Hochschule Esslingen
University of Applied Sciences

Objektorientierte Programmierung mit C++:

Klassenkonzept, Schutzmechanismen

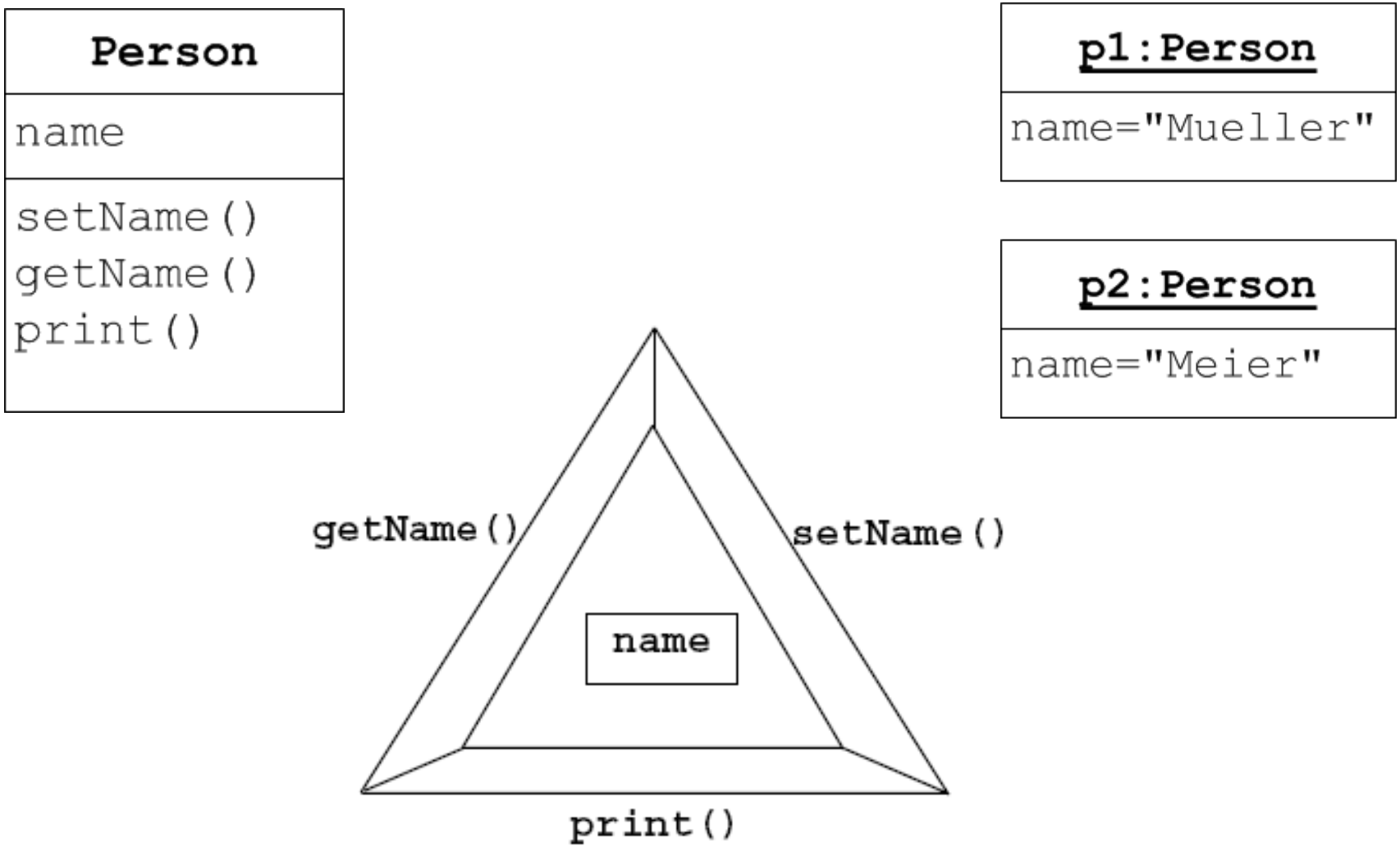
Profs. M. Dausmann, D. Schoop, A. Rößler

Fakultät Informationstechnik, Hochschule Esslingen

Agenda

- Definition einer Klasse in C++
- Gültigkeitsbereich einer Klasse
- Initialisierung von Klassenelementen
- Kapselung von Klassenelementen
- Schutzmechanismen
- Methoden und **`this`**
- Klassen vs. Strukturen: Unterschiede
- Modularisierung von Programmen

