

```
23 again = false;  
24 getline(cin, sInput);  
25 system("cls");  
26 stringstream(sInput) >> dblTemp;  
27 iLength = sInput.length();  
28 if (iLength < 4) {  
29     again = true;  
    continue;    +[iLength - 3] != '.') {
```

Thomas

C23-04 Vererbung

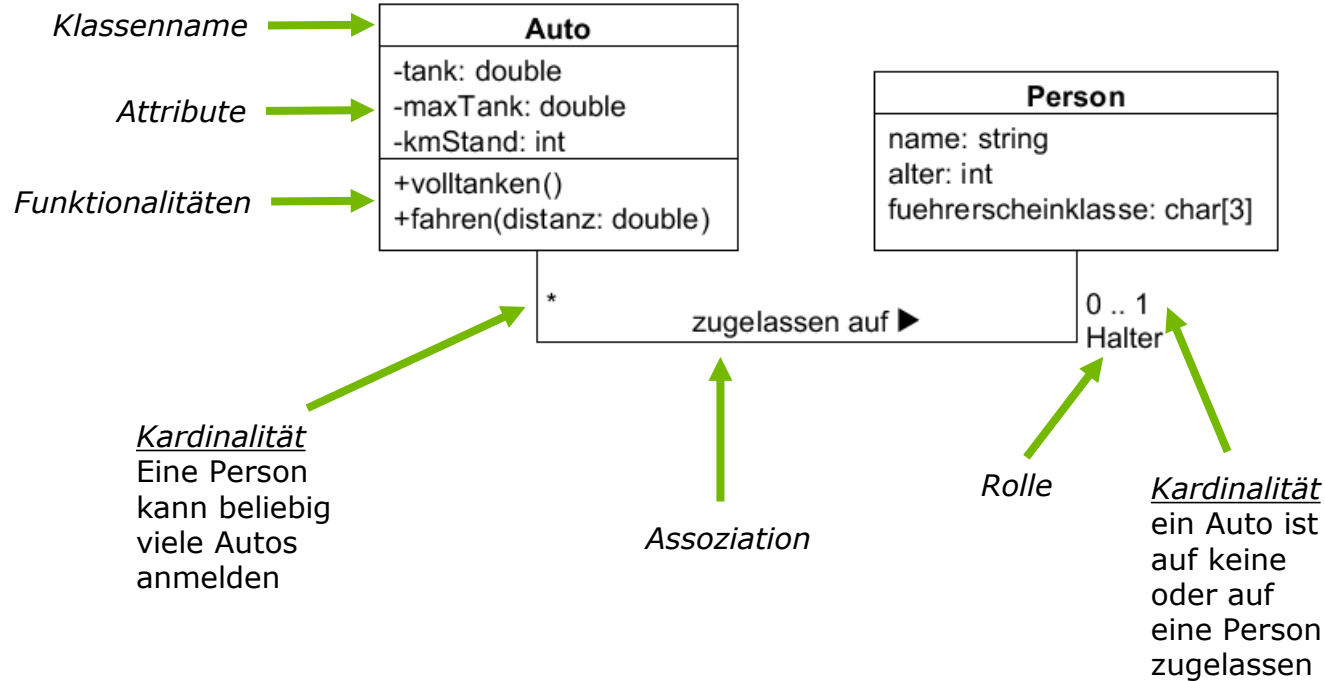
Fortgeschrittene Algorithmen und Programmierung

v4

Kurzeinführung: UML – Klassendiagramme

Übersicht

2



Beschreibung von Klassen

- Klassenbeschreibungen enthalten Member-Funktionen und Member-Variablen
- Der Zugriffsbereich von Attributen und Funktionen wird durch die vorangestellten Zeichen „+ - #“ symbolisiert → +public, -private, #protected
- Der Datentyp wird durch „ : “ hinter den Klassenelementen angegeben.
- Statische Klassenvariablen und Funktionen werden unterstrichen.
- Reine Zugriffsfunktionen (Getter- und Setter) werden nicht modelliert.

