Karlsruher Institut für Technologie

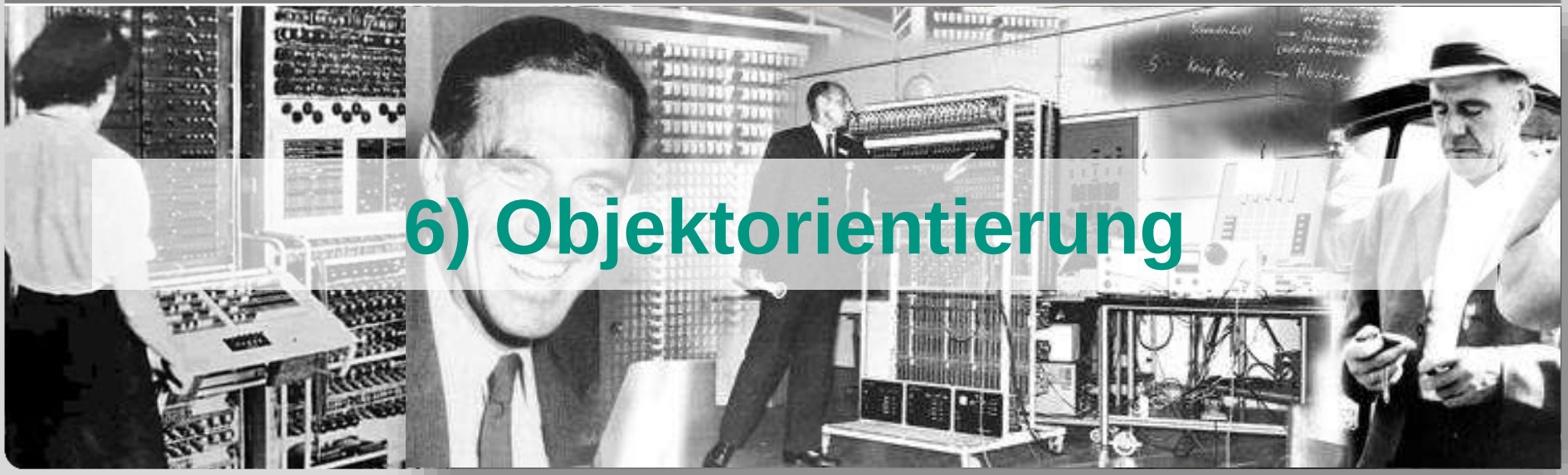
Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Vorlesung Informationstechnik (IT)

Sommersemester 2018

Institut für Technik der Informationsverarbeitung (ITIV)



6) Objektorientierung

IT Inhalt

6. Objektorientierung

- Grundidee und Motivation
  - Klassen
  - Objekte
  - Datenkapselung

7. Datenstrukturen

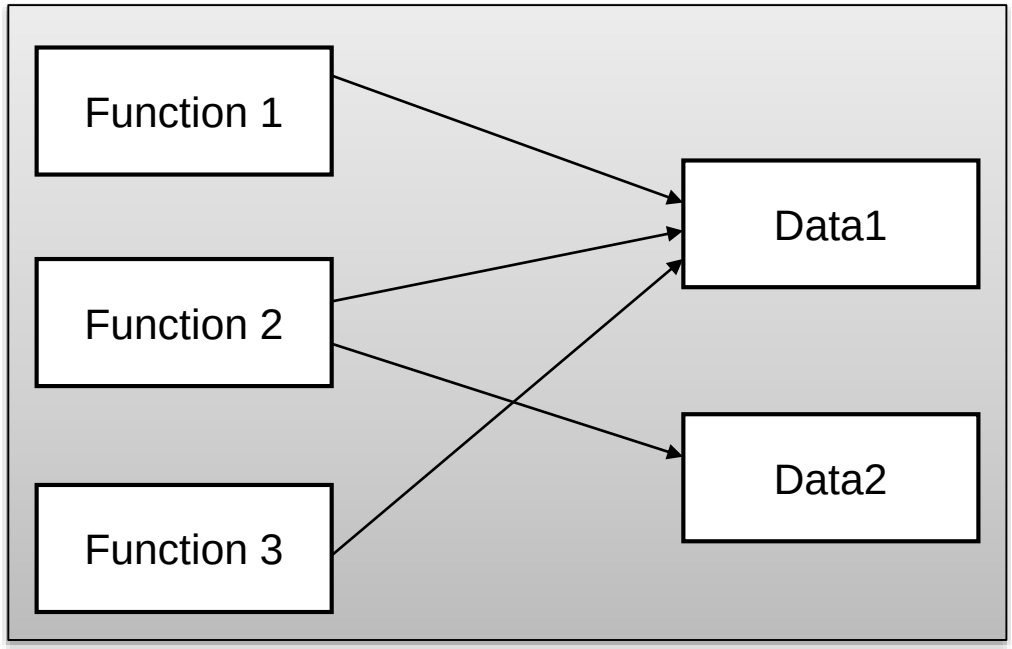
.

