Vererbung und Polymorphie

- Vererbung in C++
- Interfaces
- Zugriffe
- Laufzeitpolymorphie (Decorater Pattern)

Sonstiges über Klassen:

- Innere Klassen
- Performance
- Meyers Singleton

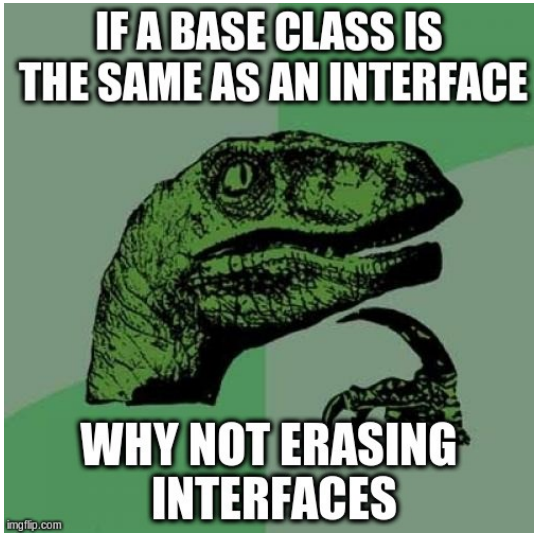
Vererbung in C++

- Vererbung wird angeben mit *class Klassenname : Basisklasse*
- Mehrverfach-Vererbung möglich. Abtrennung durch Komma
  - Vorsicht vor „Diamond“-Vererbung
  - de.wikipedia.org/wiki/Diamond-Problem
- Funktionen, die vererbt werden sollen müssen *virtual* sein
- Konstruktor der Oberklasse wird implizit aufgerufen, falls möglich.
  Ansonsten Expliziter Aufruf in der Initialisierungsliste

Vererbung in C++

- Seit C++11
  - *Override* Schlüsselwort um Überladung anzugeben
  - *Final* Schlüsselwort um nachfolgende Überladungen zu verhindern
  - Compiler kann vererbung Prüfen!!
  - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c139-use-final-sparingly
  - Übernahme von Oberklassenfunktionen mittels *using*
  - *http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c138-create-an-overload-set-for-a-derived-class-and-its-bases-with-using*
- https://godbolt.org/g/xoMxaU

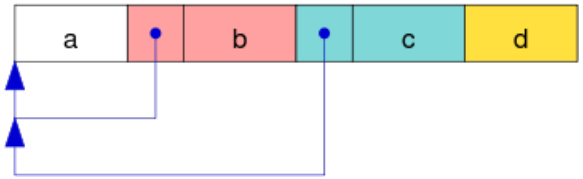
Interfaces

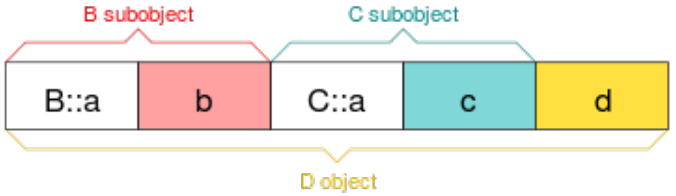


- Interfaces sind Oberklassen ohne eigene Implementierung
- Funktionen werden als *pure Virtual* bezeichnet
  - *Virtual rückgabetyp Funktionname(Paramter) = 0*
- Variablen vermeiden!!
- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c121-if-a-base-class-is-used-as-an-interface-make-it-a-pure-abstract-class
- https://godbolt.org/g/eHpxHg

Zugriffe und friends

- Zugriffskontrolle mittels *public, proteced, private* vor der Basisklasse
  - Bei public bleiben Elemente public
  - Bei protected werden public Elemente protected
  - Bei private werden alle Elemente private (Standart-Vererbungsart)
- Virtual Angabe bei Klassenname
  - Generiert Pointer auf die Oberklasse
  - Ansonsten hat das Objekt eine Kopie der Oberklasse

