OO Modellierung
AK der IK 2

Markus Stumptner

Institut für Informationssysteme (E184/2)
Paniglgasse 16

mst@dbai.tuwien.ac.at

Überblick

1. Einführung in objektorientierte Konzepte
2. objektorientierte Sprachen (OOPLs)
   Überblick, Smalltalk
3. oo „Theorie” (Vererbung)
4. Verfahren für objektorientierte Analyse und Entwurf
   (OOD)
5. oo Datenbanksysteme (OODBs)
6. Patterns

1. Einführung

1. Woher kommen Objekte
2. Objekte in der Informatik
3. oo Konzepte

Was ist ein Objekt? (1)

ein “Ding” aus der realen Welt, das

- Eigenschaften besitzt
- Verhalten besitzt
- als Einheit betrachtet wird

Beispiele:

- Orange
- Blumenhändler

Was ist ein Objekt (2)

für den Kunden ist nur das Verhalten interessant

Verhalten eines Blumenhändlers

- Blumen kaufen
- Blumen bestellen
- Blumen liefern lassen
  - Händler liefert selbst
  - liefert per Post
  - delegiert an andere

Was ist ein Objekt (3)

- **Klassifikation**
  - Objekte werden gruppiert
  - diese Klassifikation geschieht im wesentlichen nach dem Verhalten
  - Beispiel: alle Blumenhändler sind sich ähnlich
- **Vererbung**
  - *Blumenhändler* sind ein Sonderfall von *Ladenbesitzern*
- **Polymorphismus**
  - Einkaufen kann ich bei beiden

Was ist ein Objekt? (aus Informatikersicht)

Ein **Objekt** (*object*)

- ist ein konkretes oder abstraktes Ding, das die kleinste Einheit in einem objektorientierten System darstellt
- ist eindeutig identifizierbar
- hat einen internen Zustand repräsentiert durch Instanzvariable (data members, attributes, features, ...)
- hat ein Verhalten repräsentiert durch eine Menge von Operationen (= sichtbare Schnittstelle des Objektes) und deren Implementierung; Operationen werden durch das Senden einer Nachricht aktiviert (Message passing)
- ist Instanz einer Objektklasse

Wozu das Ganze?

- erleichterte ("natürliche") Modellierung durch Gruppierung zusammengehöriger Daten
- Modularisierung bei der Softwareentwicklung durch Zusammenschluß von Daten und Code
- erhöhte Software-Wartbarkeit durch Modularisierung und Datenkapselung
- Software-Wiederverwendung durch
  - Ableitung neuer Klassen aus bestehenden (Vererbung)
- Verwendung existierender Klassen (Klassenbibliotheken, Frameworks)

Objektorientiertes Paradigma

Programmiersprachen:

- Simula (1967)
  - Algol-Erweiterung, Simulationssprache
- Smalltalk-Projekt, Xerox(gestartet 1972)
  - Sprache, User Interface, Programmierumgebung

Artificial Intelligence (Kognitionswissenschaft)

- Frames

Traditionelle Datenmodellierung

- ER-Modell (Chen), TAXIS, SDM

Objektorientierte Konzepte

- Datenkapselung
- Vererbung
- Klassifizierung
- Polymorphismus

Dasselbe oder das gleiche? dasselbe!

Reprinted with permission by UFS, Inc.

aus: S. Zdonik, D. Maier: Fundamentals of Object-Oriented Databases;
in: Readings onObject-Oriented Databases, Morgan-Kaufmann, 1990.

Objektidentität

Jedes Objekt besitzt eine systemweit eindeutige **Objekt-identität** (*object identity*) .

- Vom System vergeben
- Interne Referenz - nicht lesbar für Benutzer
- Unabhängig vom aktuellen Zustand des Objektes - un-verändert über die gesamte Lebensdauer

Objektidentität ≠ Objektgleichheit

Objektidentität und Schlüssel

- Schlüssel in Datenbanksystemen: Identifikation durch eindeutigen Wert
  Bsp.: Name
- Wert und Eindeutigkeit durch Anwendung gegeben
- Wert möglicherweise Änderung unterworfen (z.B. Heirat)
- als Abhilfe: künstliche Schlüssel (*Surrogate*): Matrikelnummer, Sozialversicherungsnummer etc.
- Schlüssel werden auch in oo Systemen noch gebraucht, um Zugriff auf Objekte der realen Welt zu ermöglichen