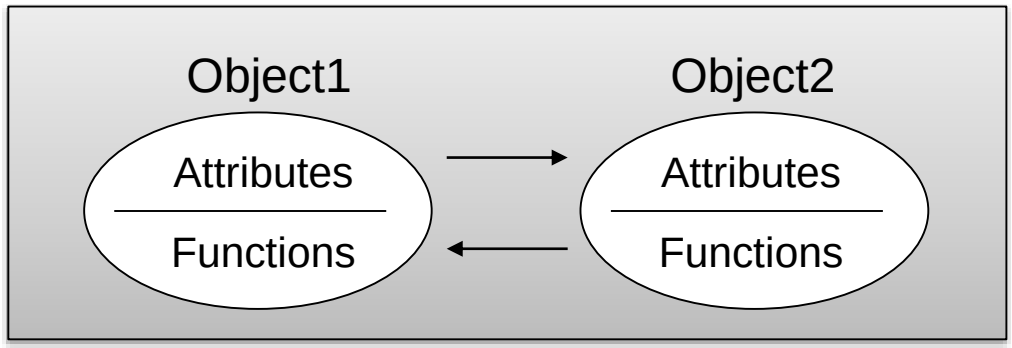
Programmierparadigmen

Paradigma	im Vordergrund stehen	Beispiel
Imperativ	**Variablen**, die einzelnen oder mehreren Speicherzellen des Rechners entsprechen, und **Anweisungen („Befehle")** als Abstraktionen der Prozessor-Funktionen	Assembler-Sprachen, FORTRAN, ALGOL
Prozedural	**Prozeduren** (Unterprogramme), an denen **Argumente (Parameter)** übergeben werden. Die Prozeduren berechnen **Werte** nach einem **Algorithmus** und geben diese zurück	Pascal, C
Funktional	**(Mathematische) Funktionen**, die einem **Vektor von Parametern** (darunter evtl. auch Funktionen) einen **Wert** zuordnen	LISP, LOGO HASKELL
Logikbasiert	**Logische Aussagen**, die im allgemeinen **freie Variablen** enthalten und von denen geprüft wird, ob und ggf. wie sie sich durch eine geeignete Bindung dieser Variablen **verifizieren** lassen	PROLOG
Objektorientiert	Autonome, interagierende **Objekte**, die durch **Botschaften** kommunizieren und mit anderen, ähnlichen Objekten in **Klassen** zusammengefasst sind	Smalltalk, C++, Java, C#

Quelle: [ApLu99]

Objekt-orientiertes Programmieren





- Traditionelles Konzept:
  - Imperatives, prozedurales Programmieren
  - Funktionen manipulieren die Daten
  - Daten und Funktionen sind separat
  - Lösungsorientierte Entwicklung
- Objekt-orientiertes Konzept:
  - Daten und Funktionen als eine Einheit
  - Problemorientierte Beschreibung
  - Evolutionäre Entwicklung

Grundidee

- Menschliche Auffassung betrachtet Realität meist als Verbund von Objekten
  - Menschen, Unternehmen, Fahrzeuge, Formulare, Geschäftsabläufe…

- Die Tanzmaus kann:
  - Tanzen
  - Singen
  - Springen
  - Freuen
  - Kreischen
  - Händeschütteln
  - …

