Konstruktor/Destruktor

Florian Adamsky, B. Sc. (PhD cand.)

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

Softwareentwicklung im WS 2014/15

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Outline

Inhaltsverzeichnis

Was ist der Unterschied zwischen `class` und `struct`?

Example (class)

```
class Rechteck {
    int hoehe;
    int breite;

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Example (struct)

```
struct Rechteck {
    int hoehe;
    int breite;

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Unterschied `class` und `struct`

[7.8] What's the difference between the keywords struct and class?

The members and base classes of a struct are public by default, while in class, they default to private. Note: you should make your base classes explicitly public, private, or protected, rather than relying on the defaults.

Struct and class are otherwise functionally equivalent.

— Quelle: The C++ FAQ

Unterschied `class` und `struct`

[7.8] What's the difference between the keywords struct and class?

The members and base classes of a struct are public by default, while in class, they default to private. Note: you should make your base classes explicitly public, private, or protected, rather than relying on the defaults.

Struct and class are otherwise functionally equivalent.

— Quelle: The C++ FAQ

Unterschied `class` und `struct`

[7.8] What's the difference between the keywords struct and class?

The members and base classes of a struct are public by default, while in class, they default to private. Note: you should make your base classes explicitly public, private, or protected, rather than relying on the defaults.

Struct and class are otherwise functionally equivalent.

— Quelle: The C++ FAQ

Was ist der Unterschied zwischen `class` und `struct`?

Example (class)