Konzept einer Klasse



Definition einer Klasse in C++

```
#include <iostream>
class Person {
  char *name;
public:
  void setName(char *n){ name = n; }
  char * getName() { return name; }
  void print(); // nur Schnittstelle
};
void Person::print(){
  std::cout << "Name: " << name << std::endl;
} // print() wird außerhalb der Klasse
  // definiert (nicht automatisch inline)
```

Benutzen einer Klasse in C++

```
void main() {
  Person p1, p2;
  // p1.name = "Mueller"; // Fehler!!!
  p1.setName("Mueller");
  p2.setName("Meier");
  p1.print();
  p2.print();
  std::cout << p1.getName() << std::endl;
  std::cout << p2.getName() << std::endl;
};
```

Vokabular

- Es gibt zwei Arten von Elementen, die eine Klasse enthalten kann: Datenelemente und Methoden
- Alternative Begriffe:
  - Datenelemente, Attribute, Instanzvariablen, Mitgliedsvariablen, Member
  - Methoden, Elementfunktionen, Instanzmethoden, Mitgliedfunktionen, Memberfunktionen
- Eine Klasse stellt eine Vorlage dar, um Objekte zu erzeugen. Objekte werden auch als Instanzen oder Exemplare der Klasse bezeichnet

Gültigkeitsbereich einer Klasse

- Klasse ist eigener Gültigkeitsbereich.
- Alle Methoden einer Klasse können implizit (d. h. ohne Parameterübergabe) auf alle Klassenelemente zugreifen.
- In C++ kann in drei Bereichen auf ein Objekt Bezug genommen werden:
  - lokal (innerhalb eines Blockes)
  - global (innerhalb einer Datei)
  - innerhalb einer Klasse

Zugriff auf Klassenelemente

```
class Person {.......};

class Bruch {
  int zaehler, nenner;
public:
  void print(){
   cout << "Quotient= " << zaehler
     << "/" << nenner << endl;
  }
};

void main(){
  Person p1, p2; Bruch b;
  p1.print(); p2.print(); b.print();
  // print() der zugehörigen Klasse wird aufgerufen
}
```

Zugriff mit Scope-Operator

```
int x = 333;                    // globales x
class Punkt {
  int x, y;
  public:
  void verschiebe(int dx, int dy){
    int x = 0;                  // lokales x
    Punkt::x += dx;             // nicht Punkt.x
    y += dy;                    // Klassenscope
    x += 3;                     // lokales x
    ::x += ::x;                 // globales x
  }
};
```

„Initialisierung“ von Klassenelementen

```
class Bruch {
  int zaehler, nenner;
public:
  void init(){                    // evtl. auch Parameter
    zaehler = 0; nenner = 1;      // Zuweisung !
  }
  void print(){
    std::cout << "Quotient= " << zaehler << "/"
    << nenner << std::endl;
  }
};
.....

Bruch b;
b.init(); b.print();
```

Kapselung von Klassenelementen

```
class Person {
 private:
  char *name;
 public:
  void setName(char *n){ name = n; }
  char * getName() { return name; }
  void print(); // nur Schnittstelle
};


void main(void) {
  Person p;
   p.name = "Mueller";
   // Fehler bei Kompilierung!!
};
```

