Programmiertechniken 2

Prof. Dr.-Ing. Zhen Ru Dai
zhenru.dai@haw-hamburg.de

Flipped Classroom

Prinzipien von Flipped Classroom:

- Sie bekommen Lernmaterialien und lernen für sich zuhause, gerne auch mit Ihren Kommilitonen.
- In der Vorlesung
  - Ich werde Fragen zu den Lernmaterialien beantworten.
  - Sie bekommen Workshops-Aufgaben und werden selbstständig an Ihren Laptops arbeiten.
  - Betreuung von mir.
- Daher unbedingt vorbereitet in die Vorlesung und Workshops kommen!

Flipped Classroom

1. Ulrich Breymann:
   Kapitel 3 – Kapitel 3.3.3 (Seite171)

2. Konstruktor, Überladen von Konstruktor
   https://youtu.be/vz1O9nRyZaY?list=PL318A5EB91569E29A
3. Destruktor, Getter, Setter
   https://youtu.be/b9wialxvcVA?list=PL318A5EB91569E29A

Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Einstieg in die OO Denkweise

1. Wählen Sie ein geeignetes Objekt aus. Bestimmen sie, was
   - typische Kenngrößen sind und
   - typische Dinge sind, die sie damit machen können.
     - Ändern sich dabei Kenngrößen?
2. Finden Sie in unserer Welt etwas, das kein Objekt ist!

Einstieg in die OO Denkweise

Betrachten Sie Dinge des täglichen Lebens, wie z.B. eine Kaffeemaschine, ein Auto, ein Magnetventil, ...

- Die Dinge haben Kenngrößen.
  - Einige Kenngrößen sind konstant.
  - Einige Kenngrößen verändern sich.
- Die Dinge sind Individuen.
- Die Dinge sind als Individuen typische Vertreter einer Menge.
- Die Dinge können typische Sachen machen.

Ziele von Objekt-Orientierung

- OO richtet sich an Sachen aus der reale Welt, statt sich mit computer-technischen Konstruktionen zu beschäftigen
- Information Hiding durch Kapselung von Attributen und Methoden
- Effiziente Wiederverwendbarkeit
  - Vererbung
  - Virtuelle Methoden
- Grundlage für SW Design heutzutage

Konzepte der OO Sichtweise

1. Klasse:
   - Beschreibung einer Menge von Objekten mit gemeinsamen Eigenschaften und Verhalten. Ist ein Datentyp!
2. Objekt:
   - Eine konkrete Ausprägung, eine Instanz, ein Exemplar der Klasse. Belegt Speicher, besitzt Identität! Objekte tun etwas.
3. Methode
   - Beschreibt das Verhalten eines Objektes. Kommunikationskanal zum Objekt.
4. Attribute
   - Beschreibt Eigenschaften eines Objektes.

Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Eigenschaften von C und C++

- C ist eine imperative Programmiersprache (70er Jahre):
  - Algorithmus als Sequenz von hintereinander auszuführenden Befehlen
- C ist eine prozedurale Programmiersprache:
  - zur Strukturierung des Quelltextes kann der Gesamtalgorithmus in Funktionen zerlegt werden
- C++ ist eine objekt-orientierte Programmiersprache (80er Jahre):
  - Zur weiteren Strukturierung können Daten von Objekten sowie die zugehörige Funktionalität als Methoden in Klassen zusammengefasst werden

Unterschiede zwischen C und C++

- C++ konzipiert als kompatible Erweiterung von C
- Eigentlich sollte jedes C-Programm auch von einem C++-Compiler übersetzt werden können
- Im Detail jedoch problematisch, z.B.
  - Variablennamen in C-Programmen können Schlüsselwörter in C++ sein, z.B. *class* , *new* , etc.
  - Typkonvertierung wegen Objekt-orientierung unterschiedlich
  - Standards entwickeln sich parallel weiter, aber gegenseitige Berücksichtigung
- C++ bietet alternative, objekt-orientierte Bibliotheken für Standardaufgaben

