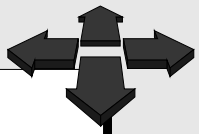
Einführung in die Softwaretechnik

Bausteine der Modellierung: Typ, ADT, Datenstruktur, Klasse

Modellierung



- **Überblick**
  - Eigenschaften von Software
  - Softwarequalität
  - Zyklus: Modellierung / Implementierung / Evaluierung
  - Typbegriff
  - Abstrakte Datentypen
  - Datenstrukturen
  - Klasse
- **Einordnung**
  - Teil 1: Große Software
  - es folgt: Architektur
- **Lernziele**
  - Qualitätsmerkmale ordnen
  - Begriff Abstrakter Datentyp und damit zusammenhängende Konzepte verstehen
  - Modellierung verstehen

Christiane Floyd, Guido Gryczan, Julian Mack

Eigenschaften von Software

- **Software ist abstrakt**
  - nicht sinnlich wahrnehmbar
- **Software besteht aus Texten. Diese Texte**
  - sind sehr umfangreich,
  - müssen detailrichtig sein
- **Software ist extrem komplex**
  - mehrere Größenordnungen, hohe Zahl von Zuständen
- **Software besteht aus einem einheitlichen "Stoff"**
  - formale Sprache
- **Software-Fehler sind schwer lokalisierbar**
- **Software ist "weich" d.h. änderbar**
- **Die Grenze zwischen Software und menschlicher Tätigkeit ist frei wählbar**

Einführung in die Softwaretechnik AB Softwaretechnik

Softwarequalität (1)

- **Gebrauchsqualität** bezieht sich auf
  - Relevanz und Effektivität
  - Betriebsqualität
  - Benutzungsqualität
- **Produktqualität** bezieht sich auf
  - Korrektheit
  - Verständlichkeit
  - Änderbarkeit
  - Wiederverwendbarkeit
- **Prozeßqualität**
  - Über den Prozeß die Produkt- und Einsatzqualität sichern!

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

Softwarequalität (2)

- **Qualität ist das übergeordnete Ziel der Softwaretechnik**
- **Qualität muß gesichert werden**
  - analytische Qualitätssicherung
    - » meßbare Merkmale des fertigen Produkts
  - konstruktive Qualitätssicherung
    - » Qualität durch den Prozeß gewährleisten
    - » Qualitätsmerkmale den Entwicklungsschritten zuordnen
    - » Qualitätskriterien und Überprüfung vorsehen
- **Qualität wird von anderen beurteilt**
  - frühzeitige Rückkopplung
  - konstruktive Kritik

Einführung in die Softwaretechnik AB Softwaretechnik

Von kleinen zu großen Programmen

- **Was heißt groß?**
  - viele Programmzeilen
  - behandelt Probleme aus einem Anwendungsbereich
  - mehrere Entwickler/innen
- **Probleme**
  - Zerlegung in ein Softwaresystem
  - Modellierung durch Abstraktion
  - Arbeitsteilung

Achtung:

wir sprechen hier zunächst von Programmierung in einem allgemeinen Sinne. Wir befassen uns zunächst nicht mit Software als Produkt!

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

Exkurs: Des Gebos-System der RWG - Große Software -

Technische Grundlagen:

OS/2 2.1
C++ und CommonView
RCS-System
Sniff+ auf Unix-Rechnern
Parts

Derzeitiger Stand:

- Lokale Server für Datenbank
- Lokale Server für Bankdienste
- transaktionsorientierte Großrechnerverbindung

Aktuelle Entwicklung:

- volle Client-Server Funktionalität
- objektorientierte ServiceCenter
- verteilte Objekte (DCE, Corba)
- Großrechnerverbindung OO
- ca. **50 Entwickler** arbeiten parallel
- Einsatz an mehr als **10.000 Arbeitsplätzen**

