Hydroinformatik - SoSe 2024
HyBHW-S1-01-V5: Klassen

Olaf Kolditz

[1]Helmholtz Centre for Environmental Research – UFZ, Leipzig
[2]Technische Universität Dresden – TUD, Dresden
[3]Center for Advanced Water Research – CAWR
[4]TUBAF-UFZ Center for Environmental Geosciences – C-EGS, Freiberg / Leipzig

Dresden, 10.05.2024

https://www.ufz.de/index.php?de=40416
https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/32518209537?10

Fahrplan für heute ...

1. Rückblick letzte Veranstaltung (HyBHW-S1-01-V4: Datentypen)

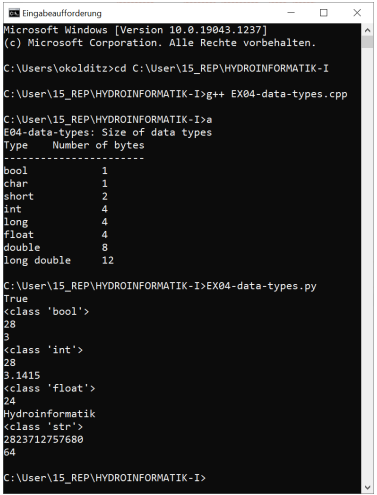
---

2. Objekt-Orientierung: Konzept
3. OpenGeoSys - ein objekt-orientiertes Programm
4. Ihre Fragen
5. Objekt-Orientierte Programmierung (OOP): Daten-Abstraktion, Klassen, Instanzen, ...
6. Übung: Instanzen einer Klasse

---

7. Ausblick auf die nächste Veranstaltung (HyBHW-S1-01-V6: Input/Output (I/O))

Datentypen: EX04-data-types



- `cmd`: Consolenanwendung starten
- `cd`: ins Verzeichnis mit den Übungen wechseln
- `git`: Aktualisieren der Übungen aus dem Repo
  - `git fetch --all`: Infos zu Änderungen
  - `git pull`: Änderungen aktualisieren
- `g++ file.cpp`: C++ compilieren
- `a.exe`: Programm ausführen
- `file.py`: Python-Programm ausführen

Voraussetzungen (Software-Installationen):

- `git`: Versionskontrolle
- `MinGW`: Compiler
- `Python`: Python

Datentypen: EX04-data-types.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
  cout << "E04-data-types: Size of data types\n";
//  system("pwd");
  cout << "Type\tNumber of bytes\n";
  cout << "-----------------------\n";
  cout << "bool\t\t" << sizeof(bool) << endl;
  cout << "char\t\t" << sizeof(char) << "\n";
  cout << "short\t\t" << sizeof(short) << endl;
  cout << "int\t\t" << sizeof(int) << endl;
  cout << "long\t\t" << sizeof(long) << endl;
  cout << "float\t\t" << sizeof(float) << endl;
  cout << "double\t\t" << sizeof(double) << endl;
  cout << "long double\t" << sizeof(long double) << endl;
  return 0;
}
```

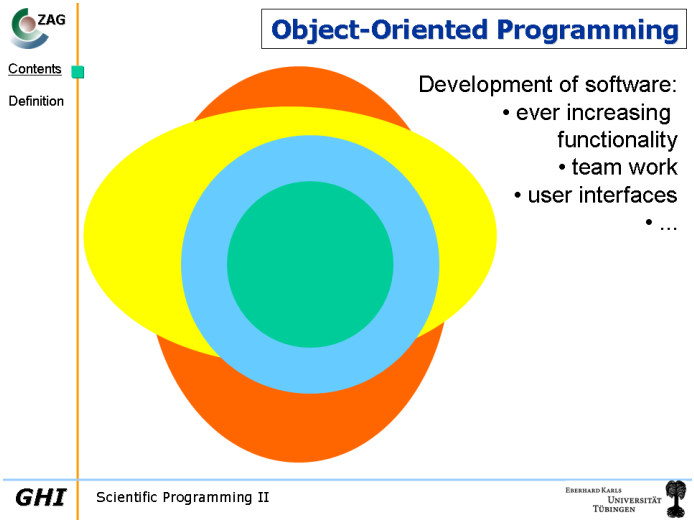
Listing: Exercise: Data types

Datentypen: EX04-data-types.py (new)

