Klassenspezifische Daten und Funktionen

- Klassenspezifische Daten
- Klassenspezifische Funktionen
- Klassenspezifische Konstanten
- Beispiel: Die Klasse `NumeriertesObjekt`

Klassenspezifische Daten und Funktionen

Klassenspezifische Daten

Klassenspezifische Daten sind Daten, die **nur einmal** für **alle** Objekte einer Klasse existieren.

Sie sind nicht an einzelne Objekte, sondern an alle Objekte einer Klasse gleichzeitig gebunden.

Beispiele:

- Verwaltungsdaten, wie die Anzahl der Objekte
- Bezugsdaten, die für alle Objekte gelten, wie ein gemeinsamer Koordinatenursprung für graphische Objekte

Klassenspezifische Daten sind innerhalb einer Klasse und nur für Objekte dieser Klasse zugreifbar.

Sie werden als `static` deklariert.

Unabhängig von der Zahl der Objekte einer Klasse wird pro `static`-Datenelement nur **ein** Speicherplatz angelegt, auf den **alle** Objekte dieser Klasse zugreifen können.

Klassenspezifische Daten

Beispiel:

```
class Test
{
  // Deklaration der klassenspez. Variablen
  public:
    static int public_int;
  private:
    static int private_int;
};

// Definition der klassenspez. Variablen

int Test::public_int = 0;
int Test::private_int = 10;

int main()
{
  Test::public_int = 145;     // ok
  Test::private_int = 12;     // falsch

  return 0;
}
```

In der Klasse `Test` werden die klassenspezifischen Variablen lediglich **deklariert**.

Definition und **Initialisierung** müssen außerhalb der Klassendefinition erfolgen.

Klassenspezifische Daten

Beispiel:

```
class Directory
{
  public:
    // Konstruktoren, Destruktor etc.
    // ...
  private:
    // Datenelemente
    static char path[];
};
```

Klassenspezifische Variablen werden von Konstruktoren nicht **initialisiert**, sondern höchstens **modifiziert**.

Die Initialisierung kann nicht in einen Konstruktor verlegt werden, weil sie sonst bei jeder Erzeugung eines Objekts durchgeführt würde.

Die Initialisierung findet außerhalb der Klasse statt.
Sie ist gleichzeitig auch die Definition der Variablen, die nur genau einmal im Programm vorhanden sein darf.

```
// Definition und Initialisierung
// der Klassenvariablen
char Directory::path[200] = "/usr/local";
```

Klassenspezifische Daten und Funktionen

Klassenspezifische Funktionen

Klassenspezifische Funktionen führen Aufgaben aus, die **an eine Klasse, nicht aber an ein Objekt gebunden sind**.

Sie können z.B. mit den `static`-Daten arbeiten.

Auf nicht-statische Daten können klassenspezifische Funktionen nicht zugreifen. Sie dürfen auch keine nicht-statischen Funktionen aufrufen.

Auch Konstruktoren sind klassenspezifische Funktionen, weil das zu konstruierende Objekt beim Aufruf noch nicht existiert.

`static`-Funktionen (und `static`-Daten) können sowohl klassenbezogen als auch objektgebunden aufgerufen (bzw. benannt) werden.

Klassenspezifische Funktionen

Beispiel:

```
class Directory
{
  public:
    // ...
    static void setpath(char const *newpath);

  private:
    static char path [];
};

char Directory::path [200] = "/usr/local";

void Directory::setpath(char const *newpath)
{
  strncpy(path, newpath, 200);
}

int main()
{
  // guter Stil: klassenbezogener Aufruf
  Directory::setpath("/etc");

  Directory dir;

  // schlechter Stil: objektgebundener Aufruf
  dir.setpath("/etc");

  return 0;
}
```

Klassenspezifische Daten und Funktionen

Klassenspezifische Konstanten

Für klassenspezifische Konstanten muss der Compiler keinen Platz anlegen, weil er direkt ihren Wert einsetzen kann.

Die Konstanten werden **innerhalb** der Klassendefinition initialisiert.