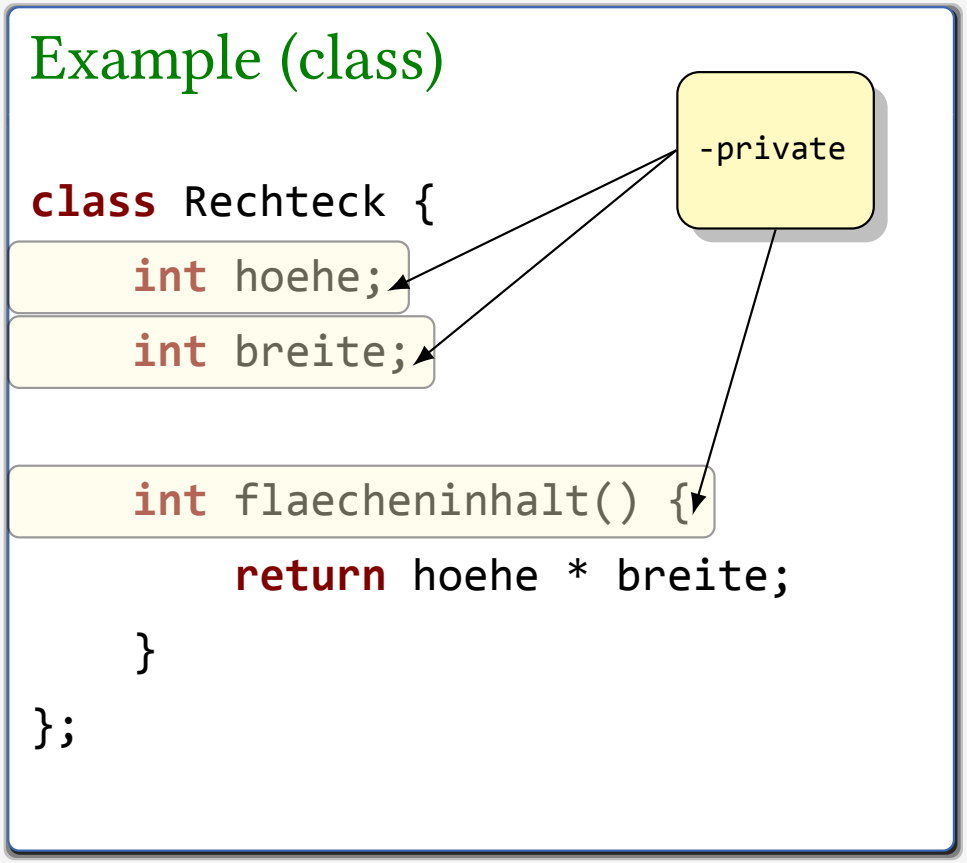
```
class Rechteck {
    int hoehe;
    int breite;

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Example (struct)

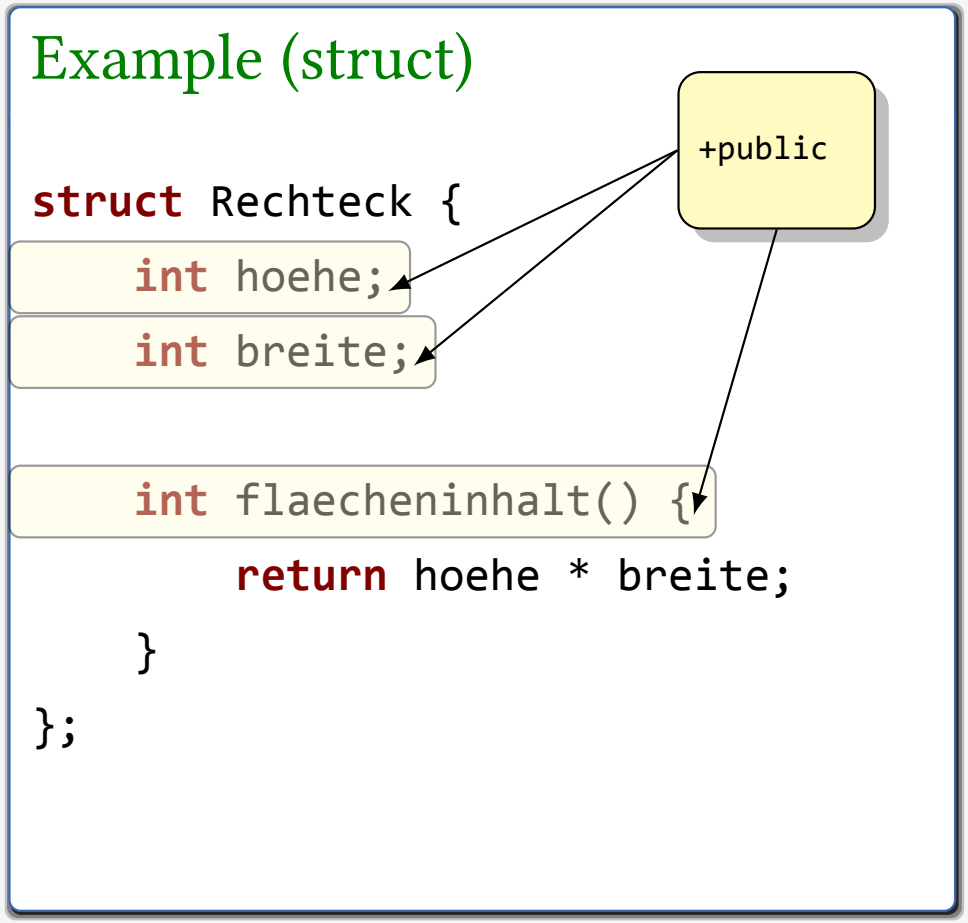
```
struct Rechteck {
    int hoehe;
    int breite;

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Was ist der Unterschied zwischen `class` und `struct`?

Example (class)



Example (struct)



Was ist der Unterschied zwischen `class` und `struct`?

Example (class)

```
class R
    int
    int

    int

    }
};
```

Example (struct)

```
e;

};
```

Tipp

Die Zugriffsschlüsselwörter (`private`, `public` oder `protected`) sollten immer explizit angegeben werden, um Verwechselung zu vermeiden.