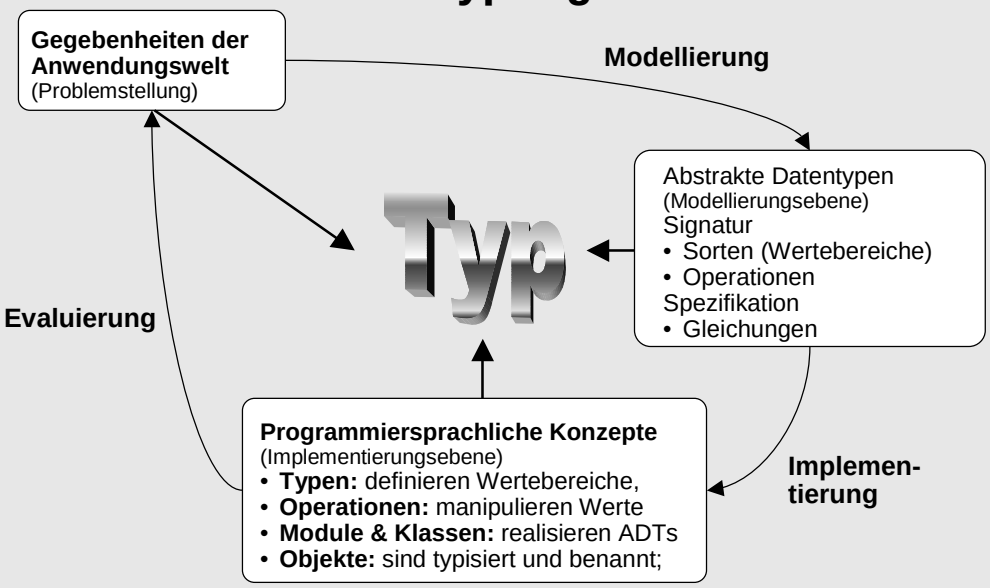
Klassen Stand 12/95:

1.556 Klassen
590 projektspezifisch
254 Materialien
244 Werkzeuge
92 Automaten

966 Basiskomponenten
43 Aspektklassen
85 Werte
69 Konverter und Tester
32 Behälter
127 Hostkommunikation
203 Unterklassen zu CV
28 DB-Anbindung
63 bankspezifische HW
105 Werkzeuge
211 Bankfachliche Bibl.

Einführung in die Softwaretechnik AB Softwaretechnik

Der Typbegriff



Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

Grundzyklus der Programmierung

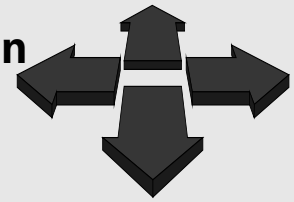
- **Wir unterscheiden vier Schritte :**
  - Zweckbestimmte **Abstraktion** anhand der Problemstellung,
  - **Modellierung** durch Entwurf und Spezifikation,
  - **Implementierung** durch Programmierung,
  - **Evaluierung** der Lösung durch Testen, Erproben und Bewerten.

Typbildung durch Modellierung

- **Von der Anwendungswelt ausgehend, gestatten Typen:**
  - die Charakterisierung von zu modellierenden Gebilden,
  - die Benennung dieser Charakterisierungen (benannte Typen),
  - die Weiterverwendung der Charakterisierung für andere, gleichartige Gebilde (weitere Exemplare desselben Typs),
  - die Beschreibung von Verwandtschaften zwischen unterschiedlichen Charakterisierungen (z.B. Spezialisierung, Benutzung von Basistypen, Vererbung).

Einführung in die Softwaretechnik AB Softwaretechnik

Abstrakte Datentypen



- **Abstrakte Datentypen sind zum einen Gegenstand der theoretischen Informatik. Dabei werden die Datentypen algebraisch formal spezifiziert.**
- **Abstrakte Datentypen sind aber auch ein wichtiges Entwurfsprinzip der objektorientierten Programmierung, um die innere Qualität von Software abzusichern. Wir skizzieren hier die Idee.**

Einführung in die Softwaretechnik AB Softwaretechnik

Abstrakter Datentyp: die Grundidee

- **Beschreibe einen *Datentyp* nur durch die *Operationen*, die auf ihn anwendbar sind und durch die *Bedingungen* ihrer Anwendung.**
- ***Verberge* die Realisierung (Implementation) des Datentyps vollständig.**

Softwaretechnisch kommen zusammen:
- Datenabstraktion,
- benutzerdefinierte Datentypen,
- Geheimnisprinzip.

