2.4 Datenabstraktion, Objektorientierung, Java

2.4.1 Grundlegende Begriffe der Objektorientierung (s. auch Alp2)

- Klassen sind wie Module eine Einheit der Softwarearchitektur.
- Große Systeme werden aus solchen Einheiten zusammengebaut.
- Meist auch Übersetzungseinheiten.
- Klassen und Module bilden Namensräume, deren Elemente vor äußerem Zugriff geschützt werden können.
- Im Gegensatz zu (klassischen) Modulen definieren Klassen gleichzeitig einen Typ.
- => Exemplare von Klassen – nicht Modulen - werden instantiiert

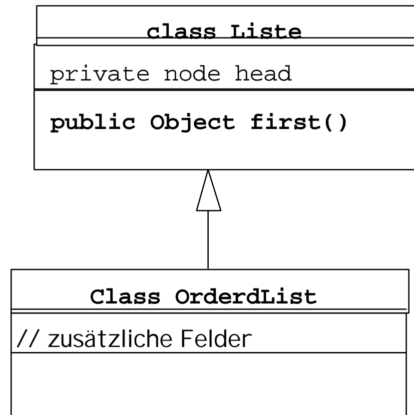
- Klassen
  - Unterklassen übernehmen automatisch die Eigenschaften ihrer Oberklassen und fördern so die Wiederverwendung.
  - Unterklassen können ererbte Eigenschaft redefinieren und erweitern und fördern so die Erweiterbarkeit.

```
class List
  // model:  [Int]
  // inv: ...
{private Node head;
 public Object first(){..}
 ...
 public Object next(){..}
 public void put (Object x){..}
}
```

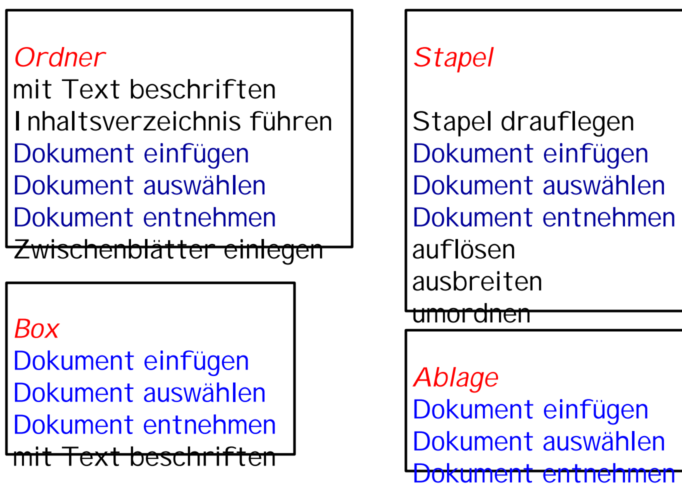
```
class OrderedList extends List
  // model:  ordered [Int]
  // inv: ...
{
  public Object min(){
      return first();}
 ...
 public void put (Object x){..}
   // redefine "put.List"
 public int somethingElse(){..}
```

Subklassen – graphische Notation

UML-Notation



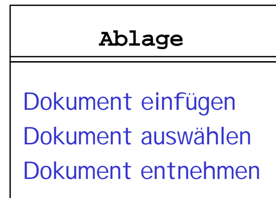
Klassenhierarchie als Modellierungsmittel



Beispiel: H. Züllekoven, 1998

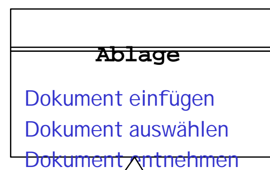
Klassenhierarchie als Modellierungsmittel

Oberbegriff



Unterbegriffe

Klassenhierarchie als Modellierungsmittel

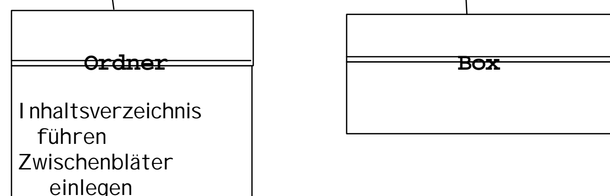


Gemeinsamkeiten geben Anlass zu neuem Oberbegriff

Archiv

beschriften

Taxonomie



Klassen

- Ober- und Unterklasse:

  Fachlich

  Ausgehend von gemeinsamen Eigenschaften bestimme Ähnlichkeiten zwischen Begriffen.

  Diese Gemeinsamkeit generalisierend in einem Oberbegriff ausdrücken

  Durch solche Generalisierung oder Klassifikation entstehen Begriffshierarchien (Taxonomien).

  Sie sind Grundlage der Aufteilung in Ober- und Unterklassen.