Objekt-Orientierung in C++

- Eine Klasse ist die Beschreibung eines Bauplans für konkrete Objekte
- Klasse = Beschreibung von Attributen (Eigenschaften) und Methoden (Funktionen)
- Eine Klasse ist nicht das Objekt selbst *(z.B. Klasse* Person*)*
- Ein Objekt ist eine Instanz einer Klasse
  (z.B. Objekt Zhen Ru Dai der *Klasse* Person*)*

Datentyp und Variable

Klasse und Objekt

Objekt-Orientierung in C++

- Quantitative Eigenschaften werden in Attributen, die selber Objekte oder einfache Datentypen sind, gespeichert.
- Über Methoden, Funktionen die an die Objekte gebunden sind, können die Attribute verändert oder ausgelesen werden.
- Objekte haben wie Variablen einen Bezeichner.
- Zugriff auf die Attribute/Methoden erfolgt über Punkt-Operator.

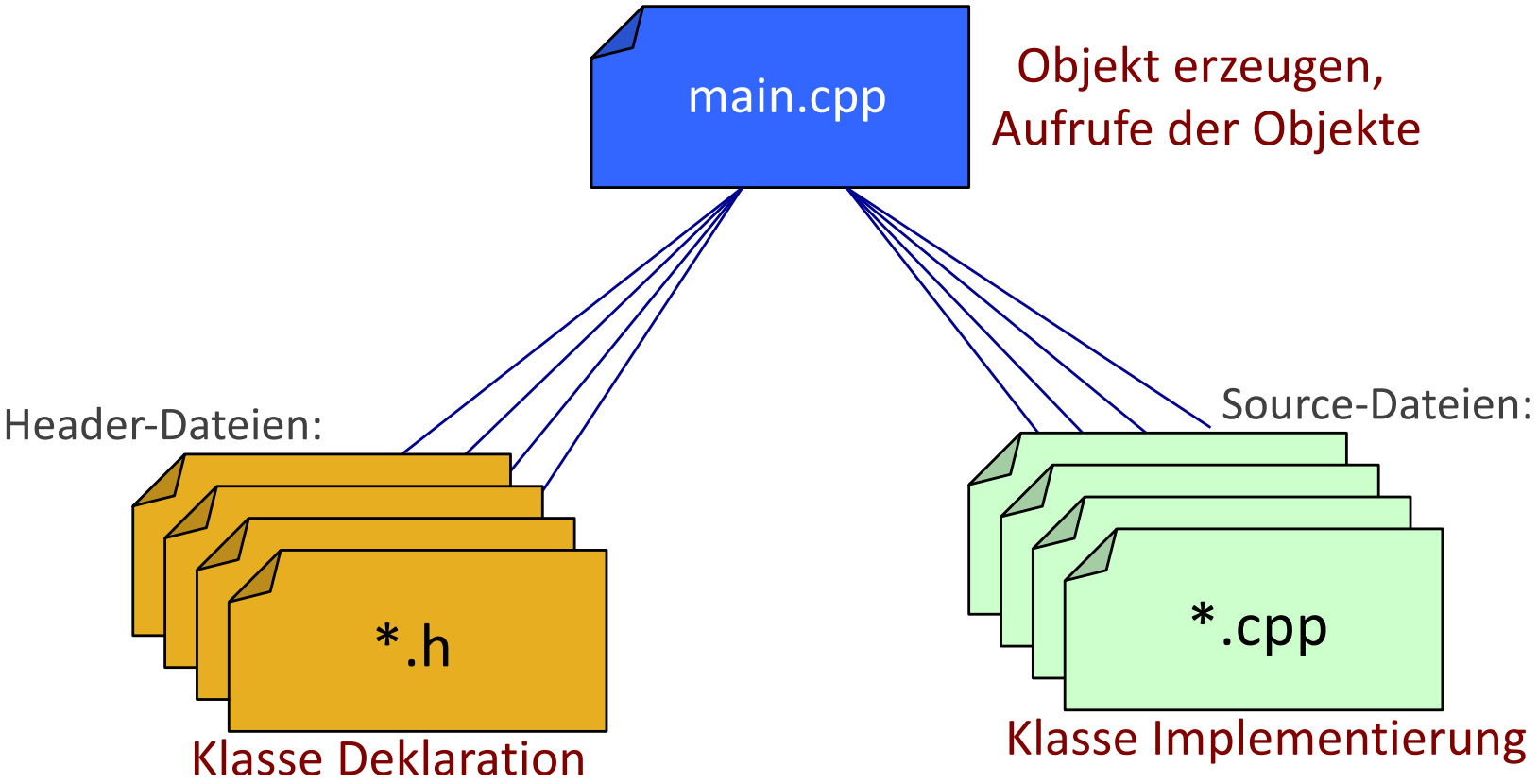
Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Klasse und Objekte