Der Begriff "abstrakter Datentyp"

- **Nach *Informatik-Duden*:**
  - **Unter einem** Datentyp **versteht man die Zusammenfassung von Wertebereichen und Operationen zu einer Einheit.**
  - **Liegt der Schwerpunkt auf den *Eigenschaften*, die die Operationen und Wertebereiche besitzen, dann spricht man von** abstrakten Datentyp **(**ADT**).**
- **Im Sinne von *Meyer besteht* ein *ADT* aus:**
  - **Typen,**
  - **Funktionen,**
  - **Axiomen,**
  - **Vorbedingungen.**

Abstrakter Datentyp (ADT)

- Eine **Signatur** Σ ist ein **Paar (S, OP)**
  - **S** = Menge von Sorten (Benennung für Wertebereich),
  - **OP** = Menge von Operationssymbolen.
- In **S** gibt es ein ausgezeichnetes Element **t**
  - die interessierende Sorte,
  - alle anderen Sorten heißen Basissorten.
- Jedes Element von **OP** bezeichnet eine Abbildung zwischen den durch die Sorten bezeichneten Wertebereichen
- Eine Spezifikation ist ein Paar **(Σ, E),** dabei ist
  - Σ eine Signatur,
  - **E** eine Menge von Axiomen oder Gleichungen, die die Eigenschaften der einzelnen Operationen und die Wechselwirkung zwischen den Operationen festlegen.
- Ein **ADT** wird durch eine Spezifikation definiert.

Wir befassen uns nicht mit den mathematischen Feinheiten.

Einführung in die Softwaretechnik AB Softwaretechnik

DAS Standardbeispiel eines ADT: der Stack

Hier die klassische Definition:

- **Keller** (engl. *stack*): Folge von Elementen eines gegebenen Datentyps mit eingeschränkten Einfüge- und Ausfügeoperationen.
- Eine **Datenstruktur** über einem Datentyp *T* bezeichnet man als Keller, Stack oder Stapel, … wenn es zwei Zugriffsoperationen gibt, von denen die eine ein Element von *T* stets *an das Ende der Folge* einfügt und die andere stets *das letzte Element der Folge entfernt* und als Ergebnis liefert. Die Einfügeoperation nennt man **push**, die Ausfügeoperation **pop**.
- Das Prinzip, daß stets das zuletzt eingefügte Element eines Kellers als erstes wieder entfernt werden muß, bezeichnet man als **LIFO**-Prinzip (engl. Last In First Out).

[nach Informatik-Duden]

Edsger Dijkstra soll laut Meyer gesagt haben: "Abstract data types are a remarkable theory, whose purpose is to describe stacks".

Stacks spielen bei der syntaktischen Analyse und beim Übersetzerbau eine große Rolle. Sie gehören zu den wichtigen "Behältern" der Softwaretechnik.

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

ADT: Sorten, Operationen, Gleichungen

```
type alpha =                          /* Typ-Name: */
S     alpha, bool, cardinal           /* Sorten */
OP
      a(): alpha                      /* erzeugende Operationen: */
      b(): alpha                      /* bauen alle zum Typ gehörenden */
      :                               /* Werte? (war:operationen) auf */
      z(): alpha
      equal(alpha,alpha): bool        /* manipulierende Operationen: */
      ord(alpha): cardinal            /* verknüpfen die Werte */
E                                     /* Gleichungen: */
      equal(a,a) = t                  /* zeigen die Wechselwirkungen */
      equal(a,b) = f                  /* zwischen den Operationen */
      :
      Für alle x , y aus alpha
      equal(x,y) = equal(y,x)
      ord(a()) = 1
      ord(b()) = 2
      :
      ord(z()) = 26
endtype alpha
```


ADT-Spezifikation eines Stacks

•TYPES
- ***STACK* [*G*]**

•FUNCTIONS
- ***push*: *STACK* [*G*] × *G* → *STACK* [*G*]**
- ***pop*: *STACK* [*G*] ↛ *STACK* [*G*]**
- ***top*: *STACK* [*G*] ↛ *G***
- ***empty*: *STACK* [*G*] → *BOOLEAN***
- ***new*: *STACK* [*G*]**

•AXIOMS

For any *x*: *G*, *s*: *STACK* [*G*]

A1 • ***top* (*push* (*s*, *x*)) = *x***

A2 • ***pop* (*push* (*s*, *x*)) = *s***

A3 • ***empty* (*new*)**

A4 • not ***empty* (*push* (*s*, *x*))**

•PRECONDITIONS
- ***pop* (*s*: *STACK* [*G*])** require not ***empty* (*s*)**
- ***top* (*s*: *STACK* [*G*])** require not ***empty* (*s*)**

•Typ **STACK** mit einem generischen Typparameter **G**

•Die zulässigen Operationen auf Exemplaren des Typs **STACK** sind als Signaturen von Funktionen deklariert. Dabei bezeichnet ↛ eine partielle Funktion.