Beispiel:

```
class KlasseMitKonstanten
{
  enum RGB {
    rot  = 0x0001,
    gelb = 0x0002,
    blau = 0x0004
  };

  static const unsigned int
    maximaleZahl = 1000;

  // Verwendung z.B.:
  static int CArray[maximaleZahl];
  // ...

};
```

Klassenspezifische Daten und Funktionen

Beispiel: Die Klasse `NumeriertesObjekt`

```
// numobj.h

#ifndef NUMOBJ_H
#define NUMOBJ_H NUMOBJ_H

class NumeriertesObjekt
{
  public:
    NumeriertesObjekt();
    NumeriertesObjekt(const NumeriertesObjekt&);
    ~NumeriertesObjekt();

    NumeriertesObjekt& operator=
      (const NumeriertesObjekt&);
    unsigned long Seriennummer() const
      { return SerienNr; }

    static int Anzahl() { return anzahl; }
    static bool Testmodus;

  private:
    static int anzahl;
    // int statt unsigned (siehe Text)
    static unsigned long maxNummer;
    const unsigned long SerienNr;
};

#endif  // NUMOBJ_H
```

Klassenspezifische Daten und Funktionen

```
// Implementierung der Klasse NumeriertesObjekt
// numobj.cpp

#include "numobj.h"
#include <iostream>
#include <cassert>
using namespace std;

// Initialisierung und Definition der
// klassenspezifischen Variablen:

int
  NumeriertesObjekt::anzahl = 0;
unsigned long
  NumeriertesObjekt::maxNummer = 0L;
bool
  NumeriertesObjekt::Testmodus = false;
```

Zur Initialisierung der `static`-Attribute genügt die Angabe von Typ, Klasse und Variablenname.

Die Initialisierung ist gleichzeitig die Definition der Variablen, die nur genau einmal im Programm vorhanden sein darf.

Klassenspezifische Daten und Funktionen

```
// Standardkonstruktor

NumeriertesObjekt::NumeriertesObjekt()
  :  SerienNr(++maxNummer)
{
  anzahl++;
  if (Testmodus)
  {
    if (SerienNr == 1)
      cout << "Start der Objekterzeugung!\n";

    cout << "  Objekt Nr. " << SerienNr
         << " erzeugt" << endl;
  }
}
```

Die Initialisierung der `static`-Variablen kann nicht in einen Konstruktor verlegt werden, weil sie sonst bei jeder Erzeugung eines Objekts durchgeführt würde.

Klassenspezifische Daten und Funktionen

```
// Kopierkonstruktor

NumeriertesObjekt::NumeriertesObjekt
  (const NumeriertesObjekt &X)
  :  SerienNr(++maxNummer)
{
  anzahl++;

  if (Testmodus)
    cout << "  Objekt Nr. " << SerienNr
         << " mit Nr. "  << X.Seriennummer()
         << " initialisiert" << endl;
}
```

Bisher diente der Kopierkonstruktor dazu, bei der Initialisierung eines neuen Objekts ein Duplikat eines anderen Objekts seiner Klasse zu erzeugen.

Das darf hier nicht sein!

Das neu erzeugte Objekt soll nicht die Seriennummer eines anderen Objekts erhalten, sondern muss eine neue bekommen.

Der Kopierkonstruktor hat dasselbe Verhalten wie der Standardkonstruktor (mit Ausnahme des Testmodus).

Klassenspezifische Daten und Funktionen

Was geschieht bei der Zuweisung eines Objekts?

```
NumeriertesObjekt NumObjekt1, NumObjekt2;
// ...
NumObjekt2 = NumObjekt1;      // ???
```

Der vom System erzeugte Zuweisungsoperator würde eine Zuweisung der Elemente von `NumObjekt1` an `NumObjekt2` bewirken.

Das einzige Element, das hier in Frage kommt, ist die private Konstante `SerienNr`. Einer Konstanten kann aber nichts zugewiesen werden, so dass der Compiler keinen Zuweisungsoperator erzeugen kann.

Es wird also ein überladener Zuweisungsoperator benötigt, der **nichts** tun darf:

```
// überladene Wertzuweisung

NumeriertesObjekt& NumeriertesObjekt::operator=
  (const NumeriertesObjekt&)
{
  return *this;
}
```

Klassenspezifische Daten und Funktionen

```
// Destruktor

NumeriertesObjekt::~NumeriertesObjekt()
{
  anzahl--;
  if (Testmodus)
  {
    cout << "  Objekt Nr. "
         << SerienNr << " gelöscht" << endl;

    if (anzahl == 0)
      cout << "letztes Objekt gelöscht!" << endl;

    if (anzahl < 0)
      cout << "FEHLER! zu oft delete aufgerufen!"
           << endl;
  }
  else
    assert(anzahl >= 0);
}

// Ende von numobj.cpp
```

Ein versehentliches zusätzliches `delete` kann vom Destruktor festgestellt werden, wenn nämlich `anzahl` negativ wird.