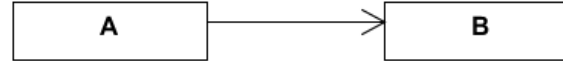
Kurzeinführung: UML – Klassendiagramme

Assoziationen zwischen Klassen

- Assoziationen zwischen Klassen können durch Beschriftungen erläutert werden.
→ Welcher Art ist die Assoziation?
- Es können Multiplizitäten bzw. Kardinalitäten ausgedrückt werden.
→ Wieviel Objekte sind bei der Assoziation beteiligt?
 - MIN ... MAX z.B.: 0 .. 1 0 .. * 1 .. 5 1 .. *
 - Genaue Anzahl z.B.: 1
 - Beliebige Anzahl *
- Rollen beschreiben das Verhältnis zwischen Klassen näher.
→ Welche Bedeutung nehmen die Klassen gegenseitig ein?

Besondere Assoziationen

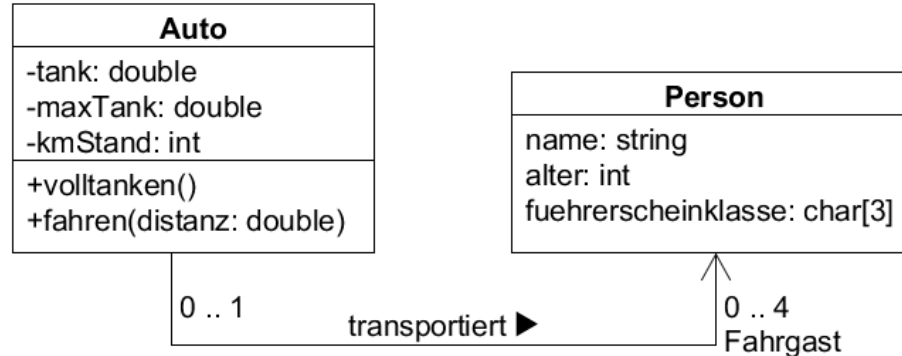
- Navigation (Spezialfall einer Assoziation):



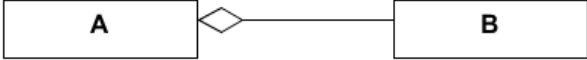
- Man kann über A auf B zugreifen.
- A hat ein Objekt von B oder eine [Liste von] Referenz oder Pointer.

Beispiel:

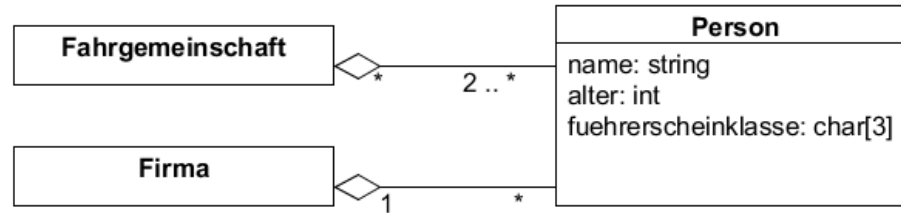
Über ein Objekt von der Klasse **Auto** kann man auf alle **Personen**-Objekte zugreifen, die gerade im Auto mitfahren.



Besondere Assoziation

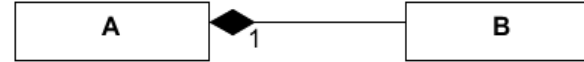
- Aggregation (Spezialfall einer Navigation):
- B ist ein Teil von A.
- Ein Objekt von B kann auch gleichzeitig Teil anderer Objekte sein.
- B kann auch ohne A existieren.
- Aggregationen werden meist durch Referenzen oder Pointer implementiert.

Beispiel: Eine Person kann gleichzeitig Teil einer Firma UND einer Fahrgemeinschaft sein.



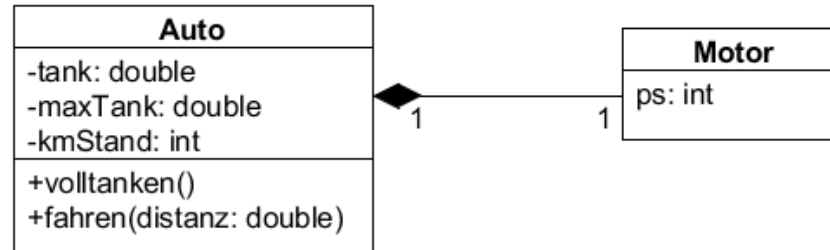
Besondere Assoziationen

- Komposition (Spezialfall einer Aggregation):



- B ist ein Teil von A.
- Ein Objekt von B kann immer nur zu einem Objekt von A gleichzeitig gehören.
- B kann nur existieren, wenn auch A existiert.
- In der Implementierung sind Objekte von B meistens Member in A.

