Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. K. D. Müller-Glaser
Prof. Dr.-Ing. J. Becker
Prof. Dr. rer. nat. W. Stork

Vorlesung: Informationstechnik (IT)

Prof. Dr.-Ing. K.D. Müller-Glaser

Institut für Technik der Informationsverarbeitung (ITIV)

Objektorientierung

Überblick

Karlsruher Institut für Technologie

- Objektorientierung: Grundidee und Motivation
- Klassen
- Objekte
- Datenkapselung
- Vererbung
- Polymorphie

Grundidee

- Menschliche Auffassung betrachtet Realität meist als Verbund von Objekten
  - Menschen, Unternehmen, Fahrzeuge, Formulare, Geschäftsabläufe…
- Programmierparadigma Objektorientierung bildet diese Objekte in Software nach

Objektorientierung ist ein Programmierparadigma.
Dabei geht es darum, dass zumeist reelle Objekte oder Konzepte im Computer abgebildet werden sollen.
Solche bestehen aber meist nicht nur in Eigenschaften, die sich als Daten auffassen lassen, sondern haben auch Fähigkeiten, etwa ihre Eigenschaften zu ändern oder mit anderen Objekten zu interagieren.
Während es bei der imperativen und prozeduralen Programmierung darum geht, Abläufe, die auf Daten angewandt werden, zu definieren, bietet die OOP „Objekte“, die eine Einheit aus Daten und Funktionen darstellen. So wird die Zuordnung von Daten einfacher.

Objekt

- Definition Objekt:
  Eine zur Laufzeit des Programms existierende Datenstruktur, die aus 0 oder mehr Werten besteht, welche Attribute genannt werden. Sie dient als computerinterne Darstellung eines abstrakten Objekts.

- Objekt ist Exemplar einer Klasse
  (Klasse ist „Fabrik“ für Objekte)

- Objektzustand – Werte der Daten,
  aktuelle Verknüpfungen mit anderen Objekten

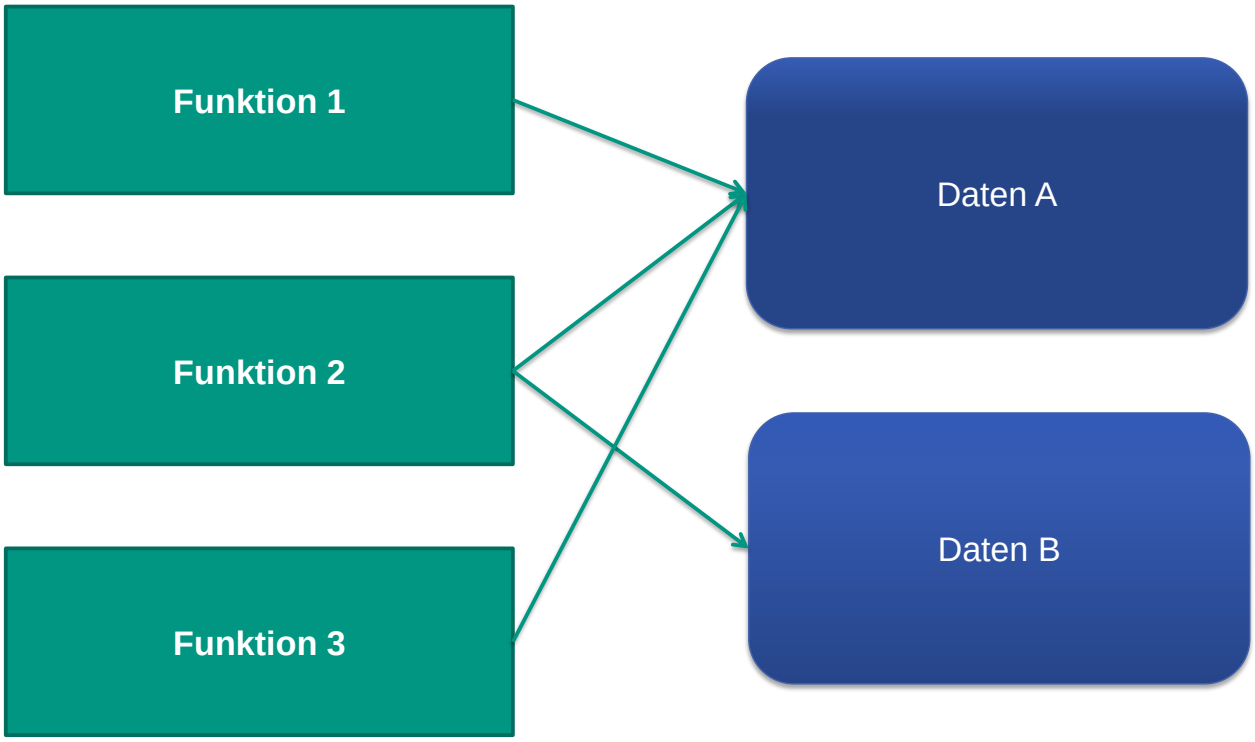
Ein Objekt ist einfach ein zusammenhängender Block von Daten. Das kann einfach eine Ganzzahl sein, es kann sich aber auch um eine Instanz einer äußerst komplexen Klasse sein, die vielleicht hunderte Attribute hat.

Prinzipien

- Daten und die zu diesen gehörenden Funktionen werden eng in einem sog. „Objekt“ verbunden
- Kapselung: Objekt definiert Zugriffsmöglichkeit auf seine Daten und schützt diese so
- Wiederverwendbarkeit: Nur einmal definieren, wie ein Objekt funktioniert, und dieses immer wieder, auch für neue Typen von Objekten, verwenden

Die Kapselung befähigt das Objekt, über seine Eigenschaften selbst zu wachen: Ein Objekt legt selbst fest, wer auf seine Eigenschaften zugreifen darf.
Objekte in Objektorientierten Sprachen haben einen Typ, der festlegt, welche Eigenschaften und Fähigkeiten sie besitzen. Man braucht den Typ nur einmal zu definieren und kann dann beliebig viele Objekte mit den entsprechenden Fähigkeiten generieren.
Neue Typen von Objekten können auch auf vorhandenen aufbauen, so dass Redundanz für den Programmierer vermieden wird.

