OOP-Vererbung, Polymorphie (Vielgestaltigkeit)

Paul Raffer

Polymorphie

	universell unendlich viele Typen eine Implementierung	**Ad-hoc** endliche Anzahl an Typen unterschiedliche Implementierungen
dynamisch Laufzeit langsamer	Inklusionspolymorphie/ Vererbungspolymorphie	Coercion
statisch Kompilezeit schneller	parametrische Polymorphie	Überladung

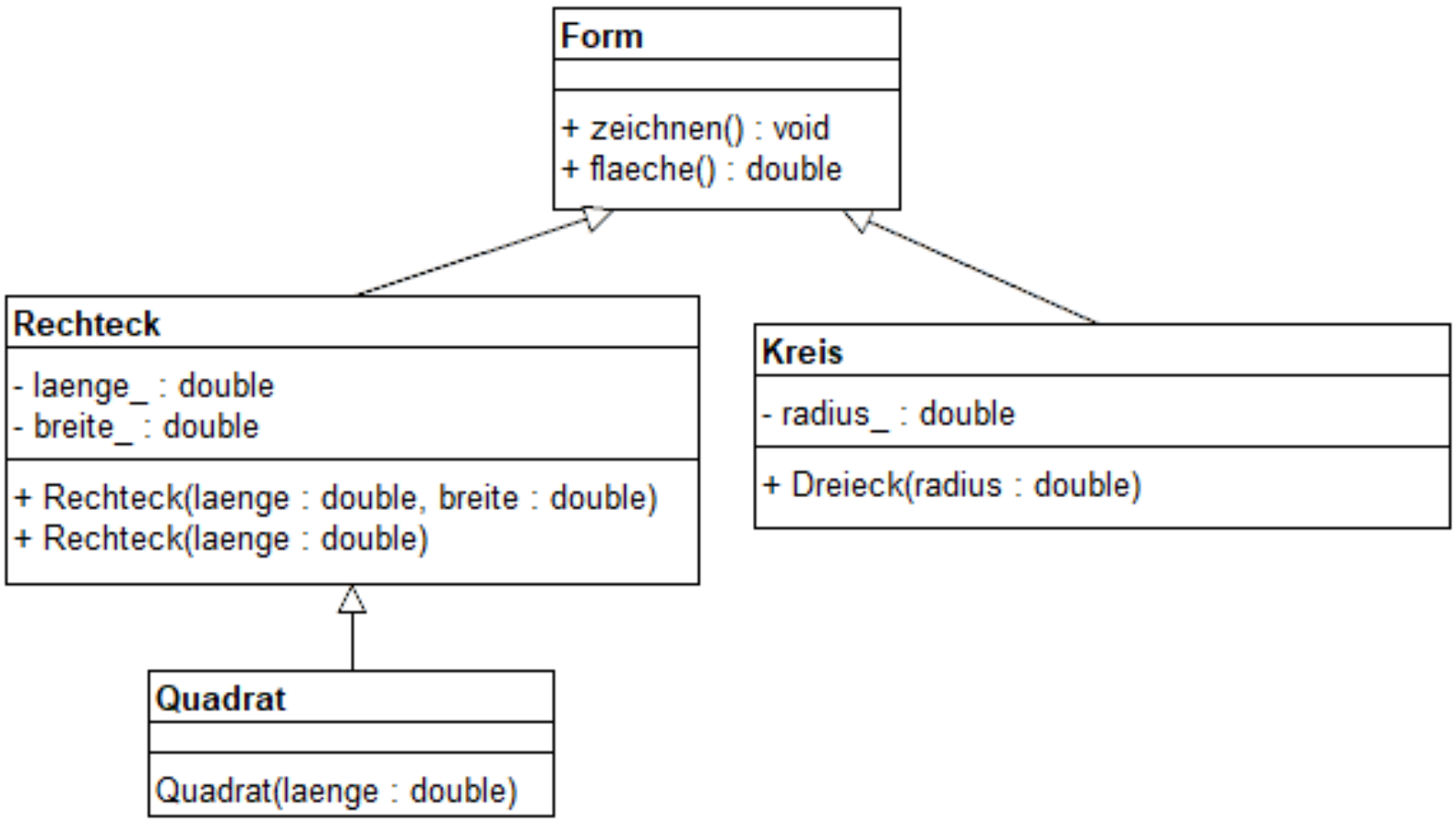
Geschichte

- ...