Beispiel: ein Motor kann immer nur in einem Auto verbaut sein. Der Motor ist ohne Auto nutzlos.

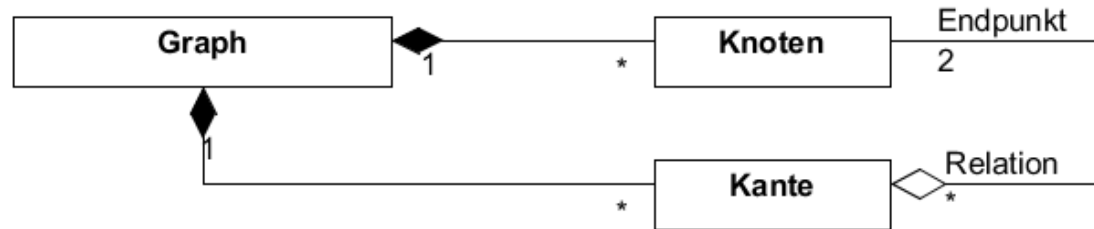


Kurzeinführung: UML – Klassendiagramme

8

Beispiel: Graph

- Ein Graph kann beliebig viele Knoten und Kanten haben
- Ein Knoten ist Teil genau eines Graphen
- Eine Kante ist Teil genau eines Graphen
- Eine Kante besteht aus je zwei Knoten
- Es können beliebig viele Kanten zu einem Knoten führen



Kurzeinführung: UML – Klassendiagramme

Weiterführende Informationen

Klassendiagramme

- <http://de.wikipedia.org/wiki/Klassendiagramm>
- [http://de.wikipedia.org/wiki/Assoziation %28UML%29](http://de.wikipedia.org/wiki/Assoziation_%28UML%29)
- http://openbook.rheinwerk-verlag.de/oop/oop_kapitel_04_003.htm
- <http://www.agilemodeling.com/artifacts/classDiagram.htm>

Freeware Programme

- Umlet: <http://www.umlet.com/>
- NClass: <http://nclass.sourceforge.net/>
- Modelio: <https://www.modelio.org/>

Redundanz(-vermeidung)

- Alle Eigenschaften und Funktionen von **Auto** sind auch in **Taxi** und **Paketbote** enthalten.
→ Der **Paketbote** IST EIN **Auto** und das **Taxi** IST EIN **Auto**!

Auto
-tank: double -maxTank: double -kmStand: int
+volltanken() +fahren(distanz: double)

Taxi
-tank: double -maxTank: double -kmStand: int -bilanzEuroCent: int -fahrpreis: double
+volltanken() +fahren(distanz: double) +serviceFahrt(distanz: double, anzFahrgaeste: int)

PaketBote
-tank: double -maxTank: double -kmStand: int -bilanzEuroCent: int -preisProPaketProKm: double
+volltanken() +fahren(distanz: double) +ausliefern(distanz: double, anzPakete: int)

Erweiterbarkeit

- Soll etwas in Auto ergänzt oder geändert werden, so müssen Taxi und Paketbote auch angepasst werden, um die gleiche Änderung zu übernehmen.
→ Code wird mehrfach geschrieben

Auto
-tank: double -maxTank: double -kmStand: int -verbrauch: double
+volltanken()

Taxi
-tank: double -maxTank: double -kmStand: int -bilanzEuroCent: int -fahrpreis: double -verbrauch: double
+volltanken() +fahren(distanz: double) +serviceFahrt(distanz: double, anzFahrgaeste: int)

PaketBote
-tank: double -maxTank: double -kmStand: int -bilanzEuroCent: int -preisProPaketProKm: double -verbrauch: double
+volltanken() +fahren(distanz: double) +ausliefern(distanz: double, anzPakete: int)

Wiederverwendbarkeit

- Soll ein weiterer Spezialfall von Auto erzeugt werden, müssen wieder alle Funktionen und Member-Variablen in die neue Klasse kopiert werden.