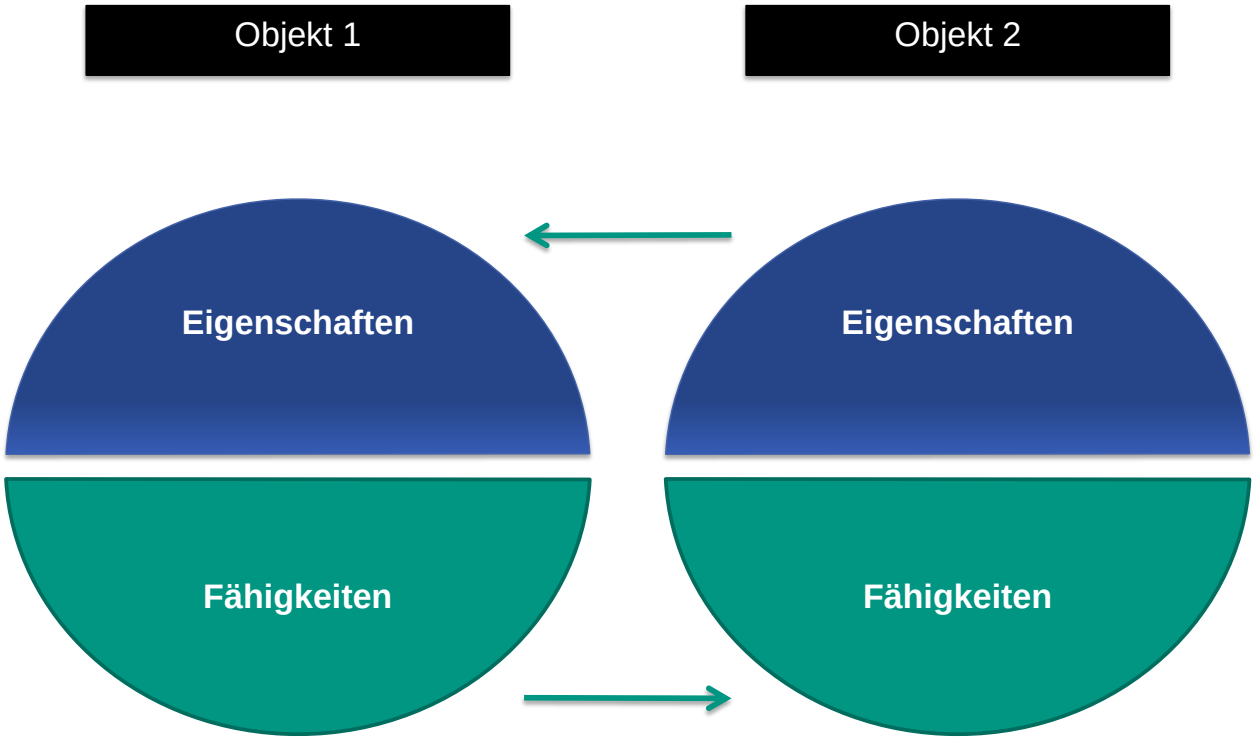
Strukturierte, Prozedurale Programmierung



Vor der Objektorientierung gab es für die meiste Software nur das klassische, prozedurale Konzept zur Programmierung:

- Es trennt Daten und Funktionen, die diese Daten bearbeiten.
- Der Programmierer muss geeignete Anfangswerte vor Verwendung und Korrektheit der Daten beim Funktionsaufruf sicherstellen.

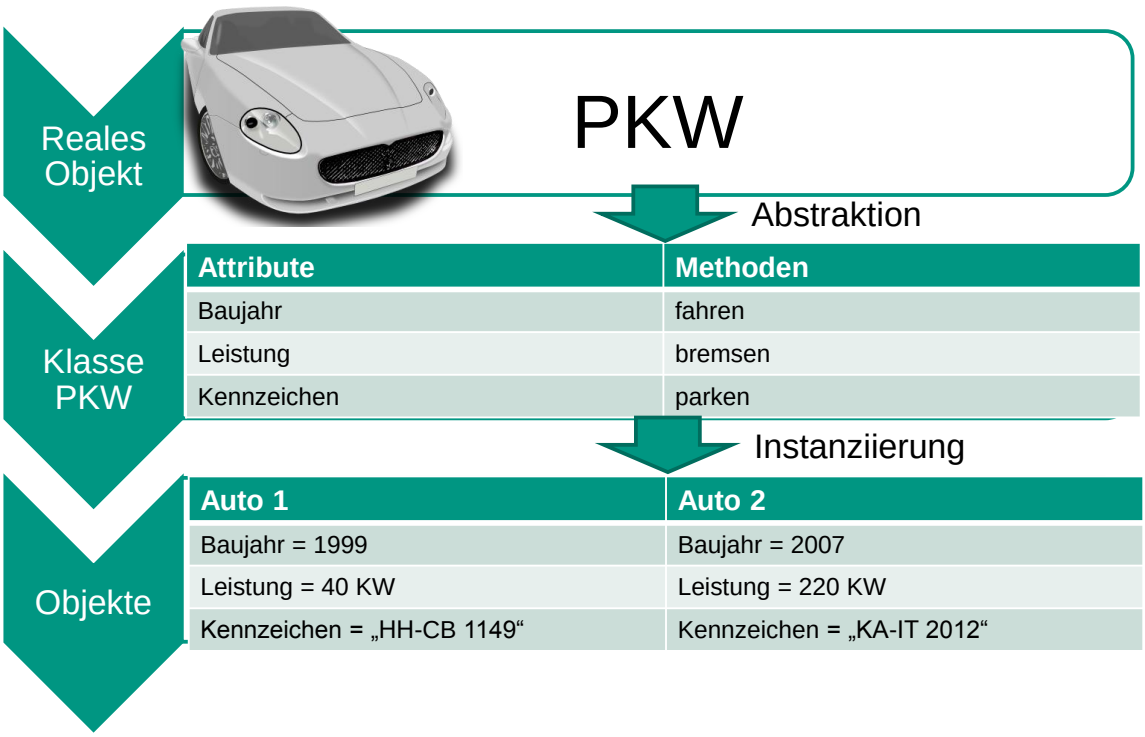
Objektorientierte Programmierung



Objektorientiertes Konzept:
Objektorientierte Programmierung
Einheit aus Daten und Funktionen

- geringere Fehleranfälligkeit
- gute Wiederverwendbarkeit
- geringer Wartungsaufwand

Abstraktion



Der Programmierer muss bei der Umsetzung eines reellen Objekts in Software zunächst einmal abstrahieren, welche Eigenschaften dieses Objekt ausmachen, und welche Fähigkeiten es besitzen muss, wenn es im Kontext der Problemstellung nützlich sein soll. Nachdem er diese Eigenschaften und Fähigkeiten notiert hat, kann er einzelne Instanzen seines Objekts ins Leben rufen:
In unserem Beispiel definiert der Programmierer, was ein Objekt vom Typ “PKW” beschreibt, und erstellt dann zwei verschiedene Objekte, die die gleichen Fähigkeiten besitzen, aber unterschiedliche Kenndaten haben.

Objekt

- Definition Objekt:
  Eine zur Laufzeit des Programms existierende Datenstruktur, die aus 0 oder mehr Werten besteht, welche Attribute genannt werden. Sie dient als computerinterne Darstellung eines abstrakten Objekts.