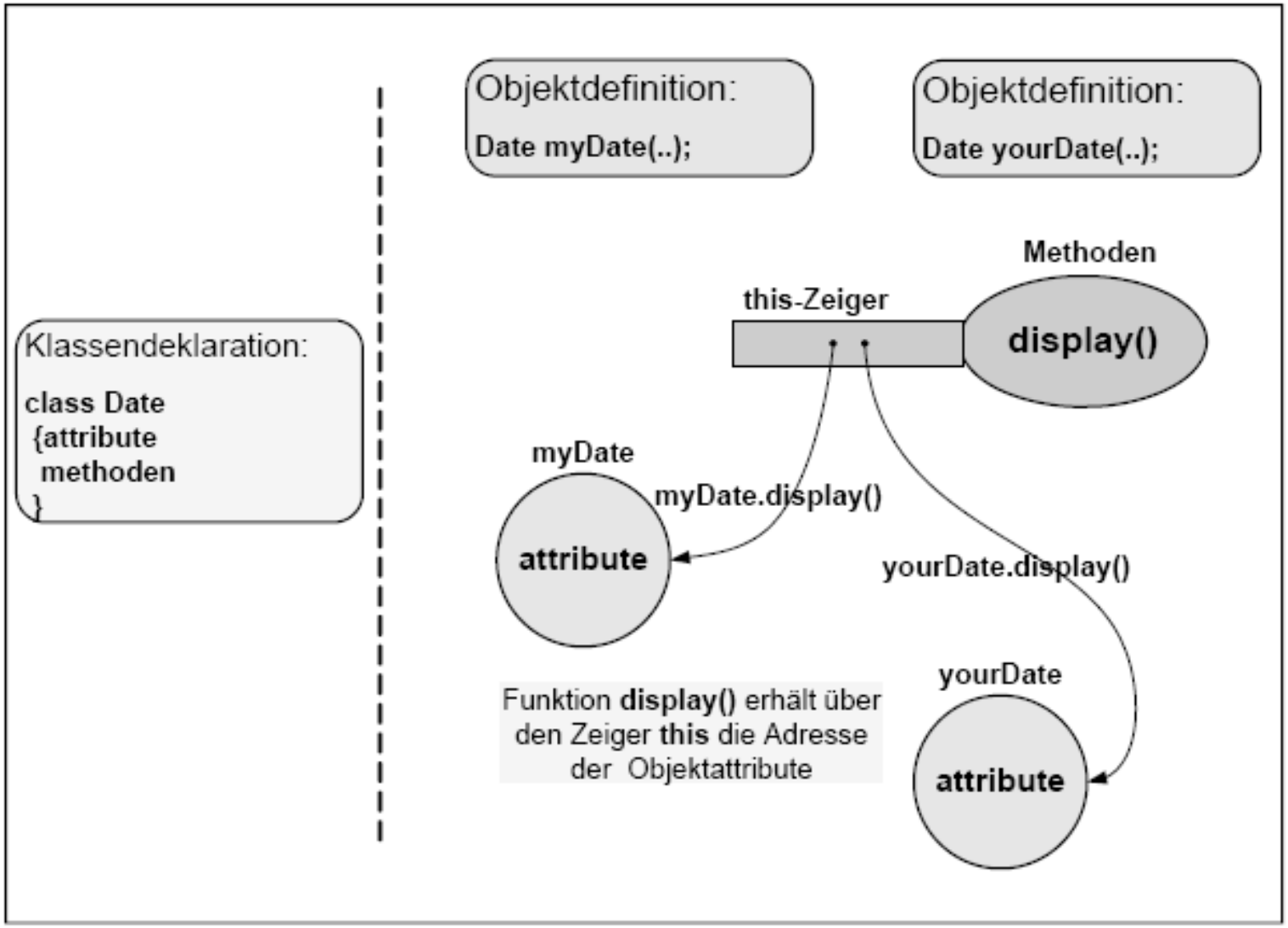
Schutzmechanismen

- Erlaubnis für Zugriff auf Elemente eines Objektes kann über das Schutzattribut gesteuert werden.
- Schlüsselwörter dafür: `public`, `private`, `protected`.
- `public` (öffentliches Element):
  - Elemente sind für alle innerhalb und außerhalb der Klasse zugänglich.
- `private` (private Elemente)
  - Elemente sind für alle innerhalb der Klasse direkt zugänglich. Elemente sind außerhalb der Klasse (einschließlich deren Nachkommen) nur über Methoden zugänglich.
- `protected` (geschützte Elemente)
  - In der Klasse und deren Nachkommen ist direkter Zugriff möglich. Elemente sind außerhalb der Klasse nur über Methoden zugänglich.

Methoden und `this`

- Die Methoden einer Klasse werden nur einmal im Speicher abgelegt. Die einzelnen Instanzen liegen ebenfalls im Speicher, enthalten aber nur die Daten, also die Attribute.
- Wird eine Methode zu einer speziellen Instanz aufgerufen, dann braucht diese Methode die Information, wo die Instanzattribute sich befinden.
- Diese Information stellt der Compiler im Zeiger **`this`** zur Verfügung, der innerhalb einer Methode auch explizit verwendet werden kann.
- Jeder Methode wird bei ihrem Aufruf der **`this`**-Zeiger übergeben. Er zeigt auf das aufrufende Objekt der Klasse, zu der die Methode gehört.

