C++ - Objektorientierte Programmierung

im Vergleich mit Java

Fachgruppe Computergraphik und Multimediasysteme

Universität Siegen – Fachbereich 12

(Folien: Nicolas Cuntz)

Inhalt

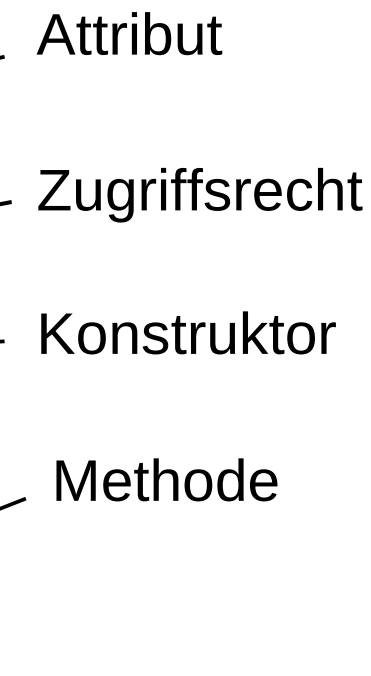
1. Klassendefinition

- Ein **Beispiel**:

```
class Zahl
{
   private:
   int x;

   public:
   Zahl()
   {
      x = 0;
   }

   int getX()
   {
      return x;
   }
};
```



1. Klassendefinition - Zugriffsrechte

- **Zugriffsrechte** werden in C++ blockweise definiert:

```
class Zahl
{
   private:
   int x;

   public:
   Zahl()
   {
      x = 0;
   }

   int getX()
   {
      return x;
   }
};
```

Zugriffsrechte:

private: Zugriff nur innerhalb der Klasse

public: Zugriff in allen Klassen

protected: Zugriff innerhalb der eigenen Klasse, in allen abgeleiteten Klassen (und in *friend*-Klassen)

Das *default*-Zugriffsrecht ist **`private`**

1. Klassendefinition

- In C++ wird die Schnittstellendefinition in der Regel von der Implementierung der Methoden getrennt:

zahl.h

```
class Zahl
{
   private:
   int x;

   public:
   int getX();
   void setX();
   …
};
```

zahl.cpp

```
#include "zahl.h"

int Zahl::getX()
{
   return x;
}
…
```

- Die zugehörige Klasse wird durch **::** zugeordnet

2. Instanziierung von Klassen

- Für die **Instanziierung** gibt es zwei Möglichkeiten:

Mit dem **new**-Operator:

```
Zahl *z = new Zahl();
```

***z** ist ein Pointer auf das Ball-Objekt

Durch **Variablendeklaration**:

```
Zahl z;
```

(Impliziter Konstruktoraufruf)

z ist ein Bezeichner für das Ball-Objekt!

- Vergleich: In Java verwendet man immer Objektreferenzen
- **Achtung:** Mit **new** erzeugte Objekte müssen mit **`delete`** gelöscht werden (kein Garbage-Collector wie in Java)
- **Achtung:** Durch Variablendeklaration erzeugte Objekte bestehen nur in ihrem Gültigkeitsbereich (kein **`delete`** nötig!)

3. Zugriff auf Methoden und Attribute

- Methoden- und Attributenzugriff über **->** oder **.**

```
Zahl *z = new Zahl();


z->getX();

(*z).getX();
```

```
Zahl z;


z.getX();

(&z).getX();
```

- Wie in Java bezeichnet **this** einen Pointer auf die eigene Klasseninstanz
- ***this** ist somit ein Bezeichner für die eigene Klasse

4. Konstruktoraufruf mit Parametern

- Bei Instanziierung mit **new** erfolgt der **Konstruktoraufruf** wie bei Java:

```
class Zahl
{
   ...
   Zahl(int);
};
```

```
Zahl *z = new Zahl(17);
```

- Bei Variablendeklaration lautet die Syntax:

```
Zahl z = Zahl(17);
```

- Falls der Konstruktor genau einen Parameter benötigt, dann ist eine Kurzschreibweise möglich:

```
Zahl z = 17;
```

4. Konstruktor-Initialisierungsliste

- Attribute dürfen **nicht** außerhalb des Konstruktors initialisiert werden (im Gegensatz zu Java!)
- Die Initialisierung ist im **Konstruktor** durchzuführen, vorzugsweise in einer **Initialisierungsliste**:

```
class Punkt
{
   int x;
   int y;

   public:
   Punkt();
};
```

```
Punkt::Punkt() : x(17), y(0)
{
}
```

- Vorteil: Die Initialisierung geschieht **vor dem Konstruktoraufruf**, also auch vor dem Aufruf von Basisklassenkonstruktoren!