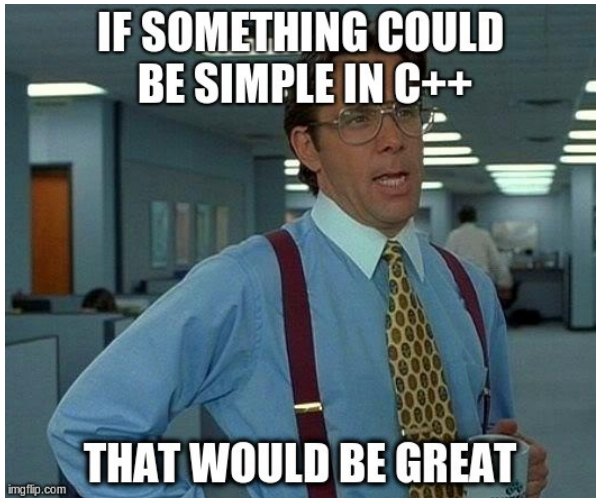




- Möglichkeit, speziellen Klassenzugriff auf Private variablen zu geben mittels *friend*
- https://godbolt.org/g/3LYJge

Laufzeitpolymorphie

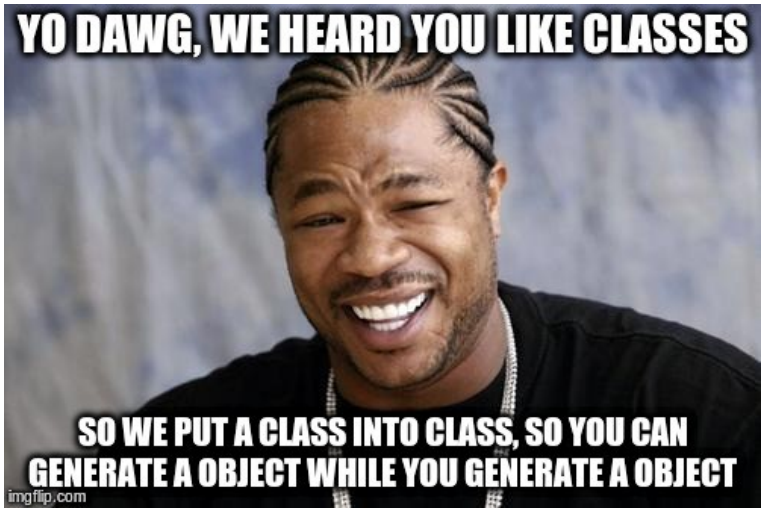
- Container mit einer Basisklasse, können auch alle vererbten Objekte beinhalten.
  - Logischerweise nur Funktionen der Basisklasse dann vom Container aus aufrufbar
  - Ansonsten: dynamic_cast
- Funktioniert nicht mit Interfaces!! Da Interfaces keinen Standartkonstruktor haben!!
  - Lösung: Über (Smart-)Pointer



- Beispiel Decorator Pattern
- https://godbolt.org/g/HpMa5N

Sonstiges über Klassen

Innere Klassen



- Verwendet um Implementierung zu verstecken
- Innere Klasse nur über Äußere Klasse zugreifbar

- Code wird unübersichtlich
- Meist ist eine separate Datei sinnvoller und übersichtlicher
- https://godbolt.org/g/JcFddE

Performance

- Variablen Anordnung
  - Variablen und Funktionen werden so angelegt, wie sie in der Klasse beschrieben werden
  - Häufig verwendete Wert nach oben
  - http://quick-bench.com/3oHpdYnBk7L18FqpTkBfDKHPt0Y
- Vererbung reduzieren
  - Erzeugt zusätzliche Einträge in der V-Table
- **ABER: LESBARER STRUKTURIERTER CODE IST WICHTIGER!!!**

Meyers Singleton

```
static Singleton& instance()
{
     static Singleton s;
     return s;
}
```

- Einfache Singleton Implementierung
- Wert wird mittels Referenz zurückgegeben
- Static in der Funktion bedeutet, dass Variable Global innerhalb der Funktion verfügbar ist
- Static ist seit C++11 Thread-Safe

Nächstes Mal:

Container und Speicherverwaltung