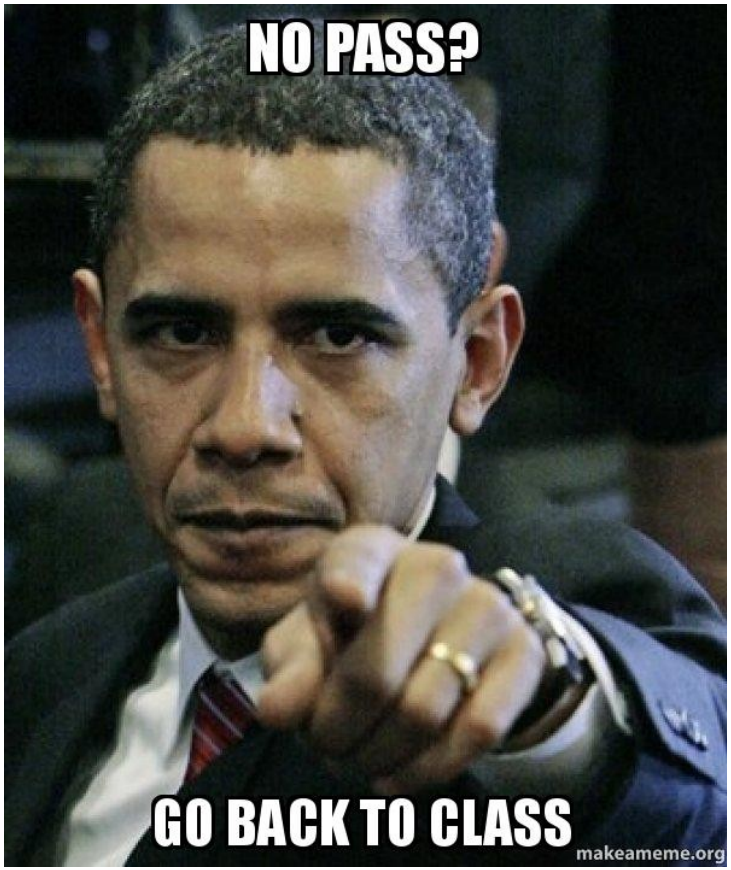
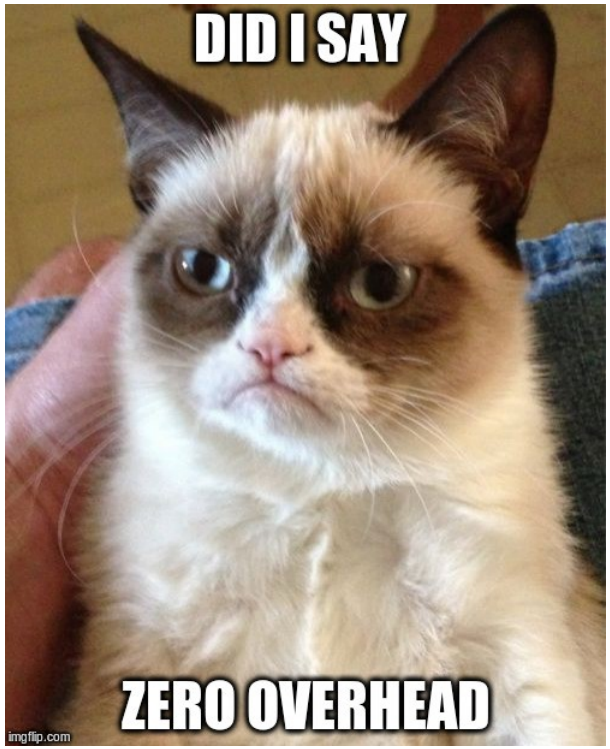
# Klassen

- Was ist eine Klasse
- Konstruktor / Destruktor (new, delete)
- Sichtbarkeit
- Operatoren
- Rule of 3 / 5 / 0 / default
- Const bei Funktionen
- noexcept



# Was ist eine Klasse

- Möglichkeit Code zu strukturieren
- Eine Struktur mit Sichtbarkeit und Funktionspointer
  - Unterschied: struct ist public, class ist private bei default
- **Kein Overhead zu struct!!**
- **http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rc-org**
- https://godbolt.org/g/nTHCpo



# Konstruktor

- Wird aufgerufen wenn ein Objekt erstellt wird
- Default Konstruktor ist ein Konstruktor ohne Übergabeparameter
  - Kann überladen werden
- Vermeidung von implizierten Konstruktoraufruf durch *explicit*
- Initialisierungsliste > Zuweisungen
- Copy- Konstruktor und Move-Konstruktor können angegeben werden
- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#S-ctor
- https://godbolt.org/g/BdL5HF