4. Default-Konstruktor

- Ein **default-Konstruktor** ist ein Konstruktor ohne Parameter

- Falls kein Konstruktor definiert wurde, dann wird ein default-Konstruktor automatisch generiert (vom Compiler)
- Falls Konstruktoren definiert wurden, jedoch kein default-Konstruktor, dann ist eine Instanziierung ohne Parameter ungültig:

```
class Zahl
{
   Zahl(int y)
   {
   }
};
```

```
Zahl z;
```

5. Destruktor

- Der **Destruktor** wird aufgerufen, wenn das Objekt zerstört wird (z.B. durch **`delete`**)

```
Zahl::~Zahl()
{
   // Deinitialisierung
}
```

Nur ein Konstruktor!
Kein Parameter!

- Im Destruktor sollte man dynamisch reservierten Speicherplatz freigeben

Ein typischer Destruktor:

```
Ball::~Ball()
{
   delete object_1;
   delete [] array_1;
}
```

6. Statische Attribute und Methoden

- Wie in Java können **statische** Attribute und Methoden definiert werden
- Statische Attribute existieren nur einmal für alle Klasseninstanzen

- Beispiel:

Definition:

```
class Ball
{
   …
   public:
   static int n;
   …
}
```

Zugriff:

```
Ball::n = 17;

// … oder
Ball b;
b::n = 17;
```

7. Friend-Methoden und friend-Klassen

- Als **`friend`** deklarierte Methoden haben uneingeschränkten Zugriff auch auf **`private`**-Methoden und -Attribute

```
class A
{
    int p;
    friend void B::setP();
    …
}
```

```
void B::setP()
{
    A a;
    a.p = 17;
}
```

- **`friend`**-Klassen haben in allen Methoden vollen Zugriff:

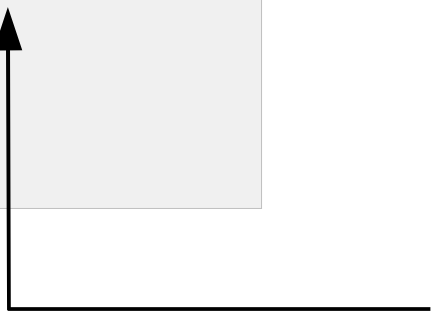
```
class A
{
    int p;
    friend B;
    …
}
```

```
void B::setP()
{
    A a;
    a.p = 17;
}
```

8. Vererbung

- Klasse **B erbt** von Klasse **A**:

```
class B : public A
{
   ...
}
```



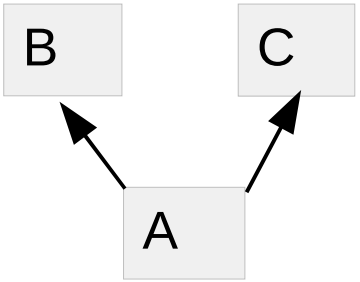
Zugriffsschutz kann für alle Funktionen von **A** noch weiter eingeschränkt werden

- Bei der Vererbung werden alle Methoden und Attribute vererbt, mit Ausnahme von Konstruktoren und **friend**-Methoden (und **operator**-Methoden)

8. Mehrfachvererbung

- Im Gegensatz zu Java ist **Mehrfachvererbung** möglich

- Beispiel:



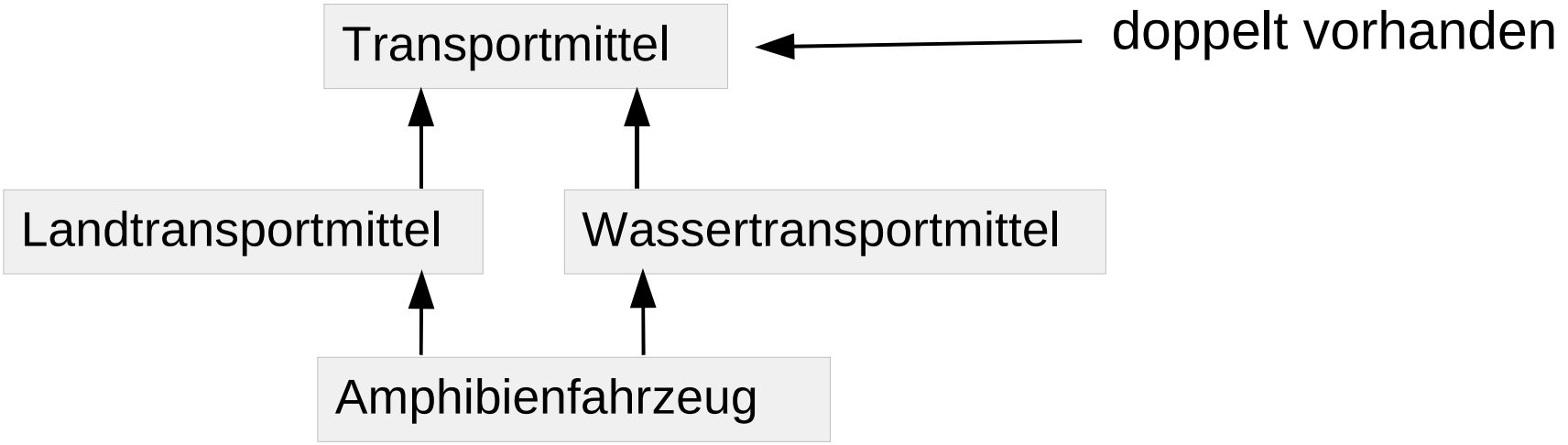

```
class A : public B, private C
{
   ...
}
```

- Die Klasse **`A`** erbt von **`B`** und von **`C`**, wobei alle geerbten Methoden von **`C`** private sind

8. Mehrfachvererbung

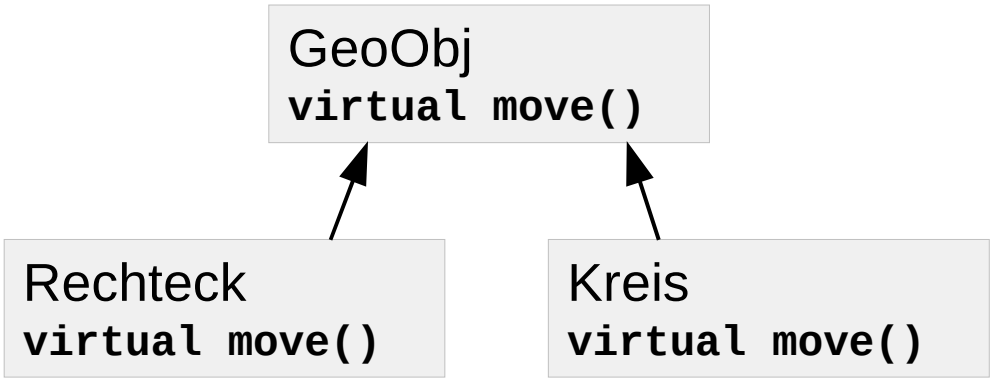
- **Achtung**: Mehrfachvererbung kann problematisch sein, z.B.

- Wenn eine **gemeinsame Basisklasse** existiert:



- Wenn zwei Basisklassen **gleichnamige** Methoden oder Attribute besitzen

- **Dynamische Methodenbindung** ist nur bei Methoden möglich, die explizit als **`virtual`** deklariert wurden (im Gegensatz zu Java)




```
GeoObj *r = new Rechteck();

r->move();
r->GeoObj::move();
```

Rechteck::move()

GeoObj::move()

10. Abstrakte Klassen

- Nur **`virtual`**-Methoden können in abgeleiteten Klassen überschrieben werden

- Abstrakte Methoden (vgl. **`abstract`** bzw. **`interface`** in Java) müssen in abgeleiteten Klassen implementiert werden und werden in der Basisklasse **nicht** implementiert

```
virtual void move() = 0;
```

11. Literatur

- Bjarne **Stroustrup**: The C++ programming language (empfehlenswert)
- Harvey M. Deitel, P. J. Deitel: C++ how to program (sehr empfehlenswert)
- Im Web:
  - http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html
  - http://www.cplusplus.com