- Objekt ist Exemplar (Instanz) einer Klasse
  (Klasse ist „Fabrik“ „Prägestempel“ für Objekte)

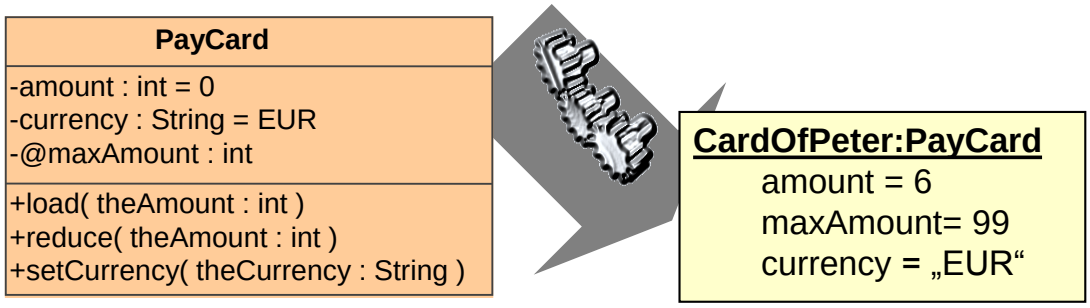
- Klassifizierung: Zuordnung eines Objektes zu einer Klasse

- Objektzustand – Werte der Daten

Ein Objekt ist einfach ein zusammenhängender Block von Daten. Das kann einfach eine Ganzzahl sein, es kann sich aber auch um eine Instanz einer äußerst komplexen Klasse sein, die vielleicht hunderte Attribute hat.

Klasse

- Bauplan für Objekte des Typs (Instanziierung, Prägestempel)
- Selbstdefinierter Datentyp, bestehend aus
  - Eigenschaften: „Attribute“
  - Fähigkeiten: „Methoden“
- Prinzip: schreibe Code nur einmal, vermeide Replikation



Klassen sind die wesentlichen Sprachelemente in C++ zur Unterstützung der Objektorientierten Programmierung (OOP).

Eine Klasse legt die Eigenschaften und Fähigkeiten von Objekten fest.

In C++ ist eine Klasse ein selbstdefinierter Datentyp, bestehend aus Datenelementen, die die Eigenschaften beschreiben, und Elementfunktionen (=Methoden), die die Fähigkeiten der Objekte darstellen.

Datenabstraktion: notwendig, um komplexe Sachverhalte in einem gegebenen Kontext darzustellen. Dinge und Vorgänge werden auf das Wesentliche reduziert. Klassen ermöglichen es, die Ergebnisse der Abstraktion direkter bei der Software-Entwicklung umzusetzen.

Datenkapselung: Bei der Definition einer Klasse wird festgelegt, welche Elemente der Klasse vor einem Zugriff von außen geschützt werden sollen (private) und welche Elemente öffentlich verfügbar sein sollen (public).

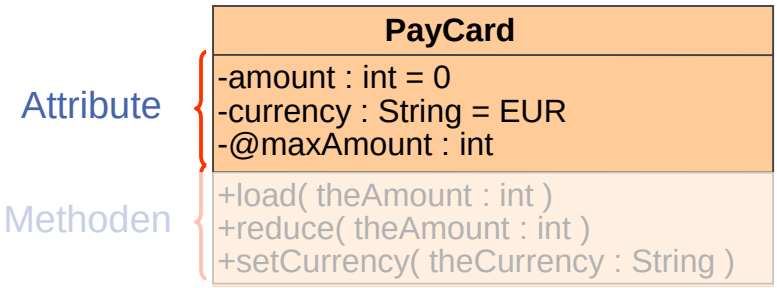
Ein Objekt kapselt seine Daten und Methoden von der Außenwelt ab und verwaltet sich mit Hilfe seiner Methoden selbst.

Ergebnis der **Analyse** der Problemstellung: Identifizierung und Beschreibung von Objekten und der Beziehungen untereinander. Das Ergebnis der Beschreibung der Objekte sind Klassen.

Instanzen einer Klasse unterscheiden sich in ihrem Zustand, das heißt in unserem Beispiel, dass es verschiedene PayCards gibt, die unterschiedliche amount haben können. Diese Attribute sind häufig einfache Zahlen oder Zeichenketten, es kann sich bei ihnen aber auch um Objekte anderer Klassen handeln, oder sie können einen Verweis auf ein beliebiges Objekt darstellen.

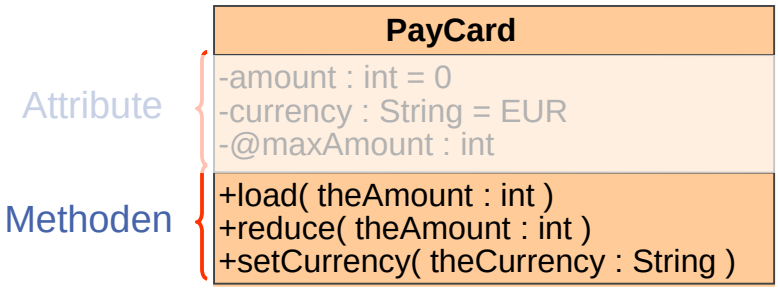
In der Objektorientierung ist es notwendig, dass eine Klasse Eigenschaften von Attributen definiert: Was kann in einem Attribute gespeichert werden (Typ)? Wer darf das Attribut sehen? Darf man es verändern?

Attribute



- Attribute definieren die Daten
- Gesamtheit der Attribute definiert den Zustand des Objekts
- Attribute haben Eigenschaften, z.B.
  - Sichtbarkeit
  - Typ
  - Wertebereich
  - Unveränderlichkeit (konstant während Objekt-Lebensdauer)

