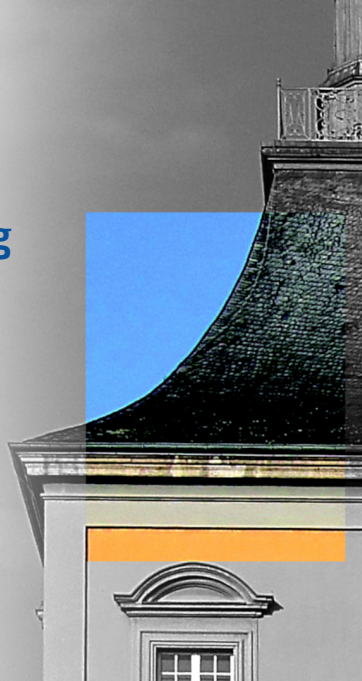
UNIVERSITÄT BONN

Algorithmen und Programmierung

Objektorientierte Programmierung

Dr. Felix Jonathan Boes
boes@cs.uni-bonn.de
Institut für Informatik



Objektorientierte Modellierung

Disclaimer

In dieser und den folgenden Vorlesungseinheiten diskutieren wir zuerst allgemeine Aspekte der objektorientierten Modellierung (z.T. in UML)

Anschließend diskutieren wir, wie diese Aspekte in einer objektorientierten Programmiersprache umgesetzt werden (hauptsächlich C++)

Objektorientierte Modellierung

Einleitung Unified Modeling Language (UML)

Offene Frage

In welcher Sprache drückt man objektorientierte Modellierung typischerweise aus?

UNIVERSITÄT BONN

Die Sprache UML

Die Sprache **Unified Modeling Language**, kurz UML, ist eine visuelle Sprache. UML erlaubt es, vielfältige Ideen im Bereich der objektorientierten Softwareentwicklung auszudrücken.

Literaturempfehlung:

- Martin Fowler. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.

UNIVERSITÄT BONN

Dialekte

Für wiederkehrende Modellierungsansätze, definiert UML **Dialekte**.

- Beim **UML-Sketching** werden Ideen konzeptionell diskutiert. Hier entsprechen die UML-Elemente den wesentlichen Projektkomponenten und beschreiben deren (konzeptionelles) Zusammenspiel.
- Beim **UML-Blueprinting** wird eine sehr konkrete Implementierung durch eine:n Softwaremodellierer:in mithilfe von UML kommuniziert.

UNIVERSITÄT BONN

Diagrammarten

Um typische Modellaspekte zu kommunizieren, definiert UML **Diagrammarten**. Beispielsweise:

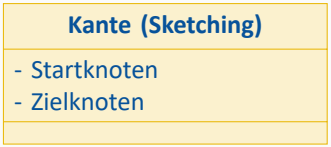
- Um den Entwurf von prozedualen Vorgängen zu kommunizieren, verwendet man **Activitätsdiagramme** und **Prozessdiagramme**.
- Um Aggregationen und Vererbungen von Objekt(typen) zu kommunizieren, verwendet man unter anderem **Klassendiagramme**.

In diesem Modul konzentrieren wir uns auf **Klassendiagramme** im **Sketching**- und im **Blueprintingdialekt**.

UNIVERSITÄT BONN

Beispiel I

Wir betrachten zwei simple Klassendiagramme für gerichtete Kanten, bevor wir **Interaktionsschnittstellen**, **Aggregation** und **Klassendiagramme** im größeren Detail besprechen.



Links ist ein Klassendiagram im unkonkreteren Sketchingdialekt und rechts ist ein Klassendiagram im konkreteren Blueprintingdialekt zu sehen.

UNIVERSITÄT BONN

Beispiel II

SimplerGerichteterGraph (Sketching)
- Knotenmenge - Kantenmenge
+ knotenEinfuegen(Knoten) + knotenEntfernen(Knoten) + kanteEinfuegen(Knoten, Knoten) + kanteEntfernen(Knoten, Knoten) + existiertKnoten(Knoten) : Boolean + existiertKante(Knoten, Knoten) : Boolean + getAuslaufendeKanten(Knoten) : Kantenmenge + getEinlaufendeKanten(Knoten) : Kantenmenge + drucke() + breitensuche(Knoten) : Knotenfolge + tiefensuche(Knoten) : Knotenfolge

Zusammenfassung

Objektorientierte Modellierung wird in UML ausgedrückt

Wir haben erste UML-Vokabeln kennen gelernt

Haben Sie Fragen?

