Prof. Dr. Timm Grams
Fachhochschule Fulda, Fachbereich Elektrotechnik und Informationstechnik
07.09.01
(korr.: 01.11.01)

Tutorial Lehrgangsverwaltung: Von UML über C zu Java

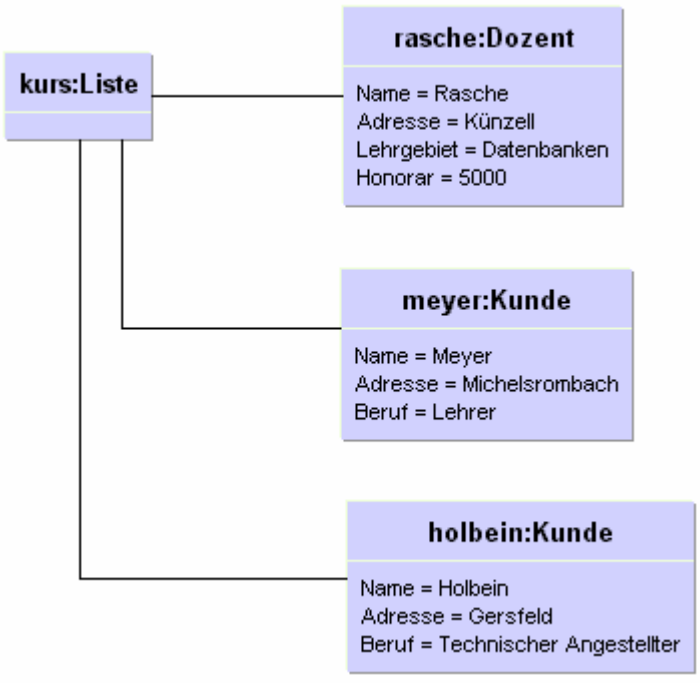
Gliederung

Die Aufgabenstellung: Lehrgangsverwaltungsprogramm

Das zu erstellende Programm dient der Verwaltung der Daten von Teilnehmern (Kunden) und Dozenten eines Lehrgangs. Funktion: Erfassung der Daten von Kunden, das sind die Teilnehmer des Kurses, und Dozenten. Nach Beendigung der Eingabe soll die Liste der Kunden und Dozenten auf dem Bildschirm ausgegeben werden. (Siehe auch die Einführung "Grundlagen der objektorientierten Software-Entwicklung" auf der CD zum "Lehrbuch Grundlagen der Informatik" von Helmut Balzert, Spektrum Akademischer Verlag, Heidelberg, Berlin 1999).

Analyse: Objektdiagramm

Die Ergebnisse der Analyse und des Designs sind Objekt- und Klassendiagramme, die nach den Regeln der Unified Modeling Language (UML) gestaltet werden. Die Diagramme werden mit dem Werkzeug GO, das dem oben angesprochenen Lehrbuch beigefügt ist, erstellt.



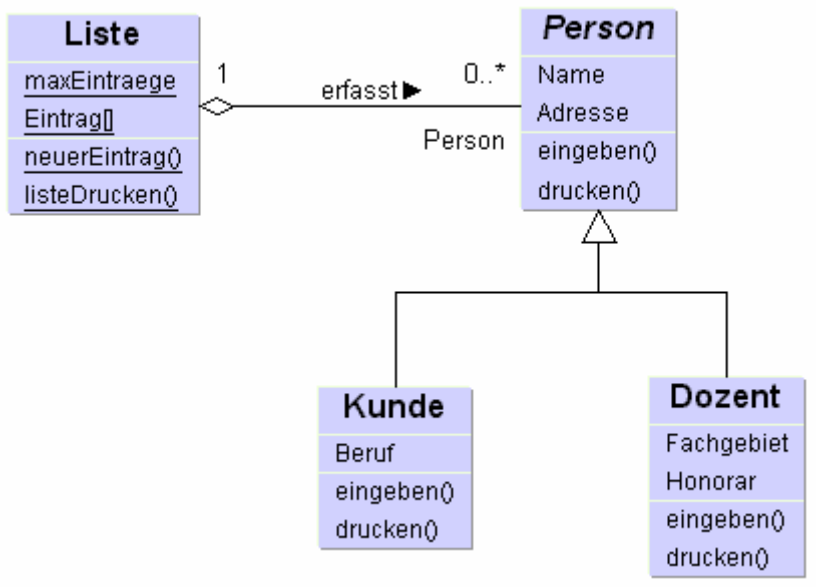
Das Objektdiagramm stellt beispielhaft dar, welche *Attribute* den *Objekten* des Systems zukommen und welche Beziehungen zwischen den Objekten bestehen. Das nebenstehende Objektdiagramm zeigt die Objekte meyer und holbein vom Typ Kunde sowie das Objekt rasche vom Typ Dozent. Das Objekt kurs vom Typ Liste enthält Bezüge auf diese Objekte.

