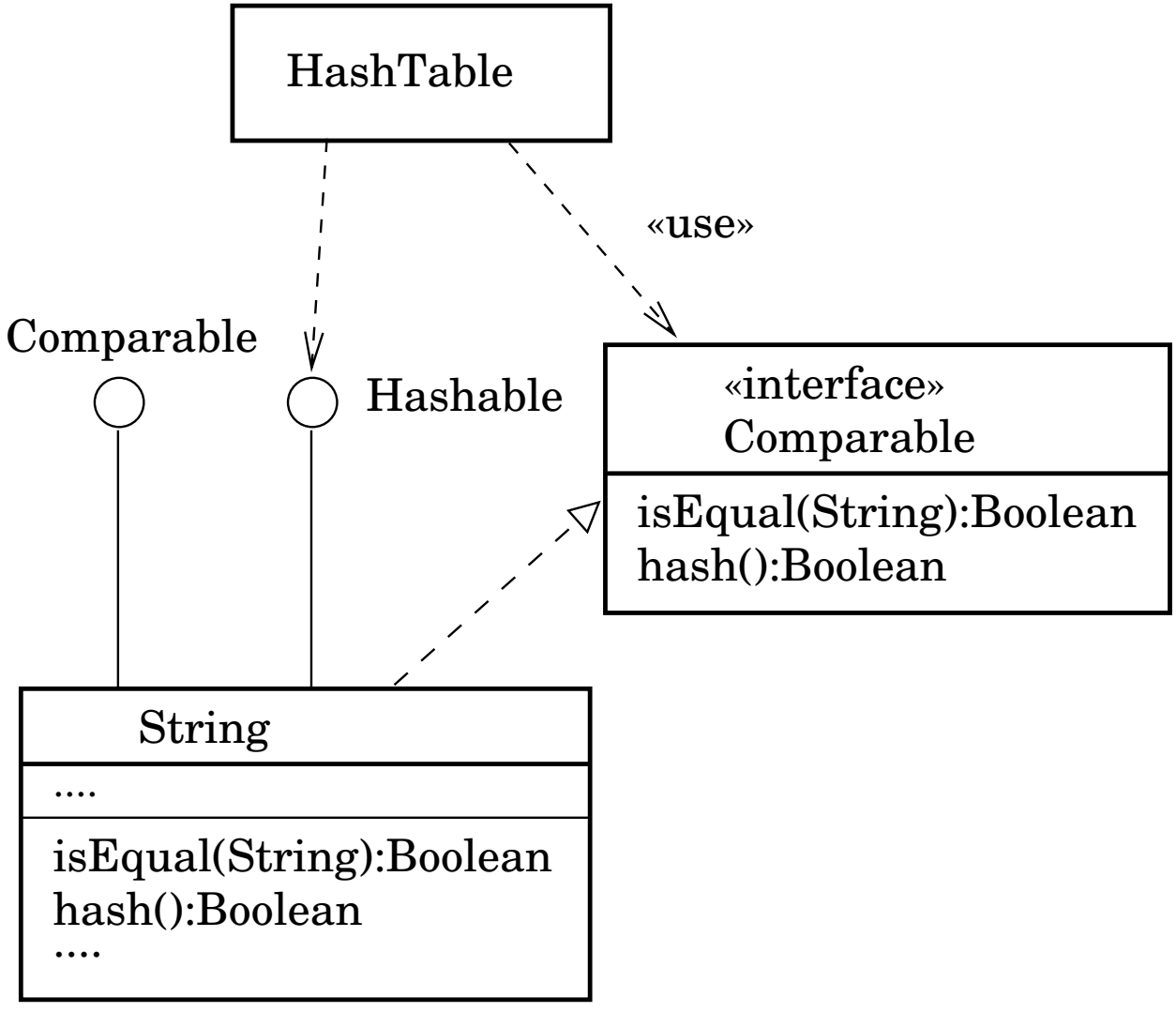
Schnittstelle = Klasse, die nur abstrakte Operationen enthält (keine Attribute, keine ausgehenden Assoziationen)

kann Ziel von Assoziationen sein

Darstellung:

- wie eine Klasse, aber mit dem Stereotyp «`interface`» , der Abschnitt für die Attribute entfällt
- kleiner Kreis, Name der Schnittstelle daneben

  (Operationen hier nicht dargestellt)


HashTable
«use»
Comparable
Hashable
«interface»
Comparable
isEqual(String):Boolean
hash():Boolean
String
....
isEqual(String):Boolean
hash():Boolean
....