Methoden und `this`



Methoden und this

```
class Person {
   ........
   int gehalt;
public:
   ........
   Person & gehaltAendern(int neues_gehalt){
     this->gehalt = neues_gehalt;
     gehalt += 50;
     return *this;
   }
};
```

Klassen vs. Strukturen: Unterschiede

- Klassen können als **`class`** oder **`struct`** definiert werden.
- **`class`** und **`struct`** sind in C++ bis auf den Defaultwert des Schutzattributes identisch.
- Defaultwert des Schutzattributes:

  `class`: **`private`**

  `struct`: **`public`**
- Was soll man nutzen (ist natürlich Geschmackssache):
  - **`class`** für Objektorientierung,
  - **`struct`** für PODs (Plain Old Data), d.h. wie in C.

Beispiel eines `struct` in C++

- Nicht wundern, wenn man in einem C++-Programm folgende Struktur findet:

```
struct Punkt {
    private: // muss sein, da default public
        int x, y;
    public:
        void setX (int n) { x = n;}
    ...
};
```

- Und folgende Benutzung:

```
Punkt p;  p.setX(10);
```

Modularisierung von Programmen

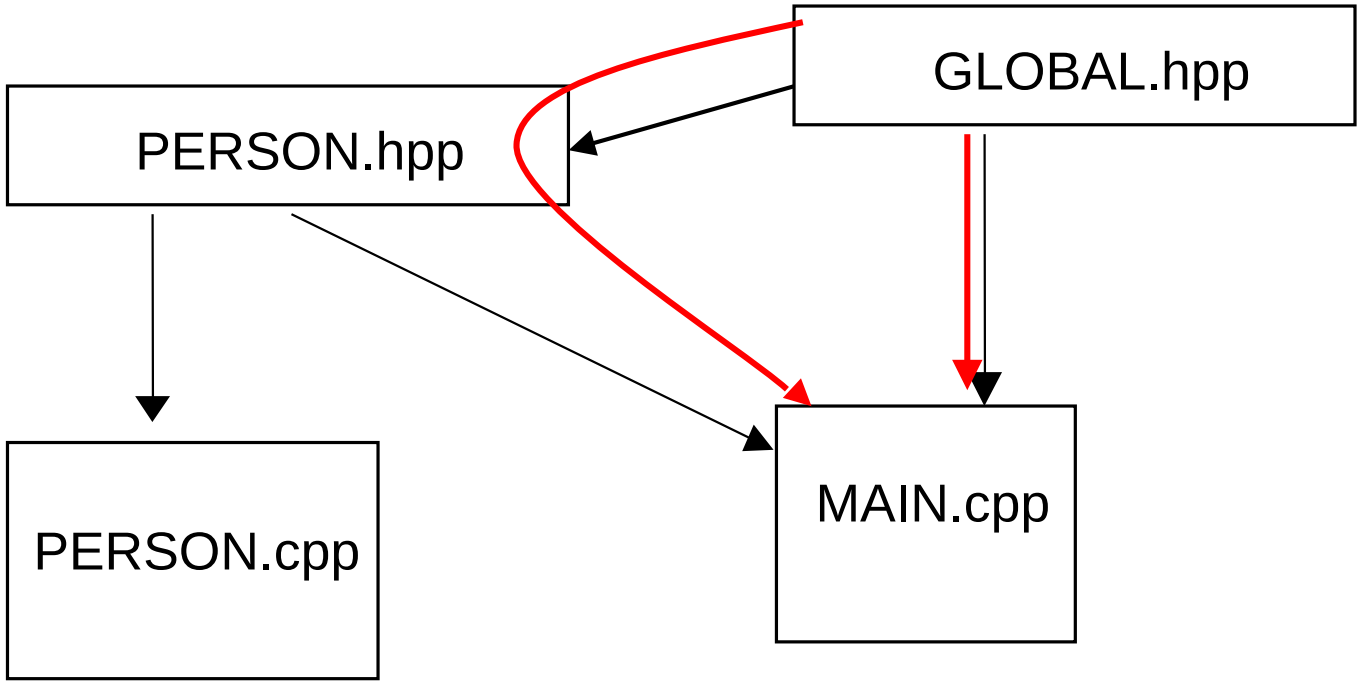
- Zerlegung eines Programms in mehrere Dateien
- Strukturierung von Programmen zur Übersichtlichkeit in
  - Deklarationsteil (<Dateiname>.h)
  - Implementierungsteil (<Dateiname>.cpp)
- Sowohl der Deklarations- als auch der Implementierungsteil kann wiederum auf verschiedene Dateien aufgeteilt werden, um die Komplexität zu verringern und Arbeitsteilung sowie Wiederverwertung zu ermöglichen.
- Einbinden eines Deklarationsteils in einer anderen Datei durch

```
#include  "<Dateiname>.hpp"
```

- Auch die cpp-Datei einer Klasse inkludiert die eigene Header-Datei.