- Information Hiding durch Trennung von Klassendefinition (`*.h`) und Implementierung (`*.cpp`)
- Da Methoden in verschiedenen Klassen den gleichen Bezeichner haben können, wird die Zugehörigkeit über den Scope-Operator „`::`“ angegeben.
  - Bei Implementierung ist Angabe des Klassennamens nötig!
  - z.B. `<Klassenname>::<Methode>`

Klasse und Objekte



C++ und ...

BMI.h

```
class BMI {
private:
    // Attribute Deklaration:
    string newName;
    int newHeight;
    double newWeight;
public:
    // Constructor Deklaration:
    BMI();
    // Destructor Deklaration:
    virtual ~BMI();
```

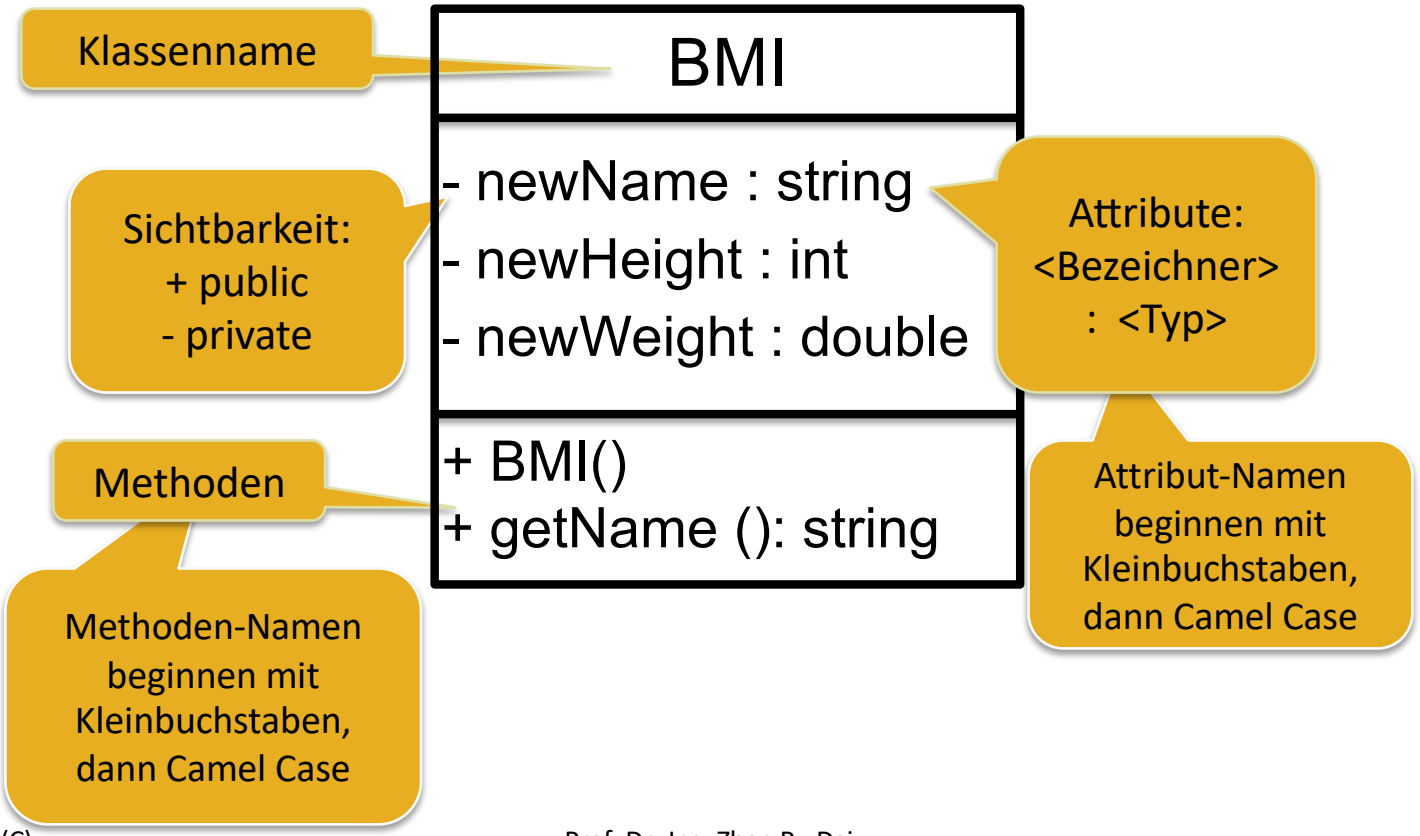
BMI.cpp

```
#include "BMI.h"

// Konstruktor Definition
BMI::BMI(){
    newName = " ";
    newHeight = 0;
    newWeight = 0.
}
// Destruktor Defi
virtual BMI::~BMI(){
}
```