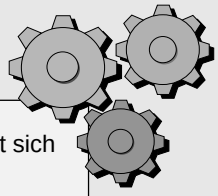
•Die Axiome definieren die Eigenschaften der Funktionen, ohne ihre Werte (Repräsentationen) festzulegen.

•Die Vorbedingungen legen den gültigen Wertebereich der partiellen Funktionen fest

nach © Meyer

Christiane Floyd, Guido Gryczan, Julian Mack

Diskussion des Zwischenergebnisses



- Die mathematisch *formale Spezifikation* eines abstrakten Datentyps ist *funktional* oder applikativ ausgerichtet. Sie läßt sich so nicht einfach in ein imperatives Programm übertragen.
- Trotzdem bleibt die *Grundidee*, einen *Datentyp* durch sein *Verhalten* und nicht durch seine *Repräsentation* zu definieren.
- Dies führt zu weiteren *Konstruktionsprinzipen*:
  - *Trenne* die *Spezifikation* einer Konstruktionseinheit von ihrer *Implementation* (Repräsentation).
  - *Verberge Implementationen* von Konstruktionseinheiten und mache sie so austauschbar.
- Die Spezifikation zeigt die Struktur nur indirekt über die Gleichungen
- Die Organisationsform bleibt offen! Verschiedene Realisierungen sind möglich! Fehler beachten !!!

top (*pop* (*push* (*pop* (*push* (*push* (
pop (*push* (*push* (*push* (*new*, *x1*), *x2*), *x3*)),
top (*pop* (*push* (*push* (*new*, *x4*), *x5*)))), *x6*)), *x7*)))

- Funktionale Verwendung des **STACK.**

nach © Meyer

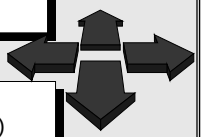
Zusammenfassung TYP-Begriff

Der Typbegriff klärt Zusammenhänge zwischen:

- den zulässigen Wertebereichen,
- den Werte erzeugenden bzw. manipulierenden Operationen,
- der Benennung von Typen, um sie zur Bildung komplexerer Typen zu verwenden,
- der Möglichkeit, Exemplare eines Typs (Objekte) anzulegen und zu verändern.

Wozu dienen Typen?

- zur Modellierung von Gegebenheiten der Anwendungswelt
- für die programminterne Verwaltung von Objekten,
- zur Kapselung von Komponenten des Basissystems
- für allgemein verwendbare Basistypen



Wo treten Typen auf?

- in einem Modul: Realisierung abstrakter Datentypen (Objekt- bzw. Typmodule)
- an der Schnittstelle: Typexport bei Typmodulen
- bei formalen Parametern: Typisierung der Parameter der exportierten Operationen
- auch Konstanten sind typisiert.

Christiane Floyd, Guido Gryczan, Julian Mack

Objekt

- **Definition: Ein Objekt ist ein Repräsentant (Exemplar) eines Typs.**
- **Bemerkung: Die Begriffe Typ und Objekt sind nicht durch eine einheitliche Theorie verbunden!**
- **Auf Modellebene gilt für Objekte:**
  - Identifizierbarkeit,
  - Typzugehörigkeit: jedes Objekt besitzt einen unveränderlichen Typ,
  - Wertebelegung: ein Objekt kann jeden Wert des Typs annehmen, der aktuelle Wert heißt Zustand;
  - Lebensdauer: Objekte werden angelegt, durchlaufen eine Geschichte und werden wieder gelöscht.