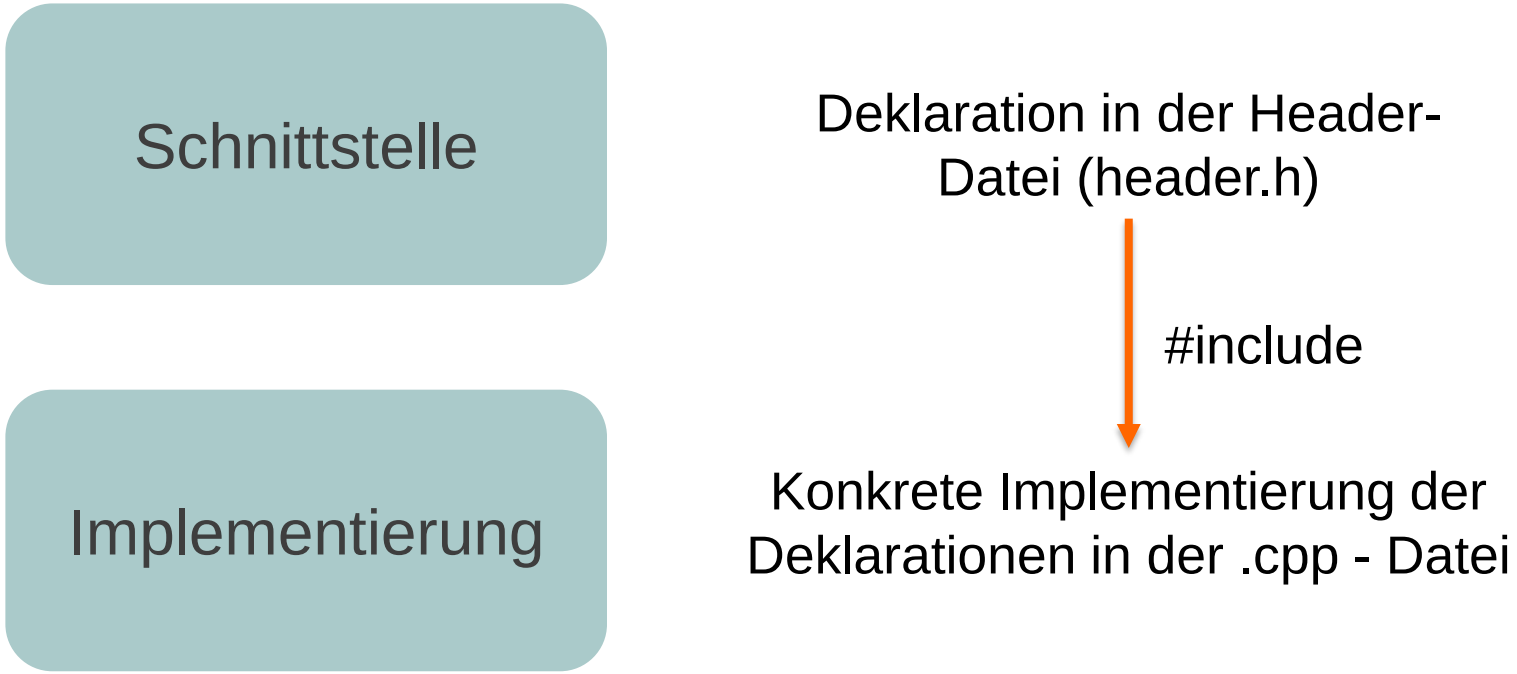
Mehrfaches Inkludieren

- Header-Dateien müssen andere Header-Dateien inkludieren, wenn der Compiler daraus Infos benötigt. Dadurch kann es vorkommen, dass Header-Dateien mehrfach inkludiert werden.