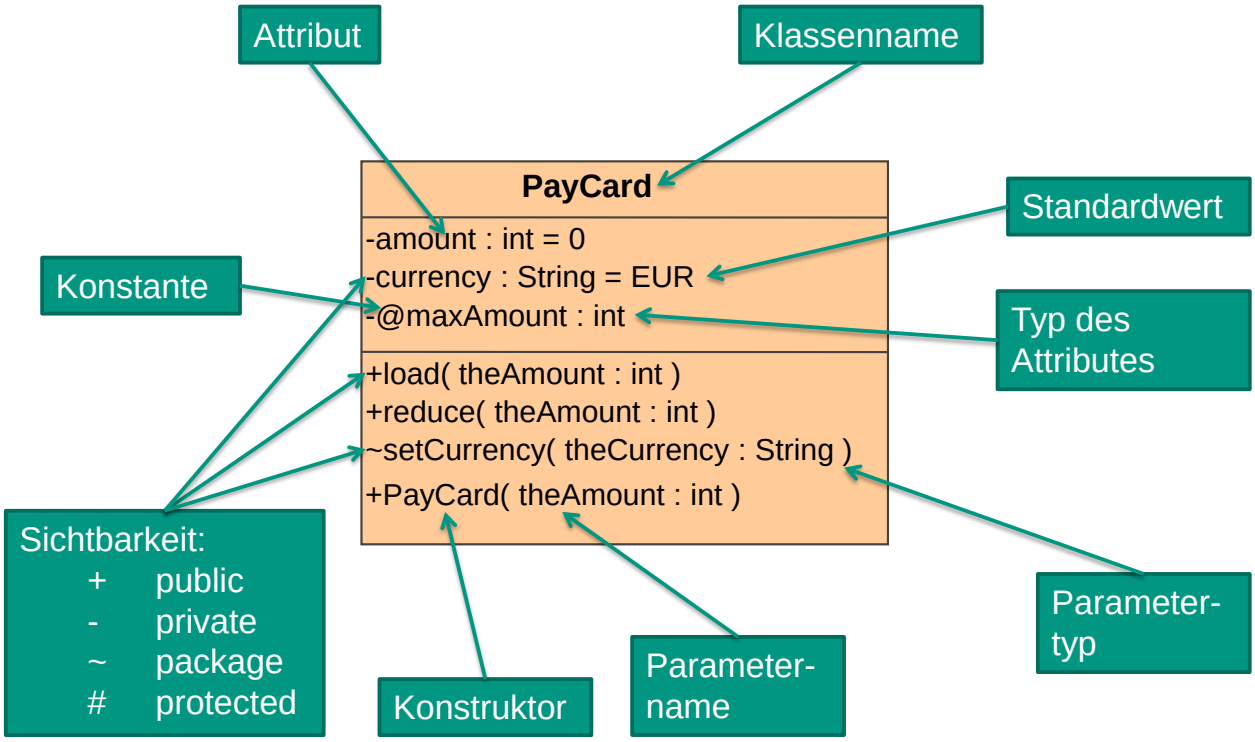
Methoden



- Funktionen (Operationen), die fest der Klasse zugeordnet sind
- Methoden verändern den Zustand / die Daten eines Objekts
- Methoden haben Eigenschaften
  - Sichtbarkeit
  - Typ des Rückgabewertes
  - Parameter

Aus der funktionalen Programmierung kennen Sie bereits Funktionen. Methoden sind zur Klasse zugehörige Funktionen. Sie wirken auf das Objekt, zu dem sie gehören: Das heißt, wenn man die Methode reduce einer Instanz der Klasse PayCard aufruft, wird die Methode auf diese Instanz angewandt (und nicht auf eine andere, und nicht auf alle Instanzen der Klasse PayCard).
Ein wesentliches Grundprinzip der OOP ist es, dass alle Instanzen einer Klasse die gleichen Fähigkeiten aufweisen. Daher ist eine Methode zwar für alle Instanzen der Klasse derselbe Programmcode, dieser wird aber aufgrund unterschiedlicher Werte der Attribute auf andere Variablen angewandt.

Klassendiagramm (Unified Modeling Language UML)



Bei der Darstellung von Klassen werden Klassendiagramme verwandt. Wir haben dies schon auf den vorhergehenden Folien getan, ohne konkret auf diese Notation eingegangen zu sein. Im Kopffeld des Diagramms steht immer der Klassenname (PayCard). Darunter folgen die Attribute zeilenweise.
Dabei steht pro Zeile zunächst ein Zeichen, das die Sichtbarkeit des jeweiligen Attributes angibt. Auf die einzelnen Bedeutungen dieser Sichtbarkeiten soll später eingegangen werden. Danach folgen weitere spezielle Eigenschaften (hier nur ein @ für eine Konstante), gefolgt vom Attributnamen (amount, currency oder maxAmount). Nach einem Doppelpunkt folgt der Typ der Variable. Der Entwickler hat die Möglichkeit, einen Standardwert für diese Attribute vorzugeben.
In der zweiten Hälfte folgen die Methoden. Hier gelten dieselben Symbole für Sichtbarkeit. In Klammern folgt auf den Variablennamen eine Liste von Parametern (separiert durch Kommata), die man der Methode beim Aufruf zu übergeben hat. Gibt die Methode einen Wert zurück, dann folgt am Ende der Zeile, analog zu den Attributen, der Typ des Rückgabewerts.

Zwischenübung01: Klassendiagramm

Erstellen Sie ein Klassendiagramm zur anfangs vorgestellten PKW-Klasse

Attribute	Methoden
Baujahr	fahren
Leistung	bremsen
Kennzeichen	parken

Definition von Klassen in C++

```
// konto.h
// Definition der Klasse Konto
#include <iostream>
#include <string>
using namespace std;

class Konto {
  private:                  //geschützte Elemente
    string name;            //Kontoinhaber
    unsigned long nr;       //Kontonummer
    double stand;           //Kontostand
  public:                   //öffentliche Schnittstelle:
    bool init( const string&, unsigned long, double );
    void display();
};
```


Sie sehen hier, wie eine Klasse in C++ definiert wird.
Die Klasse hat den Namen „Konto“.
Sie besitzt Attribute der Sichtbarkeit private (nur aus dem jeweiligen Objekt selbst heraus sichtbar), namentlich einen String name, eine vorzeichenfreie lange Ganzzahl nr und eine Gleitkommazahl doppelter Genauigkeit stand.
Desweiteren besitzt sie Methoden mit der Sichtbarkeit public (von überall aus sichtbar und aufrufbar): init, welche einen bool (Wahrheitswert) zurückliefert und drei Parameter akzeptiert; desweiteren display, das keinen Wert zurückgibt und auch keine Parameter erfordert.

Definition von Methoden in C++

```
 // konto.cpp: Definition der Methoden init() und display().
#include "konto.h"                  //Definition der Klasse
//.....
bool Konto::init( const string& i_name,
                  unsigned long i_nr,
                  double i_stand ) {
  if( i_name.size() < 1 ) {         //kein leerer Name
     return false;
  }
  name   = i_name;
  nr     = i_nr
  stand  = i_stand;
  return true;
}
//Methode display() gibt die privaten Daten aus
void Konto::display() {
   cout  << fixed << setprecision( 2 )
         << "---------------------------------" << endl
         << "Kontoinhaber:   " << name  << endl
         << "Kontonummer:    " << nr    << endl
         << "Kontostand:     " << stand << endl
         << "---------------------------------" << endl;
}
```


Nachdem man in der Datei „konto.h“ die Klasse wie auf der letzten Folie definiert hat, kann man jetzt die festgelegten Methoden mit Programmcode definieren: bool Konto::init (….) ist der Funktionskopf, der in Name, Rückgabewert und Parametern mit der Deklaration der Methode in der Klassendefinition (vorherige Folie) übereinstimmen muss.