# Destruktor

- Wird Aufgerufen wenn Objekt gelöscht wird
  - Das gilt auch, wenn ein Objekt aus dem Scope fällt
- Wird durch ~Klassenname definiert
- Wird implizit generiert, falls kein Destruktor angegeben ist
- https://godbolt.org/g/9tRaQx

# Sichtbarkeit

- Public: offene Sichbarkeit. Variablen können direkt manipuliert werden
- Private: keine Sichtbarkeit. Variablen können nur durch Funktionen verändert werden
- Protectet: keine Sichtbarkeit. Aber im Sichtbar für abgeleitete Klassen
- Standard bei Klassen ist private

# Operatoren


- Alle Arten von Operatoren können überladen werden
  - + - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &= |= << >> >>= <<= == != <= >= && || ++ -- , ->* -> ( ) [ ] new delete new[] delete[]
  - Funktionsname ist *rückgabetyp operator** (übergabetyp)*
- Nur verwenden, wenn man ganz genau weiß, was man tut !!!!
- https://godbolt.org/g/aHJFrq

# Rule of Three

- Falls mindestens der Destructor, Copy-Konstruktor oder Copy-Zuweisung implementiert ist, sollen alle drei implementiert werden
- Grund: Da diese meistens für expliziete Resoucenverwaltung verwendet werden, sollten die anderen nicht implizit generiert werden
- Keine Compilerwarnung!!!!
- https://godbolt.org/g/vHbaAn

# Rule of five

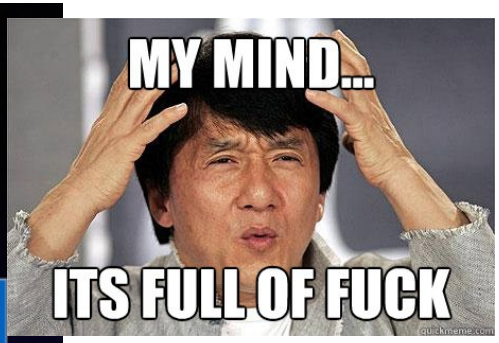
- Seit C++ 11: Rule of five
  - Zusätzlich noch Move-Konstruktor und Move-Assignement
- Vorsicht: Move Implementierung nicht trivial !!

# Rule of five

## Special Members

compiler implicitly declares

| user declares | default constructor | destructor | copy constructor | copy assignment | move constructor | move assignment |
|---|---|---|---|---|---|---|
| Nothing | defaulted | defaulted | defaulted | defaulted | defaulted | defaulted |
| Any constructor | not declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| default constructor | user declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| destructor | defaulted | user declared | defaulted | defaulted | not declared | not declared |
| copy constructor | not declared | defaulted | user declared | defaulted | not declared | not declared |
| copy assignment | defaulted | defaulted | defaulted | user declared | not declared | not declared |
| move constructor | not declared | defaulted | deleted | deleted | user declared | not declared |
| move assignment | defaulted | defaulted | deleted | deleted | not declared | user declared |




Rule of Three

Rule of Five

Rule of Zero

# Rule of Zero

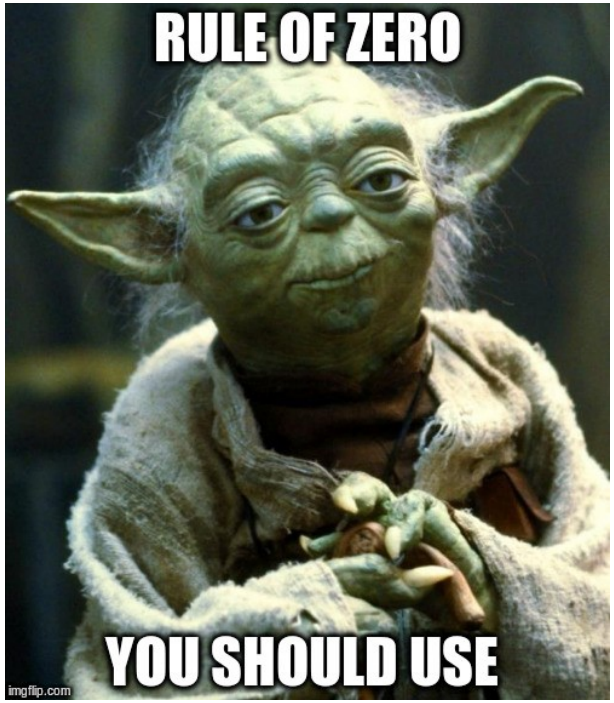
- Wenn kein Destruktor etc. implementiert ist, generiert uns der Compiler alles
- Keine Probleme mit verlorenen Pointern
  - Wenn Pointer notwendig → Smart Pointer
- Von mir empfohlene Variante !!!
- Auch später für Container wichtig
- https://godbolt.org/g/tB7hD7