Datenkapselung

Information hiding (Parnas):

- Verstecken von (für den Verwender) unwichtigen Details eines Programms

Datenkapselung (*encapsulation*):

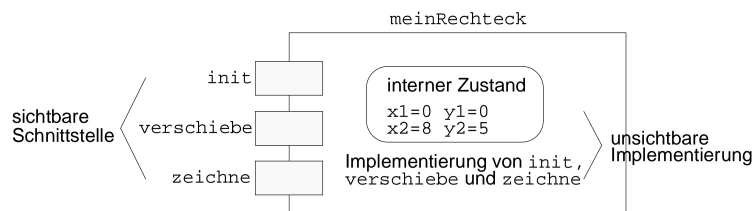
- Schutz vor unerlaubtem Zugriff auf den internen Zustand (die Struktur) einer Datenstruktur durch eine eindeutig definierte Schnittstelle (= Menge von Operationen).
- Nur mit den Operationen ist es möglich, den Zustand der Datenstruktur zu manipulieren.

abstrakter Datentyp

Ein **abstrakter Datentyp** (**ADT**, *abstract data type*) ist eine Datentypdefinition, die die Struktur von Daten zusammen mit den darauf zugelassenen Operationen bestimmt.

Der Benutzer eines ADTs kann ausschließlich über die Operationen auf die Struktur der Daten zugreifen.

Sprachen mit ADTs: CLU, Modula-2, Ada



Abstrakter Datentyp - Beispiel

```
DEFINITION MODULE REck;
TYPE Rechteck
  procedure init (x1,y1,x2,y2);
  procedure verschiebe (r:Rechteck...);
  procedure zeichne (r:Rechteck...);
END Recht.
IMPLEMENTATION MODULE REck;
TYPE Recht =
      /* interne Datenstruktur */
    x1,y1,x2,y2: Integer;
      /* Operationen */
...
END REck.
```

Modulkonzept - ADT

- Modulkonzept (Ada, Modula-2) läßt mehrere Typen (oder auch keinen) pro Modul zu
- Konzept des abstrakten Datentyps: Modul, das einen spezifischen Typ enthält und nur Operationen, die auf diesem Typ operieren
- Eine Sprache, die abstrakte Datentypen unterstützt und behandelt wie alle anderen (Eintragen in Mengen, Funktionsresultate) heißt *klassenbasiert*
- Syntaktische Sonderbehandlung dieses Typs: jede Operation eines ADT betrifft vorrangig eine Instanz dieses Typs: `r.verschiebe(x,y)` statt `verschiebe(r,x,y)`

Objektklasse (1)

- Eine **Objektklasse** (*object class*), kurz **Klasse**, legt Struktur und Verhalten einer Menge von Objekten fest ("Implementierung eines ADT")
- Eine Klasse kennt eine (meistens vordefinierte) Operation (z.B.: *create* in Eiffel, *new* in C++ und Smalltalk) zum Erzeugen von neuen Instanzen.
- Datenstruktur ist einmal pro Instanz vorhanden
- Operationen sind nur einmal bei der Klasse spezifiziert und implementiert — für alle Instanzen gleich

Objektklasse - Beispiel (1)

(Beispiel in C++)

```
class Rechteck{
      int x1,y1,x2,y2;
   public:
      void init(int,int,int,int);
      void verschiebe(int,int);
      void zeichne();
};
void Rechteck::init(int a,int b,int c,int d){
   x1 = a; y1 = b; x2 = c; y2 = d}
void Rechteck::verschiebe(int x,int y){
   ... }
void Rechteck::zeichne(){
   ... }
```


Objektklasse - Beispiel (2)

```
main{
   Rechteck meinRechteck;

   meinRechteck = new Rechteck;
   meinRechteck.init(0,0,8,5);
   meinRechteck.zeichne();
}
```

Instanzvariable

Instanzvariable (*instance variables*) beschreiben den internen Zustand eines Objektes (enthalten Datenwerte oder andere Objekte oder Verweise auf andere Objekte). (Java: fields, C++: data members, Eiffel: features, formale Ansätze: Attribute oder Properties)

Jedes Objekt hat einen eigenen, kompletten Satz der von der Klasse festgelegten Instanzvariablen

Instanzvariable haben üblicherweise einen Namen (wie Felder eines Modula-Records) `rechteck.x`

Datenkapselung: Instanzvariable sollten ausschließlich durch Aufruf von Operationen gelesen und geschrieben werden.