Achtung! "Identifizierbar" heißt nicht unbedingt "benannt"!
Unterschied:
- **explizite Namen (implementiert als deklarierte Objekte)**
- **implizite Namen (z.B. Indizes oder Referenzen)**

Einführung in die Softwaretechnik AB Softwaretechnik

Datenstrukturen

- **Datenstrukturen dienen bei der Softwareentwicklung zur Modellierung von**
  - Gebilden (Gegenständen, Vorgängen oder Sachverhalten) der Anwendungswelt,
  - technischen Betriebsmitteln oder Ressourcen, sowie von
  - Aktionen und Ereignissen, die diese Gebilde erzeugen oder ändern.
- **Die Anwendungswelt ist ein Bereich der Alltagswelt bzw. spezialisierte Fachwelt in Wirtschaft, Wissenschaft, Technik und Verwaltung)**

Einführung in die Softwaretechnik AB Softwaretechnik

Bildung von Datenstrukturen (1)

- **Betrachtung der Gebilde der Anwendungswelt als Informationsträger**
  - Information => Daten
- **Informatischer Objektbegriff**
  - Objekt = identifizierbares Exemplar einer Klasse mit typisierten informationellen Merkmalen
- **Festlegung spezieller Operationen, die Aktionen und Ereignissen der Anwendungswelt entsprechen.**
- **Herausarbeiten der wesentlichen informationellen Merkmale,**
  - Charakterisierung der Merkmale durch Attribute,
  - Rückführung der Attribute auf Datenelemente und ihre zulässigen Werte,
- **Angabe der Beziehungen zwischen einzelnen Elementen,**
- **Zusammenfügen von Elementen zu Datenstrukturen,**
  - Bestimmen der Ausdehnung der Datenstrukur

Einführung in die Softwaretechnik AB Softwaretechnik

Bildung von Datenstrukturen(2)

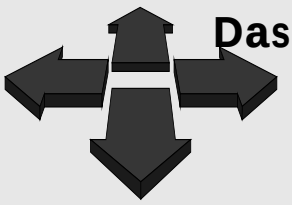
- **Aspekte:**
  - Werte der Elemente
  - Beziehungen zwischen Elementen
  - Komponentenbildung aus Elementen
  - Ausdehnung (Anzahl der Elemente)
  - Festlegung der Operationen.
- Im Prinzip werden Strukturen passend für ein Problem gewählt, aber: Es existiert ein Repertoire an standardisierten Datenstrukturen
- **Strukturkonzepte** beschreiben die Anordnung und die Beziehungen der Elemente.
- **Organisationsformen** legen fest, wie die Elemente erreicht und verändert werden können.
- Wir unterscheiden
  - **statische** und **dynamische** Datenstrukturen
  - **programminterne** und **persistente** Datenstrukturen

Einführung in die Softwaretechnik AB Softwaretechnik

Zusammenhang Datentyp / Datenstrukur / Objekt

- **Abstrakte Datentypen beschreiben**
  - die möglichen Werte
  - die zulässigen Operationen
- **Datenstrukturen**
  - Strukturkonzept
  - Organisationsform
- **Datentypen der Programmiersprache**
  - Werttypen: modellierte Gebilde
  - Referenztypen: Strukturkonzept
- **Objekte**
  - sind Exemplare eines Typs
  - haben eine Struktur und Geschichte
  - werden mit typisierten Variablen realisiert

Einführung in die Softwaretechnik AB Softwaretechnik



Das Typkonzept imperativer und objektorientierter Programmiersprachen

- Jede imperative und objektorientierte Programmiersprache besitzt elementare Datentypen, um numerische und logische Probleme lösen zu können.
- Dazu kommen bei den Daten die strukturierten und benutzerdefinierten Datentypen.
- Dynamische Datenstrukturen in klassischen imperativen Sprachen erfordern Zeigertypen.
- Darauf baut das polymorphe Typsystem der objektorientierten Programmiersprachen auf.
- Ein wesentliches Konstruktionsprinzip ist durch die abstrakten Datentypen gegeben.

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

Klassen und Objekte

- **Klasse**
  - benannte textuelle Einheit im Programm
  - definiert einen lokalen Datenraum
  - enthält *Attribute* (lokale Variablen)
  - stellt *Methoden* (Operationen) zur Verfügung
  - erlaubt, beliebig viele Exemplare anzulegen (Objekte)
- **Objekte**
  - werden zur Laufzeit angelegt und durch Referenzen identifiziert
  - sind typisiert
  - haben einen lokalen Datenraum und einen eigenen Zustand
  - tauschen sich über den Aufruf von Methoden aus.

```
class Leser {
  int gibLeserNr();
  String gibNachname();
  String gibVorname();
  ...
  mitAdresseVersehen( String ... );
  nameÄndern(String vn, String nn);
  ...
}
```


