LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Skript zur Vorlesung:

Einführung in die Programmierung

WiSe 2009 / 2010

Prof. Dr. Christian Böhm
Annahita Oswald
Bianca Wackersreuther

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Gliederung

Warum objektorientierte Programmierung?

- Elementarer Bestandteil der Algorithmen-Entwicklung ist die Modellierung eines Ausschnitts aus der realenWelt.
- Bisherige Modellierungsansätze:
  - Funktionale Programmierung:
    Die Lösung eines Problems wird als funktionaler (mathematischer) Zusammenhang zwischen Ein- und Ausgabedaten modelliert.
  - Imperative Programmierung:
    Die Lösung eines Problems wird als Menge von sequentiellen Abarbeitungsschritten modelliert.
- Beides sind wichtige Ansätze. Mit zunehmender Komplexität der Probleme (und damit der Ausschnitte der realen Welt, die man modellieren muss) führen diese Ansätze allerdings zu immer komplexeren und unübersichtlicheren Programmen.

Warum objektorientierte Programmierung?

- Objektorientierte (OO) Programmierung ist *das* Programmier-paradigma der 90er Jahre.
- OO Programmierung ist ein weiterer Ansatz, Problemstellungen der realen Welt zu modellieren.
- Der OO Ansatz orientiert sich dabei an “Dingen” (Objekte, die gewisse Eigenschaften und ein gewisses Verhalten haben), die modelliert werden müssen.
- Genau genommen ist der OO Ansatz ein Konzept, komplexe Daten und Programmzustände zu modellieren. Die tatsächlichen Algorithmen werden meist mit einer Mischung aus funktionalen und imperativen Konzepten notiert.
- Beispiel Java: Die Algorithmen werden in der Regel mittels imperativer Konzepte implementiert, wogegen die Daten und deren Zustände OO modelliert sind.

Was ist objektorientierte Programmierung?

Betrachten wir als Beispiel die (fiktive) Immobilienfirma “Teuer und Hässlich”, die exklusive Einfamilienhäuser vermittelt. Wir sind damit beauftragt, eine Software zur Verwaltung dieser Häuser zu entwickeln.

- Für jedes Einfamilienhaus werden von der Firma verschiedene Daten gespeichert, z.B. Haustyp, Name des Besitzers, Adresse, Wohnfläche, Anzahl der Bäder, Verkaufspreis, etc.
- Ist ein potentieller Kunde gefunden, müssen die verschiedenen Daten für verschiedene Häuser abrufbar sein.
- Wie können wir die Daten modellieren? Was für Pobleme treten dabei auf?

Was ist objektorientierte Programmierung?

Modellierung der Daten:

- Da wir nur Arrays zur Speicherung von Mengen von Daten zur Verfügung haben und diese auch nur gleichartige Daten speichern können, müssen wir für alle Datenarten ein eigenes Array anlegen, d.h. ein Array, in dem für alle Häuser der Haustyp gespeichert ist, ein Array, in dem für alle Häuser der Name des Besitzers gespeichert ist, etc.

Problem:

- Offensichtlich ist diese Datenmodellierung sehr unübersichtlich und fehleranfällig, insbesondere wenn mehrere Programmierer an diesem Projekt arbeiten. **Warum**?

Lösung:

- OO Modellierung: Abstraktion und Kapselung zu Objekten.

Was ist objektorientierte Programmierung?

Die Idee der OO Programmierung (Modellierung) am Beispiel der (fiktiven) Immobilienfirma “Teuer und Hässlich”:

- Jedes Einfamilienhaus ist ein *Objekt* (eine Instanz einer allgemeinen Klasse von Objekten), z.B. ein Landhaus.
- Über dieses “Landhaus”-Objekt werden von der Firma verschiedene Daten gespeichert, z.B. Haustyp, Name des Besitzers, Adresse, Wohnfläche, Anzahl der Bäder, Verkaufspreis, etc. Diese Daten werden *Attributwerte* des Objekts genannt.
- Ist ein potentieller Kunde gefunden, wird eine *Operation* oder *Methode* benötigt, die auf das “Landhaus”-Objekt angewendet werden kann und dessen Kaufpreis ermittelt.
- Auf die Attributwerte eines Objekts kann nur über die entsprechenden Methoden zugegriffen werden. Dieses Prinzip nennt man (*Daten*-)*Kapselung*.

Was ist objektorientierte Programmierung?