Kommunikation zwischen Objekten

Objekte kommunizieren miteinander durch das Senden von Nachrichten (*message passing*). Eine **Nachricht** (*message*) ist eine Aufforderung an ein Objekt, eine Operation auszuführen. Die ausgeführte Implementierung hängt vom Objekt ab.

→ Gegensatz zum traditionellen Prozeduraufruf!

```
meinRechteck.verschiebe(1,2);
```

`meinRechteck` ist Empfänger der Nachricht `verschiebe` mit den aktuellen Parametern `1` und `2`.

`meinRechteck` antwortet mit der Ausführung der Operation `verschiebe`.

Verhalten

Verhalten ist gegeben durch **Operationen** (dem Objekt zugeordnete Prozeduren oder Funktionen). Andere Bezeichnung: **Methode** (*method*).

Eine Operation hat eine sichtbare Spezifikation (= Signatur) und eine nach außen unsichtbare Implementierung.

Die **Signatur** einer Operation (*signature*) besteht üblicherweise aus dem Namen der Operation und den Klassen der Eingabeparameter und des Rückgabewertes.

```
void verschiebe(int x, y);
```

Rückgabetyp

Name der Operation

2 Eingabeparameter

Erweiterte Signatur

Die **Vor- und Nachbedingungen** einer Operation (*pre- and postconditions*) spezifizieren einen "Vertrag" zwischen der Klasse, die die Operation anbietet (= Anbieter), und der Klasse, die die Operation benutzt (= Klient; Anbieter und Klient können auch zusammenfallen). Die *Vorbedingung* muß erfüllt sein, damit die Operation ausgeführt werden kann. Nach erfolgreicher Ausführung der Operation gilt die *Nachbedingung*.

```
int pop (stack s);
  precondition:
    s.size > 0;
      /* Code der Operation */
  postcondition:
    s.size = old.s.size - 1;
```

Eiffel unterstützt Vor- und Nachbedingungen als Teil der Operationsspezifikation.
Auf den internen Zustand des Objektes vor Ausführung der Operation wird mit "`old.`" zugegriffen (s. Beispiel).

Klassenvariable und Klassenoperation

Eine **Klassenvariable** (*class variable*) ist eine globale Variable, die nur für die Instanzen einer Klasse sichtbar ist (Smalltalk: *class variable*, C++: *static*)

Eine **Klassenoperation** (class operation) ist eine Operation, die von einer Klasse ausgeführt wird und das Verhalten einer Klasse beschreibt. Das Erzeugen neuer Instanzen ist in einigen Sprachen (z.B.: Smalltalk, Objective-C) eine Klassenoperation.

Objektklasse (2)

Objektklassen werden in einigen Sprachen selbst als Objekte (*first-class objects*) verwaltet (z.B.: Smalltalk). Hat die folgenden Vorteile:

1. Klassen können zur Laufzeit definiert und als Parameter übergeben werden.
2. Klassenvariable und Klassenoperationen können einfach implementiert werden.

Verschiedene Ebenen der Kapselung

Verschiedene Ebenen der Sichtbarkeit der Schnittstellen helfen, verschiedene Zugriffsberechtigungen auf Instanzen einer Klasse zu realisieren.

Beispiele:

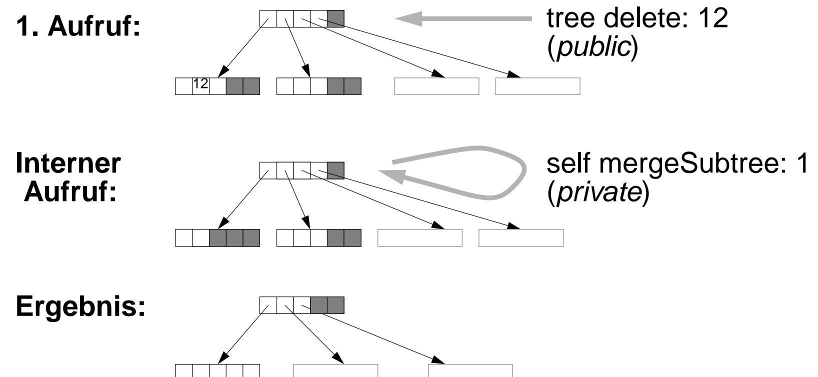
- C++, Java: Unterscheidung zwischen öffentlichen (*public*), geschützten (*protected*) und privaten (*privat*) Operationen
- Eiffel: alle *features* (Instanzvariable und Operationen) privat, wenn nicht explizit als öffentlich (*export*) markiert.
- Smalltalk: Instanzvariablen sind privat, Operationen sind öffentlich