# Rule of default / delete

- Möglichkeit „Special Memebers“ als default oder delete zu beschreiben
- *~Destruktor() = default* ( Implementiert, als würde Compilergeneriert sein)
- *~Destruktor() = delete* (Funktion nicht verfügbar)
- Große Diskussion im ISO-Komitee, was wann am besten zu verwenden ist
- https://godbolt.org/g/S9wuaL

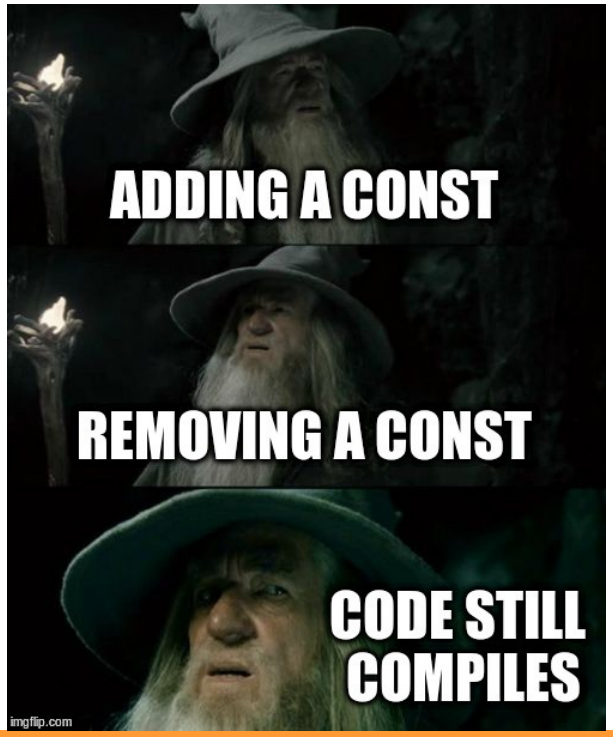
# Rules of *

- Empfehlung:
  - Versuche immer Rule of Zero zu schreiben
  - Falls nicht möglich, gib alle 5 „Special Members“ an mittels delete und default
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#S-ctor



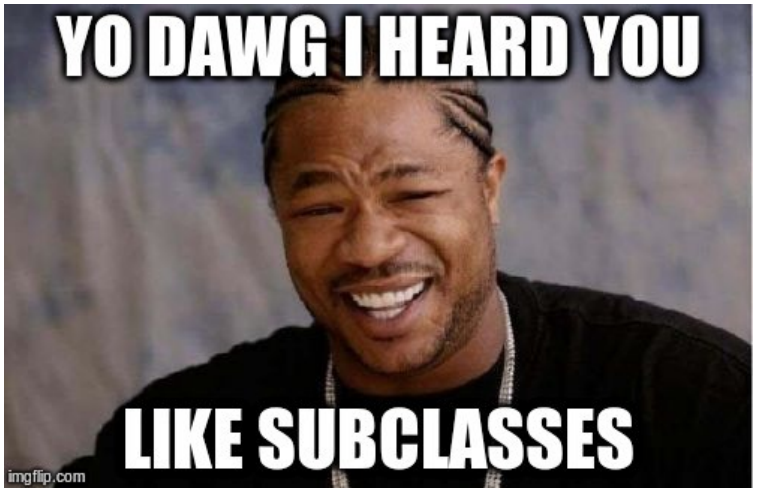
# Const Funktionen

- *Const* am Ende von Memeberfunktionen heist Membervarialben werden in dieser Funktion nicht verändert (z.B. getter)
- Erlaubt dem Compiler viele Optimierungen
- Immer angeben wenn möglich
- https://godbolt.org/g/gEkFzu



# noexcept

- *Noexcept* gibt an, dass die Funktion keine Exception wirft
- Vorsicht da:
  - Einige stl – Funktionen Exceptions werfen können und dann die Meldung verloren geht
  - Beim Refactoring oft übersehen wird
- Aber gibt Optimierungsmöglichkeiten für den Compiler
- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Re-noexcept
- https://godbolt.org/g/QZN4am




Nächstes Mal:

Vererbung, Laufzeitpolymorphie und ähnliches