```
1  #boolean
2  d = True
3  print(d)
4  type(d)
5  print(type(d))
6  print(d .__sizeof__())
7  #integer
8  a = 3
9  print(a)
10 type(a)
11 print(type(a))
12 print(a .__sizeof__())
13 #float
14 b = 3.1415
15 print(b)
16 type(b)
17 print(type(b))
18 print(b .__sizeof__())
19 #string
20 c = 'Hydroinformatik'
21 print(c)
22 type(c)
23 print(type(c))
24 id(c)
25 print(id(c))
26 print(c .__sizeof__())
```

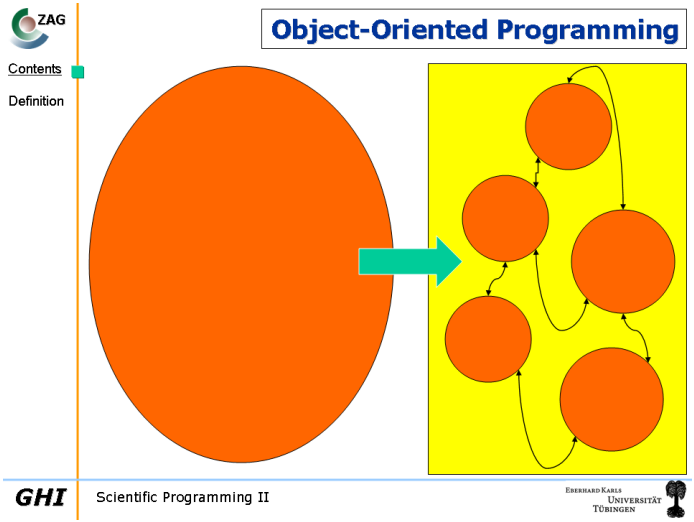
Listing: Exercise: Data types

Objekt-Orientierung



- Prozedurales Programmieren
- ...

Objekt-Orientierung



Objekt-Orientierung - Daten- und "Echte" Typen

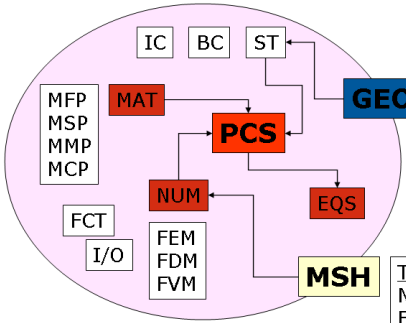
Objekt-Orientierung - Typen

Links:

https://www.youtube.com/watch?v=JBjjnqG0BP8
http://www.stroustrup.com/
https://en.wikipedia.org/wiki/Bjarne_Stroustrup

Objekt-Orientierung - OpenGeoSys (OGS)

V4: Object-Orientation:Multifield Problems



Geometry
Points
Polylines
Surfaces
Volumes
Domains

Topology
NOD Nodes
ELE Meshes

$Ax = b$
$x = A^{-1}b$

CProcess::Create()
CProcess::Config()
CProcess::Execute()

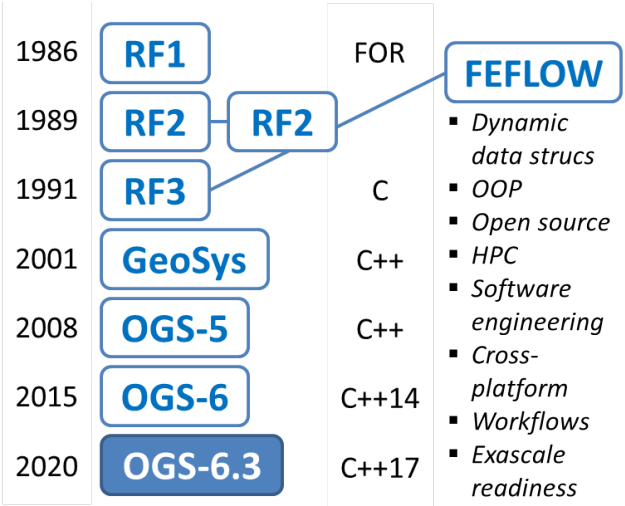
Kolditz & Bauer (2004)
J Hydroinformatics

Source code reduction: 10 (V3) -> 3 MB (V4)

HELMHOLTZ CENTRE FOR ENVIRONMENTAL RESEARCH - UFZ

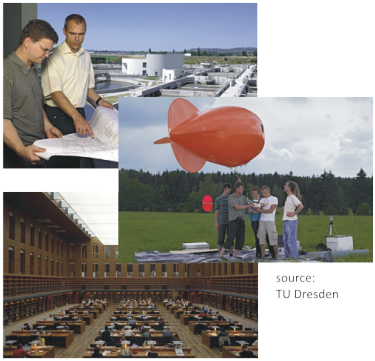
Page 14

Objekt-Orientierung - OGS - History



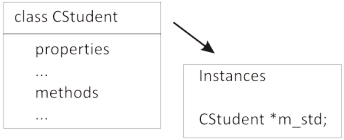
Klassen

Das Sprachelement der Klassen sind das entscheidende Kriterium von objekt-orientierten Konzepten und die objekt-orientierte Programmierung (OOP). Klassen sind eine Art Schablone für einen benutzerdefinierten Datentypen. Darüber hinaus enthält die Klasse neben den Daten auch alle Methoden (Funktionen), um mit den Daten der Klasse operieren zu können. Unser Beispiel für Klassen, das uns im Verlaufe der Vorlesung beschäftigen wird, ist - wie könnte es anders sein - `CStudent` (Abbildung). Für die Konzipierung von Klassen spielt die Abstraktion der Daten einer Klasse eine besonders wichtige Rolle.



source:
TU Dresden

Data abstraction



Daten-Abstraktion

Die Abbildung illustriert uns, dass eine Abstraktion von Daten (d.h. Eigenschaften) der Klasse Studenten eine durchaus vielschichtige Angelegenheit sein kann. Eine Aufstellung von Daten / Eigenschaften, die es aus ihrer Sicht zu berücksichtigen gilt, ist ihre nächste Hausaufgabe.

C vs. C++

Der nächste Block zeigt ihnen das Schema der Syntax der Klassen-Definition CStudent. Das Schlüsselwort für die Klassen-Definition ist class, der Name ist CStudent. Der Klassen-Rumpf ist in geschweifte Klammer eingebettet. Wichtig ist der Abschluss mit einem Semikolon. Wie bereits erwähnt, eine Klasse enthält Daten (Eigenschaften) und Methoden (Funktionen) auf den Daten. Prinzipiell geht diese Datenabstraktion auch mit anderen Sprachen wie C.

