

Programmieren (T3INF1004)

DHBW Stuttgart

Christian Alexander Holz

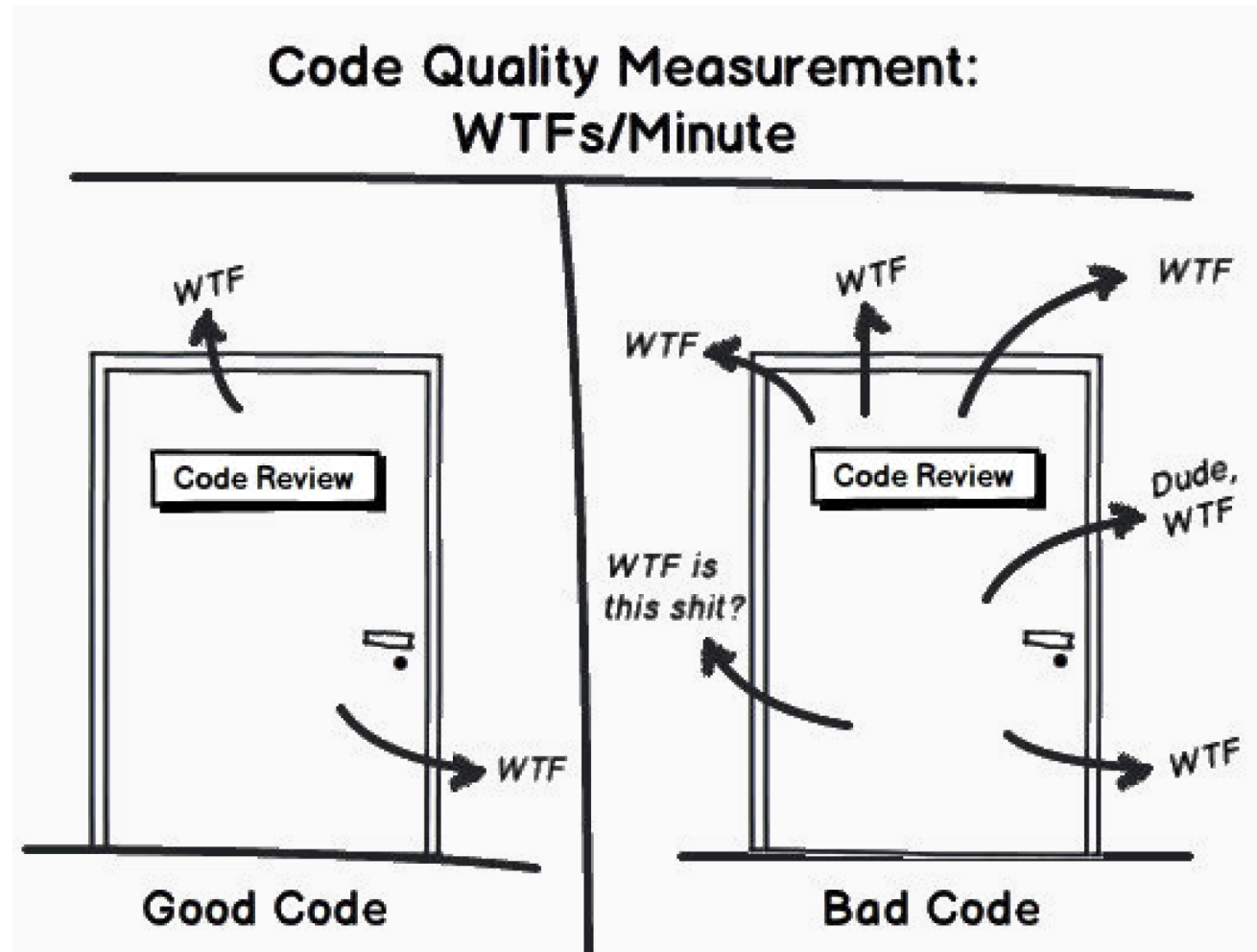
christian.holz@lehre.dhbw-stuttgart.de



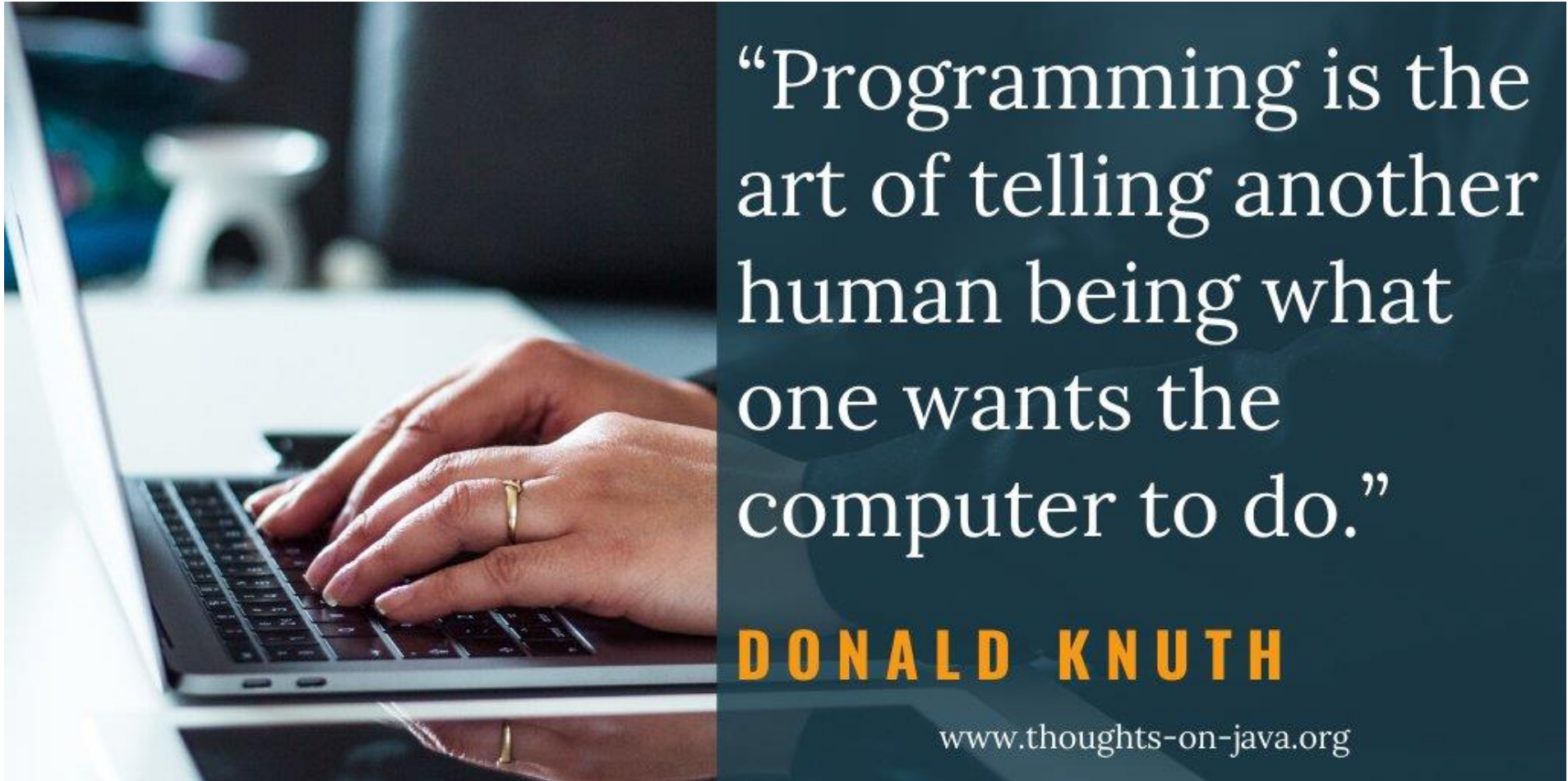
Kapitelübersicht

- 1 Einführung
- 2 Objektorientierung
- 3 Vertiefung C++
- 4 Die Standard Template Library (STL)
- 5 Clean Code
- 6 Test-driven Development

Die einzig echte Messgröße für guten Code...



Schlechte Code Qualität fährt Projekte gegen die Wand!



Kapitel 1: Einführung

Kapitel 1: Einleitung



11. „Hello C++ World!“
und Set-up

12. Wiederholung C

Was ist C++?