Interaktionsschnittstelle, Aggregation und Zugriffsbeschränkung

Offene Fragen

Wie werden Interaktionsschnittstellen beschrieben?

Wie wird die Objektaggregation (Objektzusammensetzung) beschrieben?

Wie werden Zugriffsbeschränkungen beschrieben?

UNIVERSITÄT BONN

Überblick

Wir beantworten diese Fragen in dieser und der nächsten Vorlesungseinheiten. Dabei gehen wir wie folgt vor.

Objektorientierte Modellierung (mit UML)

- Interaktionsschnittstellen
- Objektaggregation
- Zugriffsbeschränkungen

Objektorientierte Programmierung (mit C++)

- Interaktionsschnittstellen, Objektaggregation, Zugriffsbeschränkungen

Technische Umsetzung im virtuellen Adressraum

- Interaktionsschnittstellen, Objektaggregation, Zugriffsbeschränkungen

Interaktionsschnittstelle, Aggregation und Zugriffsbeschränkung

Interaktionsschnittstellen in UML beschreiben

Offene Frage

Wie werden Interaktionsschnittstellen in UML beschrieben?

UNIVERSITÄT BONN

Interaktionsschnittstelle

Die **Interaktionsschnittstelle** eines Objekttyps ist wesentlich. Sie gibt an, wie mit Objekten dieses Typs sowie verwandten Typen interagiert werden kann.

Die Interaktionsschnittstelle ist durch die **Memberfunktionen** des Objekts gegeben. In objektorientierten Programmiersprachen werden Memberfunktionen auch **Methoden** genannt.

UNIVERSITÄT BONN

Typische Modellierungsfragen

Beim Entwurf der Interaktionsschnittstelle müssen unter anderem folgende Fragen beantwortet werden.

- Was sind die Parameter der Memberfunktionen?
- Was ist die Rückgabe der Memberfunktionen?
- Werden die übergebenen Parameter möglicherweise oder auf keinen Fall verändert?
- Wird durch die Operation der Zustand des Objekts möglicherweise oder auf keinen Fall verändert?

Die ersten beiden Fragen werden oft beim Sketching festgehalten. Die letzten beiden Fragen werden nur beim Blueprinting festgehalten. Falls man kein Blueprinting verwendet, werden die letzten beiden Fragen den aufmerksamen Programmier:innen überlassen.

UNIVERSITÄT BONN

Sketching vs Blueprinting

Im folgenden Beispiel wird Sketching und Blueprinting beispielhaft gegenüber gestellt.

KlasseX (Sketching)
...Membervariablen...
+ Memberfkt1(Para1, Para2, ...) : Rück1 + Memberfkt2(...Parameter...) : Rück2 + ...

KlasseX (Blueprinting)
...Membervariablen...
+ Memfkt1(in Par1, inout Par2, out Par3, ...) : Rück1 + Memfkt2(...Parameter...) : Rück2 {query} + ...

Die Interaktionsschnittstelle ist beim Sketching skizziert. Beim Blueprinting wird durch `in`, `out`, bzw. `inout` angegeben, ob der Parameter nur gelesen, nur verändert oder gelesen und verändert werden darf. Beim Blueprinting wird durch die Angabe des Keywords `{query}` hinter dem Rückgabetyp angegeben, dass die Operation den Objektzustand nicht ändert.

Haben Sie Fragen?

Interaktionsschnittstelle, Aggregation und Zugriffsbeschränkung

Aggregation in UML beschreiben

Offene Frage

Wie wird die Objektaggregation in UML beschrieben?

UNIVERSITÄT BONN

Objektaggregation und Objektzustand

Objekte sind **Instanzen** eines bestimmten Typs. Die **Aggregation** oder **Zusammensetzung** eines Objekts oder Typs gibt an aus welchen Typen dieser wiederum besteht. Die enthaltenen Typen nennt man **Attribute** oder auch **Membervariablen**, falls die Typen benannt sind.