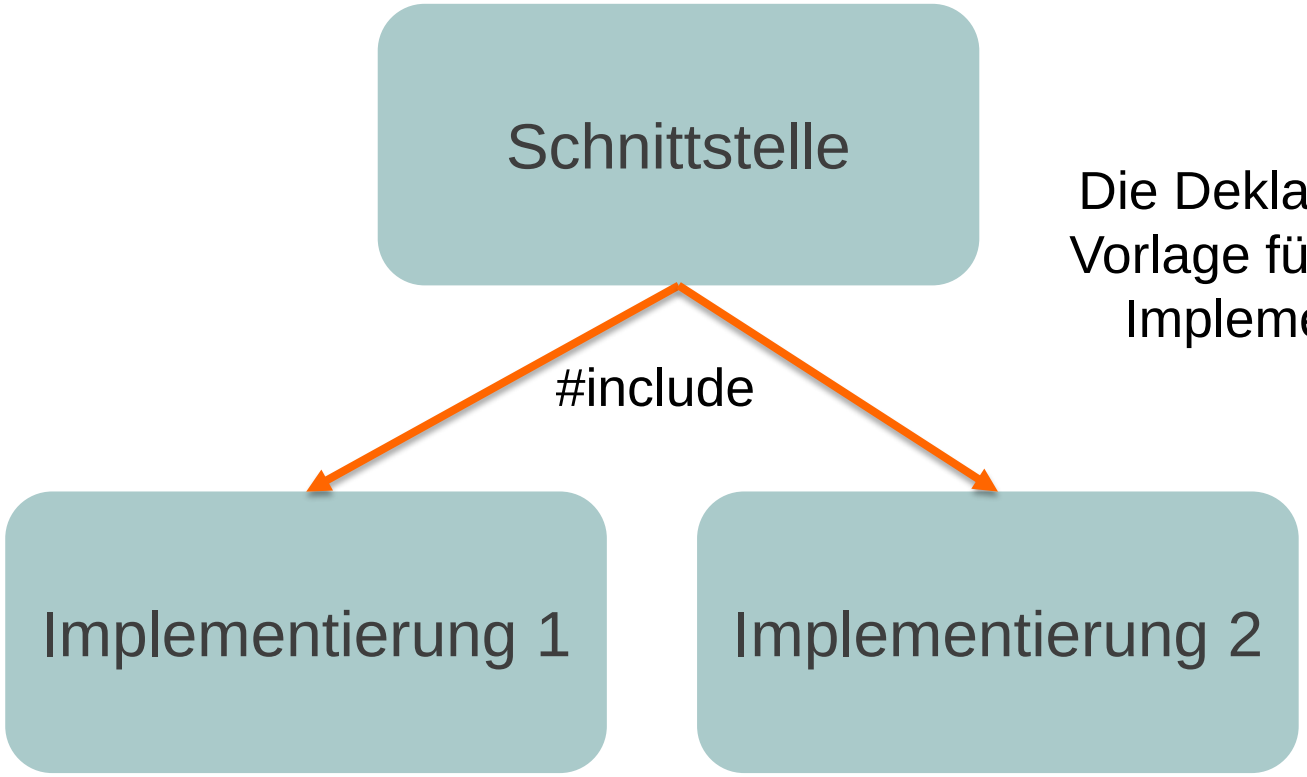
Trennung von Schnittstelle und Implementierung

- In C++ trennt man zwischen der Deklaration und der Implementierung ((Standard in der professionellen C++ Softwareentwicklung)