Grundidee

- Menschliche Auffassung betrachtet Realität meist als Verbund von Objekten
  - Menschen, Unternehmen, Fahrzeuge, Formulare, Geschäftsabläufe…
- Programmierparadigma Objektorientierung bildet diese Objekte in Software nach.
- Definition Objekt:
  - Ein Objekt (auch Instanz genannt) bezeichnet in der objektorientierten Programmierung (OOP) ein Exemplar eines bestimmten Datentyps oder einer bestimmten Klasse (auch „Objekttyp“ genannt).
  - Objekte sind konkrete Ausprägungen („Instanzen“) eines Objekttyps und werden während der Laufzeit erzeugt (Instanziierung).
- Objekt ist Exemplar (Instanz) einer Klasse
  - Klasse ist „Fabrik“ für Objekte
- Objektzustand
  - Werte der Daten und aktuelle Verknüpfungen mit anderen Objekten

Kunst-OBJEKTE

„Objekte“

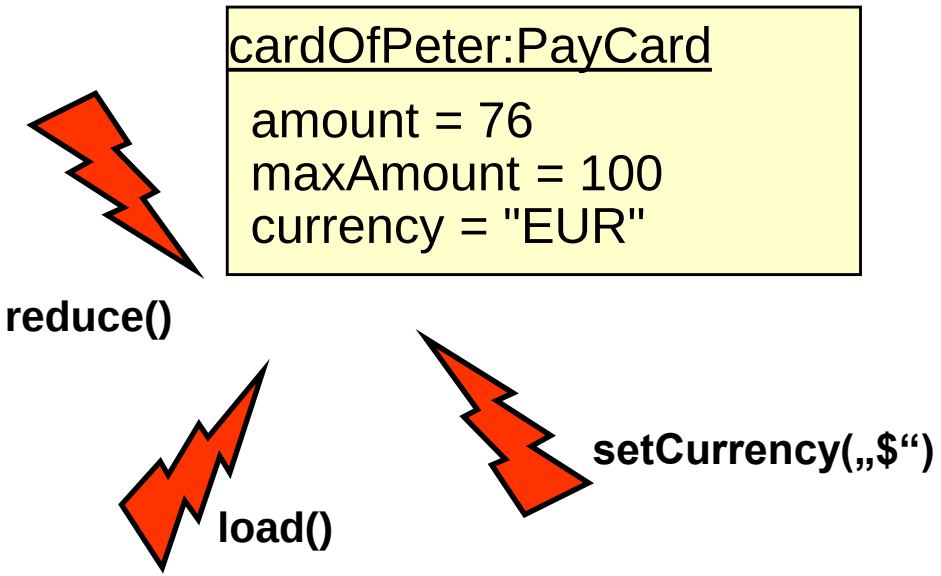
Prinzipien

- Daten und die zu diesen gehörenden Funktionen werden eng in einem Objekt verbunden
- Kapselung:
  - 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).
  - Die Kapselung befähigt das Objekt, über seine Eigenschaften selbst zu wachen.
  - Es verwaltet sich mit Hilfe seiner Methoden selbst.
- Wiederverwendbarkeit:
  - Nur einmal definieren, wie ein Objekt funktioniert, und dieses immer wieder, auch für neue Typen von Objekten, verwenden
- 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.

Klasse

- Klasse ist ein genereller Bauplan für Objekte
  - Instanzen einer Klasse unterscheiden sich in ihrem Zustand
  - Attribute sind häufig einfache Zahlen oder Zeichenketten
- Klasse ist ein selbstdefinierter Datentyp, bestehend aus
  - Eigenschaften → „Attribute“
  - Fähigkeiten → „Methoden“
- Beispiel:

PayCard
-amount : int = 0 -currency : String = EUR -@maxAmount : int
+load(theAmount : int) +reduce(theAmount : int) +setCurrency(theCurrency : String)



Klasse

- Klasse ist ein genereller Bauplan für Objekte
  - Instanzen einer Klasse unterscheiden sich in ihrem Zustand → verschiedene PayCards mit unterschiedlichen amounts (s. unten)
  - Attribute sind häufig einfache Zahlen oder Zeichenketten
- Klasse ist ein selbstdefinierter Datentyp, bestehend aus
  - Eigenschaften → „Attribute“
  - Fähigkeiten → „Methoden“
- Beispiel:

PayCard
-amount : int = 0 -currency : String = EUR -@maxAmount : int
+load(theAmount : int) +reduce(theAmount : int) +setCurrency(theCurrency : String)

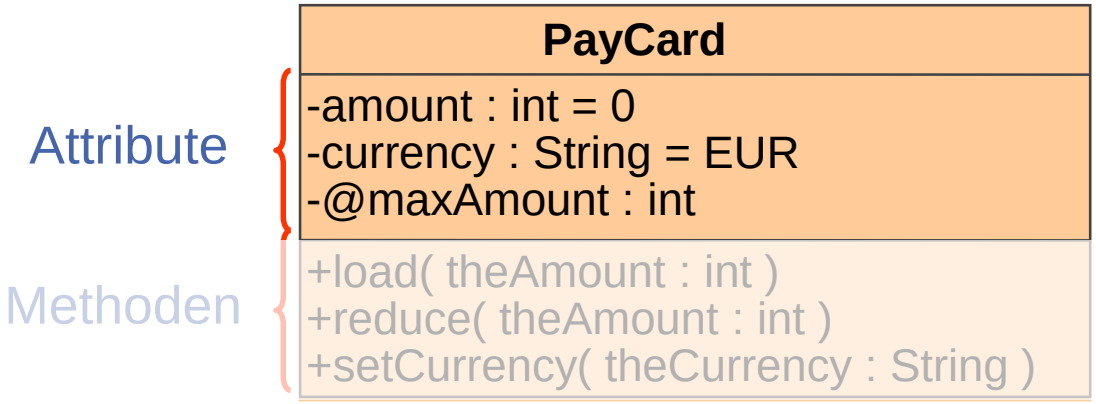
set of objects

cardOfPeter:PayCard
amount = 76
maxAmount = 100
currency = "EUR"

cardOfFoo: PayCard
amount = 6
maxAmount = 99
currency = "EUR"

cardOfTom:PayCard
amount = 6
maxAmount = 99
currency = "EUR"

Attribute

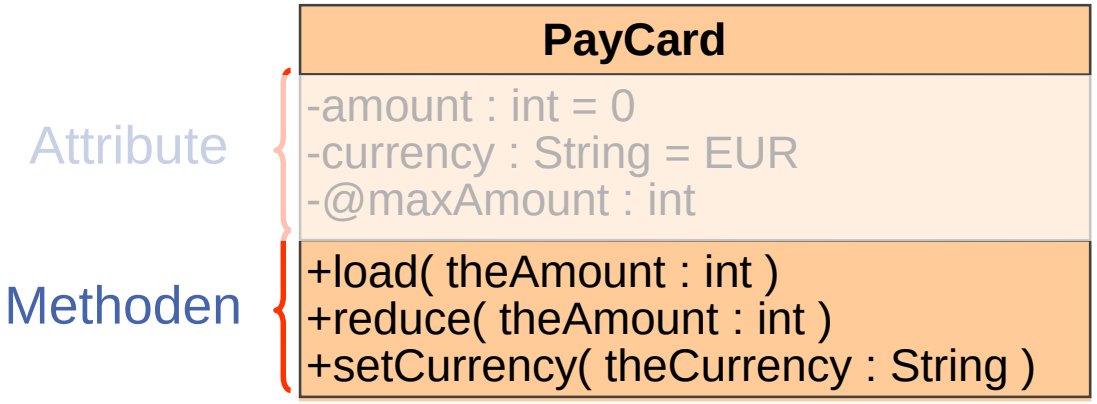


- Attribute definieren die Daten
- Gesamtheit der Attribute definiert den Zustand des Objekts
- Attribute haben Eigenschaften, z.B.
  - Typ
  - Wertebereich
  - Unveränderlichkeit (konstant während Objekt-Lebensdauer)
  - Sichtbarkeit