Scope operator

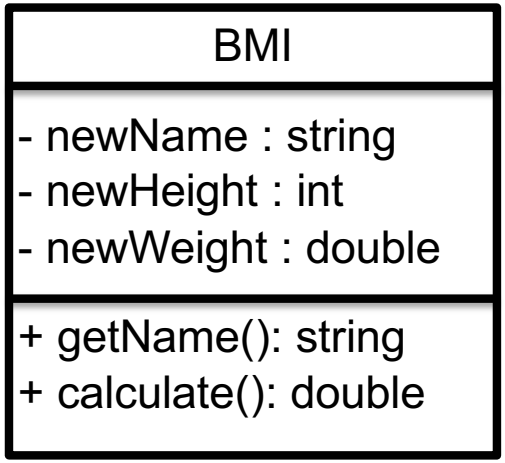
... UML Klassen

- Die Deklaration einer C++ Klasse findet man in der Header-Datei (*.h).
- UML Klassendiagramm wichtig für das Design von OO Programmen.



Methode und Attribute

- Methoden werden in der Klassendefinition deklariert.
- Methoden können wie Funktionen Parameter erhalten und Rückgabewerte liefern.
- Methoden können auf die Attribute und Methoden des Objektes zugreifen.
- Methoden sind an ein Objekt, genauer an die Klasse, gebunden.
- Namenskonvention:
  erste Buchstabe klein,
  danach Camel Case
  - z.B. ichBinHeuteEinKamel



Methode auf Attribute

- Mittels Methoden (an Objekte gebundene Funktionen) können weitere Größen berechnet werden. → Abgeleitete Attribute
- Mittels Methoden können Abfragen/Berechnungen gestaltet werden, die Aufgabe des Objektes sind.
  → Zuständigkeit gehört zum Objekt
- Mittels Methoden können die Attribute verändert werden.
  → Zustand des Objektes ändert sich

Punkt-Operator in C++

- Methoden und Attribute werden von einem Objekt aus aufgerufen
- Punkt-Operator „ **.** “
- Zugriff auf public Attribute eines Objekts (nicht zu empfehlen)
  - z.B. `Student.newHeight;`
- Zugriff auf public Methoden eines Objekts

main.cpp

```
BMI Student_1;
Student_1.setName("Max");
Student_1.setHeight(170);
Student_1.setWeight(89.4);
cout << "Mein Name: " << Student_1.getName() << endl;
```

Inhalt

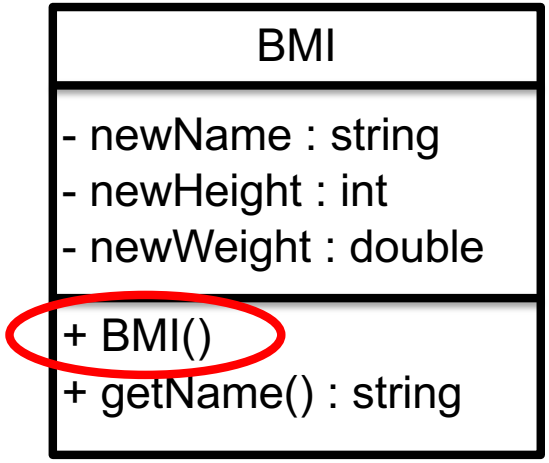
- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Konstruktoren