- 1979 von Bjarne Stroustrup entwickelt (Dänemark)
- **Stärke von C++:**
 - Effizientes, schnelles maschinennahes Programmieren
 - Optimiert für **Laufzeitperformance**
 - Mächtige Sprache mit vielen Möglichkeiten
- **Schwäche von C++:**
 - Nicht optimiert für **Entwicklungs-Performance**
 - Nicht optimiert für **Compile-time-Performance**
 - „Mit viel Macht kommt viel Verantwortung“: **Komplex** und **fehleranfällig** und damit schwerer wartbar (z.B. im Vergleich zu Python, Java, ...)
- Einsatz besonders auch im Embedded Bereich z.B. in Automobilindustrie:
Feld-Performance wichtiger als Entwicklungs- oder Compile-Time-Performance



Hello World!

Live

Integrated Development Environment (IDE)

Visual Studio Code

- In der Vorlesung wird VS-Code benutzt (eigentlich kein IDE)
- Sie können natürlich einen andere IDE nutzen, ich kann nur im Zweifel nicht helfen
- Mächtiger Code Editor für nahezu alle Programmiersprachen
- *Light Weight* und schnell, kein eigener Compiler wie z.B. bei Visual Studio
- Erweiterungen können über **Pakete** installiert werden (es gibt alles mögliche an hilfreichen Erweiterungen)
- Runterladen von *Extensions* über *Marketplace*



Mein persönlicher Lieblings Codeeditor

Set-up Visual Studio Code



- Download z.B. [hier](#)
- Zum Arbeiten mit C++ folgen Sie dieser [Anleitung](#)
- Compiler: z.B. MinGW-x64 wie in Anleitung

Example: Install MinGW-x64

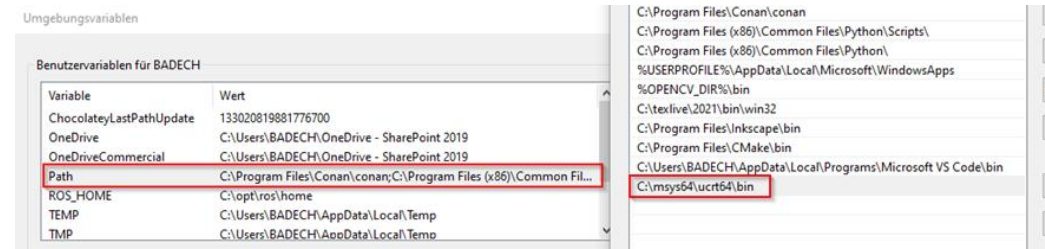
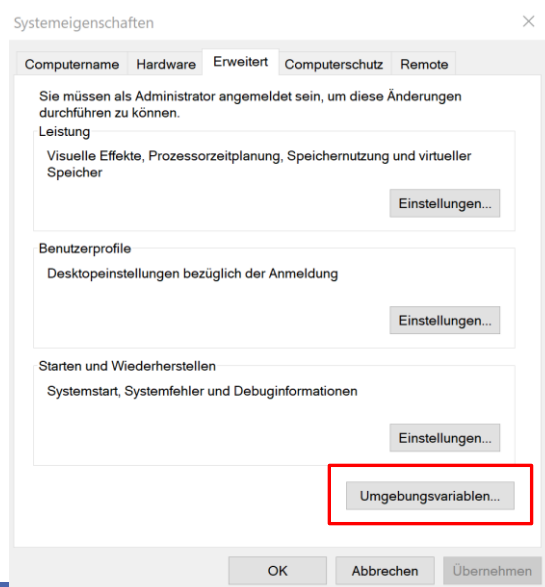
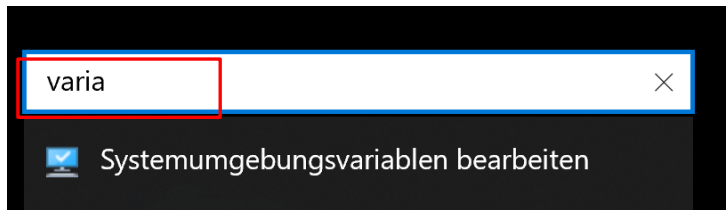
We will install Mingw-w64 via [MSYS2](#), which provides up-to-date native builds of GCC, Mingw-w64, and other helpful C++ tools and libraries. You can download the latest installer from the MSYS2 page or use this [link to the installer](#).

Follow the **Installation** instructions on the [MSYS2 website](#) to install Mingw-w64. Take care to run each required Start menu and `pacman` command, especially Step 7, when you will install the actual Mingw-w64 toolset (`pacman -S --needed base-devel mingw-w64-x86_64-toolchain`).

Set-up Visual Studio Code (Windows)