This-Pointer

- jedes Objekt hat einen Pointer auf sich selbst: `this` Pointer
- braucht nicht definiert oder initialisiert werden
- verfügbar innerhalb der Klasse

Example (This-Pointer)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    int flaecheninhalt () {
        return this->hoehe * this->breite;
    }
};
```

This-Pointer

Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```

This-Pointer

Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```

Bei identischen Variablennamen müssen die Klassen-Attribute mit dem `this` Pointer verwendet werden

This-Pointer

Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```

Bei identischen Variablennamen müssen die Klassen-Attribute mit dem `this` Pointer verwendet werden

This-Pointer

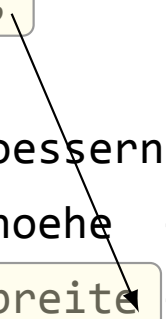
Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe += hoehe;
        this->breite += breite;
    }
};
```

This-Pointer

Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```



This-Pointer

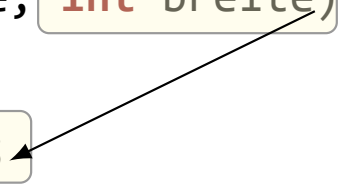
Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```

This-Pointer

Example (Rechteck)

```
class Rechteck {
public:
    int hoehe;
    int breite;

    void vergroessern(int hoehe, int breite) {
        this->hoehe  += hoehe;
        this->breite += breite;
    }
};
```



`const` Schlüsselwort

- `const` verbietet es eine Variable nach ihrer Initialisierung zu ändern
- es war angedacht dieses Schlüsselwort zuerst `readonly` zu benennen
- `const` Variablen müssen initialisiert werden

Example (Beispiel)

```
int x = 4;
x = 10;      // Änderung kein Problem

const int x = 4;
x = 10;      // error: assignment of read-only variable x
```

const Pointer

- Tipp: von rechts nach links lesen
- veränderbarer Pointer der auf eine konstante Variable zeigt

```
const int* ptrX;
ptrX  = &x;      // funktioniert, wegen veränderbaren Pointer
*ptrX = 10;      // FEHLER!!!
```

const Pointer

- Tipp: von rechts nach links lesen
- veränderbarer Pointer der auf eine konstante Variable zeigt

```
const int* ptrX;
ptrX  = &x;      // funktioniert, wegen veränderbaren Pointer
*ptrX = 10;      // FEHLER!!!
```

- konstanten Pointer der auf eine veränderbare Variable zeigt

```
int* const ptrX;
ptrX  = &x;      // FEHLER!!!
*ptrX = 10       // funktioniert, wegen veränderbarer Variable
```

const Pointer

- Tipp: von rechts nach links lesen
- veränderbarer Pointer der auf eine konstante Variable zeigt

```
const int* ptrX;
ptrX  = &x;      // funktioniert, wegen veränderbaren Pointer
*ptrX = 10;      // FEHLER!!!
```

- konstanten Pointer der auf eine veränderbare Variable zeigt

```
int* const ptrX;
ptrX  = &x;      // FEHLER!!!
*ptrX = 10       // funktioniert, wegen veränderbarer Variable
```

- konstanten Pointer der auf eine konstante Variable zeigt

```
const int* const ptrX;
ptrX  = &x;      // FEHLER!!!
*ptrX = 10       // FEHLER!!!
```

const Referenzen

- Referenz die auf konstante Variable zeigt

```
const int& refX;
```

const Referenzen

- Referenz die auf konstante Variable zeigt

```
const int& refX;
```

- Preisfrage: Was macht die folgende Zeile

```
int& const refX;
```

const Referenzen

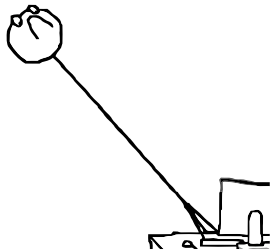
- Referenz die auf konstante Variable zeigt

```
const int& refX;
```

- Pre

```
int
```

Syntax Fehler



Error: ‘const’ qualifiers cannot be applied to ‘int&’

const Referenzen

- Referenz die auf konstante Variable zeigt

```
const int& refX;
```

- Preisfrage: Was macht die folgende Zeile

```
int& const refX;
```

- Antwort: Syntax Fehler
- eine Referenz ist von sich aus const, daher ist ein const unsinnig

Was bedeutet "const correctness"?