Objekt

- Definition Objekt:
  Eine zur Laufzeit des Programms existierende Datenstruktur, die aus 0 oder mehr Werten besteht, welche Attribute genannt werden. Sie dient als computerinterne Darstellung eines abstrakten Objekts.

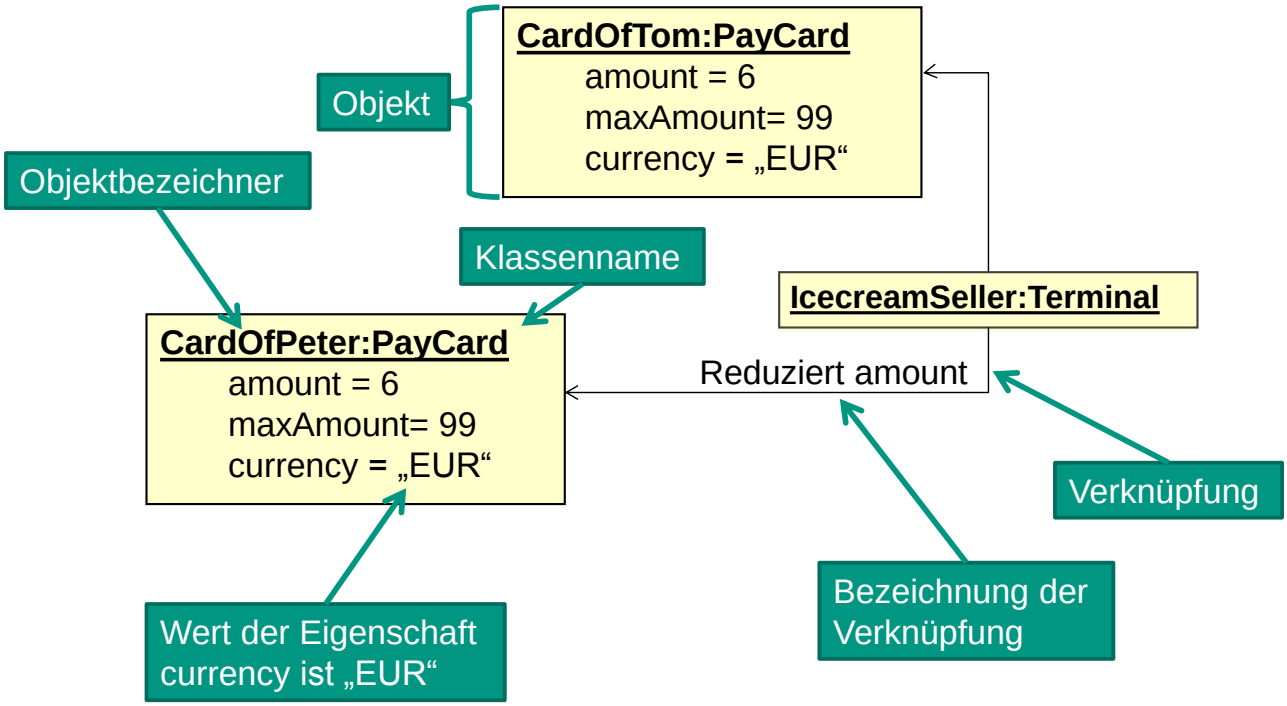
- Objekt ist Exemplar (Instanz) einer Klasse
  (Klasse ist „Fabrik“ „Prägestempel“ für Objekte)

- Klassifizierung: Zuordnung eines Objektes zu einer Klasse

- Objektzustand – Werte der Daten

Ein Objekt ist einfach ein zusammenhängender Block von Daten. Das kann einfach eine Ganzzahl sein, es kann sich aber auch um eine Instanz einer äußerst komplexen Klasse sein, die vielleicht hunderte Attribute hat.

Objektdiagramm: Details



Dargestellt ist ein Objektdiagramm. Wie man sieht, sind darin zeilenweise die einzelnen Attribute mit den jeweiligen Werten abgebildet.
Merke: Im Gegensatz zum Klassendiagramm handelt es sich bei diesen Blöcken nicht um die Beschreibung eines Typs, sondern um konkrete Instanzen dieses Typs mit konkreten Werten.

Objekte im Speicher

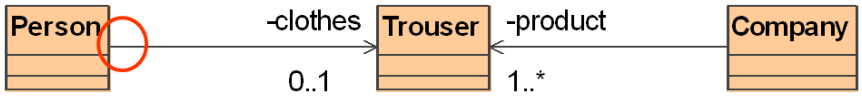
sparbuch	
name	„Lustich, Peter“
nr	42214316
stand	-2009.20

girokonto	
name	„Adams, Eva“
nr	02013650
stand	1300.65

Objekte werden zumeist einfach als Aneinanderreihung ihrer Attribute im Speicher abgelegt. Da bei Klassen bekannt ist, in welcher Reihenfolge welche Attribute vorhanden sind, liegen diese ohne spezielle Markierung hintereinander im Speicher.