Konsequenterweise besteht ein festes Objekt aus enthaltenen Objekten[1]. Die aktuellen Werte der enthaltenen Objekte bilden den aktuellen **Zustand** des umschließenden Objekts.

[1] Oder sie sind grundlegende Datentypen.

UNIVERSITÄT BONN

Beispiel I

Unsere einfache Graphenklasse besteht aus zwei Attributen. Der Zustand einer Instanz dieser Graphenklasse wird durch die Knotenmenge und die Kantenmenge gegeben.

SimplerGerichteterGraph (Sketching)
- Knotenmenge - Kantenmenge
... Memberfunktionen ...

Wir müssen sicherstellen, dass jeder, in einer Kante vorkommende, Knoten in der Knotenmenge enthalten ist. Diese Nebenbedingung kann in UML ausgedrückt werden (später evtl. mehr).

UNIVERSITÄT BONN

Beispiel II

Unsere Türme von Hanoi besteht aus einem Attribut. Der Zustand einer Instanz dieser Klasse wird durch drei Zahlenarrays gegeben. Wir müssen sicherstellen, dass die drei Zahlenarrays eine zulässige Konfiguration enthalten[2].

Türme von Hanoi (Sketching)
- Array von Zahlenarrays
+ bewege(von, nach) : bool + drucke()

[2] Vergleiche Vorlesung Imperative Programmierung 05.

Haben Sie Fragen?

Interaktionsschnittstelle, Aggregation und Zugriffsbeschränkung

Zugriffsbeschränkungen (ohne Vererbung) in UML

Offene Frage

Wie werden Zugriffsbeschränkungen in UML beschrieben?

UNIVERSITÄT BONN

Motivation

Jedes Objekt hat einen Zustand. Der Zustand ist die Wertekombinationen der Objektattribute. Nicht jede Wertekombination beschreibt einen zulässigen Zustand. Das haben wir bereits bei gerichteten Graphen und den Türme von Hanoi erkannt.

Wir wollen **methodisch sicherzustellen**, dass jedes Objekte nur **zulässige Zustände** hat und diese **nicht unbeabsichtigt** verletzt werden.

UNIVERSITÄT BONN

Kapselung

Um **methodisch sicherzustellen**, dass jedes Objekte nur **zulässige Zustände** hat

- verbieten wir den direkten Zugriff auf Objektattribute durch (fast) alle anderen Typen und
- definieren durch die Schnittestelle entsprechende Memberfunktionen, welche die erwünschte Zustandsänderung verarbeiten.

Dieses Modellierungsmuster nennt man **Kapselung** oder auch **Information Hiding**.

UNIVERSITÄT BONN

Typische Zugriffsbeschränkungen

Folgende Faustregel sorgt in fast allen Fällen für eine sinnvolle Kapelung.

- Alle Memberfunktionen des Objekts sind `public`, das heißt, jeder Typ kann mit dem Objekt über dieser Memberfunktionen interagieren.
- Alle Objektattribute sind `private`, das heißt, nur Objekte desselben Typs können auf diese Attribute zugreifen.

Wir lernen später weitere Zugriffseinschränkungen sowie partielle Einschränkungsaufhebung kennen.

UNIVERSITÄT BONN

Zugriffsbeschränkungen in UML ausdrücken

In UML drückt man durch das Präfix + aus, dass ein Attribut oder eine Memberfunktionen `public` ist. Durch das Präfix - drückt man aus, dass ein Attribut oder eine Memberfunktionen `private` ist.

Beispiel:

SimplerGerichteterGraph (Sketching)
- Knotenmenge - Kantenmenge
+ knotenEinfuegen(Knoten) + knotenEntfernen(Knoten) + kanteEinfuegen(Knoten, Knoten) + kanteEntfernen(Knoten, Knoten) + ...

UNIVERSITÄT BONN

Zugriffsbeschränkungen bei identischen Typen

Objekte desselben Typs können gegenseitig auf ihre Attribute zugreifen. Hier existieren keine Zugriffsbeschränkungen.

Dieses Verhalten ist gewünscht, denn beim Implementieren der Memberfunktionen soll (sowieso) darauf geachtet werden, dass der Zustand des Objekts zulässig ist.