- Hinzufügen von `..\mingw64\bin` zum Pfad über Windows Suchleiste




- Pfad Variable anklicken
- Manuell den Dateipfad von `mingw64\bin` einfügen (dort wo Sie es hin installiert haben, hier ist der Default Pfad angegeben)

```
MINGW64:/c/Users/druep
druep@DESKTOP-6635JKO MINGW64 ~
$ g++ --version
g++.exe (Rev10, Built by MSYS2 project) 11.2.0
Copyright (c) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```


- In einer Konsole (z.B. VSCode oder Git Bash) überprüfen ob `g++` installiert ist

Weitere VS Code Extensions


- VS Code Extensions (Beispiele):
 - C/C++ IntelliSense: Highlighting und Support
 - Schnelles Kompilieren und Ausführen von C++ Files (nicht nötig wenn Debugger genutzt wird)
 - Code Spell Checker
 - Trailing Spaces
 - vscode-icons
 - Git Graph
 - C/C++ Runner
 - ...



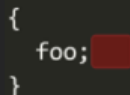
C/C++ 58ms
C/C++ IntelliSense, debugging, and code browsing.
Microsoft




C/C++ Compile Run
Compile & Run single c/c++ files easily
danielpinto8zz6



Code Spell Checker 42ms
Spelling checker for source co...
Street Side Software



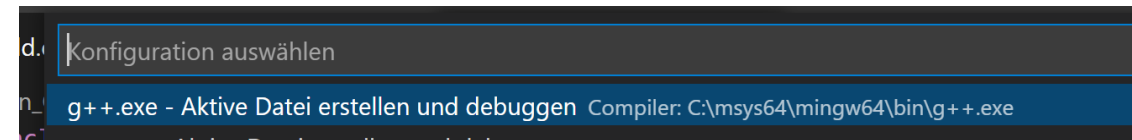
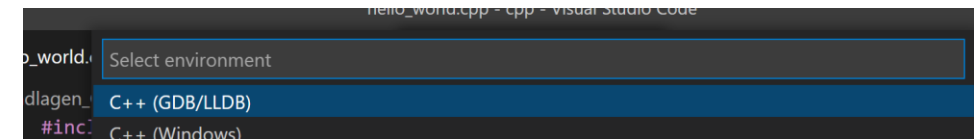
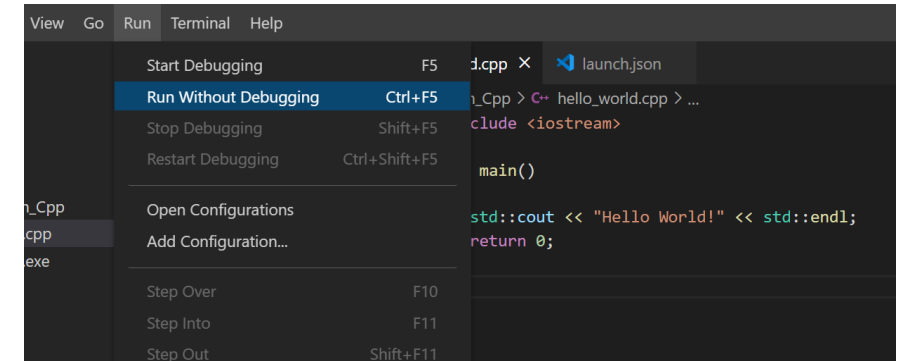
Trailing Spaces 38ms
Highlight trailing spaces and ...
Shardul Mahadik



vscode-icons 25ms
Icons for Visual Studio Code
VSCode Icons Team

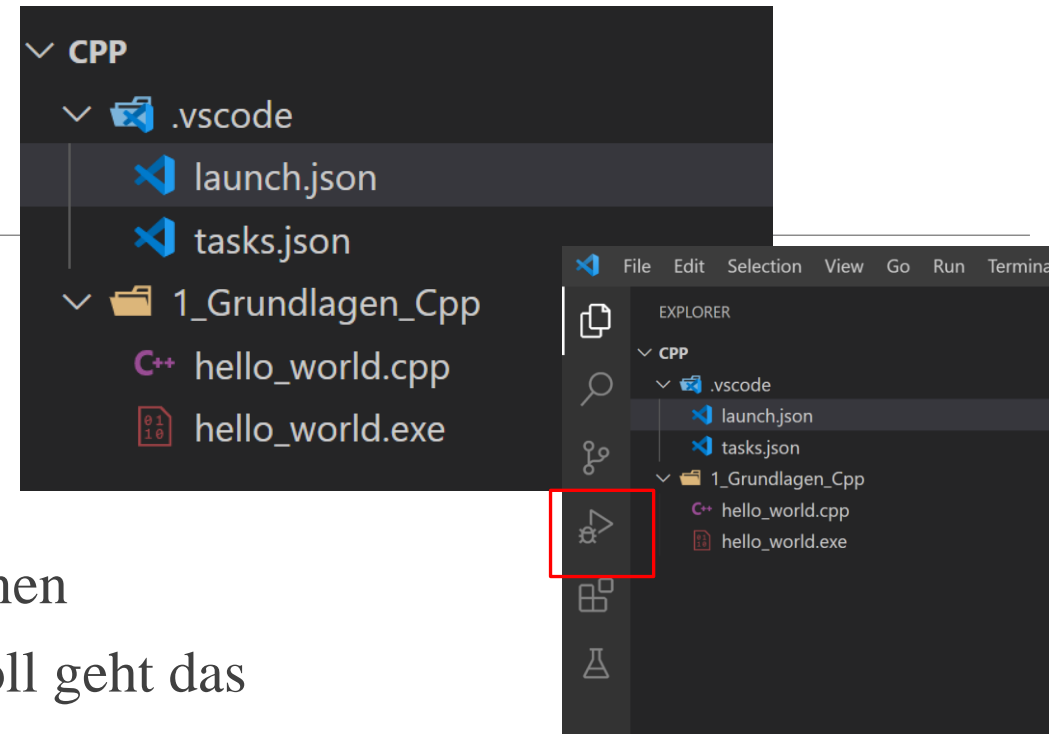
Debugging in VSCode

1. Mit VSCode den **übergeordneten** Ordner (Projekt-Ordner) öffnen indem Sie Dateien oder Unterordner speichern werden (z.B. ihren Ordner für die Cpp Vorlesung)
2. .cpp Datei öffnen (z.B. hello_world.cpp)
3. Dateipfad darf keine Lehrzeichen enthalten!
4. Run → Start Debugging (F5)
5. Select Environment C++ GDB (GNU Debugger)
6. g++.exe Aktive Datei erstellen und debuggen



Debugging in VSCode

- Jetzt sollten Sie einen `.vscode` Ordner haben mit folgendem Inhalt:
- „`launch.json`“ definiert die Launch Befehle, die über das Debug Symbol links gestartet werden können
- Wenn der letzte Befehl nochmal gestartet werden soll geht das über F5.
- „`tasks.json`“ beinhaltet tasks die direkt auch von anderen Skripten in „`launch.json`“ aufgerufen werden können. Bei uns ist das das Kompilieren.
- Hier können Sie den Compiler und die Argumente des Compilers ändern.
- **Achtung:** Wenn noch eine ausführbare exe Datei vorhanden ist bricht das Debuggen bei neuen Fehlern nicht ab (wirft zwar einen Fehler in der Konsole) und nimmt danach die alte exe fürs Debuggen.



Kapitel 1: Einleitung

11. „Hello C++ World!“
und Set-up



12. Wiederholung C

Aufgaben I: Funktionen, Kontrollstrukturen, IO

- 1) Schreiben, kompilieren und debuggen Sie das „Hello World!“ Programm.
- 2) Schreiben Sie mit Hilfe von `std::cin` ein Programm welches eine Zahl n über die Konsole bekommt und n^2 ausgibt.
- 3) Schreiben Sie eine Funktion welche $\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n$ für eine gegebenen Zahl n berechnet (wirklich über die Summe).
- 4) Schreiben Sie eine Funktion `int fib(int n)`, welche die n -te Fibonacci-Zahl F_n bestimme.
Hinweis: Die Fibonacci-Folge F_n ist definiert durch $F_0 = 0$, $F_1 = 1$ und $F_n = F_{n-2} + F_{n-1}$ für $n > 1$.
- 5) Schreiben Sie eine Funktion `bool prim(int n)`, die prüft ob n eine Primzahl ist (nur durch sich selbst und Eins teilbar).
Hinweis: `bool` ist ein elementarer Datentyp der genau zwei Werte (`true/false`) annehmen kann.

Aufgaben II: Funktionen, Kontrollstrukturen, IO

1) Implementieren Sie folgende Funktion und erklären Sie den Output:

```
C++ hello_world.cpp C++ test_integer.cpp ×
1_Grundlagen_Cpp > C++ test_integer.cpp > ...
1  #include <iostream>
2
3  int main()
4  {
5      for (int n = 0; n >= 0; n += 100000)
6      {
7          std::cout << n << std::endl;
8      }
9  }
10
```

2) Implementieren Sie folgende Funktion und erklären Sie den Output:

```
+ hello_world.cpp C++ test_integer.cpp C++ test_float.cpp ×
_Grundlagen_Cpp > C++ test_float.cpp > ...
1  #include <iostream>
2
3  int main()
4  {
5      for (float x = 0.0; x != x + 1; x += 100)
6      {
7          std::cout << x << std::endl;
8      }
9  }
10
```

Kapitel 2: Objektorientierung

Kapitel 2: Objektorientierung



21. Grundlagen

22. Vererbung

23. Polymorphismus

Exkurs: Git

C++ Basics: Header Files

C++ Basics: Pointer und Referenzen

C++ Basics: Speicherverwaltung

2.1 Grundlagen

Datenkapselung (C-Beispiel)

Was war nochmal prozedural?

- Trennung zwischen Funktionalität & Daten
- Zur Strukturierung stehen lediglich Funktionen (Prozeduren) zur Verfügung
- Variablen sind entweder lokal (innerhalb einer Funktion) oder, wenn übergreifend benötigt, auch global verfügbar
- Jede globale Variable kann überall bearbeitet werden
- Funktionen sind (sobald einmal deklariert) überall verfügbar
- Jede Funktion kann von überall mit jedem (vom Datentyp passenden) Parameter aufgerufen werden

Objektorientierung bietet Datenkapselung und Abstraktion

Paradigmen der Objektorientierung

Prozedural

1. Orientiert sich an Funktionsweise von Rechnern (Ausführen von Anweisungen und Funktionen)