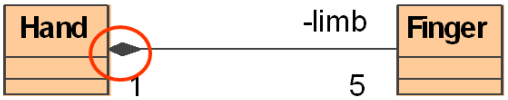
Assoziationen

Modellieren Beziehungen zwischen Objekten auf Klassenebene

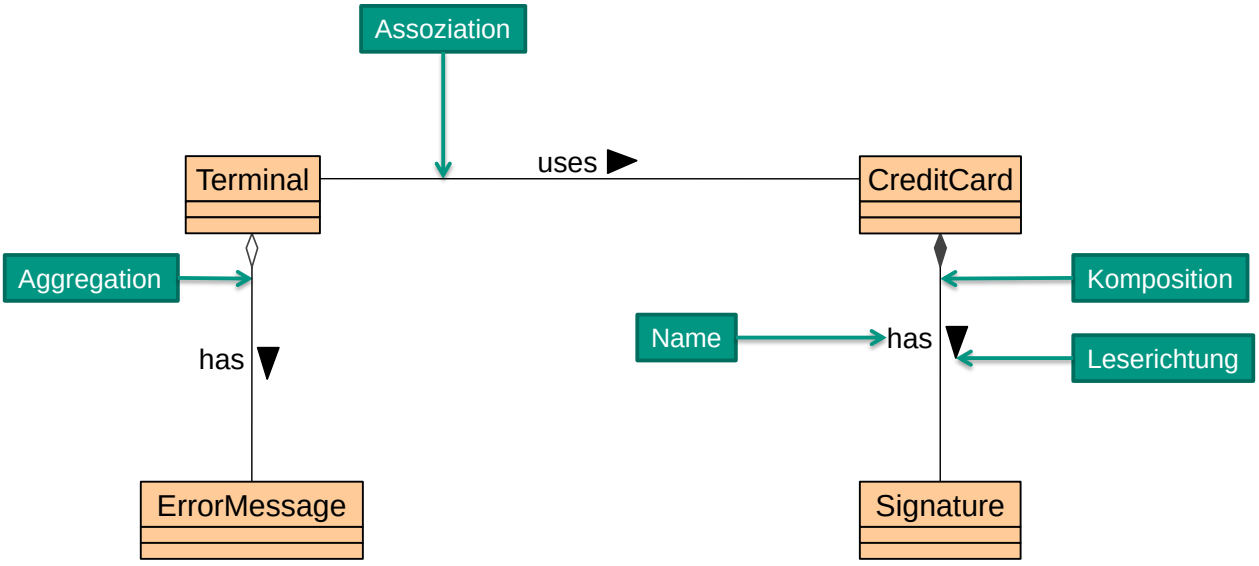


- Aggregation
  - Modelliert "Teile und Ganzes"-Beziehung

- Komposition
  - Strikter als Aggregation
  - „Teile und Ganzes –Beziehung" bilden eine Einheit mit gemeinsamer Lebensdauer
  - Teile können nur Teil eines einzigen „Ganzen" sein.



Beispiel: Kommunikation

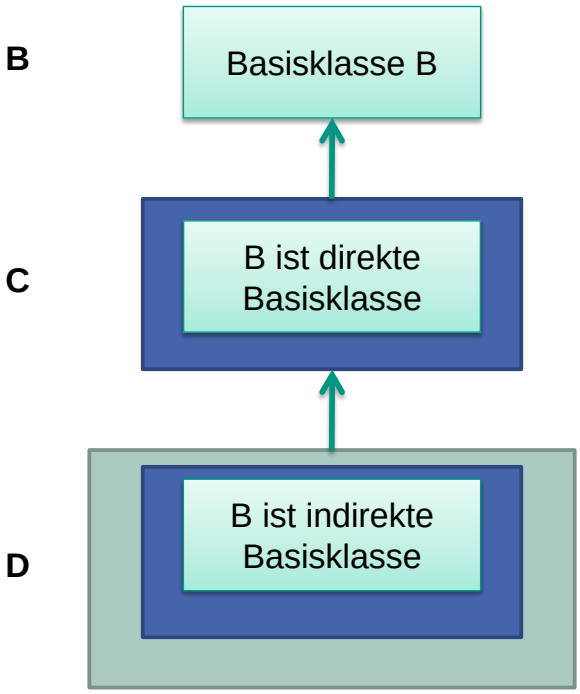


Multiplizität

- Gibt an, wie oft eine Entität in einem gegebenen Kontext vorkommt

Symbol	Bedeutung
1	Exakt ein mal
0..1	Kein oder ein mal
* (or 0..*)	Kein mal oder häufiger
1..*	Ein mal oder häufiger
3..*	Drei mal oder häufiger
4..6	Vier bis sechs mal
2, 4, 6	Zwei, vier oder sechs mal
1..5, 8, 10..*	Ein bis fünf mal, acht mal oder zehn mal oder häufiger

Vererbung



Ein wesentliches Ziel der Objektorientierung ist die Wiederverwendbarkeit von Code. Aus diesem Grund wurde die Vererbung eingeführt:

Hat man eine Basisklasse (andere Bezeichnungen Oberklasse, Superklasse, Elternklasse, Mutterklasse), die man um weitere Fähigkeiten oder Eigenschaften erweitern möchte, wäre es entgegen dem Sinn der Objektorientierten Programmierung, den Code der alten Klasse zu kopieren und entsprechend zu erweitern.

Stattdessen definiert man eine Tochterklasse (Unterklasse, Subklasse, Child Class, abgeleitete Klasse), die automatisch alle Fähigkeiten und Eigenschaften der Mutterklasse erhält / erbt. Dadurch ist auch der Zusammenhang zwischen diesen beiden Klassen klar, so dass man dort, wo ein Objekt der Basisklasse gefordert ist, auch ein Objekt der abgeleiteten Klasse benutzen kann, da diese nur einen Spezialfall der Basisklasse darstellt.

Vererbung ist auch mehrstufig möglich, in manchen Programmiersprachen (z.B. C++) ist sogar das Erben von mehreren Basisklassen gleichzeitig möglich.

Möchte der Programmierer einer Tochterklasse nicht nur als Erweiterung der Basisklasse sehen, sondern auch das Verhalten schon in der Basisklasse vorhandener Methoden ändern, so ist dies auch möglich, indem er die entsprechenden Methoden in der Tochterklasse mit eigenen Implementierungen überschreibt.

Definition der abgeleiteten Klasse

- Beispiel von vorheriger Folie:

```
class C : public B {
  private:
    /* Deklaration der zusätzlichen
       privaten Datenelemente und
       Elementfunktionen */
  protected:
    /* Deklaration der zusätzlichen
       geschützten Datenelemente und
       Elementfunktionen */
  public:
    /* Deklaration der zusätzlichen
       öffentlichen Datenelemente und
       Elementfunktionen */
};
```


Hier wird, wie auf der vorigen Folie angedeutet, eine Klasse C definiert, welche von B erbt.

Mit dem Schlüsselwort „public“ vor dem Namen B werden die Zugriffsrechte festgelegt: die öffentliche Schnittstelle von B als Basisklasse wird in die abgeleitet Klasse C übernommen.

Objekte der abgeleiteten Klasse C können auch die public-Methoden der Basisklasse aufrufen.

Die privaten Elemente der Basisklasse bleiben in jedem Fall geschützt. Eine Methode der abgeleiteten Klasse kann nicht auf die „private“_Elemente der Basisklasse zugreifen.

Die öffentliche Schnittstelle der abgeleiteten Klasse C besteht aus
den „public“-Elementen der Basisklasse und
den in der abgeleiteten Klasse zusätzlich definierten „public“-Elementen.