Objektbezeichner beginnen mit einem Kleinbuchstaben, Klassenbezeichner mit einem Großbuchstaben.

Analyse: OOA-Klassendiagramm

Das OOA-Klassendiagramm ist eine Darstellung der Objekttypen (Klassen), und deren Beziehungen zueinander. Die Klassen sind als dreigeteilte Rechtecke gezeichnet. Darin stehen jeweils

1. der Name der *Klasse*,
2. die *Attribute* (Variablen) und
3. die *Methoden* (Funktionen).



Unterstrichen sind die *statischen* (static) Attribute und Methoden. Diese gehören zur Klasse und sind auch ohne Objekt existent und erreichbar.

Kursiv geschriebene Klassennamen bezeichnen *abstrakte* Klassen. Das sind Klassen ohne eigene Realisierungen (Objekte).

Im Klassendiagramm bleiben die Beziehungen zwischen den Objekten sichtbar. An den Verbindungslinien stehen Angaben über die mögliche Anzahl der Objekte und die Art der Verbindung. Eine offene Raute zeigt eine Aggregation an; sie ist auf der Seite desjenigen Objekttyps angebracht, der die "Verantwortung für das Ganze" trägt.

Verbindungen mit einem offenen Dreieck stehen für eine Vererbungsbeziehung. Das Dreieck zeigt auf die Oberklasse. Unterklassen *erben* die Attribute und Methoden der Oberklasse und fügen ihnen eventuell weitere hinzu. Methoden der Unterklassen *überschreiben* Methoden der Oberklasse, vorausgesetzt sie haben dieselbe Signatur (identischen Namen und dieselben Parametertypen) und denselben Typ des Rückgabewertes.

Den Referenzen oder Zeigern (Pointern) der Klasse Liste ist nicht anzusehen, ob sie gerade auf einen Kunden oder auf einen Dozenten verweisen. Neben dem durch die Deklaration festgelegten statischen Typ haben solche Referenzen noch einen *dynamischen Typ*. Dieser ist gegeben durch den Typ des Objekts, auf das gerade verwiesen wird; er ist veränderlich. Dieser *Polymorphismus* von Referenzen ist von zentraler Bedeutung für die objektorientierte Programmierung.

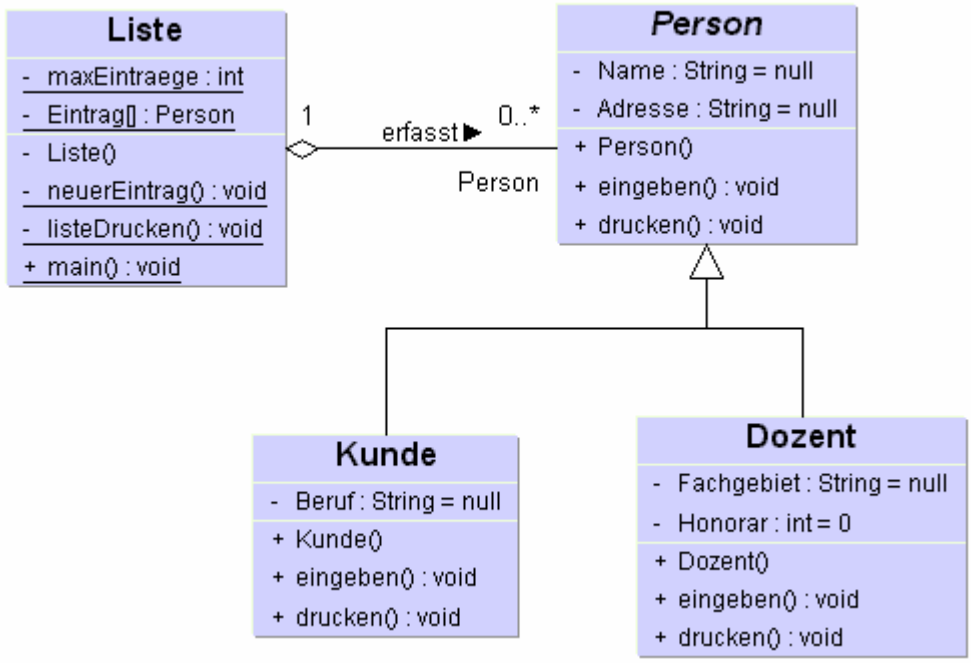
Der Aufruf überschriebener Methoden richtet sich grundsätzlich nach dem dynamischen Typ der Referenz. Anders ausgedrückt: Der Methodenaufruf einer überschriebenen Methode hängt vom Typ des Objekts ab und nicht etwa vom statischen Typ der Referenz.

