Draft der C++-Vorlesung vom 17.Juni 2022

Klassenhierarchie §12

Keywords: Vererbung, virtuelle Methoden, Überladen von Methoden, Polymporphismus
Keywords: Basisklasse, abgeleitete Klasse, abstrakte/konkrete Klasse, VMT
Keywords: Basisklassenpointer, `shared_ptr`, `unique_ptr`, dynamic casting

Finales Ziel: STL auf Containern mit polymorphen Elementen. *v_ 10c_ shared/*

Angestelltenhierarchie §12.1

Einführungsbeispiel: Employee, Worker, salesPerson, Manager. *v_ 9b/* Zur Erinnerung:

- Entwurf (IS-A; HAS-A)
- Initialisierung der Basisklasse via *member initialization list* im Konstruktor der abgeleitetten Klasse.
- Zugriffsrechte: private, protected, public
- Überschreiben von Methoden (*virtual method*).
- Polymorphismus mit Referenz &Employee in Parameterliste und VMT zur Auswahl der überschriebenen Methode zur Laufzeit.
  Kein Polymorphismus mit Kopie Employee, da erfolgt ein Upcasting (Reduktion auf Eigenschaften der Basisklasse).

Polymorphismus §12.2

- Funktion `Ausgabe(Employee const&)` in :
  - demonstriert Polymorphismus (VMT).
  - funktioniert nur mit Referenz (oder Pointer); nicht bei Copy (UpCasting).
    Copy ist bei abstrakten Basisklassen nicht möglich!
- Example *v_ 9b_ try* mit abgeleiteter Klasse `BoxPromoter`.
  - Dynamic bindung §12.2.3
  - Funktionen mit Polymorphismus (`Employee &`) können neue Klasse benutzen, ohne daß diese Funktionen neu übersetzt werden müssen.
  - Vorkompilierte Bibliotheken können einfach genutzt werden.
- Einige Worte zum Casting in Klassenhierarchien:
  - UpCasting per Copy (abgeleitete Klasse → Basisklasse): möglich, da Eigenschaften wegfallen [Compilezeit].

  - DownCasting per Copy (Basisklasse → abgeleitete Klasse): nur via spezielle Konstruktoren möglich [Compilezeit].
    Fehler im Klassendesign.
  - Casting via Referenz: `dynamic_cast<other_class&>(my_instance)`
    Fehler über `catch (std::bad_cast& bc)` abfangen [Laufzeit]
  - Casting via Pointer: `dynamic_cast<other_class*>(&my_instance)`
    Fehler über Test des Ergebnisses auf `nullptr` abfangen [Laufzeit]
  - Example *v_ 9b_ cast*

- Allgemein Casting:
  - `static_cast<new_type>(instance)`
  - `const_cast<type>(const type)` [sehr selten benötigt]
  - `dynamic_cast<new_type&>(type&)` in Klassenhierarchien [Spezialfälle]
    auch mit Pointer.
  - `reinterpret_cast<new_type>(instance)` auf Ihre Verantwortung [alles zu spät]

STL und Klassenhierarchie §12.3

Polymorphismus funktioniert nur zur **Laufzeit** über **Basisklassenreferenzen** oder **Basisklassenpointer**.
STL-Container mit Referenzen sind nicht möglich, daher müssen wir Basisklassenpointer als Elemente der Container benutzen um den Polymorphismus einer Klassenhierarchie auszunutzen.

- Example *v_ 9b_ poly*:
- Benutze STL-algorithms, aber stets mit Lambda-Funktion (o.ä.) als unäreren oder binären Operator.
- klassischer Basisklassenpointer `*Employee`

- besser: `shared_ptr<Fahrzeug>` [*v_ 10c_ shared*]
- oder: `unique_ptr<Fahrzeug>` [*v_ 10c_ unique*]

Literatur

[Haase22] Gundolf Haase: Einführung in die Programmierung mit C++ (2022), *www*[1].

[Stroustrup10] Bjarne Stroustrup: Einführung in die Programmierung mit C++. Pearson Studium, München (2010).

[1] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`