Vererbungspolymorphie

- Vererbung
  - virtuelle Funktionen
  - abstrakte Klassen
  - Mehrfachvererbung
    - virtuelle Vererbung

Beispiel: Formen



Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Beispiel: Formen

Beispiel: Formen

```
class Form { // abstrakte Bsisklasse, gemeinsames Interface
public:
    virtual auto zeichnen() const -> void = 0;
    [[nodiscard]] virtual auto flaeche() const -> double = 0;
};
```

Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Beispiel: Formen

```
class Form { // abstrakte Bsisklasse, gemeinsames Interface
public:
    virtual auto zeichnen() const -> void = 0;
    [[nodiscard]] virtual auto flaeche() const -> double = 0;
};

class Rechteck : public Form {
    double laenge_, breite_;
public:
    Rechteck(double laenge, double breite) : laenge_{laenge}, breite_{breite} {}
    explicit Rechteck(double laenge) : Rechteck{laenge, laenge} {}

    auto zeichnen() const -> void override { /*...*/ }
    auto flaeche() const -> double override { return laenge_ * breite_; }
};
```

Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Beispiel: Formen

```
class Form { // abstrakte Bsisklasse, gemeinsames Interface
public:
    virtual auto zeichnen() const -> void = 0;
    [[nodiscard]] virtual auto flaeche() const -> double = 0;
};

class Rechteck : public Form {
    double laenge_, breite_;
public:
    Rechteck(double laenge, double breite) : laenge_{laenge}, breite_{breite} {}
    explicit Rechteck(double laenge) : Rechteck{laenge, laenge} {}

    auto zeichnen() const -> void override { /*...*/ }
    auto flaeche() const -> double override { return laenge_ * breite_; }
};

class Quadrat : public Rechteck {
public:
    explicit Quadrat(double laenge) : Rechteck{laenge} {}
};
```

Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Beispiel: Formen

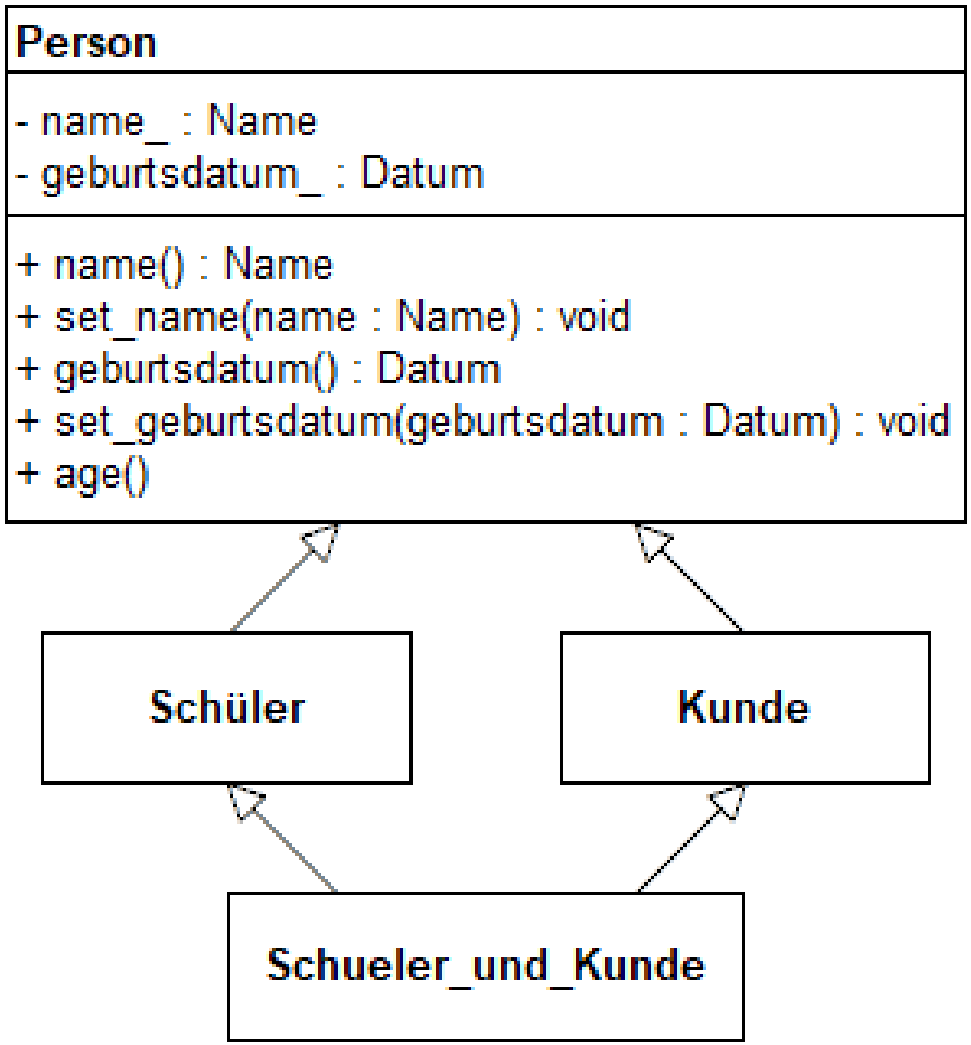
```
class Form { // abstrakte Bsisklasse, gemeinsames Interface
public:
    virtual auto zeichnen() const -> void = 0;
    [[nodiscard]] virtual auto flaeche() const -> double = 0;
};

class Rechteck : public Form {
    double laenge_, breite_;
public:
    Rechteck(double laenge, double breite) : laenge_{laenge}, breite_{breite} {}
    explicit Rechteck(double laenge) : Rechteck{laenge, laenge} {}

    auto zeichnen() const -> void override { /*...*/ }
    auto flaeche() const -> double override { return laenge_ * breite_; }
};

class Quadrat : public Rechteck {
public:
    explicit Quadrat(double laenge) : Rechteck{laenge} {}
};

class Kreis : Form {
    double radius_;
public:
    explicit Kreis(double radius) : radius_{radius} {}

    auto zeichnen() const -> void override { /*...*/ }
    auto flaeche() const -> double override
    {
        return std::pow(radius_, 2) * std::pi_v<double>; // A = r^2 * pi
    }
};
```

Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Mehrfachvererbung

- Funktionalität von mehreren Klassen kombinieren

Virtuelle Vererbung



Vollständiges Beispiel: `https://godbolt.org/z/heEWxW`

Coercion

- implizite Typumwandlungen

Coercion

- implizite Typumwandlungen
  - eingebaute Datentypen

```
5 + 3.2 /* 5 wird von int zu double gecastet */
```

Coercion

- implizite Typumwandlungen
  - eingebaute Datentypen

```
5 + 3.2 /* 5 wird von int zu double gecastet */
```

  - eingene Datentypen

Coercion

- implizite Typumwandlungen
  - eingebaute Datentypen

```
5 + 3.2 /* 5 wird von int zu double gecastet */
```