Trennung von Schnittstelle und Implementierung

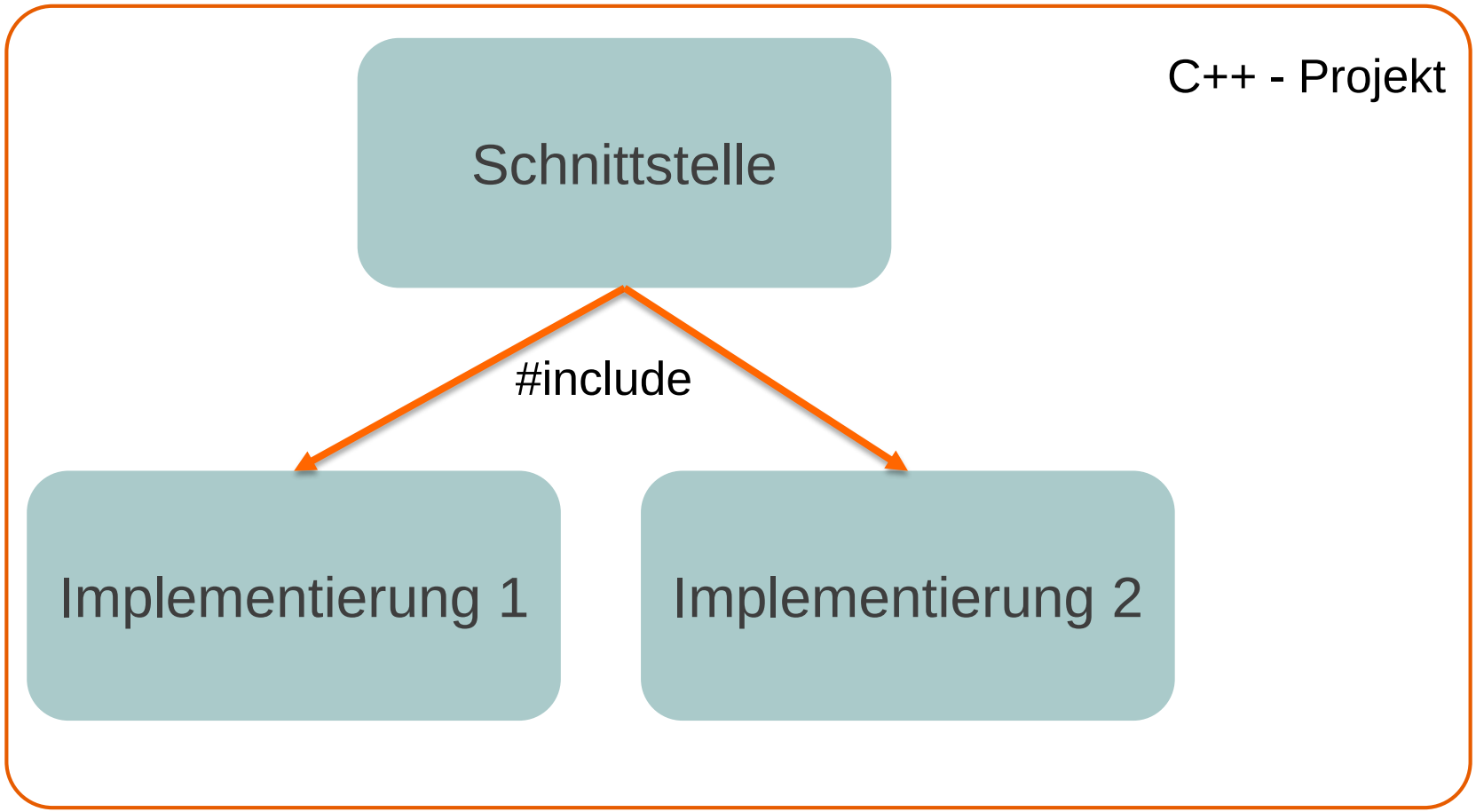
- In C++ trennt man zwischen der Deklaration und der Implementierung



