Fachhochschule Nordwestschweiz
Hochschule für Technik

Programmieren in C++

Christian Lang (Lac)

25. Oktober 2019

Vererbung Teil 2

Inhalt

- Interfaces
- Virtueller Destruktor
- Aufruf von virtuellen Methoden
- Forward Declarations
- Mehrfachvererbung
- Virtual Inheritance

Interfaces

- C++ kennt keinen “Interface” Typ
- abstrakte Klasse ohne jegliche Implementation
- pure virtual-Methoden

```
1  struct BicycleInterface {
2    virtual void Drive() const = 0;     // pure virtual
3  };
4
5  class FastBicycle : public BicycleInterface {
6   public:
7    void Drive() const override { ... }
8  };
9
10 std::shared_ptr<BicycleInterface> b = std::make_shared<FastBicycle>();
11 b->Drive();
```

Virtueller Destruktor

- keine automatisch synthetisierte Methode ist `virtual`
- aus Gründen der Performance → vtable erzeugt Overhead
- bei Interfaces ist deshalb der generierte Destruktor auch nicht `virtual`

```
1 std::shared_ptr<BicycleInterface> b = std::make_shared<FastBicycle>();
2 b.reset();
3 // Aufruf von BicycleInterface::~BicycleInterface
4 // anstatt von FastBicycle::~FastBicycle
```

Deshalb muss mindestens der Destruktor in jeder Basis-Klasse und jedem Interface `virtual` sein!

```
1 struct BicycleInterface {
2   virtual ~BicycleInterface() = default;  // nutze generierten Body
3   virtual void Drive() const = 0;
4 };
```

Aufruf von virtuellen Methoden

- virtuelle Methoden zeigen auf komplexe Klassenhierarchien
- sollten je nach Kontext nicht aufgerufen werden
- in Konstruktor sind abgeleitete Instanzen noch nicht initialisiert
- in Destruktoren sind abgeleitete Instanzen bereits zerstört

→ virtuelle Methoden werden nur bis zur aktuellen Klasse aufgelöst.

Best-Practice

nie direkt oder indirekt eine virtuelle Methode in einem Konstruktor oder Destruktor aufrufen

Beispiel: virtuelle Methoden in ctor/dtor

```cpp
struct Base {
  Base() { Init(); }
  virtual ~Base() { Purge(); }

 protected:
  // ruft nie Derived::Init()
  // auf sondern Base::Init()
  virtual void Init() { ... };

  // undefined behavior
  // weil kein Base::Purge()
  // implementiert ist
  virtual void Purge() = 0;
};
```

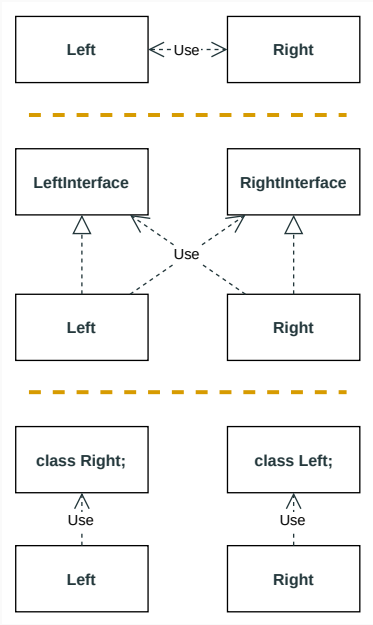
```cpp
struct Derived : Base {
  virtual ~Derived() = default;

 protected:
  void Init() override {
    Base::Init();
    // Init spezielle Resourcen
  }

  void Purge() override {
    // Purge spezielle Resourcen
    Base::Purge();
  }
};
```

Forward Declarations

- auch `#include`-Abhängigkeiten können zyklisch sein
- brechen mit Interfaces oder Forward Declaration

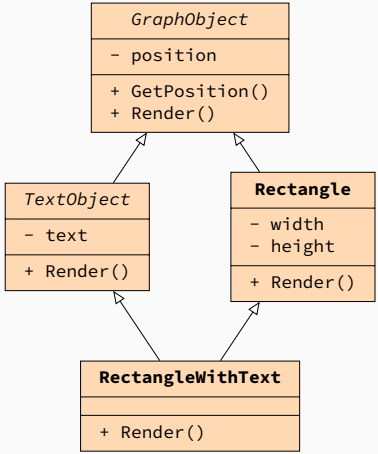
```
1  class Right;          // Forward declaration
2  class Left {
3    Right* r;
4    Right instance;     // Compiler-Fehler
5  };
```

- Compiler kennt nur Namen aber kein Memory-Layout
  → funktioniert nur mit Pointer/Referenzen
- Auch sinnvoll für Compiletime-Optimierung



Mehrfachvererbung

- Mehrfachvererbung von zwei Basen
- Zudem gemeinsame Basisklasse
  - Diamond Problem
- Methode `Render()` ist überall überschrieben
- Methode `GetPosition()` ist uneindeutig (Ambiguous)



Wenn möglich vermeiden

- Meistens nur für Interfaces sinnvoll

Probleme bei Mehrfachvererbung

```
1  Rectangle rect(0, 0, 20, 50);
2  RectangleWithText text(10, 5, 60, 60, "Text");
3  rect.Render();   // Aufruf von Rectangle::Render()
4  text.Render();   // Aufruf von RectangleWithText::Render()
5
6  Position Pos1 = rect.GetPosition();
7  Position Pos2 = text.GetPosition();    // Compile-Fehler
```

Ursache

a) `GetPosition()` ist als vererbte Implementation in `TextObject` und `Rectangle` vorhanden.

b) Dito für `position`

Compiler weiss nicht, welche Instanz er verwenden soll.

Virtual Inheritance

```
// Anstatt
class Rectangle : public GraphObject {
// virtuelle Vererbung (auch bei TextObject)
class Rectangle : virtual public GraphObject {
```

- Compiler legt nur eine Instanz der Basis `GraphObject` in `RectangleWithText` an
- Aufruf des Konstruktors von `GraphObject` wandert von `Rectangle` und `TextObject` nach `RectangleWithText`

```
class RectangleWithText : public TextObject, public Rectangle  {
public:
  RectangleWithText(int x, int y, int w, int h, std::string text)
      : TextObject(-2, -2, text)
      , Rectangle(-1, -1, w, h)
      , GraphObject(x, y)
  {}
};
```