Beispiel public/private:: B*-Baum

1. Aufruf:

Interner Aufruf:

Ergebnis:



Die Operation mergeSubtree: sollte nicht von außerhalb ausgelöst werden.

Vererbung

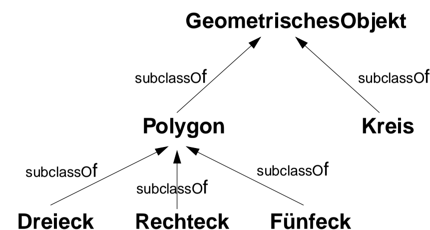
Vererbung (*inheritance*): Definition neuer Klassen aus existierenden Klassen. Eine aus einer bestehenden Klasse (= **Oberklasse**) abgeleitete **Unterklasse** (Subklasse) erbt alle Instanzvariablen und Operationen (Spezifikation und Implementierung) der Oberklasse.

Eine Unterklasse kann

1. neue Instanzvariable und/oder Operationen definieren
2. Implementierung geerbter Operationen überschreiben
3. geerbte Operationen durch eigenen Code ergänzen

Klassenhierarchie

Durch Einführung **erweiterbarer Klassen** (durch Vererbung) entstehen **Klassenhierarchien** als Basis objektorientierter Systementwicklung. Eine Hierarchie besteht aus Klassen mit ähnlichen Eigenschaften (z.B.: Lebewesen, Fahrzeuge, geometrische Objekte, etc)



Vererbung - Beispiel (1)

```
class GeometrischesObjekt {
      string farbe;
   public:
      void zeichne();
      void verschiebe(...); }
   { /* Implementierung von zeichne und verschiebe */}
class Polygon: public GeometrischesObjekt {
      int kantenAnzahl;
   public:
      float fläche(); }
   { /* Impl. von fläche und Überschreiben von zeichne */}
```

`Polygon` ist Unterklasse von `GeometrischesObjekt`.

`Polygon` erbt von `GeometrischesObjekt` die Instanzvariable `farbe` und die Operationen `zeichne` und `verschiebe`.

Vererbung - Beispiel (2)

```
class Rechteck: public Polygon {
      int x1, y1, x2, y2;
   public:
      void init(int,int,int,int);
      Gerade diag(); }
   { /* Impl. von init, diag, Überschreiben von fläche und zeichne
*/}
```

`Rechteck` ist Unterklasse von `Polygon`.

`Rechteck` erbt von `Polygon` die Instanzvariablen `kantenAnzahl` und `farbe`, und die Operationen `fläche`, `zeichne` und `verschiebe`.

- Vererbung ist transitiv definiert

Überschreiben von Operationen

Das **Überschreiben** (*overriding*) von Operationen unterstützt eine Neuimplementierung von geerbten Operationen in Unterklassen.

Beispiel:

`Rechteck` erbt die Operation `zeichne` und überschreibt sie mit einer für die spezielle Klasse angepaßten Implementierung.

Vorteil:

- Spezifikation d. Verhaltens (der Operationen) lokalisiert
- Implementierung des Verhaltens an internen Zustand der Klassen, die dieses Verhalten haben, angepaßt

Polymorphismus (1)

Polymorphismus (*polymorphism*) ist die Fähigkeit, verschiedene Gestalt anzunehmen.

Eine **polymorphe Operation** (*polymorphic operation*) kann auf Objekten verschiedener Klassen ausgeführt werden und jeweils eine andere Semantik haben.