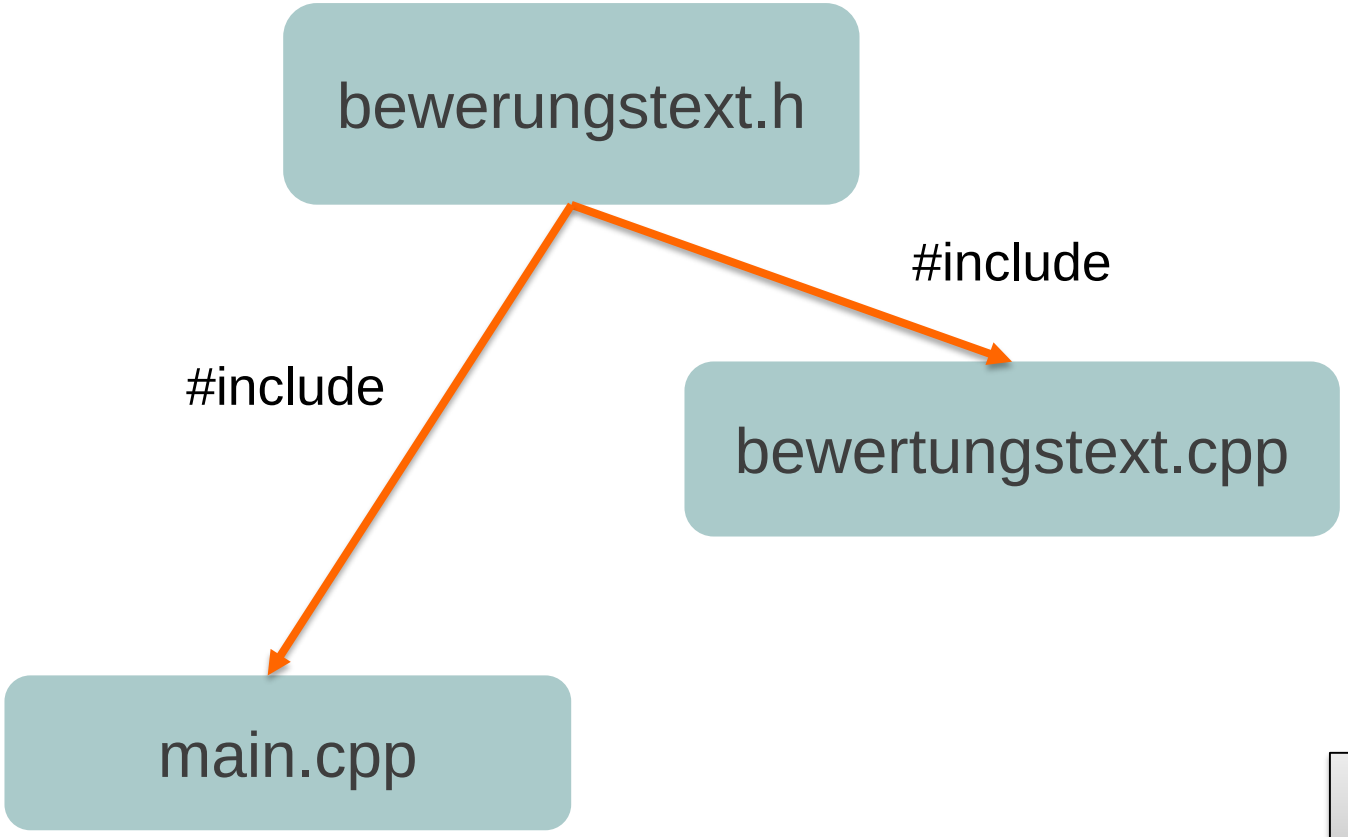
Die Deklaration ist eine Vorlage für alle anderen Implementierungen

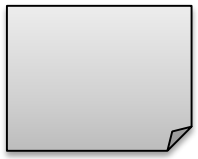
Trennung von Schnittstelle und Implementierung

- In C++ trennt man zwischen der Deklaration und der Implementierung



Trennung von Schnittstelle und Implementierung





bewertungstext (projekt)

Verhindern einer mehrfachen Deklaration

- Problem: Klassendeklaration (Header-Datei) wird über **`#include`** indirekt mehrfach eingebunden.

  → Kompilermeldung: **`multiple declaration`**

- Abhilfe mit Präprozessoranweisungen:

```
#ifndef classname_h
#define classname_h
class classname {
   …
}
#endif
```

Pragma once

- Nachteil der Lösung mit `#define`: die Namen müssen projektweit eindeutig sein!
- Eine andere Lösung: Man schreibt in die Header-Datei:

```
#pragma once
```

- Vorteil: Das Pragma kann u.U. effizienter implementiert werden, da bei der Lösung mit `#define` die Datei immer zuerst geöffnet werden muss, bis man merkt, dass man sie eigentlich gar nicht braucht.
- Nachteil: es ist kein Standard und daher compilerabhängig.
- Visual Studio und GCC beispielsweise akzeptieren das Pragma.

Zusammenfassung

- Klassenelemente sind (Daten-)Attribute und Methoden.
- Klassenelemente können geschützt werden gegen Zugriff von außen.
- Die Implementierung der Methoden kann innerhalb oder außerhalb der Klasse erfolgen.
- Erfolgt die Implementierung außerhalb, muss der Scope-Operator benutzt werden, um sie der Klasse zuzuordnen.
- Der Scope-Operator kann auch benutzt werden, um Namensprobleme zu lösen.
- Methoden haben Zugriff auf die Attribute des Objekts, für das sie aufgerufen werden, direkt oder über den this-Zeiger.

Zusammenfassung

- Die Klassendefinition kann in Header- und Source-Datei(en) aufgeteilt werden.
- Die Source-Datei einer Klasse inkludiert ihre eigene Header-Datei.
- In jeder Datei, in der eine Klasse benutzt wird, wird deren Header-Datei inkludiert.
- Um mehrfaches Inkludieren zu vermeiden, kann man mit Include-Guards (`#ifndef guard - #define guard`) arbeiten oder mit dem compilerabhängigen `#pragma once`.