2. Strukturierung von Programmen nur über Funktionen und Structs
3. Trennung von Funktionalität und Daten
4. Daten sind nur lokal in Funktion oder Global verfügbar
5. Funktionen sind global verfügbar

Wartbarkeit schwierig!

VS

Objektorientiert

1. Orientiert sich an der „echten Welt“
2. Strukturierung über Funktionen, Structs und Objekte
3. Zusammenführen von Daten und Funktionalität
4. Kapselung: Daten-Verfügbarkeit definiert über Objekte
5. Kapselung: Funktions-Verfügbarkeit definiert über Objekte

Vorteile: 4 Säulen

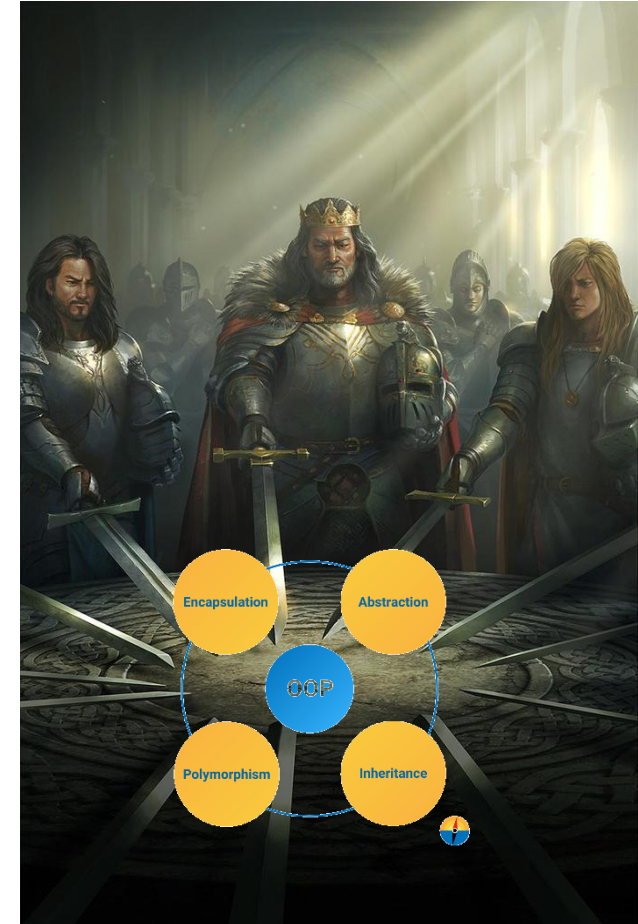
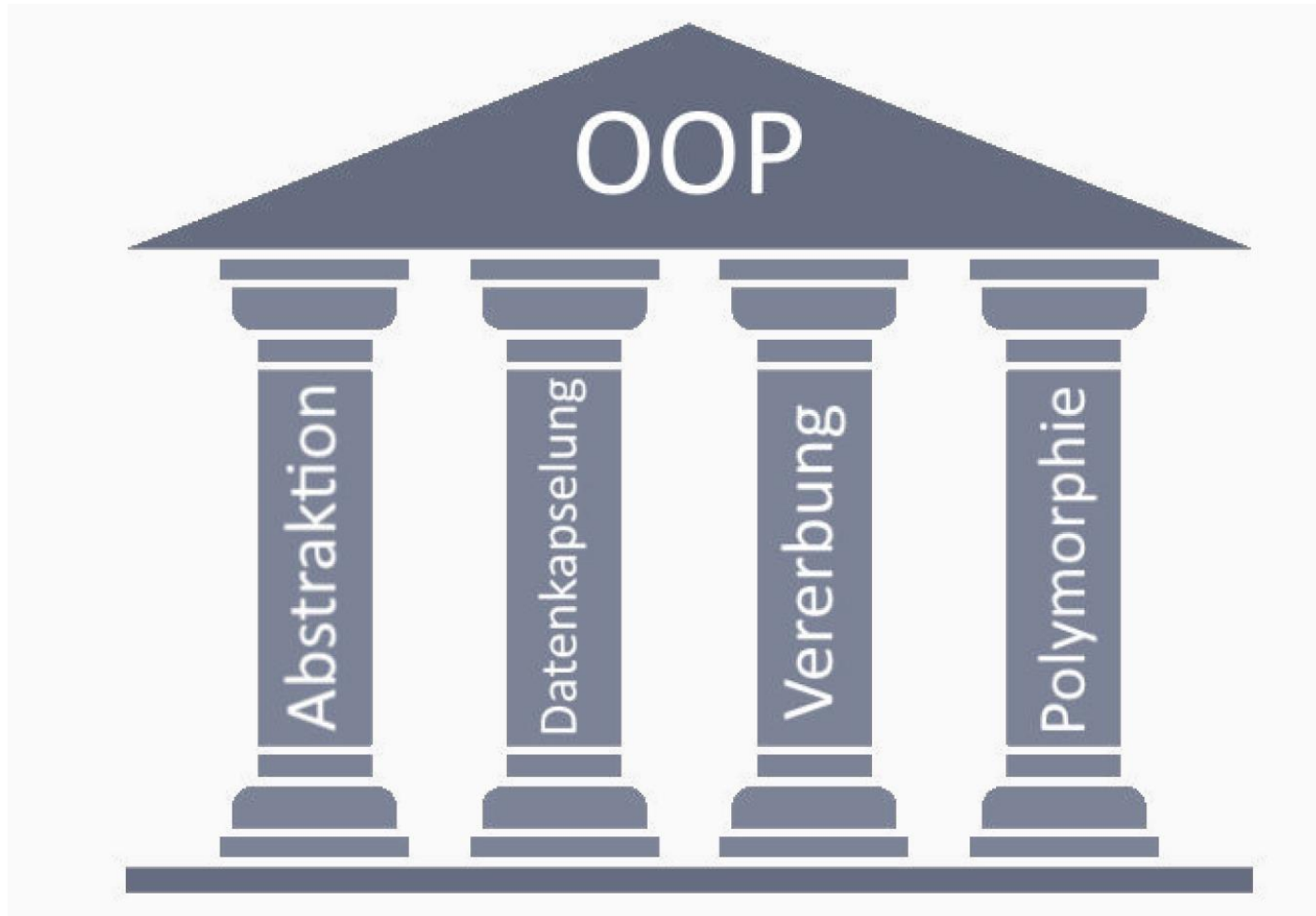
Einige Vorteile OOP

Bei richtiger Anwendung

- Gute Lesbarkeit
- Gute Erweiterbarkeit
- Intuitiver code

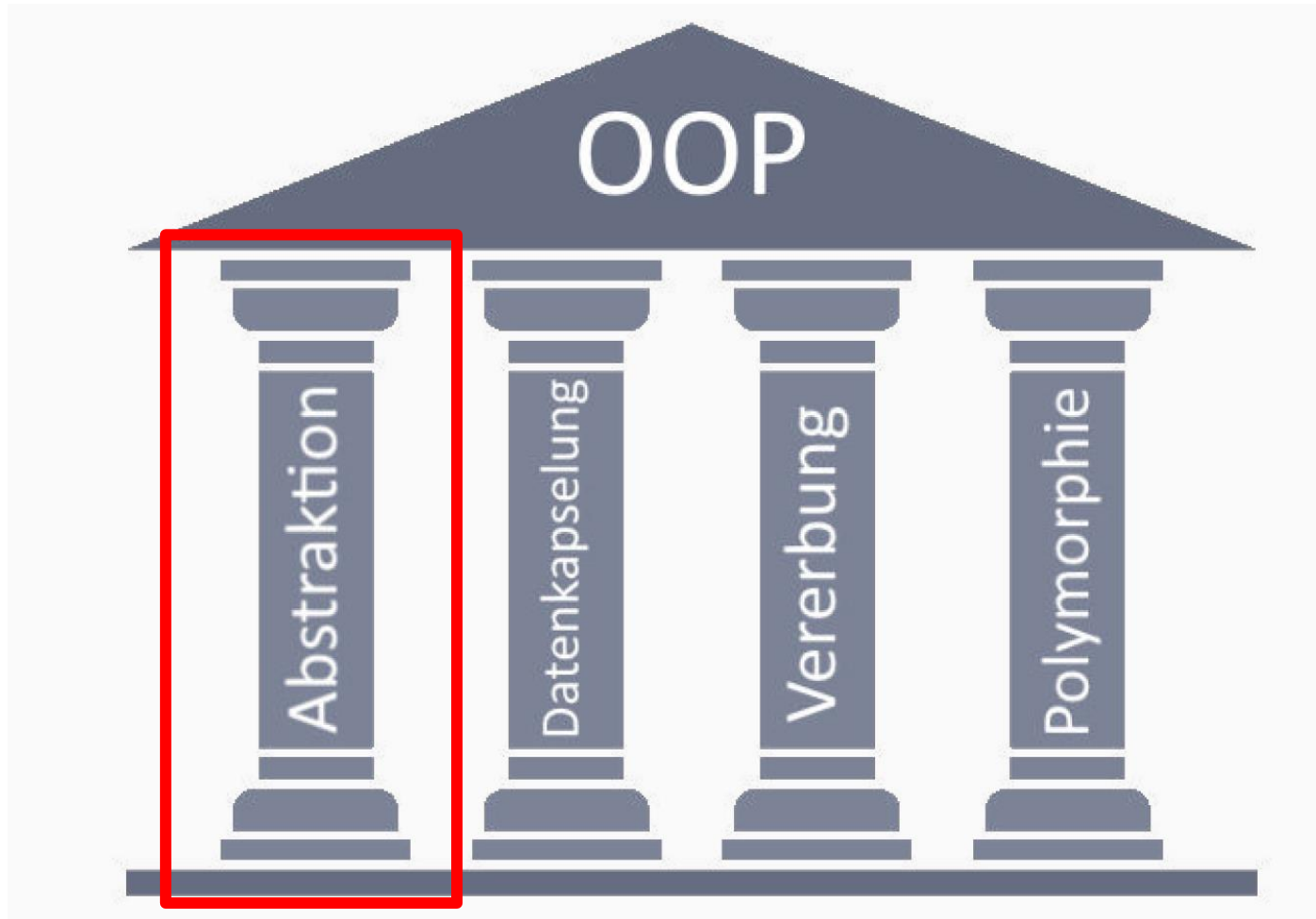


Die vier Säulen der objektorientierten Programmierung



<https://blog.devgenius.io/mastering-object-oriented-programming-and-solid-principles-a-comprehensive-guide-with-python-74acd2fb81c1>

Die vier Säulen der objektorientierten Programmierung



Objekte

Name	"Schiggy"
Nr. im Pokédex	7
Lebenspunkte	50
Level	1
Trainer	"Ash"



Was ist OBJEKTORIENTIERTE PROGRAMMIERUNG?
(Mit Pokémon erklärt) | [#Programmierung](#)

[Was ist OBJEKTORIENTIERTE PROGRAMMIERUNG?](#)
(Mit Pokémon erklärt) | [#Programmierung - YouTube](#)