Auto
-tank: double -maxTank: double -kmStand: int -verbrauch: double
+volltanken() +fahren(distanz: double)

Firmenwagen
-tank: double -maxTank: double -kmStand: int -verbrauch: double -firma: string -mitarbeiter-ID: unsigned int
+volltanken() +fahren(distanz: double)

Vererbung

Grundbegriffe

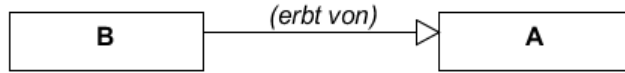
- Die Klasse, von der geerbt wird, heißt **Basisklasse** oder Elternklasse.
- Die erbende Klasse heißt **abgeleitete Klasse** oder Kindklasse.
- Durch Vererbung übernimmt eine abgeleitete Klasse alle Member-Variablen und Member-Funktionen ihrer Basisklasse.
- Wird die Basisklasse verändert, betrifft das auch alle abgeleiteten Klassen!

Vererbung

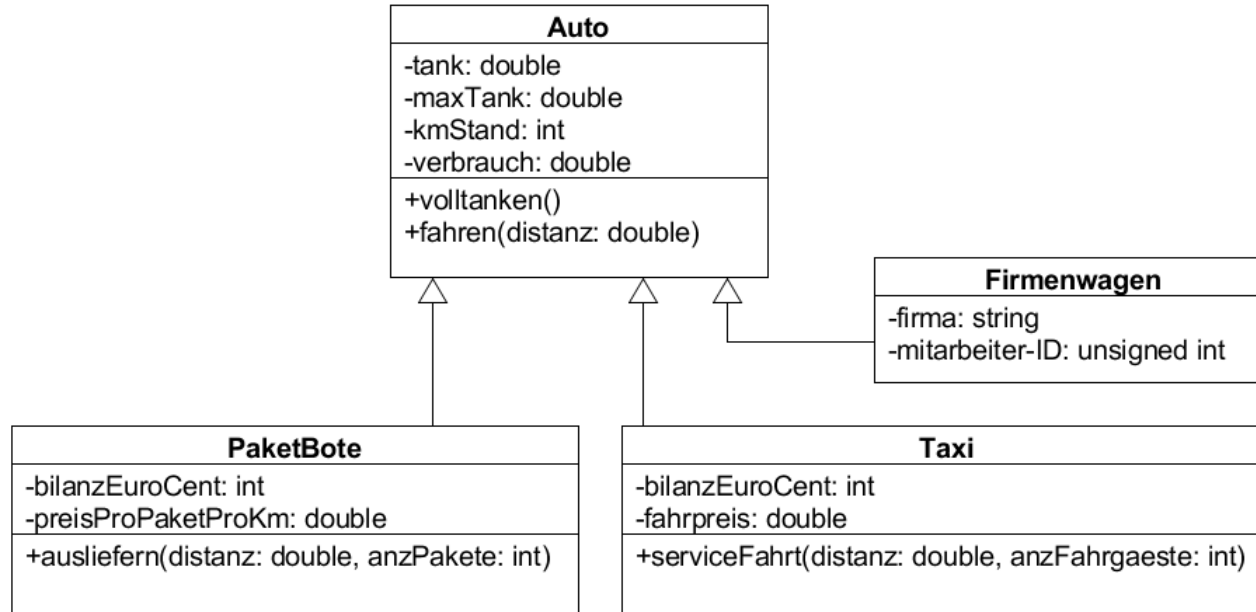
Beschreibung in UML

14

Syntax



Beispiel



Entwurf von Vererbungsstrukturen

- Beim Softwareentwurf entstehen Vererbungshierarchien zwischen Klassen. Der Prozess ist iterativ. Es werden zwei grundlegende Prinzipien angewendet:

Generalisierung

Wenn in einem Softwareprojekt mehrere Klassen die gleichen Eigenschaften und Funktionen haben, dann wird eine gemeinsame Basisklasse erstellt, die diese gemeinsamen Elemente enthält.

→ Die ursprünglichen Klassen werden zu abgeleiteten Klassen.
Sie erben von der neuen Basisklasse.

Spezialisierung (Verfeinerung)

Eine existierende Klasse soll um speziellere Daten erweitert werden. Diese neuen Eigenschaften passen aber nicht zu den bisher erstellten Objekten der ursprünglichen Klasse.