Beispielsweise greift man beim Vergleichen von zwei Objekten desselben Typs oder bei einer Zuweisung auf alle Attribute beider Objekte zu.

UNIVERSITÄT BONN

Beispiel Vergleichsfunktion in Java

```java
public class Studi {
  private String vorname;
  private String nachname;

  // Konstruktor
  public Studi(String vorname, String nachname) { ... }

  // Vergleichsfunktion, welche bei der Expression
  //   studi1.equals(studi2);
  // verwendet wird.
  public boolean equals(Object other) {
    if (other == null || getClass() != other.getClass()) {
      return false;
    } else {
      Studi s = (Studi) other;
      // Zugriff auf Attribute desselben Typs sind immer erlaubt
      return vorname.equals(s.vorname) && nachname.equals(s.nachname);
    }
  }
}
```

UNIVERSITÄT BONN

Beispiel Zuweisungsoperator in C++

```
class Studi {
public:
  // Konstruktor
  Studi(const std::string& vorname, const std::string& nachname) { ... }

  // Falls nicht selbst explizit definiert, erstellt der Compiler
  // den folgenden "Zuweisungsoperator", welcher bei der Expression
  //   studi1 = studi2;
  // verwendet wird.
  Studi& operator=(const Studi& other) {
    vorname = other.vorname;
    nachname = other.nachname;
    ...
  }

private:
  std::string vorname;
  std::string nachname;
}
```

UNIVERSITÄT BONN

Öffentliche Attribute

Durch die Vergabe von Zugriffsbeschränkungen erklärt man den Nutzer:innen, wie mit dem Objekt interagiert werden darf. Durch typische Kapselung ist der Zugriff auf jedes Attribut unzulässig.

Es gilt die Faustregel: Wenn eine Attribut zugreifbar (`public`) ist, dann darf es zu jeder Zeit, von jedem anderen Objekt einen beliebigen Wert erhalten ohne dass der Zustand invalide wird.

Ein Objekt, dessen Attribute alle zugreifbar sind und das (im Sinne des Sketchings) über keine wesentlichen Memberfunktionen verfügt nennt man **Data Transfer Object**.

Data Transfer Objects werden sehr selten verwendet.

UNIVERSITÄT BONN

Beispiel

Die allgemeinen Koordinaten eines Punkts im dreidimensionalen Raum sind für alle Gleitkommawerte zulässig. Außerdem verfügen die Koordinaten über keine wesentlichen Memberfunktionen.

3DKoordinaten
+ x : Double + y : Double + z : Double

Also stellen diese Koordinaten ein Data Transfer Object dar.

UNIVERSITÄT BONN

Private Memberfunktionen

Durch die Vergabe von Zugriffsbeschränkungen erklärt man den Nutzer:innen, wie mit dem Objekt interagiert werden darf. Bei der typischen Kapselung ist der Zugriff auf jede Memberfunktion erlaubt.

Es gilt die Faustregel: Wenn die Implementierung der Schnittstelle eine interne Hilfsfunktion benötigt, dann sollte diese nicht von außen zugreifbar sein. So eine Hilfsfunktion wird üblicherweise als nicht öffentliche (`private`) Memberfunktion implementiert. Solche Hilfsfunktion werden beim UML-Sketching oft nicht genannt.

Ausblick
Zugriffsbeschränkungen bei Vererbung

Wie sich Zugriffsbeschränkungen bei verwandten Typen verhalten können und sollen diskutieren wir später.

Motivierend könnte man sich Folgendes fragen. Gegeben eine gerichtete Kante und eine gefärbte, gerichtete Kante.

- Auf welche Attribute soll die gerichtete Kante in der gefärbten, gerichteten Kante direkt zugreifen dürfen?
- Auf welche Attribute soll die gefärbte, gerichtete Kante in der gerichteten Kante direkt zugreifen dürfen?

Ausblick
Zugriffsbeschränkungen umgehen

Stark typisierte Programmiersprachen wie Java und C++ halten Zugriffsbeschränkungen strikt ein. Das kann in sehr wenigen Ausnahmen sehr unpraktisch sein. Um gesetzte Zugriffsbeschränkungen zu umgehen, stellen Java und C++ unterschiedliche Konstruktionen zur Verfügung.