Stufen der Objektorientierung

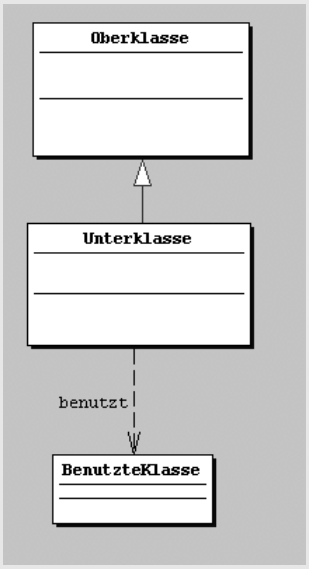
- ***objektbasiert*: modulare Architektur**
  - Datenabstraktion mit Modulen
  - Trennung von Definition und Implementation von Operationen
- ***klassenbasiert*: es gibt Klassen und Objekte**
  - Klassen sind statische Komponenten: Struktur
  - Objekte sind dynamische Komponenten: Interaktion
  - Operationen heißen i. d. Regel Methoden
  - Benutzt-Beziehung zwischen Klassen über Methoden
  - Klassen dienen zur Implementierung abstrakter Datentypen
- ***objektorientiert*: es gibt Klassen und Vererbung**
  - Klassen können Eigenschaften (Attribute und Methoden) anderer Klassen übernehmen (“erben”)
  - bei gleicher Definition (Spezifikation) kann Implementation von Methoden (Verhalten) variieren.
- **Erst die Vererbung bringt eine neue Qualität!**

Christiane Floyd, Guido Gryczan, Julian Mack

Vererbung

- **Vererbung stellt eine Beziehung zwischen zwei Klassen her:**
  - *Unterklasse* und *Oberklasse*,
  - statische Beziehung.
- **Die Unterklasse**
  - übernimmt alle Attribute und Methoden der Oberklasse,
  - kann weitere Attribute und Methoden hinzufügen,
  - kann Methoden der Oberklasse redefinieren (nur Verhalten).
- **Abstrakte Klassen:**
  - immer Oberklassen,
  - mindestens eine Methode ist nur definiert, aber nicht implementiert,
  - abstrakte Methoden werden in Unterklassen implementiert.

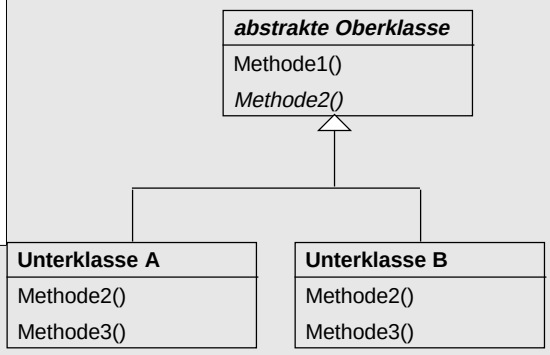
=> Spezialisierung bzw. Generalisierung



Generizität und Polymorphie

- **Polymorphie heißt Vielgestaltigkeit.**
- ***Generizität* (statische Polymorphie) betrifft die Struktur (Klassen): Eine Klasse heißt generisch, wenn sie eine Datenstruktur mit variablen Elementtypen spezifiziert.**

- ***Dynamische* Polymorphie bedeutet, daß ein- und dieselbe Referenz zur Laufzeit auf Objekte unterschiedlicher (aktueller) Typen zeigen kann.**
  - Die aktuellen Typen müssen Subtypen des deklarierten Referenztyps sein.
  - Die zugehörigen Methoden werden dynamisch gebunden.



Klassen als Ausgangspunkt des objektorientierten Typsystems

- Wir betrachten das Typsystem sog. *stark-typisierter objektorientierter Programmiersprachen*; d.h. kurz gesagt, Sprachen, in denen jedes Programmobjekt und jeder Bezeichner einen explizit festgelegten Typ hat.
- Bisher haben wir vor allem den *Klassenbegriff* betrachtet.
  - Auf der Konstruktionsebene legt eine Klasse fest, wie sich ihre Exemplare verhalten und wie sie aufgebaut sind.
  - Dabei drücken *Klassenhierarchien* die Generalisierung oder Spezialisierung dieses Verhaltens und des Aufbaus aus.
  - Compiler und Laufzeitsystem stellen sicher, daß die richtigen *Exemplare* von »vollständigen« Klassenbeschreibungen erzeugt werden.
- Demgegenüber ist ein *Typ* vor allem ein Spezifikationskonzept, das eine »äußere« Sicht auf deklarierte Bezeichner und Programmobjekte festlegt.