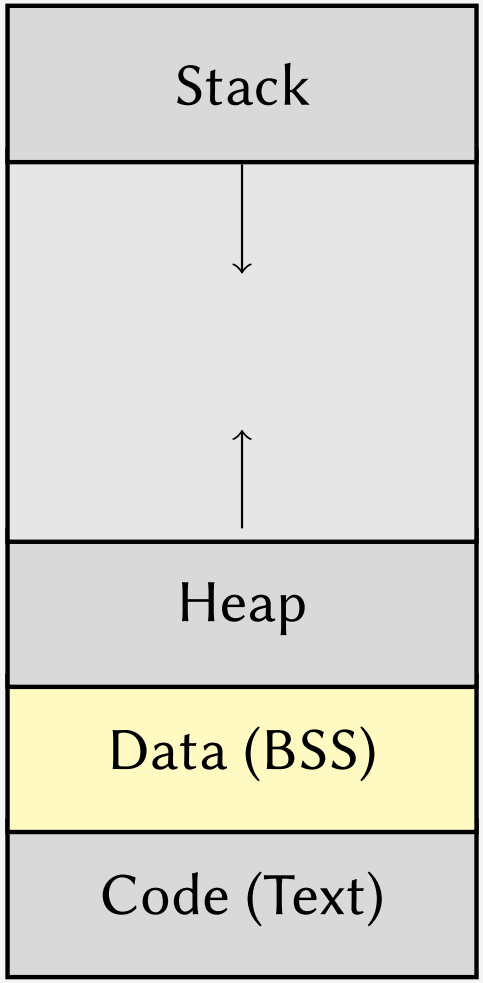
- erfahrene C++ Programmierer verwenden const überall dort wo Variablen nicht verändert werden

```
void m1(const string& refStr); // Pass by reference-to-const
void m2(string* const ptrStr); // Pass by pointer-to-const
void m3(string str);           // Pass by value
```

- ein versehentliches Änderung des Parameters führt bei m1 und m2 zu einem Fehler zur Kompilezeit

statische Variablen mit `static`

- statische Variablen speichert der Compiler in dem Data-Segment
- in C++ werden statische Variablen mit dem Schlüsselwort `static` gekennzeichent
- Lebensdauer endet bis das Programm beendet ist
- Initialisierung wird nur einmal aufgerufen



normale Variablen in einer Methode

Example (Klasse)

```
class Klasse {
public:
    int zaehler;

    Klasse() : zaehler(0) {}

    void methode() {
       zaehler++;
       cout << zaehler << endl;
    }
};
```

normale Variablen in einer Methode

Example (Klasse)

```
class Klasse {
public:
    int zaehler;

    Klasse() : zaehler(0) {}

    void methode() {
       zaehler++;
       cout << zaehler << endl;
    }
};
```

Example (Main)

```
int main(void) {
    Klasse a;
    a.methode();
    a.methode();
    a.methode();

    Klasse b;
    b.methode();
    b.methode();
}
```

normale Variablen in einer Methode

Example (Klasse)

```
class K
public:
    int

    Kla

    voi

    }
};
```

Example (Main)

```
}
```

Ausgabe

```
$ ./programm
1
2
3

1
2
```

statische Variablen in einer Methode

Example (Klasse)

```
class Klasse {
public:
    static int zaehler;

    void methode() {
        zaehler++;
        cout << zaehler << endl;
    }
};

int Klasse::zaehler = 0;
```

statische Variablen in einer Methode

Example (Klasse)

```
class Klasse {
public:
    static int zaehler;

    void methode() {
       zaehler++;
       cout << zaehler << endl;
    }
};

int Klasse::zaehler = 0;
```