  - eingene Datentypen
    - Konvertierungskonstruktoren

```
Bruch::Bruch(int zaehler = 0, int nenner = 1) // Konvertierungskonstruk
    : zaehler_{zaehler}, nenner_{nenner} {}

auto print(Bruch const & bruch) -> void
{
    std::cout << '(' << bruch.zaehler << '/' << bruch.nenner << ')';
}

auto main() -> int
{
    print(42); // impliziter Cast zu Bruch{42, 1}
    // Output: (42/1)
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Coercion

- implizite Typumwandlungen
  - eingebaute Datentypen

```
5 + 3.2 /* 5 wird von int zu double gecastet */
```

  - eingene Datentypen
    - Konvertierungskonstruktoren

```
Bruch::Bruch(int zaehler = 0, int nenner = 1) // Konvertierungskonstruk
    : zaehler_{zaehler}, nenner_{nenner} {}

auto print(Bruch const & bruch) -> void
{
    std::cout << '(' << bruch.zaehler << '/' << bruch.nenner << ')';
}

auto main() -> int
{
    print(42); // impliziter Cast zu Bruch{42, 1}
    // Output: (42/1)
}
```

    - Konvertierungsoperatoren

```
Bruch::operator double() // Konvertierungsoperator (Bruch zu double)
{
    return zaehler / nenner;
}

auto main() -> int
{
    std::cout << Bruch{1, 4} + 4.25; // Output: 4.5
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

parametrische Polymorphie

- Templates/Generics
  - Funktionstemplates
  - Klassentemplates
  - Konzepte

Funktionstemplates

```
template <typename T1, typename T2>
auto summe(T1 const & t1, T2 const & t2) -> decltype(t1 + t2)
{
    return t1 + t2;
}

// ODER

auto summe(auto const & t1, auto const & t2)
{
    return t1 + t2;
}


auto main() -> int
{
    std::cout << summe("Hallo", " Welt!"); // Hallo Welt!
    std::cout << summe(7, 3.5); // 10.5
}
```

Vollständiges Beispiel: https://godbolt.org/z/

Klassentemplates

Klassentemplates

```
template <typename T>
class Stack {
    std::vector<T> elements_;
public:
    auto push(T const & t) -> void;
    auto pop() -> T;
    [[nodiscard]] auto is_empty() const -> bool;
};
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```
template <typename T>
class Stack {
    std::vector<T> elements_;
public:
    auto push(T const & t) -> void;
    auto pop() -> T;
    [[nodiscard]] auto is_empty() const -> bool;
};


template <typename T>
auto Stack<T>::push(T const & element) -> void
{
    elements_.push_back(element);     // Element auf Stack legen
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```
template <typename T>
class Stack {
    std::vector<T> elements_;
public:
    auto push(T const & t) -> void;
    auto pop() -> T;
    [[nodiscard]] auto is_empty() const -> bool;
};


template <typename T>
auto Stack<T>::push(T const & element) -> void
{
    elements_.push_back(element);     // Element auf Stack legen
}


template <typename T>
auto Stack<T>::pop() -> T
{
    assert(!is_empty());              // Stack darf nicht leer sein!
    auto element = elements_.back(); // letztes Element
    elements_.pop_back();             // letztes Element l schen
    return element;                   // gel schtes Element zur ckgeben
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```
template <typename T>
class Stack {
    std::vector<T> elements_;
public:
    auto push(T const & t) -> void;
    auto pop() -> T;
    [[nodiscard]] auto is_empty() const -> bool;
};


template <typename T>
auto Stack<T>::push(T const & element) -> void
{
    elements_.push_back(element);     // Element auf Stack legen
}


template <typename T>
auto Stack<T>::pop() -> T
{
    assert(!is_empty());              // Stack darf nicht leer sein!
    auto element = elements_.back(); // letztes Element
    elements_.pop_back();             // letztes Element l schen
    return element;                   // gel schtes Element zur ckgeben
}


template <typename T>
auto Stack<T>::is_empty() const -> bool
{
    return elements_.empty();         // ist der Stack leer?
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```cpp
#include <string>
#include <iostream>
#include "stack.hpp"

auto main() -> int
{
    auto stack_int = Stack<int>{}; // Leeren Stack von Integern erzeugen.
    stack_int.push(42);            // 42 auf den Stack legen.
    std::cout                      // Letztes Element (42) loeschen und ausgeben.
        << stack_int.pop() << '\n';
    std::cout                      // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```
#include <string>
#include <iostream>
#include "stack.hpp"

auto main() -> int
{
    auto stack_int = Stack<int>{}; // Leeren Stack von Integern erzeugen.
    stack_int.push(42);            // 42 auf den Stack legen.
    std::cout                      // Letztes Element (42) loeschen und ausgeben.
        << stack_int.pop() << '\n';
    std::cout                      // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";

    auto stack_string = Stack<std::string>{}; // Leeren Stack von Strings erzeugen.
    stack_string.push("Hallo, Welt!");        // "Hallo, Welt!\n" auf den Stack legen.
    std::cout                                 // Letztes Element ("Hallo, Welt!")
        << stack_string.pop() << '\n';        // loeschen und ausgeben.
    std::cout                                 // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Klassentemplates