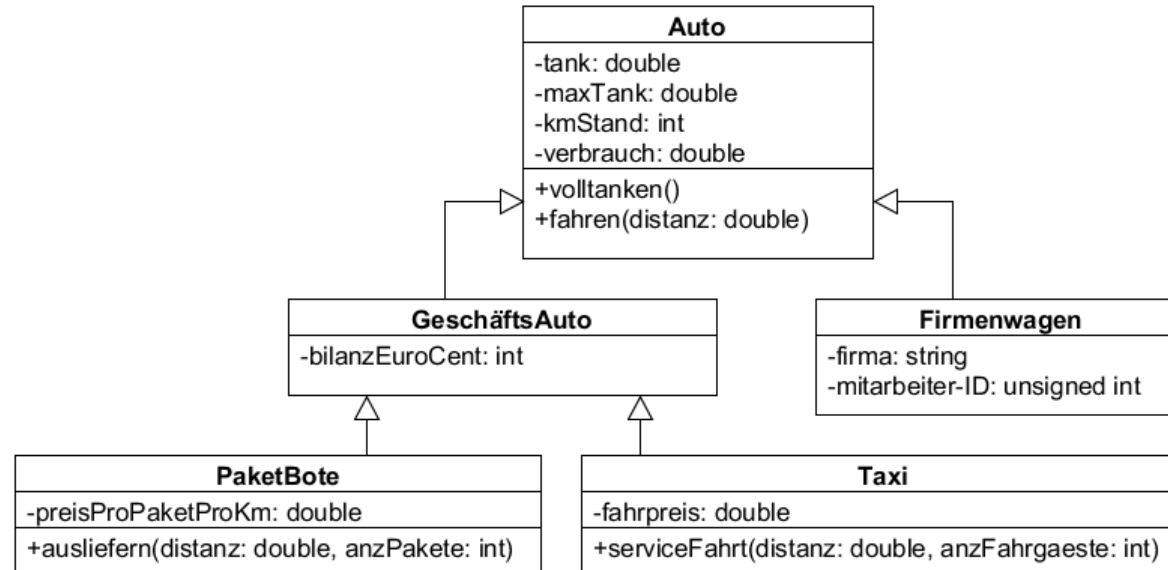
→ Es wird eine neue abgeleitete Klasse erstellt, die die neuen Eigenschaften bekommt. Die ursprüngliche Klasse wird zur Basisklasse.

Vererbung

Generalisierung

Beispiel

- Die Klassen Paketbote und Taxi enthalten beide die Member-Variable ‚bilanzEuroCent‘.
→ Diese Eigenschaft wird in eine neue Basisklasse ausgelagert