In OO Systemen: Polymorphismus durch

- Vererben
- Überschreiben

Polymorphismus (2)

Eine **polymorphe Variable** (*polymorphic variable*) kann im Laufe der Ausführung eines Programmes Referenzen auf Instanzen verschiedener Klassen enthalten. Eine polymorphe Variable hat eine

1. **statische Klasse**: wird bei der Deklaration spezifiziert, und ist zur Übersetzungszeit bekannt (fix!)
2. **dynamische Klasse**: jeweils die Klasse des Objektes, das die Variable zur Laufzeit referenziert (variabel!)

Binden

Binden (*binding*) bezeichnet die Zuordnung einer Nachricht (Operationsname) zum Code (Implementierung einer Operation), der ausgeführt wird.

Statisches Binden (*static binding*) ist Binden zur Kompilierungszeit.

Dynamisches Binden (*dynamic binding*, *late binding*) ist Binden zur Laufzeit.

Einige Sprachen unterstützen statisches *und* dynamisches Binden (z.B.: Simula, C++). In Eiffel, Objective-C, Javaund Smalltalk wird nur dynamisches Binden unterstützt.

Dynamisches Binden - Beispiel

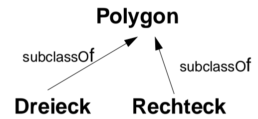
Dynamisches Binden erlaubt die Ausnützung von Vererbungsbeziehungen. Beim Senden einer Nachricht an eine polymorphe Variable wird bei der dynamischen Klasse mit der Codesuche begonnen und - falls dort nicht gefunden - rekursiv in den Oberklassen fortgesetzt.

(Fortsetzung des Polymorphismus-Beispiels)

```
main {
...;
for i in polygone do polygone[i]->zeichne;
...;
}
```

Für die Operation `zeichne` gibt es eine Spezifikation in der Klasse `GeometrischesObjekt` und individuelle Implementierungen in den Subklassen.

Polymorphismus - Beispiel




```
main {
Polygon p;
Rechteck r;
Dreieck d;
Polygon *polygone[2];
...;
r = new Rechteck; r.init(...);
d = new Dreieck; d.init(...);
r.zeichne();
d.zeichne();        /* operation zeichne ist polymorph */
p = r;              /* variable p ist polymorph */
polygone[1] = r;
polygone[2] = d;       /* variable polygone
}                         ist ein polymorphes array */
```

Achtung: "`r = p`" ist verboten, weil ein Polygon nicht alle Operationen von Rechteck versteht; sehr wohl aber umgekehrt.

Vererbung - 2 Sichten

Gegeben eine Klasse *K* und eine Unterklasse *K'*.
Vererbung ist:

- *Erweiterung* auf der *intensionalen Ebene* (= Klassenebene):
  Menge der Instanzvar. und Op. von *K'* ist eine Obermenge der Menge der Instanzvar. und Op. von *K*

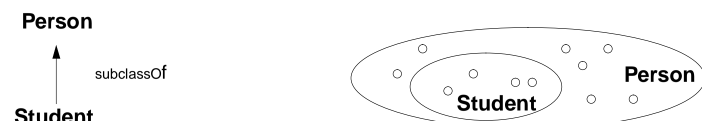
  Beispiel:

Person:	Name	Student:	Name
	Adresse,		Adresse
			MNR
			Kennummer etc.

Vererbung - Sichten

- *Einschränkung* auf der *extensionalen Ebene* (= Instanzebene):

  (Menge der Instanzen von *K'*) ⊆
  (Menge der Instanzen von *K*)



Vorteile der Vererbung

1. Wiederverwendung:
   Instanzvariable und Operationen einer Klasse in
   Unterklassen verwendbar
2. Modellierungshilfe:
   Kategorisierung als Form der Wissensrepräsentation
3. Inkrementelle Entwicklung:
   Schrittweise Entw. vom allgemeinen zum speziellen

Vererbung - ein überladener Begriff

Viele verschiedene Arten von Vererbung und daraus resultierende Klassenhierarchien

1. Vererbungsstruktur:
   Einfachvererbung (Klassenhierarchie ist ein Baum),
   Mehrfachvererbung (gerichteter Graph)
2. Qualität: Spezialisierung,
   Spezifikationsvererbung, Codevererbung
3. Granularität:
   Klassenvererbung, Instanzvererbung

➥ ein eigenes Kapitel

Vorteile

von Vererbung, Polymorphismus und dynamischen Binden:

1. ein Name für semantisch ähnliche Operationen: verständliche Programme, leichter änderbar
2. Programmierer braucht die exakten Klassen der Elemente einer Menge nicht zu kennen
3. neue Klasse kann hinzugefügt werden, ohne das Programm ändern zu müssen
4. Möglichkeit inkrementeller Entwicklung

Typkonzept in Programmiersprachen (1)

Ein **Typ** (*type*) spezifiziert das Verhalten einer Menge von Datenwerten (in konventionellen Sprachen) oder Objekten (in oo Sprachen).