© Züllighoven

Klasse und Typ

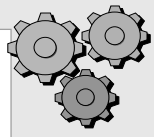
- Eine *Klasse* definiert neben der Schnittstelle das Verhalten und den Aufbau ihrer Exemplare.
- Ein *Typ* dient zur Spezifikation von Objekten und zur Deklaration von Bezeichnern.
- Der *Typ* eines Objekts zur Laufzeit bezieht sich zunächst nur auf seine *Schnittstelle* (die Menge der zulässigen Botschaften).
- Um neben der Schnittstelle auch das *Verhalten* von Exemplaren eines Typs festzulegen, können Typen durch *Klassen* realisiert werden.
- Ein *Objekt* kann verschiedene Typen haben, und Objekte von unterschiedlichen Klassen können denselben Typ haben.
- Ein Teil der Schnittstelle eines Objekts kann durch einen Typ charakterisiert sein und andere Teile durch andere Typen.
- Zwei Objekte des gleichen Typs brauchen nur mit einem Teil ihrer Schnittstellen übereinzustimmen.

Christiane Floyd, Guido Gryczan, Julian Mack

Typ, objektorientiert

- Ein **Typ** bezeichnet objektorientiert eine Spezifikation des Verhaltens von Objekten im Sinne eines abstrakten Datentyps. Ein Typ ist eine syntaktische und semantische Abstraktion. Er dient zur Deklaration von Bezeichnern und Größen eines Programms zum Zwecke der Überprüfung ihrer (typsicheren) Verwendung.
- Ein Typ spezifiziert syntaktisch eine *Schnittstelle*, d.h. benennt die Operationen, mit denen ein Exemplar des Typs aufrufbar ist. Darüber hinaus kann ein Typ im Sinne eines *Protokolls* das Verhalten von Objekten beschreiben
- Ein Typ enthält keine Information über die Repräsentation des Zustands und die Implementierung der Operationen.

In gängigen objektorientierten Programmiersprachen werden Klasse und Typ gleichgesetzt. Dies ist praktisch, verwischt aber die konzeptionellen Unterschiede, die wir im Auge behalten müssen, wenn wir softwaretechnisch sauber arbeiten wollen.



Zusammenhang Klasse und Typ

- Was ist die Bedeutung von Vererbung im Zusammenhang mit Typisierung?

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

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

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

Unterklassen erben alle Eigenschaften (Deklaration von Objekten, Prozeduren und Funktionen) ihrer Oberklassen und damit auch die Schnittstelle, die den Typ festlegt.

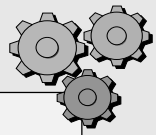
Typhierarchie

- Eine **Typhierarchie** besteht aus Sub- und Supertypen. Ein *Subtyp* bietet zumindest die Operationen an, die von seinem *Supertyp* spezifiziert sind. Dabei kann es folgende Varianten geben:
  - Die *Operationen* stimmen genau in ihren *Signaturen* überein.
  - Die Typen der *Parameter- und Ergebnisobjekte* können selbst von einem *Sub- oder Supertyp* des ursprünglichen Typs sein (Ko- und Kontravarianz).
- Ein *Exemplar eines Subtyps* muß zur Laufzeit die Stelle eines Exemplars des *Supertyps* einnehmen können. Dabei ist *Verhaltensgleichheit* oder *Verhaltensähnlichkeit* gefordert:
  - Ein Exemplar eines Subtyps führt zur keiner feststellbaren Veränderung des Programms.
  - Ein Exemplar des Subtyps ist insofern verhaltensähnlich, als es das erwartete Verhalten und niemals einen Laufzeitfehler produziert.

Verhaltensähnlichkeit ist eine viel “weichere” Formulierung als die von vielen Theoretikern geforderte Verhaltensgleichheit.