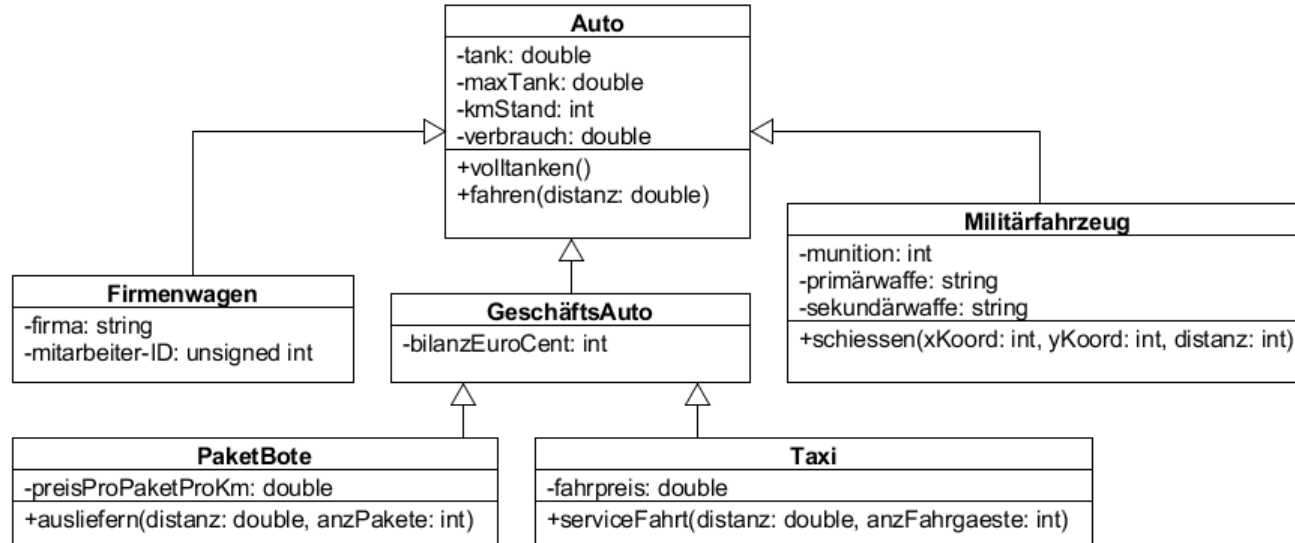


Vererbung

Spezialisierung

Beispiel

- Es sollen Eigenschaften für einen militärischen Einsatz von Autos hinzugefügt werden. Diese speziellen Eigenschaften passen nicht zu den anderen abgeleiteten Fahrzeugklassen.
→ Es wird eine neue abgeleitete Klasse „Militärfahrzeug“ erstellt.



Vererbung

Vererbung in C++

Syntax

```
class AbgeleiteteKlasse : [private|protected|public] Basisklasse  
{  
    ...  
};
```

- Durch optionale Angabe eines Zugriffsbereiches (public, protected oder private) kann die abgeleitete Klasse die Sichtbarkeit der geerbten Elemente einschränken.
- Falls keine Angabe gemacht wird, ist die Vererbung private.
- Achtung: friend-Beziehungen werden nicht mit vererbt.

Vererbung in C++

Unterschied zwischen private und protected

- Eine Klasse kann mit den Zugriffsbereichen `private` oder `protected` unterscheiden, welche Elemente in abgeleiteten Klassen direkt zugänglich sind.
 - Abgeleitete Klassen können nicht auf `private` Elemente der Basisklasse zugreifen. (Trotzdem werden die Elemente vererbt!)
 - Abgeleitete Klassen dürfen auf alle `protected` Elemente der Basisklasse direkt zugreifen.
 - Nach ‚außen‘ (aus Sicht der Objekte) verhält sich `protected` wie `private`.

→ Sowohl Basisklassen als auch abgeleitete Klassen haben Einfluss auf die Sichtbarkeit der Klassenelemente bei der Vererbung!

Vererbung in C++

```
class Basisklasse { };  
  
class A : public    Basisklasse { };  
class B : protected Basisklasse { };  
class C : private   Basisklasse { };  
class D :           Basisklasse { };
```

Ist ein Element in der Basisklasse ...	public	protected	private
... wird es in A	public	protected	unzugänglich
... wird es in B	protected	protected	unzugänglich
... wird es in C oder D	private	private	unzugänglich

Vererbung

Beispiel

Beispiel 1

StudentPublic.cpp

```
#include <iostream>
using namespace std;

class Person // Basisklasse
{
public:
    int alter;
};

class Student : public Person
{
public:
    int semester;
    float note;
    // int alter; -> wird geerbt!
};
```

Fortsetzung folgt ...

Beispiel 1

StudentPublic.cpp (Fortsetzung)

```
int main()
{
    Student s;

    s.alter = 20; // von Person geerbt!
    s.semester = 1;
    s.note = 2.0;

    cout << "\n" << "Alter: " << s.alter
         << "\n" << "Semester: " << s.semester
         << "\n" << "Note: " << s.note
         << endl;

    return 0;
}
```

```
Alter: 20
Semester: 1
Note: 2
```

Beispiel 2 (Vererbung mit Protected)

StudentProtected1.cpp

```
#include <iostream>
using namespace std;

class Person // Basisklasse
{
public:
    int alter;
};

class Student : protected Person
{
public:
    int semester;
    float note;
    // Zugriff auf Public-Elemente der Basisklasse:
    int getAlter() { return alter; }
    void setAlter(int alter) { this->alter = alter; }

    // Durch protected - Vererbung implizit:
    // protected:
    //     int alter;
};
```

Fortsetzung folgt ...

Beispiel 2 (Vererbung mit Protected)

StudentProtected1.cpp (Fortsetzung)

```
int main()
{
    Student s;

    // s.alter = 20;    -> Geht nicht: Ist in Student protected!
    s.setAlter(20);
    s.semester = 1;
    s.note = 2.0;

    cout << "\n" << "Alter: " << s.getAlter()
         << "\n" << "Semester: " << s.semester
         << "\n" << "Note: " << s.note
         << endl;

    return 0;
}
```

```
Alter: 20
Semester: 1
Note: 2
```


Vererbung

Umdefinieren von Zugriffsbereichen

- Bei private- oder protected- Vererbung werden alle public-Elemente der Basisklasse nach außen (für die Objekte) unzugänglich.
- Mit using lassen sich einzelne Elemente umdefinieren. So können Elemente selektiv wieder nach außen sichtbar gemacht werden.