Klassen und Objekte

Klassen: „Bauplan“

- Erweiterung von structs
- Enthält:
 - Daten
 - **Funktionalität**
- Kann Zugriff auf Werte verhindern
→ Kapselung!
- „Bauplan“ für das Erstellen von Objekten

Name
Nr. im Pokédex
Lebenspunkte
Level
Trainer

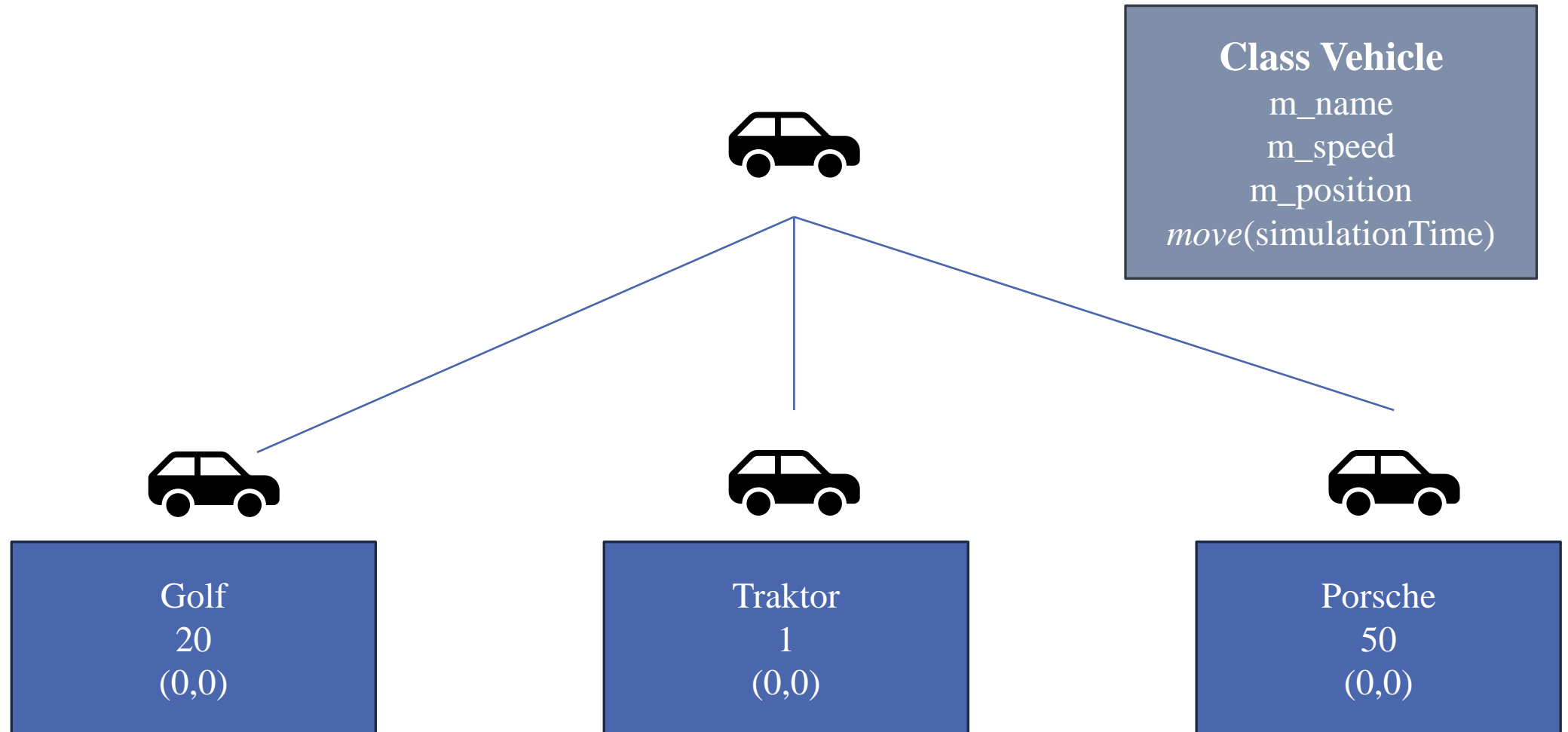
Objekte: „Instanz“

- Konkrete Instanz einer Klasse
- Enthält:
 - Zustand (Daten)
 - Verhalten (Methoden)
- Identität (z.B. über Speicherort)

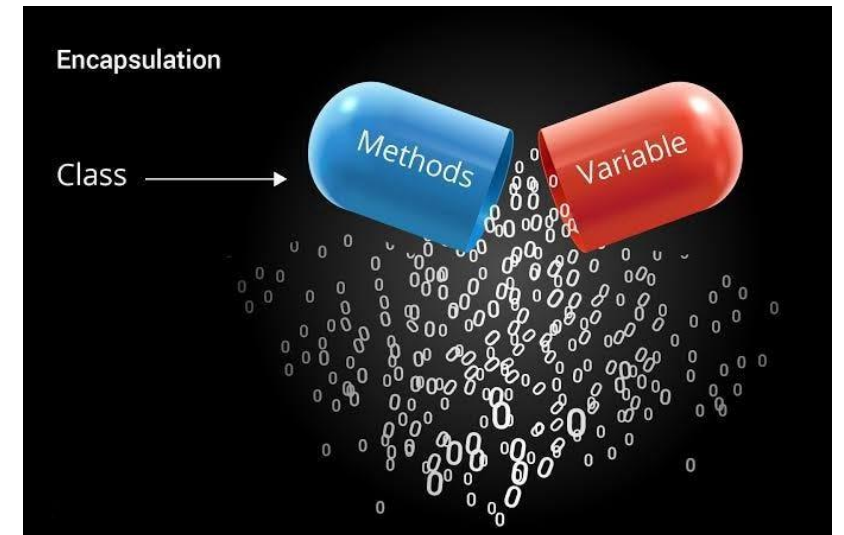
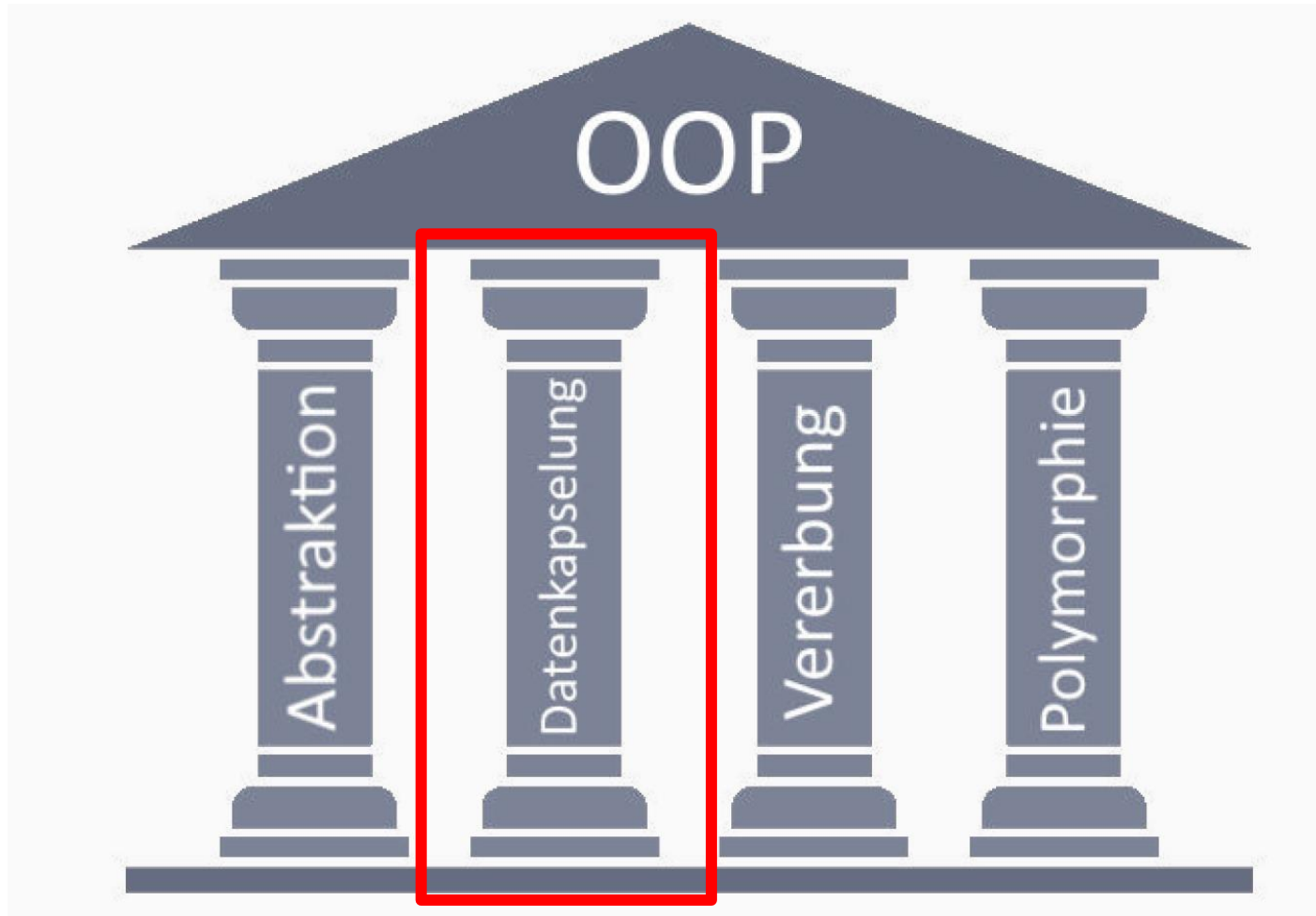
"Schiggy"
7
50
1
"Ash"



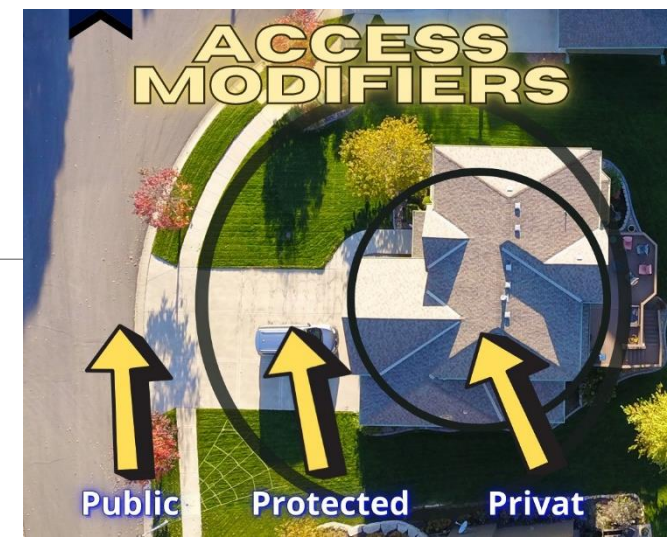
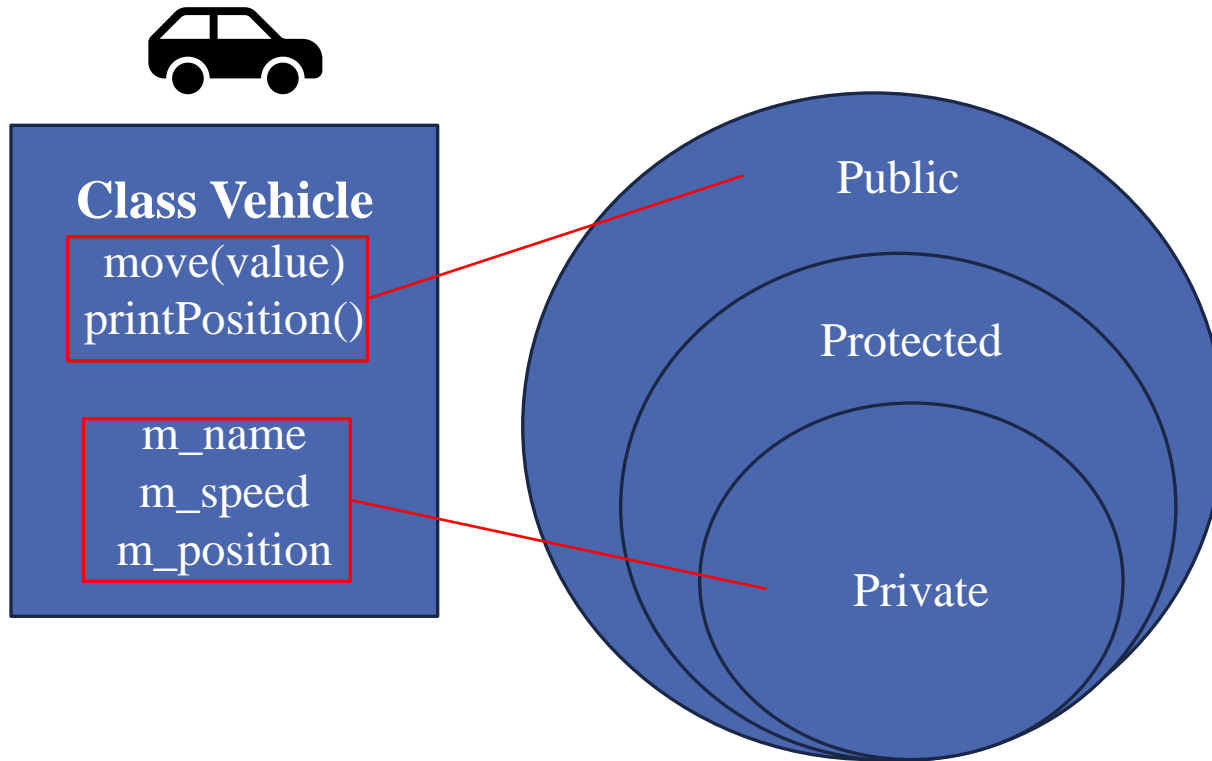
Klassen und Objekte



Die vier Säulen der objektorientierten Programmierung



Kapselung: Zugriffe



- **Public:** kann von außen aufgerufen werden, von einer Instanz können die Methoden etc. in diesem Bereich aufgerufen werden
- **Private:** kein Aufruf von außen möglich, Instanzen erreichen die Methoden und Membervariablen in diesem Bereich nicht (**auch Kinderklassen nicht**)
- **Protected:** Instanzen (reale Objekte) erreichen die Inhalte des Bereichs nicht, Kinderklassen erreichen diesen jedoch schon —> **Public für Kinder**

Konstrukturen & Initialisierungslisten & Überladen

Konstrukturen