Einführung in die Softwaretechnik AB Softwaretechnik

Realisierungsmöglichkeiten von Klasse und Typ

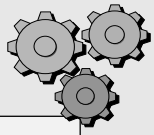


- **Benannte Schnittstelle**:
  Die Trennung von Klasse und Typ finden wir in Java mit seinen Interfaces und in Objective C mit seinen Protocols. Eine Klasse kann neben ihrer Oberklasse angeben, welche benannten Schnittstellen sie erfüllt. Klassen können nicht nur nach Klassenhierarchien, sondern auch nach ihren Schnittstellen gruppiert werden.
- **Klasse als Typ**:
  C++ und Eiffel benutzen das Klassenkonstrukt, um sowohl den Typ eines Objekts als auch seine Implementierung festzulegen. Allerdings gibt es auch die Möglichkeit, reine Spezifikationsklassen zu schreiben, von denen keine Exemplare erzeugt werden können.
- **Fehlende Typdeklaration**:
  In Smalltalk gibt es keine Typdeklarationen für Bezeichner und Programmeinheiten, d.h., bei der Übersetzung erfolgt keine Typkontrolle. Kennt ein Objekt eine gerufene Operation nicht, führt dies zu einem Laufzeitfehler.

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack

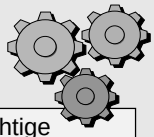
Erfahrungen mit Klassen- und Typkonzepten



- *Typsysteme* stellen eine große Unterstützung für die sichere Konstruktion von Softwaresystemen dar.
- *Dynamisch typisierte Sprachen* bieten während der reinen Entwicklungsphase mehr Flexibilität als statisch typisierte Sprachen. Wir konstruieren so, als ob die Sprache statisch typisiert wäre. Zur Sicherheit dienen dynamische Typprüfungen an entscheidenden Stellen.
- *Vererbungshierarchien* sollten fachlichen Hierarchien entsprechen und damit auch ein ähnliches Verhalten festlegen (Klassenhierarchien entsprechen Typhierarchien).
- *Benannte Schnittstellen* können eine sinnvolle Alternative zur Mehrfachvererbung sein. Entwurfskonventionen müssen dann sicherstellen, daß eine *Verhaltensähnlichkeit* in der Verwendung gewährleistet ist.

Einführung in die Softwaretechnik AB Softwaretechnik

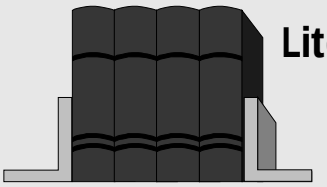
Zwischenergebnis und Diskussion



- Das *objektorientierte polymorphe Typsystem* ist eine wichtige Voraussetzung, um das *Prinzip der abstrakten Datentypen* umzusetzen:
  - Das Verhalten und die Struktur aller *Exemplare* einer Klasse kann durch die *Klasse* festgelegt werden.
  - Klassen realisieren *benutzerdefinierte Typen*, die zur Deklaration verwendet werden können.
  - *Oberklassenbildung* und *Polymorphie* erlauben, unterschiedliche Realisierungen unter einem gemeinsamen Typ anzusprechen.
- Gleichzeitig löst das objektorientierte Typsystem das *Offen-Geschlossen-Dilemma*:
  - Eine Klasse kann gegenüber ihren Klienten *geschlossen* sein, indem sie ein stabile Dienstleistung anbietet.
  - Eine Klasse ist für die Weiterentwicklung *offen*, indem sie sich durch Erweiterung und Redefinition in Unterklassen verändern läßt.

Einführung in die Softwaretechnik AB Softwaretechnik

Christiane Floyd, Guido Gryczan, Julian Mack



Literaturhinweise

Hoare, C.A.R; *Notes on Data Structuring.* In: Dahl, Dijkstra, Hoare: Structured Programming. Academic Press, 1972.

[Einer DER Klassiker über Datenstrukturen.]

Robert W. **Sebesta**, *Concepts of Programming Languages.* Benjamin Cummings, 2. Auflage, 1993.

Ken **Arnold**, James **Gosling**: *The Java Programming Language.* 2nd Edition, Addison-Wesley, 1998 (dt. "Die Java Programmiersprache", ISBN 3-8273-1034-2).

Bertrand **Meyer**: *Object-oriented Software Construction*. Second Edition. Prentice Hall, 1997.

Peter **Rechenberg**, Gustav **Pomberger,** *Informatik-Handbuch.* Hanser-Verlag, 1977.

Heinz **Züllighoven**: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz*. dpunkt-Verlag, 1998.