Design: OOD-Klassendiagramm

Das OOD-Klassendiagramm enthält weitere Präzisierungen. Das sind insbesondere

1. die Typen der Attribute und deren Initialisierung,
2. die Typen der Parameter und Rückgabewerte der Methoden, und
3. die Festlegungen gemäß *Geheimnisprinzip*: Private Attribute und Methoden sind mit einem Minuszeichen markiert. Öffentliche (public) Elemente erhalten ein Pluszeichen. Private Elemente sind nur innerhalb einer Klasse und innerhalb deren Objekte sichtbar. Die öffentlichen Elemente bilden die Schnittstelle nach außen.
4. Konstruktoren unterscheiden sich von den Methoden dadurch, dass es keine Rückgabewerte gibt. Sie werden bei der Generierung von Objekten aufgerufen und dienen hauptsächlich der Initialisierung der Attribute des Objekts. Konstruktoren werden durch den Klassennamen bezeichnet: Der Konstruktor der Klasse Kunde beispielsweise ist Kunde().

Festlegungen für die "objektorientierte Programmierung mit C" in Stichworten



Klasse (physikalisch) = Modul = *KlassenName*.h + *KlassenName*.c

Klasse (als Struktur) = `struct`-Datentyp

Attribut: Membervariable für Daten

Methode: Funktionspointer als Membervariable

Objekt: dynamische Variable des `struct`-Datentyps

Vererbung und Polymorphismus: Untypisierte Pointer (void*) auf Container für die Attribute der Unterklassen. Die Container sind im Allgemeinen vom `struct`-Datentyp ("Puppen-in-Puppen"-Prinzip).

Geheimnisprinzip:

- Interface = Header-Datei (.h) mit öffentlichen Deklarationen
- Implementation = Programmdatei (.c) mit privaten Definitionen

Konstruktoren: Der Konstruktor für Objekte der Klasse `Kunde` beispielsweise wird im C-Programm nicht `Kunde()` geschrieben, sondern `nKunde()`, sprich: "neuer Kunde".

Die Realisierung in C: Schnittstellen (.h-Dateien) der Klassen[1]

```
/** Person.h - abstrakte Klasse

Das Modul bietet die Deklaration der Klasse Person und einen
Konstruktor fuer diese Klasse. Die Klasse ist eine Struktur. Die
Membervariablen des Schnittstellenteils (Interface) sind
Funktionspointer (oeffentliche Methoden). Als weitere Membervariable
gibt es einen untypisierten Pointer auf den Container fuer die
Attribute der Klasse. Dieser Container wird in der Implementierungs-
datei Person.c definiert. Er enthaelt die privaten Attribute sowie
einen Pointer zum Anhaengen der Container von Subklassen. Ueber die
oeffentlichen Methoden setSub() und getSub() sind diese Container
zugaenglich.
*****************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

#ifndef Person
#define Person struct Person

/** Deklaration der Klasse */
Person {
  /** Oeffentliche Methoden (Interface der Klasse) */
  void  (*eingeben)(Person* self);
  void  (*drucken)(Person* self);
  void  (*setSub)(Person* self, void* sub);
  void* (*getSub)(Person* self);

  /*Pointer auf privaten Container der Klasse Person*/
  void* con;
};
#endif

/** Konstruktor */
Person* nPerson();


/* Kunde.h */
#include "Person.h"

/** Konstruktor */
Person* nKunde();


/* Dozent.h */
#include "Person.h"

/** Konstruktor */
Person* nDozent();
```

Implementierungsdateien (.c-Dateien) der Klassen

```
/** Person.c*/
```

[1] Bei den Realisierungen ist aus Gründen der Übersichtlichkeit darauf verzichtet worden, mögliche Bereichsüberschreitungen beim Eingeben der Liste abzufangen.