- Saubere Initialisierung eines Objekts →man kann erzwingen, dass nur initialisierte Instanzen erzeugt werden
- ggf. Bereitstellung von dynamischen Speicherplatz
- ggf. Benachrichtigung eines anderen Objekts über Erzeugung (Registrierung)
- Durch Überladen des Konstruktors (Overload): bequeme Möglichkeit zur Initialisierung

Konstruktoren

- Konstruktoren sind spezielle Methoden
- Konstruktoren dienen zur Erzeugung eines Objekts
- Der Name des Konstruktors ist derselbe wie der Name der Klasse, z.B. `BMI();`
- Konstruktoren werden automatisch bei der Erzeugung (Instanziierung) des Objektes aufgerufen.



BMI.h

```
public:
   // Default Constructor Deklaration
   BMI();
```

BMI.cpp

```
// Default Constructor Implementation
BMI::BMI(){
     newName = " ";
     newHeight = 0;
     newWeight = 0.0;
}
```

Konstruktoren

- Der Konstruktor besitzt keinen Rückgabewert, auch nicht `void`.
- Konstruktoren enthalten Standardwerte, so dass mit dem Objekt sinnvoll gearbeitet werden kann.
- Ein Konstruktor ohne formale Parameter wird als Standard-Konstruktor bezeichnet.
- Hat eine Klasse keinen Konstruktor, so erzeugt der Compiler im Hintergrund automatisch einen Standard-Konstruktor.
- Konstruktoren können überladen werden
  - Bei mehreren (überladenen) Konstruktoren wird der ausgewählt, der am besten zur Signatur passt.

Überladen von Methoden/Konstruktoren

- Die Signatur einer Funktion/Methode beinhaltet den Bezeichner, die Datentypen der formalen Parameter und deren Reihenfolge.

Beispiel:

```
void show(int start, int ende);
```

Welche Signaturen sind gleich?

```
void show(int ende, int start);
int show(int start, int ende);
void show(int start, int length);
void show(int start, double size);
void showAll(int start, int ende);
```

- Überladen von Methoden
  - beim gleichen Methodennamen unterschiedliche Anzahl von Parametern oder verschiedenen Datentypen

Methoden Überladen (Overload)

- Konstruktoren können überladen werden, d.h. je nach Bedarf können unterschiedliche Initialisierungen angeboten werden.
- Konstruktoren müssen sich in der Parameterliste unterscheiden.

BMI.h

```
public:
  // Default Constructor Deklaration
  BMI();

  // Overload Constuctor Deklaration
  BMI(string name, int height, double weight);
```

BMI.cpp

```
// Overload Konstruktor Implementierung
BMI::BMI(string name, int height, double weight) {
        newName = name;
        newHeight = height;
        newWeight = weight;
```

main.cpp

```
// Aufruf der überladenen Konstruktors:
BMI Student_1(name, height, weight);

// Aufruf der Default Konstruktors:
BMI Student_2;
```

Destruktoren

- Destruktoren sind spezielle Methoden, sie haben den gleichen Bezeichner wie die Klasse mit einer führenden „~“.
- Destruktoren werden automatisch bei der Löschung des Objektes aufgerufen.
- Destruktoren haben keinen Rückgabewert und keine Parameter!
- Macht Aufräumarbeiten
  - z.B. Schließen von Dateien
  - z.B. Abmeldung bei anderen Objekten
  - z.B. Freigabe von dynamischen Speicher, falls vorher angefordert wurde
- Typischerweise `virtual`.