- In Java können jegliche Zugriffsbeschränkungen mithilfe von *Reflections* umgangen werden.
- In C++ können jegliche Zugriffsbeschränkungen für ausgewählte **`friend`**-Funktionen aufgehoben werden.
- Außerdem lässt der C++-Standard eine sichere (aber komplizierte) Möglichkeit[3] offen, Zugriffsbeschränkungen vollständig aufzuheben.

[3] http://bloglitb.blogspot.com/2011/12/access-to-private-members-safer.html

Haben Sie Fragen?

Zusammenfassung

Wir haben gelernt, wie Interaktionsschnittstellen, Objektaggregationen und Zugriffsbeschränkungen in UML ausgedrückt werden

Um den korrekten Zustand eines Objekts methodisch sicherzustellen, verwendet man Kapselung

Data Transfer Objects sind Objekte, die nur öffentliche Attribute und keine wesentlichen Memberfunktionen besitzen

Interaktionsschnittstelle, Aggregation und Zugriffsbeschränkung

Schnittstellen, Aggregation und Zugriffsspezifikation (ohne Vererbung) in C++

Offene Fragen

Wie werden Interaktionsschnittstellen in C++ beschrieben?

Wie wird die Objektaggregation in C++ beschrieben?

Wie werden Zugriffsbeschränkungen in C++ beschrieben?

UNIVERSITÄT BONN

Überblick

In C++ werden Attribute durch Membervariablen definiert. Diese werden alle in der Klassendefinition angegeben.

In C++ wird die Interaktionsschnittstelle durch Memberfunktionen definiert. Diese werden alle in der Klassendefinition angegeben.

Die Zugriffsspezifikation auf Membervariablen und Memberfunktionen wird in der Klassendefinition durch **`private`**-Sections und **`public`**-Sections angegeben.

UNIVERSITÄT BONN

Beispiel

Das folgende Beispiel haben wir in der vergangenen Vorlesung bereits diskutiert.

```
class Kante {
public: // public-Section für (öffentliche) Memberfunktionen
  // Konstruktur
  Kante(const Knoten& startknoten, const Knoten& zielknoten);

  // Erhalte den Startknoten (als Referenz)
  const Knoten& get_startknoten() const;

  // Erhalte den Zielknoten (als Referenz)
  const Knoten& get_zielknoten() const;

  // Vergleichsoperator der die Indizes zweier Knoten vergleicht
  bool operator== (const Kante& other) const;

private: // private-Section für (nicht öffentliche) Attribute
  const Knoten start;
  const Knoten ziel;
};
```

UNIVERSITÄT BONN

Zugriff

Der gewünschte Zugriff auf eine Membervariable `x` oder eine Memberfunktionen `f` eines Objekts `ob`, wird durch die Expression `ob.x` bzw. `ob.f(...)` ausgedrückt.

Falls der Zugriff zulässig ist, kann die Expression ausgewertet werden. Falls der Zugriff verboten ist, kann die Expression nicht ausgewertet werden. Hier stoppt der Compiler die Übersetzung.

UNIVERSITÄT BONN

Data Transfer Objects

Erinnerung: *Data Transfer Objects* sind Objekte bei denen alle Membervariablen und Memberfunktionen öffentlich zugänglich sind und die keine wesentlichen Memberfunktionen besitzen.

In C++ werden Data Transfer Objects typischerweise als **`struct`** definiert. Hierbei wird statt dem Keyword **`class`** das Keyword **`struct`** verwendet. Außerdem wird typischerweise auf die **`private`**-Section verzichtet und es werden keine wesentlichen Memberfunktionen definiert.

UNIVERSITÄT BONN

Beispiel

```
struct 3DKoordinaten {
  double x;
  double y;
  double z;
};
```

Zusammenfassung

Wir haben gelernt, wie Interaktionsschnittstellen, Objektaggregationen und Zugriffsbeschränkungen in C++ ausgedrückt werden

Die Einhaltung der Zugriffsbeschränkungen wird durch den Compiler sicher gestellt

Haben Sie Fragen?