Example (Main)

```
int main(void) {
    Klasse a;
    a.methode();
    a.methode();
    a.methode();

    Klasse b;
    b.methode();
    b.methode();
}
```

statische Variablen in einer Methode

Example (Klasse)

```
class K
public:
    sta

    voi

    }
};

int Klasse::zaehler = 0;
```

Example (Main)

```
}
```

Ausgabe

```
$ ./programm
1
2
3

4
5
```

Inhaltsverzeichnis

Initalisierungs-Problem

- bisherige Klassen hatten keine Initialisierung, nur eine Definition

```
Rechteck* quadrat = new Rechteck();

quadrat->hoehe  = 23;
quadrat->breite = 23;
```

- wie kann man `hoehe` und `breite` initialisieren wenn diese `private` sind?

Was ist ein Konstruktor?

- eine Methode die den selben Namen trägt wie die Klasse
- wird automatisch aufgerufen wenn ein Objekt erstellt wird
- Besonderheit: hat keinen Rückgabewert
  - gibt das Objekt selber zurück

Beispiel eines Konstruktors

Example (Klasse)

```
class Rechteck {
private:
    int hoehe;
    int breite;

public:
    Rechteck(int h, int b) {
        hoehe  = h;
        breite = b;
    }

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Beispiel eines Konstruktors

Example (Klasse)

```
class Rechteck {
private:
    int hoehe;
    int breite;

public:
    Rechteck(int h, int b) {
        hoehe  = h;
        breite = b;
    }

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Example (Objekt)

```
int main(void) {
  // auf dem Heap
  Rechteck* q = new Reckteck(10, 40);
  cout << q->flaecheninhalt();

  // auf dem Stack
  Rechteck f(10, 40);
  cout << f.flaecheninhalt() << endl;
}
```

Default-Konstruktor

- ein Konstruktor der keine Argumente benötigt wird *Default Konstruktor* genannt
- hauptsächlich zur Initialisierung von Attributen benötigt:

Example (Default-Konstruktor)

```
class Zaehler {
private:
    int zaehler;
    int array[10];

public:
    Zaehler() {
        zaehler = 0;
        array   = {0};
    }
};
```

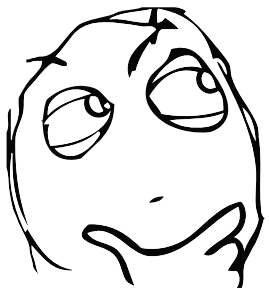
Default-Konstruktor

- ein Konstruktor der keine Argumente benötigt wird *Default Konstruktor* genannt
- hauptsächlich zur Initialisierung von Attributen benötigt:

Exampl

```
class Za
private:
    int
    int

public:
    Zaehler() {
        zaehler = 0;
        array   = {0};
    }
};
```

Mhhh



Was ist eigentlich mit Referenzen oder `const` Variablen die sofort initialisiert werden müssen?

Initialization-Lists

Example (Syntax)

```
Konstruktor(T varname) : varname(value) {}
```

- Erforderlich bei:
  - `const` Variablen
  - Referenzen
  - Aufruf des Konstruktors der Elternklasse
- Optional bei:
  - allen anderen Variablen

Beispiel der Initialization-List

Example (Klasse)

```
class Rechteck {
private:
    const int hoehe;
    int& breite;

public:
    Rechteck(int h, int& b) : hoehe(h), breite(b) {
        cout << "Konstruktor" << endl;
    }

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```

Beispiel der Initialization-List

Example (Klasse)

```
class Rechteck {
private:
    const int hoehe;
    int& breite;

public:
    Rechteck(int h, int& b) : hoehe(h), breite(b) {
        cout << "Konstruktor" << endl;
    }

    int flaecheninhalt() {
        return hoehe * breite;
    }
};
```