Klassen

Technisch

Alle Beschreibungen der Oberklasse sind durch Vererbung zunächst auch Beschreibungen ihrer Unterklassen

In Unterklasse können geerbten Beschreibungen spezialisiert werden.

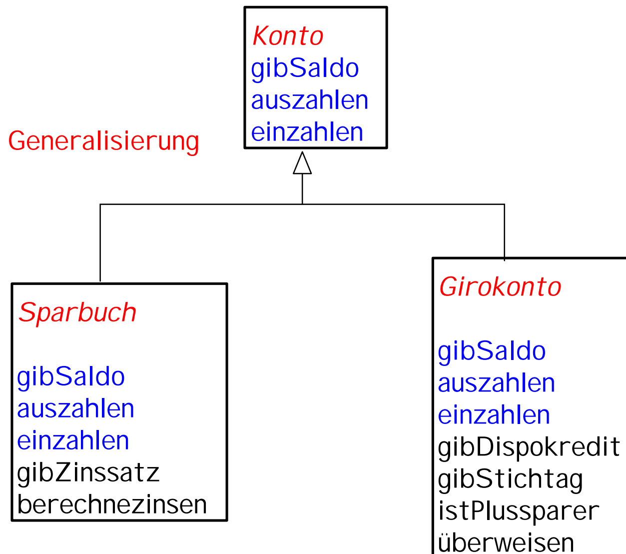
Dabei werden Operationen

- redefiniert, wenn eine neue Implementation in der Unterklasse eine vorliegende Implementation in Oberklasse ersetzt (überschreibt, verdeckt).
- hinzugefügt, wenn noch keine namensgleiche Operation in einer Oberklasse existiert.
- definiert (oder implementiert), die in Oberklassen nur spezifiziert sind.

Auch Attribute können in Unterklassen hinzugefügt werden.

Klassen

Nochmal: Generalisierung



Klassen und Vererbung

```
public class Konto {
 public Konto() { ... };
 public void  einzahlen(float b){...}
 public float gibSaldo(){...}
 public float auszahlen(float b){...}
 protected float saldo;
 };
```

```
public class Sparbuch extends Konto {
public Sparbuch(Zinssatz z){...}
public Zinssatz gibZinssatz(){...};
public void berechneZinsen(){saldo = saldo +..;};
public void überweisen(float b, Konto z){...};
protected Zinssatz zinssatz;
};
```

Exkurs: Sichtbarkeit

sichtbar in	`public`	`protected`	(empty)	`private`
gleicher Klasse	ja	ja	ja	ja
anderswo in gleichem Paket	ja	ja	ja	nein
Unterklasse in **anderem** Paket	ja	**ja**	**nein**	nein
anderswo in **anderem** Paket	ja	nein	nein	nein

Verwendung von Klassen

- Ein Objekt der Klasse **`Sparbuch`** kann neben allen Operationen der Klasse **`Sparbuch`** auch alle Operationen der Klasse **`Konto`** verwendet werden.

```
Sparbuch einSparbuch = new Sparbuch(3.0);
einSparbuch.einzahlen(1000);
einSparbuch.berechneZinsen();
if (einSparbuch.gibSaldo()) > 500)
{
einSparbuch.auszahlen(500);
System.out.print("Der neue Saldo beträgt ");
System.out.println(einSparbuch.gibSaldo());
}
```

Klassen und Typen

2.4.2 Polymorphie

```
Sparbuch einSparbuch = new Sparbuch(3.0);
Konto einKonto;
KontoDrucker = new KontoDrucker();
KontoDrucker.setzeKonto(einSparbuch);
...
einKonto = einSparbuch;
```

Wie passt das zum Typsystem?

```
public class Konto {
...}
```

```
public class KontoDrucker{
public KontoDrucker();
public void setzeKonto(Konto k);
..};
```

```
public class Sparbuch extends Konto {..}
```


Polymorphie

- Polymorphe Zuweisung

  Objektvariable einer abgeleiteten Klasse kann an Variable einer Oberklasse zugewiesen werden

  - direkte Zuweisung
  - Parameterübergabe, da Objektreferenzen mit "call-by-value" übergeben werden:
    formalerParam = aktuellerParam

- Fragen:
  - Zusammenhang Typ – Klasse?
  - Zuweisung von Objekt der Oberklasse an Variable der Unterklasse möglich?
    y = new einKonto; einSparbuch x = y
  - wie findet Laufzeitsystem die "richtige" Implementierung (**redefine**!)
  - wie steht es mit Rückgabewerten?