```cpp
#include <string>
#include <iostream>
#include "stack.hpp"

auto main() -> int
{
    auto stack_int = Stack<int>{}; // Leeren Stack von Integern erzeugen.
    stack_int.push(42);            // 42 auf den Stack legen.
    std::cout                      // Letztes Element (42) loeschen und ausgeben.
        << stack_int.pop() << '\n';
    std::cout                      // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";

    auto stack_string = Stack<std::string>{}; // Leeren Stack von Strings erzeugen.
    stack_string.push("Hallo, Welt!");        // "Hallo, Welt!\n" auf den Stack legen.
    std::cout                                 // Letztes Element ("Hallo, Welt!")
        << stack_string.pop() << '\n';        // loeschen und ausgeben.
    std::cout                                 // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";

    auto stack_string = Stack</*???*/>{}; // Leeren Stack von beliebigem Tpy erzeugen.
    stack_string.push(/*???*/);           // Wert auf den Stack legen.
    std::cout                             // Letztes Element loeschen und ausgeben.
        << stack_string.pop() << '\n';
    std::cout                             // Ausgeben ob der Stack leer ist.
        << "Der Stack ist "
        << (stack_int.is_empty() ? "" : "nicht ")
        << "leer!\n\n";
}
```

Vollständiges Beispiel: `https://godbolt.org/z/`

Überladung

- Funktionsüberladungen
  - Operatorüberladungen

Quellen

- FSST-Mitschrift 2. und 3. Klasse
- https://www.ics.uci.edu/~jajones/INF102-S18/readings/05_stratchey_1967.pdf
- https://www.youtube.com/watch?v=uTxRF5ag27A&list=PLrAXtmErZgOdP_8GztsuKi9nrraNbKKp4
- https://soundcloud.com/lambda-cast
- https://de.wikipedia.org/wiki/Polymorphie_(Programmierung)
- https://lec.inf.ethz.ch/ifmp/2019/slides/lecture14.handout.pdf
- https://courses.cs.washington.edu/courses/cse331/11wi/lectures/lect06-subtyping.pdf
- https://de.wikipedia.org/wiki/Vererbung_(Programmierung)
- https://wiki.haskell.org/Polymorphism
- http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf
- C++ Templates: The Complete Guide
- C++: Das umfassende Handbuch
- Grundkurs C++
- C++: Die Sprache der Objekte
- Exceptional C++: 47 engineering puzzles, programming problems, and solutions
- Effektiv C++. 50 Specific Ways to Improve Your Programs and Designs
- https://www.youtube.com/watch?v=HddFGPTAmtU
- https://www.bfilipek.com/2020/04/variant-virtual-polymorphism.html

Vollständiges Beispiel: https://godbolt.org/z/heEWxW