Vorteile von getypten Sprachen:

1. *Typüberprüfung:* Typen spezifizieren Bedingungen und Regeln, um die Kompatibilität zwischen Ausdrücken bestimmen zu können.
2. *Verifikation:*Typen spezifizieren invariante Bedingungen, die die Instanzen des Typs erfüllen müssen.
3. *Implementierung:* Typen spezifizieren den benötigten Speicherplatz und die Struktur ihrer Instanzen.

Typersetzbarkeit

Betrachtung von Vererbungsbeziehungen aus der Perspektive von Typüberprüfungen.

Wenn $S \leq T$ *(S ist Subtyp von T)*, dann kann an jeder Stelle, an der eine Instanz von *T* erwartet wird, eine Instanz von *S* verwendet werden.

➥ Prinzip der Typersetzbarkeit

In getypten oo Sprachen fällt die Subtyphierarchie in der Regel mit der Vererbungshierarchie zusammen (z.B.: C++, Eiffel).

Typüberprüfung (1)

Statisch getypte Sprachen (*statically typed languages*) sind Sprachen, bei denen der Typ jedes Ausdrucks zur Übersetzungszeit bekannt ist
→ statische (zur Übersetzungszeit) Typüberprüfung!

Streng getypte Sprachen (*strongly typed languages*) sind Sprachen, bei denen der Typ jedes Ausdrucks nicht unbedingt zur Übersetzungszeit bekannt sein muß, aber jeder Ausdruck typkompatibel ist
→ statische und dynamische (zur Laufzeit) Typüberprüfung!

Schwach getypte Sprachen (*weakly typed languages*) sind Sprachen, bei denen der Typ jedes Ausdrucks erst zur Laufzeit bekannt ist → dynamische Typüberprüfung!

Die 4 wünschenswerten Eigenschaften

1. Typersetzbarkeit
2. Statische Typüberprüfung
3. Veränderbarkeit
4. Subtypenbildung durch Spezialisierung

Aber: nicht alle 4 sind gleichzeitig erfüllbar -> siehe Kapitel über Vererbung

Erwartungen an Software

1. Korrektheit/Verifizierbarkeit
2. Robust (bzgl. abnormaler Fälle)
3. Erweiterbar (bes. wichtig für große Systeme)
4. Wiederverwendbar
5. Sicherheit
6. kompatibel mit existierenden Systemen
7. Effizient
8. Portabel
9. Leicht verwendbar
10. Wartbar

Erwartungen und OO

1. Korrektheit: Zusicherungen (Vor-, Nachbedingungen, Invarianten)

2.,4.,5.,6. Robust, wiederverwendbar, sicher, kompatibel: durch Modularität, Datenkapselung, Abstrakte Datentypen, Schnittstellen.

3. Erweiterbar: Modularität, Vererbung)

8.,9.,10. Portabel, leicht verwendbar, wartbar: Abstraktion, Modellierung semantischer Zusammenhänge (Kohäsion)Modellierung der realen Welt

Vorteile OO

1. Korrektheit (Zusicherungen)
2. Robustheit
3. Erweiterbarkeit
4. Wiederverwendbarkeit (und Entwicklung von Klassen-bibliotheken und Frameworks)
5. Sicherheit (Einschränkung fehlerhafter Zugriffe)
6. Reduzierte Datenabhängigkeit bei Aenderungen
7. Direkte Abbildung Analyse-Design-Implementierung
8. Kompatibilität (Middleware)
9. Leichte Verwendbarkeit (allerdings: Lernkurve)

Zusammenfassung

Alt: Datenstrukturen + Algorithmus = Programm

Neu: Daten + Verhalten = Objekt
System = Interagierende Objekte