Regeln für die Anwendung für Konstruktoren und Destruktoren (Exkurs)

- Allgemein
  - Bei mehreren globalen Objekten oder mehreren lokalen Objekten innerhalb eines Blockes werden
    - Die Konstruktoren in der Reihenfolge der Datendefinitionen und
    - Die Destruktoren in der umgekehrten Reihenfolge aufgerufen.
  - Globale Objekte
    - Konstruktor wird zu beginn der Lebensdauer (vor `main()`) aufgerufen
    - Destruktor wird hinter der schließenden Klammer von `main()` aufgerufen

Regeln für die Anwendung für Konstruktoren und Destruktoren (Exkurs)

- Lokale Objekte
  - Konstruktor wird an der Definitionsstelle des Objekts aufgerufen
  - Destruktor wird beim Verlassen des definierenden Blocks aufgerufen
- Dynamische Objekte
  - Konstruktor wird bei `new` aufgerufen
  - Destruktor wird bei `delete` für zugehörigen Zeiger aufgerufen

Regeln für die Anwendung für Konstruktoren und Destruktoren (Exkurs)

- Objekte mit Klassenkomponenten
  - Konstruktor der Komponenten wird vor dem der umfassenden Klasse aufgerufen
  - Am Ende der Lebensdauer werden Destruktoren in umgekehrter Reihenfolge aufgerufen
- Feld von Objekten
  - Konstruktor wird bei Datendefinition für jedes Element beginnend in Index 0 aufgerufen
  - Am Ende der Lebensdauer werden Destruktoren in umgekehrter Reihenfolge aufgerufen

Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Sichtbarkeiten und Zugriffskontrollen

- Grundprinzip der SW-Entwicklung: Information Hiding
- Der Anwender einer Klasse sieht nur das, was er zur Benutzung der Objekte benötigt.
- Elemente, die intern gebaucht werden, werden versteckt.

Sichtbarkeiten und Zugriffskontrollen

- Arten der Zugriffskontrollen
  - `private:`
    - Attribute wegen Information Hiding als `private` deklariert.
    - Zugriff nur innerhalb einer Klasse erlaubt.
    - Scope bis Ende der Klasse oder bis `public` auftaucht.
  - `public:`
    - Zugriff außerhalb der Klasse erlaubt.
    - Methoden als `public`
- Zugriffskontrollen auf Attributen und Methoden
  - z.B. `private: string name;`
  - z.B. `public: int berechnen(int gewicht);`

Getter/Setter Methoden

- Wie kann man trotz allem Zugriff auf private Elemente ermöglichen?
  - Sicherer Zugriff auf Attributen nur über Methoden mit Getter und Setter-Methoden
- Getter sind Methoden, die lesend auf Attribute zugreifen. Typischerweise keine Parameter, einen Rückgabewert und den Präfix „get“. Gegebenenfalls wird der Wert aus dem Zustand des Objektes berechnet (abgeleitetes Attribut).
- Setter sind Methoden, die schreibend auf Attribute zugreifen. Typischerweise keinen Rückgabewert und den Präfix „set“. Sie kontrollieren die Änderung des Wertes.

Getter-Methoden

- Attribute sollen `private` bleiben wegen Information Hiding
- Mittels Getter-Methoden können Attributwerte abgefragt werden.

BMI.h

```
class BMI {
private:
    string newName;
    int newHeight;
public:
    string getName() const;
    int getHeight() const;
}
```

Attribut nach außen nicht sichtbar

main.cpp

```
BMI Student_1;
Student_1.newName=„Max“;
```

Nicht erlaubt!

Getter-Methoden

- Attribute sollen `private` bleiben wegen Information Hiding
- Mittels Getter-Methoden können Attributwerte abgefragt werden.
- Mit `const` damit der Wert wirklich nicht verändert wird.

BMI.h

```
string getName() const;
double setWeight() const;
int setHeight() const;
```

BMI.cpp

```
string BMI::getName() const {
        return newName;
}
int BMI::getHeight() const {
    return newHeight;
}
double BMI::getWeight() const {
    return newWeight;
}
```

Setter-Methoden

- Manipulation von Objekt-Attributen ausschließlich indirekt durch Methoden.
- Mittels Setter-Methoden können Attribute verändert werden.