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "Person.h"

#define Container struct Container
static Container{char* Name; char* Adresse; void* sub;} *c;

Container* container(Person* self) {return (Container*)self->con;}

void setSub(Person* self, void* sub) {container(self)->sub=sub;}

void* getSub(Person* self) {return container(self)->sub;}

static void eingeben(Person* self) {
  char buf[80];

  printf("\n? Name: "); gets(buf);
  (c=container(self))->Name=(char*)malloc(strlen(buf)+1);
  strcpy(c->Name, buf);

  printf("\n? Adresse: "); gets(buf);
  c->Adresse=(char*)malloc(strlen(buf)+1);
  strcpy(c->Adresse, buf);
}

static void drucken(Person* self) {
 printf("\n! Name: %s", (c=container(self))->Name);
 printf("\n! Adresse: %s", c->Adresse);
}

Person* nPerson() {
  Person* p=(Person*)malloc(sizeof(Person));
  c=(Container*)malloc(sizeof(Container));
  c->Name=c->Adresse=c->sub= NULL;

  p->eingeben=eingeben;
  p->drucken=drucken;
  p->setSub=setSub;
  p->getSub=getSub;
  p->con=c;

  return p;
}
```

/* Kunde.c

```
Finale Klasse. Unterklassen sind nicht vorgesehen.
*************************************************/
#include "Kunde.h"

static Person* super=NULL; /*erschliesst die Methoden der Superklasse*/

#define Container struct Container
static Container {char* Beruf;} *c;

static Container* container(Person* self) {
  return (Container*)super->getSub(self);
}

static void eingeben(Person* self){
  char buf[80];

  super->eingeben(self);
  printf("\n? Beruf: "); gets(buf);
  (c=container(self))->Beruf=(char*)malloc(strlen(buf)+1);
  strcpy(c->Beruf, buf);
}
```

```
static void drucken(Person* self) {
  super->drucken(self);
  printf("\n! Beruf: %s", container(self)->Beruf);
}

Person* nKunde() {
  Person* p=nPerson();
  if(super==NULL)super=nPerson();
  c=(Container*)malloc(sizeof(Container));
  c->Beruf= NULL;
  super->setSub(p, c);

  /** Ueberschreiben der Methoden (overriding) */
  p->eingeben= eingeben;
  p->drucken= drucken;
  p->setSub= NULL;
  p->getSub= NULL;

  return p;
}
```

/* Dozent.c

```
Finale Klasse. Unterklassen sind nicht vorgesehen.
************************************************/
#include "Dozent.h"

static Person* super=NULL; /*erschliesst die Methoden der Superklasse*/

#define Container struct Container
static Container {char* Fachgebiet; char* Honorar;} *c;

static Container* container(Person* self) {
  return (Container*)super->getSub(self);
}

static void eingeben(Person* self){
  char buf[80];

  super->eingeben(self);

  printf("\n? Fachgebiet: "); gets(buf);
  (c=container(self))->Fachgebiet=(char*)malloc(strlen(buf)+1);
  strcpy(c->Fachgebiet, buf);

  printf("\n? Honorar: "); gets(buf);
  c->Honorar=(char*)malloc(strlen(buf)+1);
  strcpy(c->Honorar, buf);
}

static void drucken(Person* self) {
  super->drucken(self);
  printf("\n! Fachgebiet: %s", (c=container(self))->Fachgebiet);
  printf("\n! Honorar: %s", c->Honorar);

  /*Denkfalle: Die Variante
    printf("\n! Fachgebiet: %s\n! Honorar: %s",
    ((c=container(self))->Fachgebiet, c->Honorar);
  funktioniert nicht, da Berechnungsreihenfolge der Argumente undefiniert!
  ************************************************************************/
}

Person* nDozent() {
  Person* p=nPerson();
  if (super==NULL) super=nPerson();
  c=(Container*)malloc(sizeof(Container));
```

```
  c->Fachgebiet= c->Honorar= NULL;
  super->setSub(p, c);

  /** Ueberschreiben der Methoden (overriding) */
  p->eingeben=eingeben;
  p->drucken=drucken;
  p->setSub=NULL;
  p->getSub=NULL;

  return p;
}
```

Das C-Hauptprogramm

```
/** Liste.c, Timm Grams, Fulda, 6.9.01 (28.10.01)

Hauptklasse des C-Projekts "Tutorial - ein Lehrgangsverwaltungssystem.
DasProjekt dient der Demonstration einiger Grundkonzepte der
objektorientierten Programmierung.

Das Hauptprogramm main entspricht der "public static main"-Methode einer
Java Applikation.