```c
typedef struct
{
  char* name_first;
  char* name_last;
  long matrikel_number;
} TDStudent;
TDStudent *student = NULL;
```

Listing: C - Type definition

```cpp
class CStudent
{
  data:
  ...
  methods:
  ...
};
```

Listing: C++ - Class definition

Daten-Abstraktion

Ein weiterer Vorzug von OO-Sprachen ist z.B. die Sichtbarkeit / Zugreifbarkeit von Daten zu regeln. Der nachfolgende Block zeigt das Datenschutz-Konzept von C++ (Sicherheitsstufen): Daten können öffentlich sein (public) oder gezielt für 'Freunde' verfügbar gemacht werden (protected) oder nur exklusiv für die eigene Klasse sichtbar zu sein (private).

```
1 class CStudent
2 {
3   private:
4   ...
5   protected:
6   ...
7   public:
8   ...
9 };
```

Listing: C++ - Data protection

Klassen-Deklaration

Im vorangegangenen Abschnitt haben wir uns mit der Datenabstraktion mittels Klassen beschäftigt. So sollte konsequenterweise jede Klasse auch ihre eigenen - sorry - eigenen Quelldateien besitzen. Die Deklaration von Klassen erfolgt üblicherweise in einer sogenannten Header-Datei *.h. Für die Methoden / Funktionen der Klasse ist eine *.cpp Datei reserviert. Für uns bedeutet dies, zwei Dateien anlegen:

- student.h - die Deklaration der Klasse CStudent
- student.cpp - die Methoden der Klasse CStudent

Klassen-Deklaration

Um mit der Klasse arbeiten zu können, müssen wir das entsprechende Header-File inkludieren. Dies erfolgt mit der Anweisung #include "student.h" am Anfang unseres Main-Files.

```
1 #include "student.h"
2 int main
3 {
4   return 0;
5 }
```

Listing: C++ - Declararion

Instanzen einer Klasse

An dieser Stelle möchten wir unsere Eingangsgraphik erinnern. Instanzen sind Kopien einer Klasse mit denen wir arbeiten können, dass heißt diese bekommen echten Speicher für ihre Daten (die natürlich für jede Instanz einer Klasse unterschiedlich sein können).

Instanzen einer Klasse

Es gibt zwei Möglichkeiten, Instanzen einer Klasse zu erzeugen:

```
1 #include "student.h"
2 void main()
3 {
4   // Creating an instances of a class - 1
5   CStudent m_std_A;
6   // Creating an instances of a class - 2
7   CStudent *m_std_B;
8 }
```

Listing: Creating instances

Instanzen einer Klasse

Der direkte und der mittels eines sogenannten Zeigers (hierfür gibt ein Extra-Kapitel). Wir werden sehen, dass der zweite Weg oft der bessere ist, da wir z.B. die Initialisierung und das Speichermanagement für unsere Daten selber in die Hand nehmen können. Dies können wir mittels sogenannter Konstruktoren und Destruktoren erledigen. Damit beschäftigen wir uns im nächsten Abschnitt.

Instanzen einer Klasse: EX05a

```cpp
#include "student.h"
#include <iostream>
using namespace std;
int main()
{
  CStudent *m_std_cpp; // pointer to an instance
  cout << "E41: Instances of classes" << endl;
  cout << "What have we created?\t : " << m_std << endl;
  cout << "What size has it?\t : " << sizeof(m_std) << endl;
  TDStudent *m_std_c; // pointer to TD
  return 0;
}
```

Listing: Exercise: Creating instances

Übungen: Übersicht

- EX05a: Instanzen einer Klasse - Kreieren
- EX05b: Instanzen einer Klasse - Speicher
- EX05c: Eigenschaften einer Klasse
- EX05d: Weitere Eigenschaften einer Klasse
- EX05e: Aktion "Gummibärchen" (Datenbank häcken) ...