BMI.h

```
void setName(string name);
   // setName -Name setzen
    // @param string- Name der Person
void setWeight(double weight);
    // setWeight- Gewicht setzen
    // @param double- Gewicht der Person
void setHeight(int height);
    // setHeight- Größe setzen
    // @param int- Größe der Person
```

BMI.cpp

```
void BMI::setName(string name){
     newName = name;
}
void BMI::setHeight(int height){
     newHeight = height;
}
void BMI::setWeight(double weight){
     newWeight = weight;
}
```

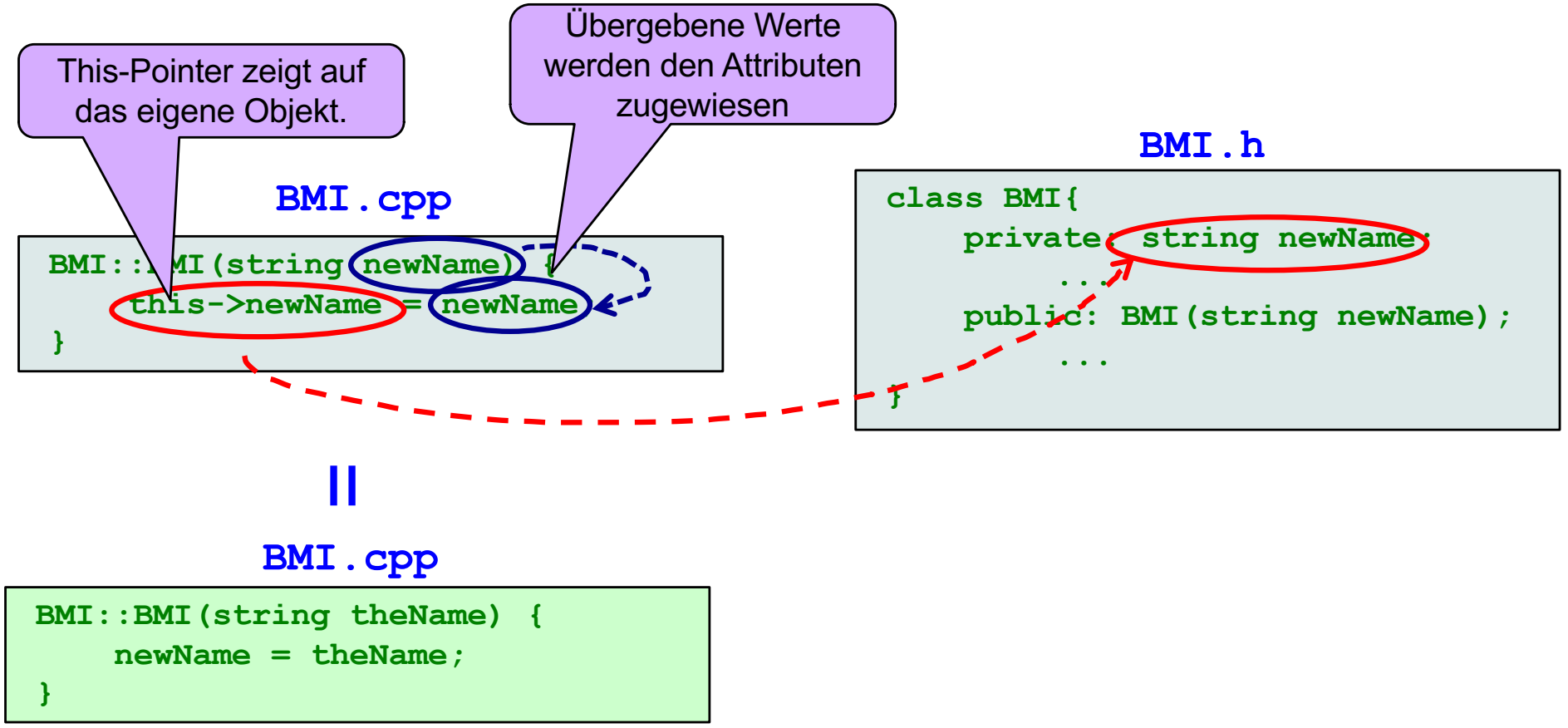
Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

`this`-Zeiger in C++

- `this`-Zeiger zeigt auf die Adresse des eigenen Objekts.
- Mittels des `this`-Zeiger kann zwischen den Attributen des Objektes und den formalen Parametern bei Namensgleichheit unterschieden werden.
- Oft vorzufinden in einem Konstruktor.

this-Zeiger in C++



this-Zeiger in C++

Wie lautet die Ausgabe in der main-Funktion?