```
public:
    // Default constructor
    Vehicle() : m_name(), m_speed(1), m_position(){};

    // Overloaded constructor
    Vehicle(std::string name, int speed, Position position)
    {
        : m_name(name)
        , m_speed(speed)
        , m_position(position)
    }

    // Destructor
    ~Vehicle()
    {
    }
```

- Allokieren Speicher und initialisieren Werte
- Initialisierungslisten brauchen keine extra Kopie: Guter Stil.
- Es gibt typischerweise verschiedene Konstrukturen je nachdem welche Argumente zur Verfügung gestellt werden
- Destruktor wird aufgerufen wenn Objekt gelöscht wird (wenn z.B. Funktion in dem es gebraucht wurde geschlossen wird)

Initialisierungslisten

```
1  #include <iostream>
2
3  class A
4  {
5  public:
6      A()
7      {
8          std::cout << "A";
9      }
10
11     A(int i)
12     {
13         std::cout << "X" << i;
14     }
15 };
16
17 class B
18 {
19 public:
20     B()
21     : m_a1()
22     , m_a2(2)
23     {
24         std::cout << "B";
25     }
26
27 private:
28     A m_a1;
29     A m_a2;
30 };
31
32 int main()
33 {
34     B();
35     return 0;
36 }
37
```

Frage: Was ist der Output dieses Programms?



Programmers. Everyday.



Überladen