StudentProtected2.cpp

```
#include <iostream>
using namespace std;

class Person // Basisklasse
{
public:
    int alter;
};

class Student : protected Person
{
public:
    int semester;
    float note;
    using Person::alter; // Umdefinieren des Zugriffsbereichs
};
```

Fortsetzung folgt ...

Vererbung

Umdefinieren von Zugriffsbereichen

StudentProtected2.cpp (Fortsetzung)

```
int main()
{
    Student s;

    s.alter = 20; // wurde wieder in public umdefiniert!
    s.semester = 1;
    s.note = 2.0;

    cout << "\n" << "Alter: " << s.alter
         << "\n" << "Semester: " << s.semester
         << "\n" << "Note: " << s.note
         << endl;

    return 0;
}
```

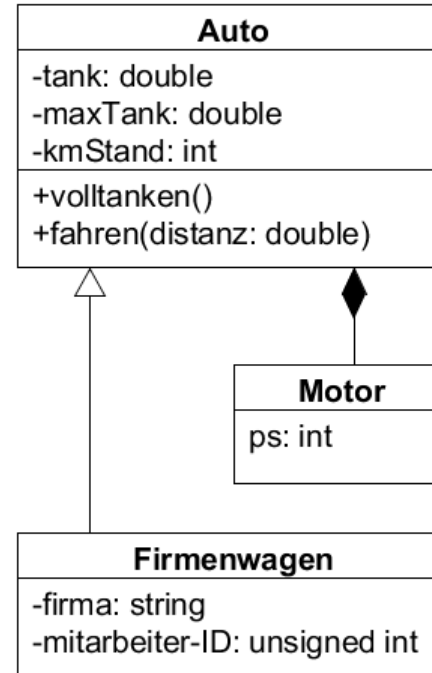
```
Alter: 20
Semester: 1
Note: 2
```

Vererbung

Vererbung oder Komposition?

Faustregel

- Liegt eine „IST-EIN“ Beziehung vor, ist Vererbung die richtige Wahl.
Merkmal: Alle Attribute der Basisklasse passen auch zu den abgeleiteten Klassen.
- Bei einer „HAT-EIN“ Beziehung wird eine Klasse das Member-Objekt der anderen Klasse (Komposition).



Vererbung

Initialisierung von Basisklassen und Member-Objekten

- Hat eine Basisklasse keinen Standardkonstruktor, muss sie in der Initialisierungsliste der abgeleiteten Klasse initialisiert werden!
- Member-Objekte müssen ebenfalls in einer Initialisierungsliste initialisiert werden, wenn die Klasse des Member-Objektes keinen Standardkonstruktor hat.
- Konstruktoren von Basisklassen oder Member-Objekten können nur über Initialisierungslisten explizit verwendet werden.

Vererbung

Initialisierung von Basisklassen und Member-Objekten

Beispiel

ErstwagenZweitwagen.cpp

```
#include <iostream>
using namespace std;

class Motor
{
public:
    Motor(int ps) : m_ps(ps) { } // kein Standardkonstruktor

    int getPS() {return m_ps;}

private:
    int m_ps;
};
```

Vererbung

Initialisierung von Basisklassen und Member-Objekten

Beispiel

ErstwagenZweitwagen.cpp (Fortsetzung)

```
class Fahrzeug
{
public:
    Fahrzeug(int radAnzahl) : m_radAnzahl(radAnzahl) { }

    int getRadanzahl() { return m_radAnzahl; }

private:
    int m_radAnzahl;
};
```

Vererbung

Initialisierung von Basisklassen und Member-Objekten

Beispiel

ErstwagenZweitwagen.cpp (Fortsetzung)

```
class Auto: public Fahrzeug    // Auto ist ein Fahrzeug
{
public:
    Auto(int radAnzahl, int ps)
        : Fahrzeug(radAnzahl), m_motor(ps) {}

    Auto() : Fahrzeug(4), m_motor(98) { }

    int getPS() { return m_motor.getPS(); }

private:
    Motor m_motor;    // Auto hat einen Motor
};
```