Sichtbarkeit von Methoden und Attributen

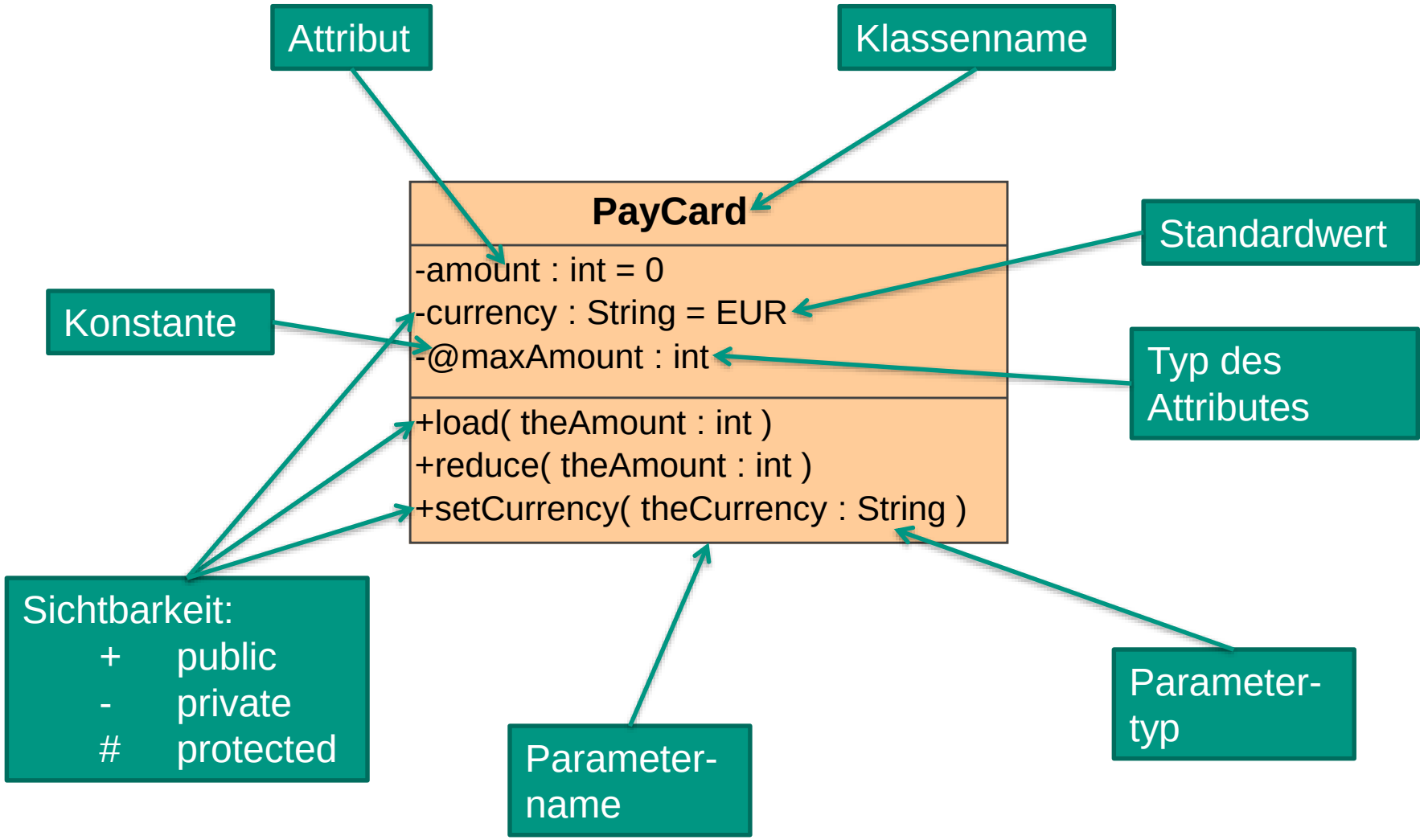
- *private*
  - Methoden oder Variablen vom Typ private sind nur in der aktuellen Klasse sichtbar.
  - Sowohl für Aufrufer von Objekten der Klasse als auch für abgeleitete Klassen bleiben sie unsichtbar.
- *protected*
  - Methoden oder Variablen vom Typ protected sind in der aktuellen Klasse und in abgeleiteten Klassen sichtbar.
- *public*
  - Attribute und Methoden vom Typ public sind im Rahmen ihrer Lebensdauer überall sichtbar.
  - Sie können daher in der eigenen Klasse und von beliebigen Methoden anderer Klassen verwendet werden.
  - Das Attribut public ist auch bei der Klassendefinition selbst wichtig, denn nur Klassen, die als public deklariert wurden, sind außerhalb der Klasse sichtbar, in der sie definiert wurden.
- Remark: *Beschränkung des Zugriffs auf Klassenelemente durch Sichtbarkeitsebenen*
  - *private-Elemente sollten immer dann verwendet werden, wenn Implementierungs abhängige Details zu verstecken sind, die auch in abgeleiteten Klassen nicht sichtbar sein sollen.*
  - *protected-Elemente sind vor Zugriffen von außen geschützt, können aber von abgeleiteten Klassen verwendet werden.*
  - *public-Elemente bilden die für alle sichtbaren Teile einer Klassendefinition und können daher als ihre Schnittstelle angesehen werden.*