```
2_Objektorientierung > C++ overload.cpp > main()
1  #include <iostream>
2
3  class Overload
4  {
5  public:
6      void foo()
7      {
8          std::cout << "noArgs ";
9      }
10
11     void foo(int)
12     {
13         std::cout << "int ";
14     }
15
16     void foo(int, int)
17     {
18         std::cout << "twoInts ";
19     }
20
21     void foo(double)
22     {
23         std::cout << "double ";
24     }
25 };
26
27 int main()
28 {
29     Overload obj;
30     obj.foo(4);
31     obj.foo(4.);
32     obj.foo();
33     obj.foo(3, 4);
34
35     return 0;
36 }
```

- Funktionen sind überladen wenn es mehr als eine Funktion mit dem gleichen Namen aber unterschiedlichen Parameter Listen gibt.
- Der Linker sucht nach der Funktion mit der am besten passenden Signatur.
- **Frage:** Welcher Output wird erzeugt?
- **Frage:** Welcher Output wird erzeugt wenn die letzte Funktion auskommentiert wird?

Aufgaben: Objektorientierung - Einführung

- 1) Erstellen Sie eine Klasse Vehicle.
 - a) Dieses erhält beim Erstellen einen Namen, eine Position (nur x Wert als double), und einen durchschnittlichen Verbrauch (l/100km) und einen Tank mit 50 Litern Benzin.
 - b) Es soll zu einem Ort (x') fahren können wenn es genug Benzin für die Fahrt hat, sonst wird die Fahrt nicht angetreten.
 - c) Es soll betankt werden können.
 - d) Es soll ausgeben können wie viele Liter es seit Erstellung insgesamt verbraucht hat.

Testen Sie Ihre Klasse durch Aufruf in der `main` Funktion.
Testen Sie beim Entwickeln nach jeder neu hinzugefügten Eigenschaft!

Kapitel 2: Objektorientierung

21. Grundlagen

22. Vererbung

23. Polymorphismus



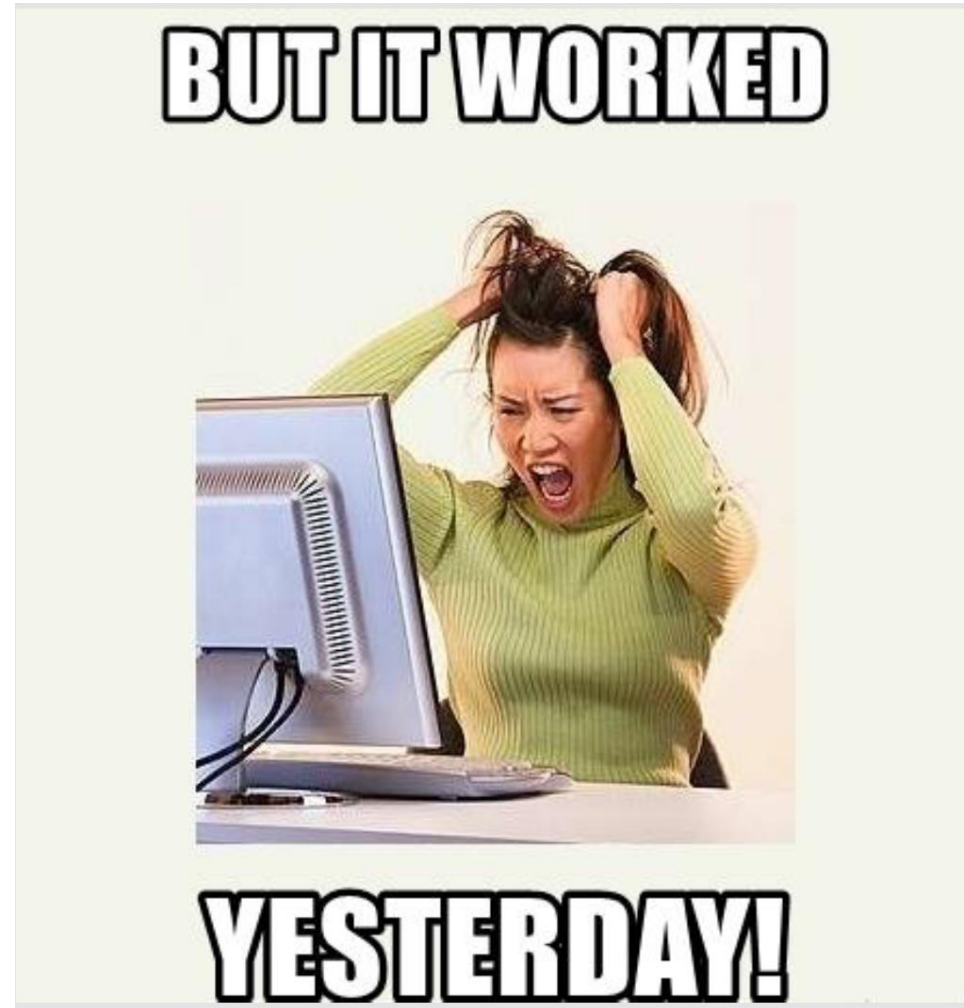
Exkurs: Git

C++ Basics: Header Files

C++ Basics: Pointer und Referenzen

C++ Basics: Speicherverwaltung


Versionsverwaltung mit Git

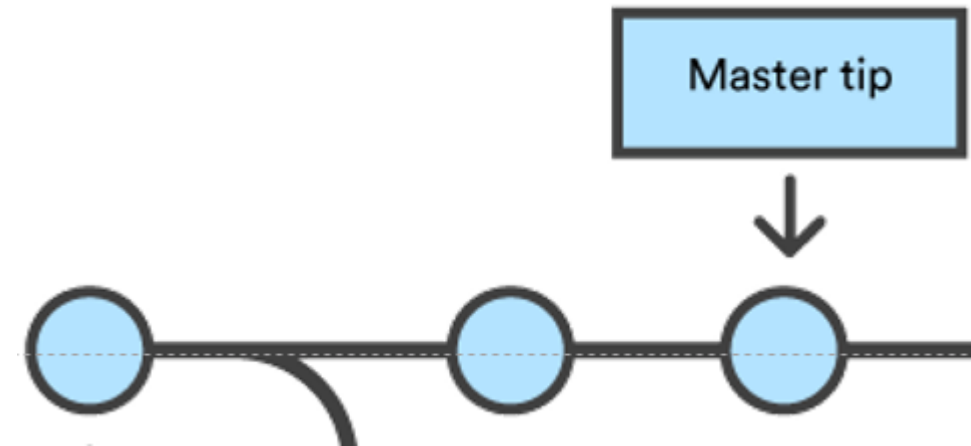


Git

- Open-source Versionskontroll-Software um Änderungen zu verfolgen
- Bietet Möglichkeit ...
 - Zu jedem gespeicherten alten Stand zurück zu kehren
 - Den Unterschied zwischen zwei Ständen zu visualisieren
 - Gleichzeitiges Arbeiten am gleichen Code mit Handeln von Konflikten etc.
- Hier:
 - Erste Erfahrungen mit Git
 - Regelmäßige Sicherheitskopien
 - Arbeiten zu zweit an Übungsprojekt

Git - Commits

- Commit: 
 - im Git Repository „registrierte“ Sammlung an Änderungen
 - + Autor
 - + Zeitstempel
 - + Commit Message
 - Entspricht einer „Momentaufnahme“ der Projektcodes



Git – Demo: Commits

Inhalt der Demo

- Show git repository in github and in git bash
- Add new file, make changes
- Commit and push changes in bash and in VS Code
- Show changes online

Git – Demo: Neues Repository erstellen

- Github → New repository
- Set name, description, private, and add .gitignore file for C++

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * mdrue04 / Repository name * cpp_dhbw ⚠

Great repository names are The repository cpp_dhbw already exists on this account. [out curly-umbrella?](#)

Description (optional)

Cpp lecture 2022

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

☒ Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

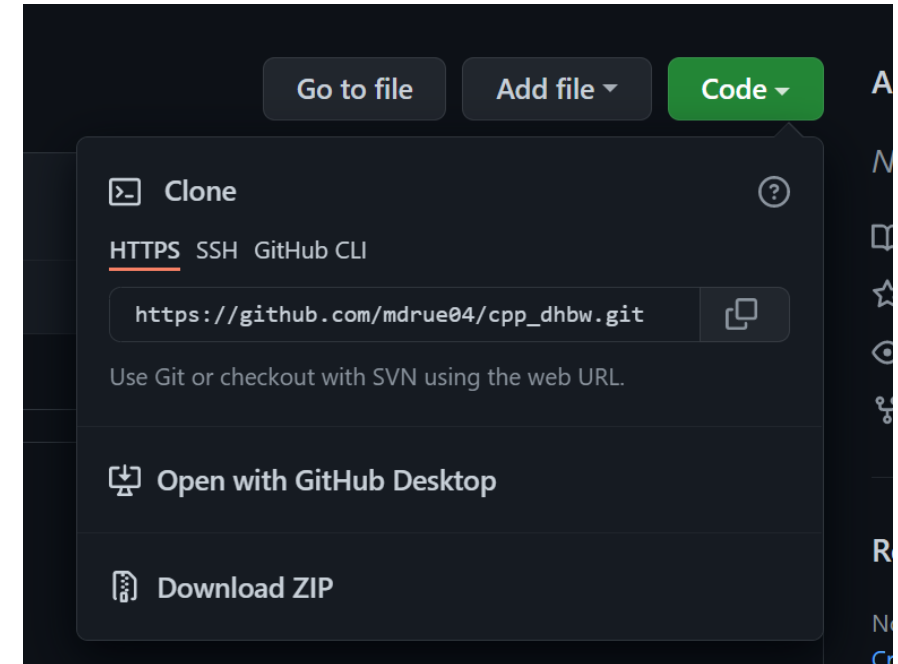
.gitignore template: C++

☐ Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

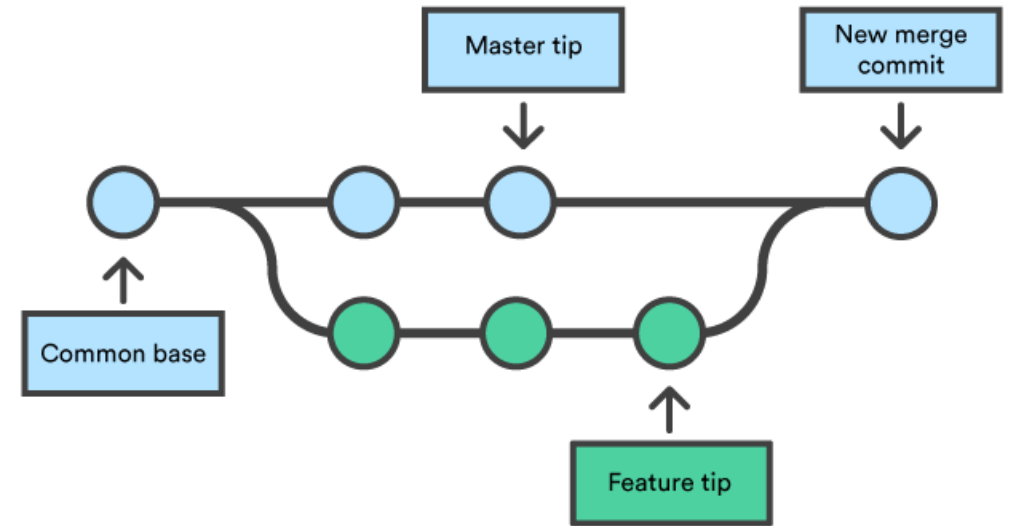
Github – Neues Git Repository

- Gehen Sie auf Code
 - Dann unter Clone, den Text unter HTTPS in die Zwischenablage kopieren
- In einer Git Console (z.B. git Bash)
 - change directory (cd) um zum richtigen Ordner zu kommen
- Dann in der Konsole:
`$git clone <https_text>`
- Dies legt bei Ihnen eine lokale Kopie des Git-Repos an



Git - Branches

- Um gleichzeitig an unterschiedlichen Ständen arbeiten zu können gibt es Branches
- Der Master-Branch ist der Haupt-Branch
- Davon können neue Branches abzweigen (z.B. für ein Feature, d.h. eine in sich abgeschlossene Erweiterung eines Programms)
- Darauf können erneut Commits erstellt werden
- Diese Commits sind nur dort verfügbar
- Ist das Feature fertig implementiert, kann man mit einem „merge“ (oft über einen Pull-Request) die Commits auf den Master „zurückführen“
- Um Stabilität zu gewährleisten wird der Master Branch gewöhnlich geschützt (Code Reviews, Pull-Request Builds, ...)



Achtung: Es gibt immer eine Server Version (origin/) eines Branches und eine lokale Version. Diese sind z.B. so lange unterschiedlich, bis lokale Commits gepusht werden oder Server Commits gepullt werden (nächste Folie)

Git – wichtige Befehle

- Git ist entweder über Konsole oder über ein GUI steuerbar, hier die Konsolen Befehle
- `git status` : Zeigt den Status des lokalen Repositories an
- `git add <fileName>/--all` : fügt <fileName>/alles der Staging Area hinzu
- `git commit -m „Commit Message“` : Erstellt einen Commit (erst noch lokal)
- `git push` : lädt die lokalen Commits des aktuellen Branchs hoch
- `git checkout <branchName>` : wechseln auf branchName
- `git fetch` : aktualisiert die lokale Kopie von origin/<current_branch>
- `git pull` : hole die neuesten Commits des aktuellen Branchs vom Server auf die lokale Kopie des Branches

Git – Demo: Branches

- Mehr Details auch z.B. in dieser [Anleitung](#)

Inhalt der Demo:

- Create new branch in git bash: `git checkout -b <branch_name>`
- Show new branch in github: `git checkout <branch_name>`
- Make, commit and push changes
- Open PR and merge branch
- Git Bash: checkout, status, fetch, pull
- Checkout older version of branch and remove new changes: `git reset --hard <commit_id>`

Download Git for Windows

- **Git**

- Git bash for Windows Download [hier](#)
- Installiert Git und eine Konsole (Git Bash)

- **Github**

- Account kann [hier](#) erstellt werden

- **Alternative Anbieter zu Github**

- Gitlab.com (für kleine Teams gratis)
- Bitbucket.org (für kleine Teams gratis)



Git - Wann wird committed?

- Ein Commit ist die kleinste Einheit von Änderungen die verglichen und z.B. auch wieder zurück genommen werden können. Diese sollte nicht zu klein auch auf keinen Fall zu groß sein.
- Wenn eine logisch zusammenhängende Programmänderung oder ein zusammenhängender Unterpunkt davon abgeschlossen ist (diese kann aus mehreren File-Änderungen bestehen)
- Wenn ich meinen aktuellen Arbeitsstand speichern möchte (weil ich z.B. den Laptop über das Wochenende zu klappe)
- Die Git Message sollte immer eingetragen werden und am besten den Grund für eine Änderung angeben („new commit“ oder „implemented change“ sagen gar nichts aus)

Git – Was wird committed?

- Vorwiegend **Textdateien** (.cpp, .hpp, README, .gitignore, ...)
- Grafiken und Schaubilder (können in README eingebettet werden)
- Keine großen Daten (lediglich Beispieldaten)
- Keine Executables
- Änderungen können im Binärformat nicht nachvollzogen werden
→ nicht sinnvoll für Versionsverwaltung

Aufgaben: Git

- 1) Legen Sie sich bei einem Anbieter ihrer Wahl einen Git Account und ein Repository für die Vorlesung an (Support nur für Github).
- 2) Checken Sie dieses aus und machen Sie sich mit der Funktionsweise vertraut. Es genügt, wenn Sie danach in der Lage sind:
 - a) Einen Commit mit einer Nachricht anzulegen und zu pushen
 - b) Die Änderungen eines älteren Commits einzusehen
 - c) Zu einem älteren Commit zurückzuspringen könnt (`git checkout <commit_id>`)
 - d) Einen Branch zu erstellen, dort zu commiten und zu pushen und diesen wieder zu mergen
- 3) Comitten Sie den Code aus der letzten Aufgabe in Ihrem Repo und machen Sie mindestens eine neue Änderung die auch gepusht wird.