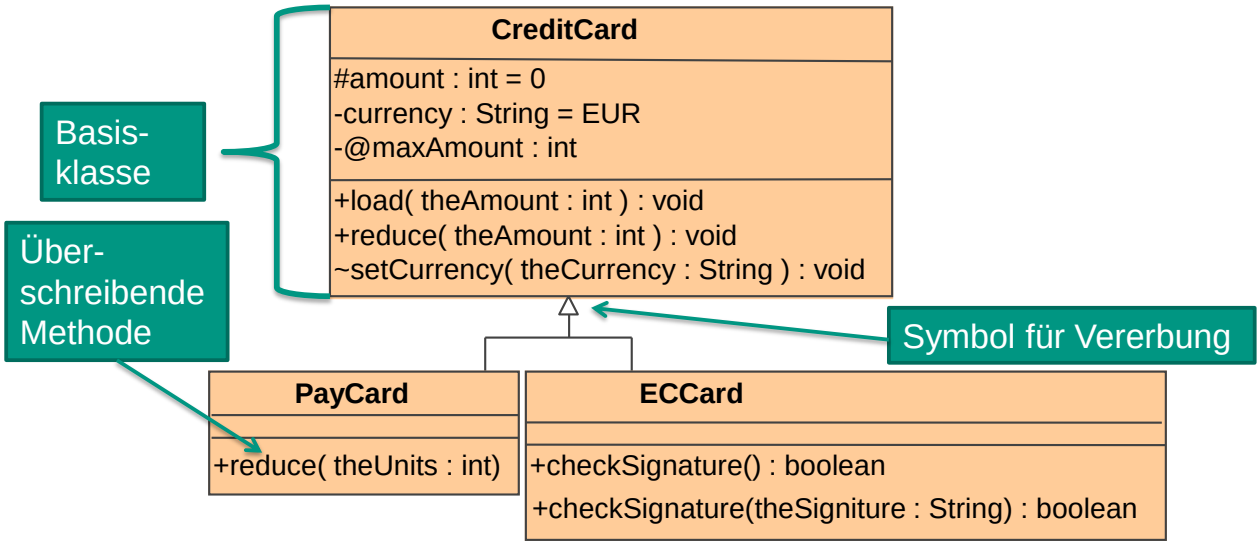
Bei der public-Vererbung werden die public- und die protected-Schnittstelle der Basisklasse als solche in die abgeleitete Klasse übernommen.
Ein Objekt der abgeleiteten Klasse ist ein spezielles Objekt der Basisklasse.

Vererbung mir „protected“ und „private“ , z.B. class C : protected B { ……. }

Bei der protected-Vererbung werden die public- und die protected-Elemente der Basisklasse zu protected Elementen in der abgeleiteten Klasse.

Bei der private-Vererbung werden die public- und die protected-Elemente der Basisklasse zu privaten

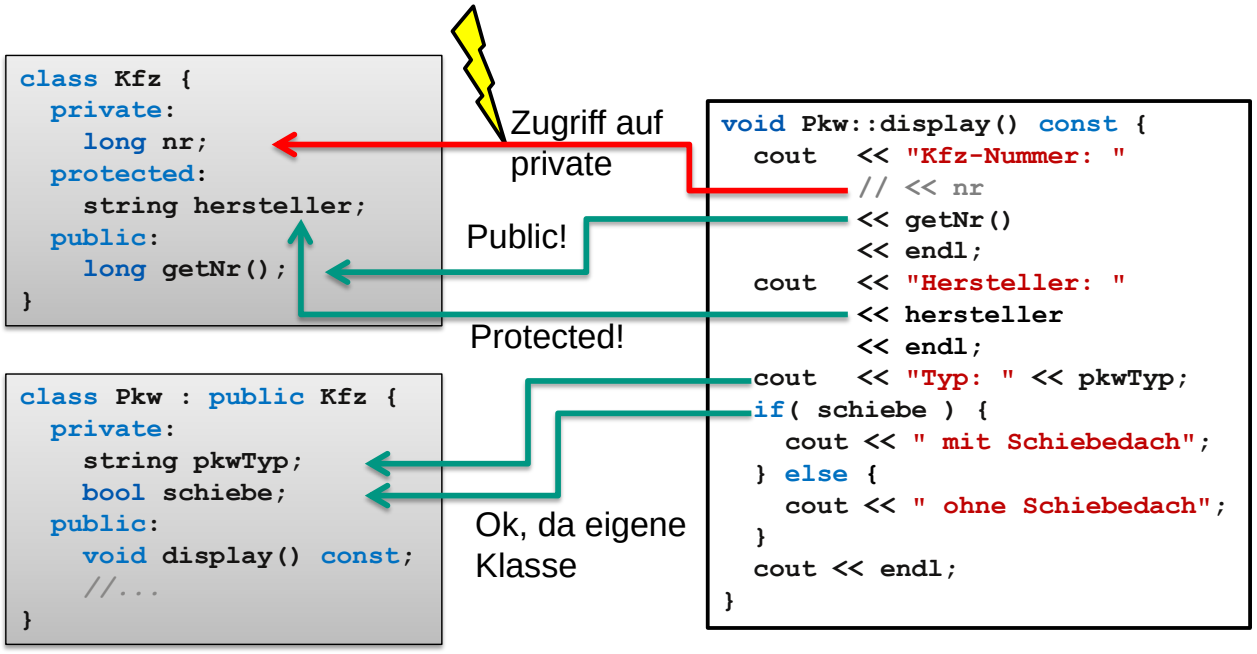
Vererbung - Klassendiagramm



- PayCard und ECCard spezialisieren die Basisklasse CreditCard
  - PayCard überschreibt die Funktionalität für Methode reduce
  - ECCard prüft die Signatur in zwei Varianten

Elementen der abgeleiteten Klasse.

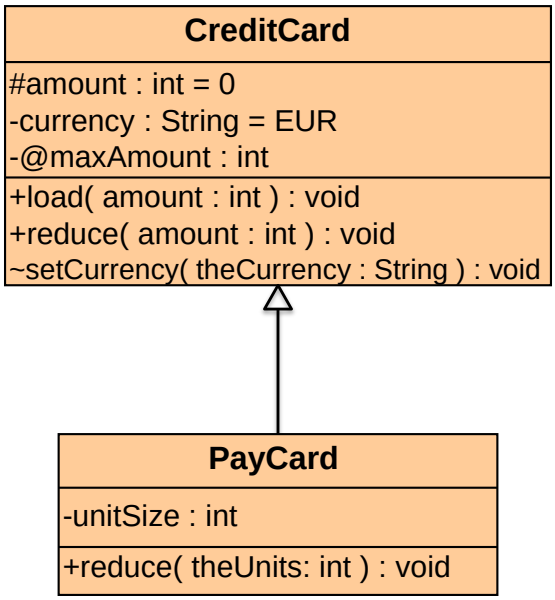
Zugriff – C++ Beispiel



Im Diagramm erkennt man, dass PayCard eine Tochterklasse (Spezialisierung) von CreditCard ist, bzw. CreditCard ist die Basisklasse (Generalisierung) von PayCard.
Dabei überschreibt es die Methode reduce der Basisklasse.
Auch ECCard erbt von CreditCard, allerdings überschreibt es nicht reduce, sondern ergänzt die Klasse um zwei weitere Methoden checkSignature.