Methoden



- Funktionen (Operationen), die fest der Klasse zugeordnet sind
- Methoden verändern den Zustand / die Daten eines Objekts
  - 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).
- Alle Instanzen einer Klasse weisen die gleichen Fähigkeiten auf.
  - Daher ist eine Methode zwar für alle Instanzen der Klasse derselbe Programmcode, dieser wird aber aufgrund unterschiedlicher Werte der Attribute auf andere Variablen angewandt.
- Methoden haben Eigenschaften
  - Sichtbarkeit
  - Typ des Rückgabewertes
  - Parameter

Klassendiagramm (Unified Modeling Language UML)



Definition von Klassen in C++

Konto
-name : string -nr : unsigned long -stand : double
+init(i_name : string, i_nr : unsigned long, i_stand : double) : bool +display() : void

```
// 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 i_name, unsigned long i_nr, double i_stand);
    void display();
};
```

Definition von Methoden in C++

Konto
-name : string -nr : unsigned long -stand : double
+init(i_name : string, i_nr : unsigned long, i_stand : double) : bool +display() : void

- Die Klasse „Konto“ wurde in **`konto.h`** mit folgenden Attributen der Sichtbarkeit private definiert.
  - einen String **`name`**
  - eine vorzeichenfreie lange Ganzzahl **`nr`**
  - eine Gleitkommazahl **`stand`** doppelter Genauigkeit
- Die festgelegten Methoden mit Programmcode definiert **`konto.cpp`**:
  - **`bool Konto::init (….)`**
    ist der Funktionskopf, der in Name, Rückgabewert und Parametern mit der Deklaration der Methode in der Klassendefinition übereinstimmen muss.
  - **`void Konto::display()`**
    void (englisch für *nichtig*, *ungültig*, *leer*) wird in einigen Programmier-sprachen anstelle eines Datentyps benutzt, um anzugeben, dass keine Daten übergeben werden oder der Typ der Daten nicht angegeben ist.
- **`size()`** ist eine Methode der Klasse string aus der Standardbibliothek und gibt die Länge des Strings als Anzahl an charactern bzw. Bytes zurück (s. backup)

Definition von Methoden in C++

Methoden

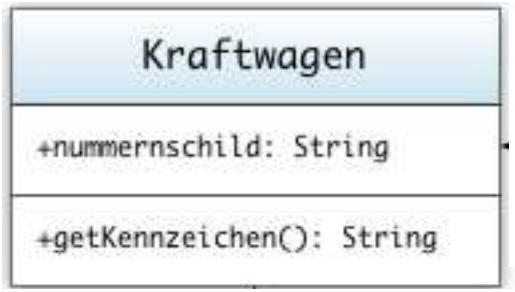
Konto
-name : string -nr : unsigned long -stand : double
+init(i_name : string, i_nr : unsigned long, i_stand : double) : bool +display() : void

```
 // 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 ) {          //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  << "Kontoinhaber:   " << name  << endl
         << "Kontonummer:    " << nr    << endl
         << "Kontostand:     " << stand << endl;
}
```