Die Attribute und Methoden der Hauptklasse "Liste" werden hier global und
extern definiert. Das entspricht in Java den als static deklarierten
Attributen und Methoden. Ein Konstruktor wird nicht benoetigt.
*************************************************************************/
#include "Person.h"
#include "Kunde.h"
#include "Dozent.h"

#define maxEintraege 200

/** Vektor der Listeneintraege*/
Person* Eintrag[maxEintraege];

void neuerEintrag(char c) {
 int i=0;  Person* p;
 while (i<maxEintraege&&Eintrag[i]!=NULL) i++;
 switch (c) {
   case 'd': Eintrag[i]=p=nDozent(); p->eingeben(p); break;
   case 'k': Eintrag[i]=p=nKunde(); p->eingeben(p); break;
 }
}

void listeDrucken() {
 int i;
 for (i=0; i<maxEintraege; i++) if (Eintrag[i]!=NULL)
 Eintrag[i]->drucken(Eintrag[i]);
}

main() {
  char buf[80], c;
  printf("\nLehrgangsverwaltung - ein Tutorial der ooP");
  do {
    printf("\n? <D>ozent, <K>unde>, <S>chluss ");
    gets(buf);
    c=tolower(*buf);
    if (c=='k'||c=='d') neuerEintrag(c);
    else if (c!='s') printf("\nVertippt?");
  } while(c!='s');
  printf("\n\n! Ausgabe der Liste");
  listeDrucken();
  return 0;
}
```

Java-Programmtexte

IO.java

```
import java.io.*;

public class IO {
  static byte buf[]=new byte[129];

  public static String inStr() {
    int i=0;
    try {
      i= System.in.read(buf);
    }catch(IOException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }catch(NumberFormatException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }
    return new String(buf, 0, i);
  }

  public static double inDbl() {return new Double(inStr()).doubleValue();}
  public static void outStr(String s) {System.out.print(s);}
  public static void outInt(int s) {System.out.print(Integer.toString(s));}

}
```

// Person.java

```
public abstract class Person {
  private String Name = null;
  private String Adresse = null;

  public  Person() {}

  public void eingeben() {
    IO.outStr("? Name: ");
    Name=IO.inStr();

    IO.outStr("? Adresse: ");
    Adresse=IO.inStr();
  }

  public void drucken() {
    IO.outStr("! Name: "); IO.outStr(Name);
    IO.outStr("! Adresse: "); IO.outStr(Adresse);
  }

}
```

// Kunde.java

```
public class Kunde extends Person {
  private String Beruf = null;

  public  Kunde() {}
  public void eingeben() {
    super.eingeben();
    IO.outStr("? Beruf: ");
    Beruf=IO.inStr();
  }
  public void drucken() {
    super.drucken();
    IO.outStr("! Beruf: "); IO.outStr(Beruf);
  }
}
```

```
// Dozent.java

public class Dozent extends Person {
  private String Fachgebiet = null;
  private int Honorar = 0;

  public  Dozent() { }

  public void eingeben() {
    super.eingeben();
    IO.outStr("? Fachgebiet: "); Fachgebiet=IO.inStr();
    IO.outStr("? Honorar: "); Honorar=(int)IO.inDbl();
  }

  public void drucken() {
    super.drucken();
    IO.outStr("! Fachgebiet: "); IO.outStr(Fachgebiet);
    IO.outStr("! Honorar: "); IO.outInt(Honorar); IO.outStr("\n");
  }

}

/** Liste.java, Timm Grams, 5.9.2001

Hauptklasse der Java-Version des Programms zu "Tutorial - ein
Lehrgangsverwaltungssystem".

Das Projekt dient der Demonstration von Grundkonzepten der
objektorientierten Programmierung.

Neben der Java-Version existiert eine C-Version des Programms.

*************************************************************************
*/

public class Liste
{
  static private int maxEintraege=200;

  /** Vektor der Listeneintraege */
  static private Person[] Eintrag=new Person[maxEintraege];

  static private void neuerEintrag(char c) {
   int i=0;  Person p;
   while (i<maxEintraege&&Eintrag[i]!=null) i++;
   switch (c) {
     case 'd': Eintrag[i]=p=new Dozent(); p.eingeben(); break;
     case 'k': Eintrag[i]=p=new Kunde(); p.eingeben(); break;
   }
  }

  static private void listeDrucken() {
    int i;
    for (i=0; i<maxEintraege; i++) if (Eintrag[i]!=null)
    Eintrag[i].drucken();
  }

  public static void main (String[] args) {
    char c;
    IO.outStr("\nLehrgangsverwaltung - ein Tutorial der ooP");
    do {
      IO.outStr("\n? <D>ozent, <K>unde, <S>chluss ");
      c=Character.toLowerCase(IO.inStr().charAt(0));

      if (c=='k'|| c=='d') neuerEintrag(c);
      else if (c!='s') IO.outStr("\nVertippt?");
```

```
    } while(c!='s');
    IO.outStr("\n\n! Ausgabe der Liste\n");
    listeDrucken();
  }
}
```