Daher ist `anzahl` vom Typ `int` und nicht `unsigned int`.

Klassenspezifische Daten und Funktionen

```
// Demonstration von numerierten Objekten: nummain.cpp
#include "numobj.h"
#include <iostream>
using namespace std;

int main()
{
  // Testmodus für alle Objekte der Klasse einschalten
  NumeriertesObjekt::Testmodus = true;

  NumeriertesObjekt dasNumObjekt_X;       //...wird erzeugt
  cout << "Die Seriennummer von dasNumObjekt_X ist: "
       << dasNumObjekt_X.Seriennummer() << endl;

  // Anfang eines neuen Blocks
  {
    NumeriertesObjekt dasNumObjekt_Y;  //...wird erzeugt

    // schlechter Stil: objektgebundener Aufruf:
    cout << dasNumObjekt_Y.Anzahl()
         << " Objekte aktiv" << endl;

    // *p wird dynamisch erzeugt:
    NumeriertesObjekt *p = new NumeriertesObjekt;

    // schlechter Stil:objektgebundener Aufruf über Zeiger:
    cout << p->Anzahl()
         << " Objekte aktiv" << endl;
    delete p;    // *p wird gelöscht

    // besser: klassenbezogener Aufruf:
    cout << NumeriertesObjekt::Anzahl()
         << " Objekte aktiv" << endl;

    delete p;    // Fehler: ein delete zuviel!
  }              // Blockende: dasNumObjekt_Y wird gelöscht
```

Klassenspezifische Daten und Funktionen

```
// Demonstration von numerierten Objekten:
// nummain.cpp (Fortsetzung)

  cout << " Kopierkonstruktor: " << endl;
  NumeriertesObjekt dasNumObjekt_X1 = dasNumObjekt_X;

  cout << "Die Seriennummer von dasNumObjekt_X ist: "
       << dasNumObjekt_X.Seriennummer() << endl;

  cout << "Die Seriennummer von dasNumObjekt_X1 ist: "
       << dasNumObjekt_X1.Seriennummer() << endl;

  // Ohne den überladenen Zuweisungsoperator wird
  // die folgende Anweisung wegen const SerienNr
  // vom Compiler verboten.
  dasNumObjekt_X1 = dasNumObjekt_X;

}               // dasNumObjekt_X wird gelöscht
```

Klassenspezifische Daten und Funktionen

Ausgabe des Programms:

```
Start der Objekterzeugung!
  Objekt Nr. 1 erzeugt
Die Seriennummer von dasNumObjekt_X ist: 1
  Objekt Nr. 2 erzeugt
2 Objekte aktiv
  Objekt Nr. 3 erzeugt
3 Objekte aktiv
  Objekt Nr. 3 gelöscht
2 Objekte aktiv
  Objekt Nr. 0 gelöscht
  Objekt Nr. 2 gelöscht
letztes Objekt gelöscht!
 Kopierkonstruktor:
  Objekt Nr. 4 mit Nr. 1 initialisiert
Die Seriennummer von dasNumObjekt_X ist: 1
Die Seriennummer von dasNumObjekt_X1 ist: 4
  Objekt Nr. 4 gelöscht
letztes Objekt gelöscht!
  Objekt Nr. 1 gelöscht
FEHLER! zu oft delete aufgerufen!
```

Klassenspezifische Daten und Funktionen

Ausgabe des Programms, wenn die Zeile

```
delete p;    // Fehler: ein delete zuviel!
```

in `main()` auskommentiert oder gelöscht wird:

```
Start der Objekterzeugung!
  Objekt Nr. 1 erzeugt
Die Seriennummer von dasNumObjekt_X ist: 1
  Objekt Nr. 2 erzeugt
2 Objekte aktiv
  Objekt Nr. 3 erzeugt
3 Objekte aktiv
  Objekt Nr. 3 gelöscht
2 Objekte aktiv
  Objekt Nr. 2 gelöscht
 Kopierkonstruktor:
  Objekt Nr. 4 mit Nr. 1 initialisiert
Die Seriennummer von dasNumObjekt_X ist: 1
Die Seriennummer von dasNumObjekt_X1 ist: 4
  Objekt Nr. 4 gelöscht
  Objekt Nr. 1 gelöscht
letztes Objekt gelöscht!
```