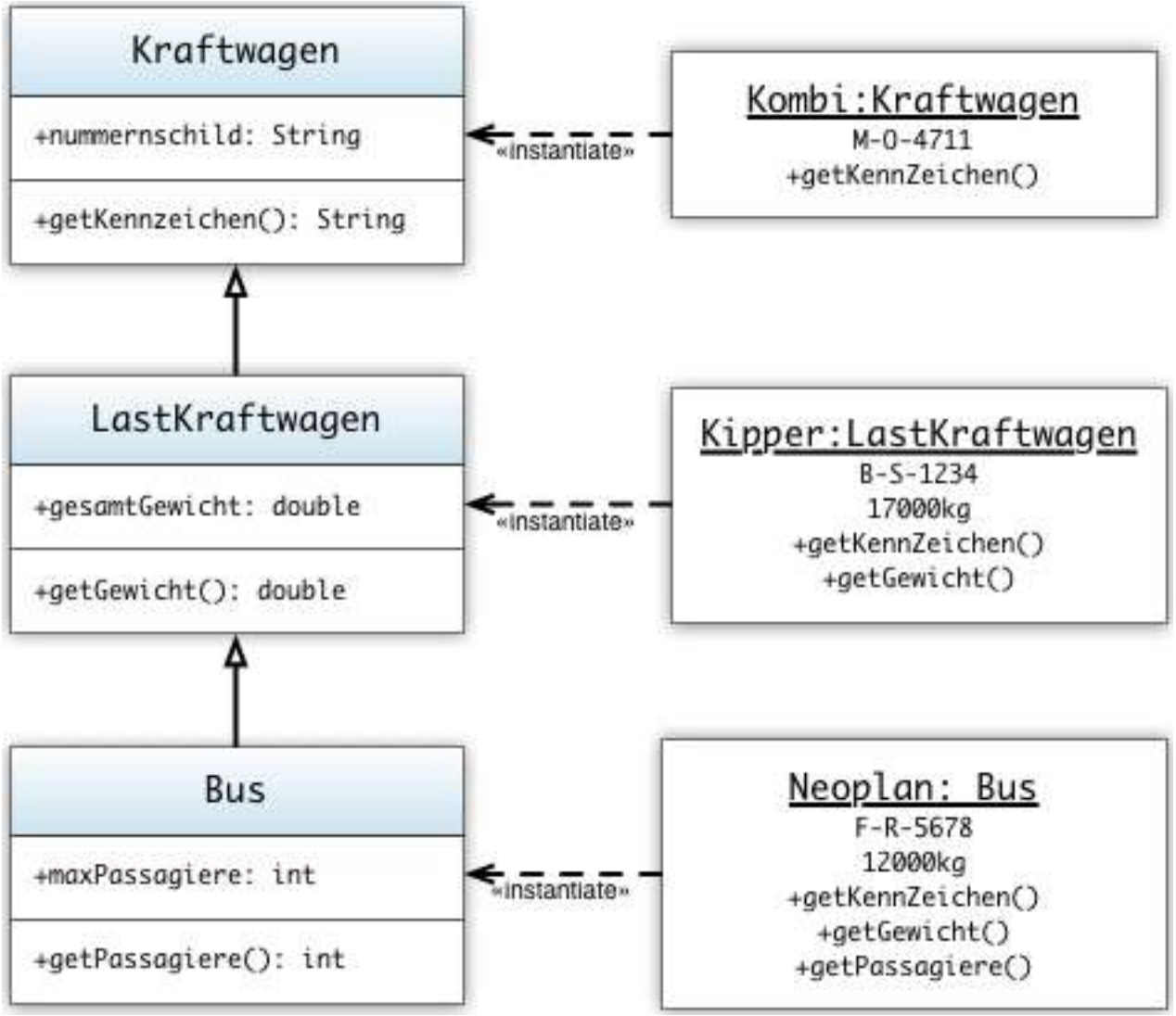
Vererbung

- Superklassen vererben Eigenschaften an ihre jeweiligen Kindklassen, die Subklassen.
  - Es gilt: Die Subklasse erbt die Attribute ihrer Elternklasse.
- Vorteile
  - reduzierter Programmieraufwand
  - leicht verständlicher Quellcode
  - Effizienz
  - übersichtliche Programmstrukturen
  - Performancemaximierung
- Durch Vererbung gemeinsame Attribute nur einmal modellieren
  - Eigenschaft auch nur ein einziges Mal zu testen!
  - Dadurch reduziert sich auch die Möglichkeit Programmfehler zu integrieren deutlich!
  - Ebenso sind Korrekturen im Quellcode nur an einer einzigen Stelle erforderlich – bei komplexen Programmen lassen sich Kosten für die Programmierung somit spürbar reduzieren.