BMI.cpp

```
class BMI {
public:
   // print the adress of this object
   void BMI::printThis() const{
       cout<< this <<endl;
   }
```

main.cpp

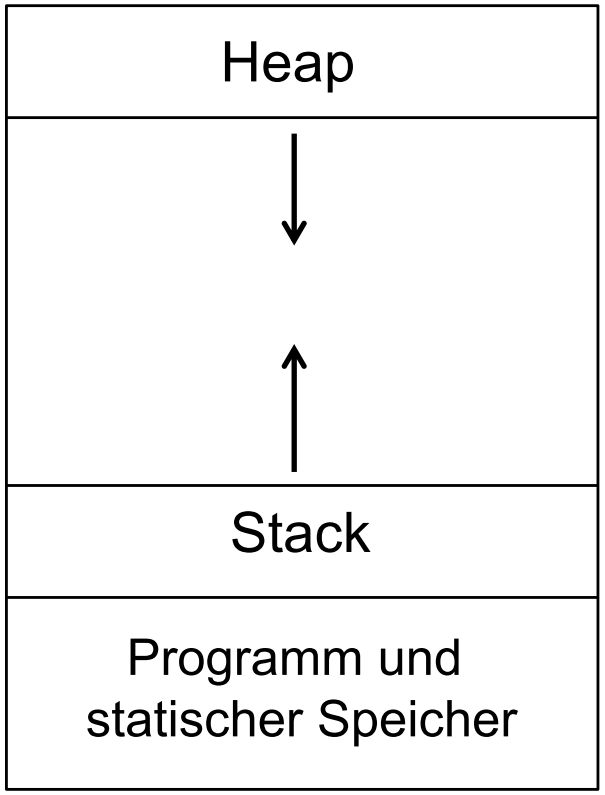
```
BMI pStudent_this;
pStudent_this.printThis();
cout << &pStudent_this << endl;

BMI* pStudent_that = new BMI;
pStudent_that->printThis();
cout << pStudent_that << endl;
```

Inhalt

- Einführung in Objekt-Orientierung
- Einführung in C++
  - Klassen und Objekte
  - Konstruktor und Destruktor
  - Getter und Setter Methoden
  - `This`-Zeiger
  - Dynamische Speicherverwaltung

Speicherverwaltung mit Heap und Stack

- Dynamische Speicherverwaltung mit Heap (Zeiger)
- Statische Speicherverwaltung mit Stack (Variablen, Array)



- Heap wächst nach „unten“
- Stack wächst nach „oben“
- Wenn Heapgrenze auf Stackgrenze trifft, dann „Out of Memory Error“
- Stack bereinig sich selbst, für Heap ist der Programmierer verantwortlich.

Dynamische Speicherverwaltung in C++

- Dynamisches Erzeugen von Objekten zur Laufzeit
  - „**new**“-Operator
  - z.B. `BMI *pStudent = new BMI;`
- Dynamisches Löschen von erzeugten Objekten zur Laufzeit
  - Dynamisch erzeugte Objekte müssen auch wieder gelöscht werden! Keine automatische Speicherbereinigung!
  - „**delete**“-Operator
  - z.B. `delete pStudent;`
- Dynamische Speicherverwaltung mit Heap muss vom Programmierer gemacht werden.

Pfeil-Operator „->“ in C++

- Zeiger auf Objekte von Klassen (Objektzeiger).
- De-Referenzieren und Zugriff auf Elementen des Objekts.
  - Direkt mit Zeigern arbeiten, anstatt auf Objekten.
- Vorteil von Objektzeiger: jederzeit auf eine andere Adresse zeigen können.
- `(*objPtr).x` ist identisch zu `objPtr->x`

main.cpp

```
BMI* pStudent = new BMI;
pStudent->setName("Moritz");
cout << „Student Name: " << pStudent->getName() <<
endl;
```