Klassen und Typen

- Typ ist Spezifikationskonzept
  - Legt äußere Sicht auf Bezeichner und Programmobjekte des Typs fest
  - Definiert erlaubte Operationen (besonders: ADT)
- Klasse legt fest, wie Objekte ("Exemplare") der Klasse
  - aufgebaut sind (Konstruktion)
  - sich verhalten
  - ... und den Typ des Objekts (... und Spezifikation)
- Klassenhierarchie:
  Generalisierung / Spezialisierung von Typ und Konstruktion

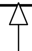
  - Unterschiedliche OO-Typsysteme möglich
  - Betrachten hier nur Typsystem sog. stark-typisierter objektorientierter Programmiersprachen (Java, NICHT Smalltalk)

Klassen, Typen, Vererbung

- Unterklassen erben alle Eigenschaften (Deklarationvon Objekten, Prozedurenund Funktionen) ihrer Oberklassen und damit auch die Schnittstelle, die den Typ festlegt
- Das Verhalten von Objekten der Unterklasse soll hinsichtlich des gemeinsamen Teils der Schnittstelle mit der Oberklasse "gleich" sein
  Aber: Semantik redefinierter Methoden?

Klassen, Typen, Vererbung

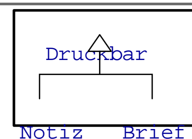
```
class Druckbar {
public void anDenAnfang() {...}
public String nächsteZeile() {...}
public boolean amEnde(){...}
}
```



```
class Brief extends Druckbar {
public void anDenAnfang() {...}
public Adresse empfänger() {...}
}
```

```
class Notiz
    extends Druckbar {
public void
    anDenAnfang() {...}
public void
    anfügen(String[]t)
{...}
}
```


Klassen, Typen, Vererbung




```
private Drucker dr;
private Druckbar d;
private Notiz n;
private Brief b;
private Druckbar d1, d2;
...
dr = b;
dr.drucken(n);
d1 = d2;
d = n;
d = b;
dr.drucken(n);
dr.drucken(b);
```

beliebige Zuweisung: unzulässig oder fehleranfällig

typgleiche Zuweisung: restriktiv und sicher

polymorphe Zuweisung: flexibel und (meist) sicher

Statischer und dynamischer Typ

- Bezeichner haben durch Deklaration **statischen Typ**

```
private Druckbar d; private Notiz n; private Brief b;
```

- Als Folge polymorpher Zuweisungen kann ein Bezeichner verschiedenartige Objekte benennen oder "referenzieren" (spätes Binden)
- Typ des gerade referenzierten Objekts ist der **dynamische Typ** des Bezeichners
- Menge der dynamisch möglichen Typen in statisch typisierten objektorientierten Sprachen durch Vererbung eingeschränkt
- Keine dynamischen Typen in statisch getypten, nicht objektorientierten Sprachen (Pascal, Modula-2)

"frühes Binden " - durch Übersetzer

Statischer und dynamischer Typ

```
class Drucker{
  public void drucken(Druckbar d){...}
  ...
}
private Druckbar d;
private Notiz n;
private Brief b;
...
d = n;
d = b;
dr.drucken(n);
dr.drucken(b);
```

Ist hier **d.empfänger()** erlaubt?? Schließlich ist der dynamisch Typ von d Brief

Nein! Statischer Fehler (Syntax)

Statischer und dynamischer Typ

```
private Druckbar d1,d2;
private Brief b;
...
d = b;
d.anDenAnfang():
```

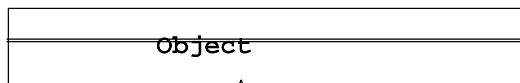
typsicher, da nur (dynamisch) Methoden von `class Druckbar` verwendet werden können

```
d = (Brief) b;
(Brief) d.empfänger();
d2 = (Brief) d1;
```

unsicher, da (dynamisch) Methoden von `class Brief` auf Objekt angewendet werden, auf das Variable vom Typ Druckbar zeigt.

Kann zu `ClassCastException` führen

Statischer und dynamischer Typ




```
class Druckbar {
public void anDenAnfang() {...}
public String nächsteZeile() {...}
public boolean amEnde(){...}
}
```

```
class Brief extends Druckbar {
public void anDenAnfang() {...}
public Adresse empfänger() {...}
}
```

Aufruf einer Methode an Objekt der Klasse A wird in Klasse A gesucht, wenn nicht gefunden, nacheinander in den Oberklassen.
Methode muss typkorrekt sein (Signatur, Typkompatibilität)

Kompatibel: gleich oder in Vererbungsbeziehung (sie oben)

Object

```
class Object

public String toString()
// Ausgabe des Objects als String
public boolean equals (Object obj)
// Referenzvergleich
protected Object clone ()
// Objektkopie
protected void finalize ()
// Aufräumen vor Garbage Collection
...
```