Beispiel zu Vererbung



Beispiel zu Vererbung

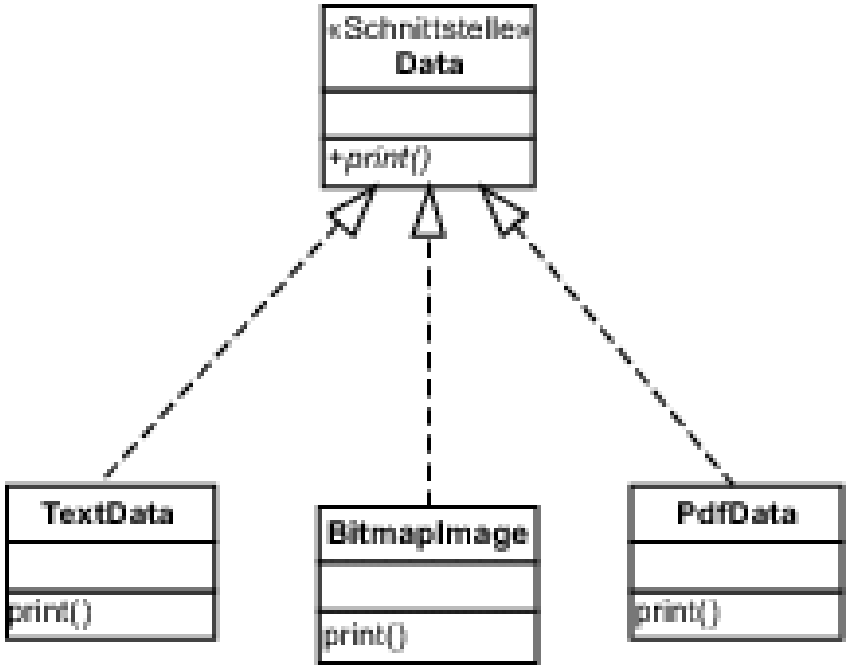


Polymorphismus
Vielgestaltigkeit

- Verschiedene Objekte bei Aufruf derselben Operation zeigen unterschiedliches Verhalten
  - Da Kindklassen von ihrer Elternklassen sämtlichen Eigenschaften erben, besitzt die Subklasse automatisch sämtliche Methoden der Superklasse.
  - Folglich würde die Subklasse nach dem Konzept der **Vererbung** ohne Methodendefiniton die Orginalmethode der Superklasse erben.
- Wenn wir die Methode in der Subklasse nun polymorph neu definieren, die Methode also überschreiben, ersetzen wir in Instanzen unserer Subklasse die allgemeine Methode der Superklasse durch eine spezielle Methodenfassung.
- Somit erhalten wir durch das Konzept der Polymorphie die Chance spezialisierte Funktionen in einer Kindklasse zu definieren.

Polymorphismus
Beispiel: print()

- Die Operation print() wird auf die Klasse `data` definiert.
- Wenn print() auf ein Exemplar …
  - … von `TextData` aufgerufen wird, wird die Methode print() der Klasse `TextData` aufgerufen.
  - … von `BitmapImage` aufgerufen wird, wird die Methode print() von `BitmapImage` aufgerufen.
  - … Von `PdfData` …



- Von der Realität zum Objekt
- Aufbau einer Klasse
  - Attribute
  - Methoden

Backup

std::string::size

C++98 | C++11

```
size_t size() const;
```

Return length of string

Returns the length of the string, in terms of bytes.

This is the number of actual bytes that conform the contents of the string, which is not necessarily equal to its storage capacity.

Note that `string` objects handle bytes without knowledge of the encoding that may eventually be used to encode the characters it contains. Therefore, the value returned may not correspond to the actual number of encoded characters in sequences of multi-byte or variable-length characters (such as UTF-8).

Both `string::size` and string::length are synonyms and return the same value.