Zwischenübung02: Vererbung

Gegeben sei die Klasse PayCard, welche von CreditCard erbt. Welche der folgenden Zugriffe sind erlaubt, welche sind verboten?

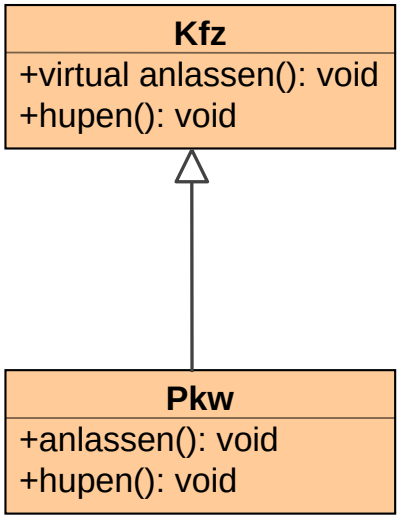



```
void PayCard::reduce( int theUnits) {
  amount -= theUnits * unitSize;
  cout   << "Neuer Kontostand: "
         << amount
         << " "
         << currency
         << endl;
}
```

Polymorphie

- Gleicher Name für Methoden mit gleichen Aktivitäten
- Es wird eine Methode bei einem Objekt aufgerufen, ohne die genaue Klasse dieses Objekts zu kennen
- Late binding: Entscheidung zur Laufzeit
- in C++ Schlüsselwort *virtual* notwendig

Polymorphie in C++




```
Kfz* auto = new Pkw();
   //ok, da Pkw Spezialisierung von Kfz

auto->anlassen();
   //Pkw::anlassen()
   //wie erwartet, da virtual

auto->hupen();
   //== Kfz::hupen()

((Pkw*)auto)->hupen();
   //umständlich,
   //aber notwendig ohne virtual, wenn
   //Pkw::hupen() aufgerufen werden soll
```

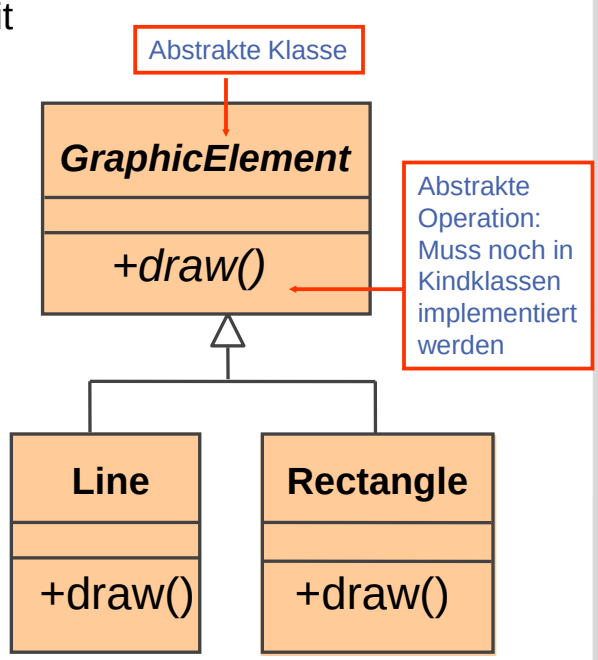

Polymorphie (von griechisch „Poly“ viel(fach) und „Morphos“ Gestalt) ist der Gedanke, dass abgeleitete Klassen ein und derselben Basisklasse verschiedene Aufgaben erfüllen sollen. Daher müssen diese Tochterklassen verschieden funktionierende Methoden besitzen. Da der Programmierer aber mit allen diesen Klassen gleich umgehen können möchte, sollten Methoden, die das Gleiche (auf verschiedene Art und Weise) tun, sich unter dem gleichen Namen aufrufen lassen.
Überschreibt nun eine Tochterklasse eine gleichnamige Methode der Basisklasse, dann stellt sich folgendes Problem: Der Compiler kann aus dem Quelltext ja noch nicht feststellen, welche Methode denn nun wirklich aufgerufen werden soll.
Diese Entscheidung muss zur Laufzeit getroffen werden, man spricht dann von „late binding“, weil der Funktionsaufruf nicht schon früher (beim Linken) an eine konkrete Funktionsadresse gebunden werden kann.
Damit der C++ Compiler „late binding“ vorsieht, muss ihm mitgeteilt werden, dass eine Funktion polymorph sein soll, sonst setzt er an die Stelle des symbolischen Aufrufs den konkreten Funktionsaufruf. Dies geschieht mittels Schlüsselwort virtual.

Polymorphie

- Gleicher Methodenname für ähnliche aber unterschiedliche Operationen
- Beim Aufruf der Methode muss der Typ des Objekts nicht bekannt sein
- Late binding: Entscheidung zur Laufzeit
- In C++ Schlüsselwort *virtual* benötigt

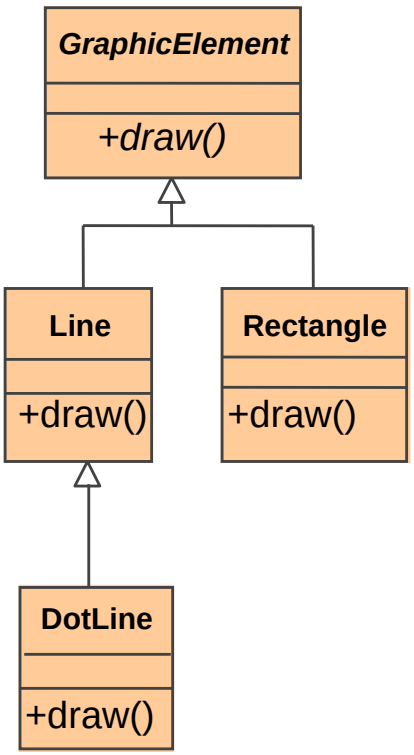
- Beispiel:

```
GraphicElement* g1 = new Line();
GraphicElement* g2 = new Rectangle();
GraphicElement* g3 = new Line();
g1->draw();
g2->draw();
g3->draw();
```



Beispiel: late binding

```
GraphicElement* g1 = new GraphicElement();
GraphicElement* g2 = new Line();
GraphicElement* g3 = new Rectangle();
GraphicElement* g4 = new DotLine();
Line* g5 = new Line();
Line* g6 = new DotLine();
g1->draw(); // Syntax error
g2->draw(); // ----------------
g3->draw(); // □
g4->draw(); // . . . . . .
g5->draw(); // ----------------
g6->draw(); // . . . . . .
```



Fazit

- OOP vereint Daten und Funktionen zu Klassen
- Klassen sind Baupläne („Fabriken“) für Instanzen
- Klassen können voneinander Eigenschaften und Fähigkeiten erben
- Mit Polymorphie lässt sich in einer Tochterklasse ein von der Basisklasse abweichendes Verhalten definieren