Vererbung

Initialisierung von Basisklassen und Member-Objekten

Beispiel

ErstwagenZweitwagen.cpp (Fortsetzung)

```
int main()
{
    Auto erstwagen(6, 400);
    Auto zweitwagen;

    cout << "Erstwagen: " << endl
          << "PS: " << erstwagen.getPS() << endl
          << "Anzahl der Raeder: " << erstwagen.getRadanzahl() << endl;

    cout << "\nZweitwagen: " << endl
          << "PS: " << zweitwagen.getPS() << endl
          << "Anzahl der Raeder: " << zweitwagen.getRadanzahl() << endl;

    return 0;
}
```

```
Erstwagen:
PS: 400
Anzahl der Raeder: 6

Zweitwagen:
PS: 98
Anzahl der Raeder: 4
```


Vererbung

Konvertierung

- Implizite Konvertierung zur Basisklasse ist immer möglich!

```
void printFahrzeug(const Fahrzeug& rFahrzeug)
{
    cout <<"Ein Fahrzeug mit " << rFahrzeug.getRadanzahl()
          <<" Raedern." <<endl;
}

int main()
{
    Auto erstwagen(6, 400);
    printFahrzeug(erstwagen);
}
```

Ein Fahrzeug mit 6 Raedern.

Vererbung

Überschreiben von Member-Funktionen

- **Abgeleitete Klassen können Member-Funktionen der Basisklasse überschreiben!**
(d.h., die Inhalte der Funktion bei gleichbleibendem Interface neu definieren)
- Eine überschriebene Member-Funktion einer Basisklasse ist in der abgeleiteten Klasse über den Scope-Operator `::` erreichbar.
- Soll ein Objekt der abgeleiteten Klasse eine überschriebene Funktion der Basisklasse nutzen, muss es explizit in die Basisklasse konvertiert (gecastet) werden.

Vererbung

Überschreiben von Member-Funktionen

Beispiel

StudentOverride.cpp

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

class Person // Basisklasse
{
public:
    Person(const string& rFirstname, const string& rLastname)
        : m_firstname(rFirstname), m_lastname(rLastname) { }

    // Person::toString
    string toString() const { return m_firstname + " " + m_lastname; }
private:
    string m_firstname;
    string m_lastname;
};
```

Vererbung

Überschreiben von Member-Funktionen

Beispiel

StudentOverride.cpp (Fortsetzung)

```
class Student : public Person
{
public:
    Student(const string& rFirstname, const string& rLastname, int semester = 1)
        : Person(rFirstname, rLastname), m_semester(semester) { }

    // Student::toString überschreibt Person::toString
    string toString() const {
        stringstream s;
        // Zugriff auf überschriebene Funktion der Basisklasse über Scope-Operator!
        s << Person::toString() << " ist im " << m_semester << ". Semester.";
        return s.str();
    }

private:
    int m_semester;
};
```

Vererbung

Überschreiben von Member-Funktionen

Beispiel

StudentOverride.cpp (Fortsetzung)

```
void printPerson(const Person& rPerson) {  
    // Person::toString  
    std::cout<<"Name: " <<rPerson.toString() <<std::endl;  
}  
  
int main()  
{  
    Student s1("Jens", "Meier");  
    cout << s1.toString() <<endl;  
  
    Student s2("Guo", "Chuang", 5);  
    cout << s2.toString() <<endl;  
    // Zugriff auf Basisklasse durch explizite Konvertierung  
    cout << "Name: " << ((Person) s2).toString() << endl;  
  
    Student s3("Steve", "Durand", 3);  
    printPerson(s3); // implizite Konvertierung  
  
    return 0;  
}
```

```
Jens Meier ist im 1. Semester.  
Guo Chuang ist im 5. Semester.  
Name: Guo Chuang  
Name: Steve Durand
```

Vererbung

Best Practice

- Vererbung ist sinnvoll, wenn eine „IST EIN“-Beziehung zwischen Klassen vorliegt. Ansonsten ist Komposition das geeignete Werkzeug.
- Vererbung sollte in der Regel `public` sein, da die Basisklasse erweitert und nicht eingeschränkt werden sollte.
- Überschriebene Funktionen einer Basisklasse sollten in der abgeleiteten Klasse erweitert, aber nicht grundsätzlich (sinnentstellend) verändert werden.



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

www.htw-berlin.de