- Außer dem “Landhaus”-Objekt bietet “Teuer und Hässlich” noch andere Einfamilienhäuser an:

:Einfamilienhaus
Haustyp = „Landhaus“ Besitzer = „Dr. Kaiser“ Adresse = „Solln“ Wohnfläche = 400 qm Anzahl Bäder = 3 Verkaufspreis = 2 Mio. EUR
Verkaufspreis_anzeigen():double

:Einfamilienhaus
Haustyp = „Stadthaus“ Besitzer = „Herzog“ Adresse = „Laim“ Wohnfläche = 220 qm Anzahl Bäder = 2 Verkaufspreis = 1 Mio. EUR
Verkaufspreis_anzeigen():double

:Einfamilienhaus
Haustyp = „Bungalow“ Besitzer = „Miller“ Adresse = „Freimann“ Wohnfläche = 250 qm Anzahl Bäder = 2 Verkaufspreis = 1,2 Mio. EUR
Verkaufspreis_anzeigen():double

- Alle Objekte besitzen zwar unterschiedliche Attributwerte, die aber alle vom gleichen Typ sind. Man spricht von *Attributen*.
- Alle Objekte besitzen die Operation “Verkaufspreis anzeigen”.
- Die Objekte werden daher zu einer *Klasse* “Einfamilienhaus” zusammengefasst.

Was ist objektorientierte Programmierung?

- Eine Klasse definiert die Attribute und Methoden ihrer Objekte.
- Der *Zustand* eines konkreten Objekts wird durch seine Attributwerte und Verbindungen (*Links*) zu anderen konkreten Objekten bestimmt.
- Das mögliche *Verhalten* eines Objekts wird durch die Menge von Methoden beschrieben.
- Die wichtigsten Konzepte der OO Programmierung sind:
  - Abstraktion
  - Kapselung
  - Wiederverwendung
  - Beziehungen
  - Polymorphismus

Abstraktion

- Eines der wichtigsten Konzepte der OO Programmierung ist die Trennung zwischen Konzept und Umsetzung (einem konkreten Objekt).
- Diese *Abstraktion* manifestiert sich in der Unterscheidung zwischen Objekt und Klasse. Ein Objekt ist ein existierendes "Ding" aus der Anwendungswelt des Programms (z.B. das Landhaus von Dr. Kaiser). Die Klasse ist eine Beschreibung eines oder mehrerer ähnlicher Objekte.
- Eine Klasse beschreibt mindestens drei wichtige Dinge:
  - Wie ist das Objekt zu bedienen?
  - Welche Eigenschaften hat das Objekt und wie verhält es sich?
  - Wie wird das Objekt hergestellt?

Abstraktion

- Jedes erzeugte Objekt einer Klasse hat seine eigene Identität.
- Beispiel: Lichtschalter in einem Haus
  - Alle Lichtschalter sind gleich zu bedienen.
  - Alle Lichtschalter sind gleich konstruiert.
  - Dennoch unterscheidet sich der Lichtschalter für die Flurbeleuchtung vom Lichtschalter fürs Badezimmer: Es ist nicht *derselbe* Lichtschalter.
- Abstraktion hilft, Details zu ignorieren, und reduziert damit die Komplexität des Problems. Dadurch werden komplexe Apparate und Techniken beherrschbar.

Kapselung

- Eine Klasse ist die Zusammenfassung einer Menge von Daten (Attributen, auch: *Membervariablen* oder *Instanzvariablen*) und darauf operierender Funktionen (Methoden).
- Die Attribute werden für jedes konkrete Objekt neu angelegt bzw. belegt. Sie repräsentieren den Zustand der Objekte.
- Die Methoden sind nur einmal realisiert / definiert und operieren bei jedem Aufruf auf den Daten eines bestimmten konkreten Objekts. Sie repräsentieren das Verhalten der Objekte.
- *Kapselung* bedeutet, dass (von gewollten Ausnahmen abgesehen) die Methoden die einzige Möglichkeit darstellen, mit einem konkreten Objekt zu kommunizieren und so Informationen über dessen Zustand zu gewinnen oder diesen zu verändern.

Kapselung

- Kapselung hilft, die Komplexität der Bedienung eines Objekts zu reduzieren.
- Durch Kapselung werden die Implementierungsdetails von Objekten verborgen.
- Dadurch können Daten nicht bewusst oder versehentlich verändert werden.
- Kapselung ist daher auch ein wichtiger Sicherheitsaspekt: Ein direkter Zugriff auf die Daten wird unterbunden, der Zugriff erfolgt nur über definierte *Schnittstellen*.

Wiederverwendung

- Durch Abstraktion und Kapselung wird die *Wiederverwendbarkeit* von Programmteilen gefördert.
- Beispiel: Sog. *Collections* sind Objekte, die Sammlungen anderer Objekte aufnehmen und auf eine spezielle Art und Weise verarbeiten können.
  - Collections sind meist sehr kompliziert aufgebaut.
  - Collections haben meist eine einfache Art der Bedienung (Schnittstelle).
  - Werden Collections als Klasse realisiert, werden durch die Kapselung die komplexen Details “wegabstrahiert”.
  - Dies erleichtert die Wiederverwendung: Wenn ein Programm eine spezielle Collection benötigt, muss ein Objekt der passenden Klasse erzeugt werden. Das Programm kann über die einfache Schnittstelle (Methoden der Klasse) darauf zugreifen.
- Wiederverwendbarkeit hilft, effizienter und fehlerfreier zu programmieren.

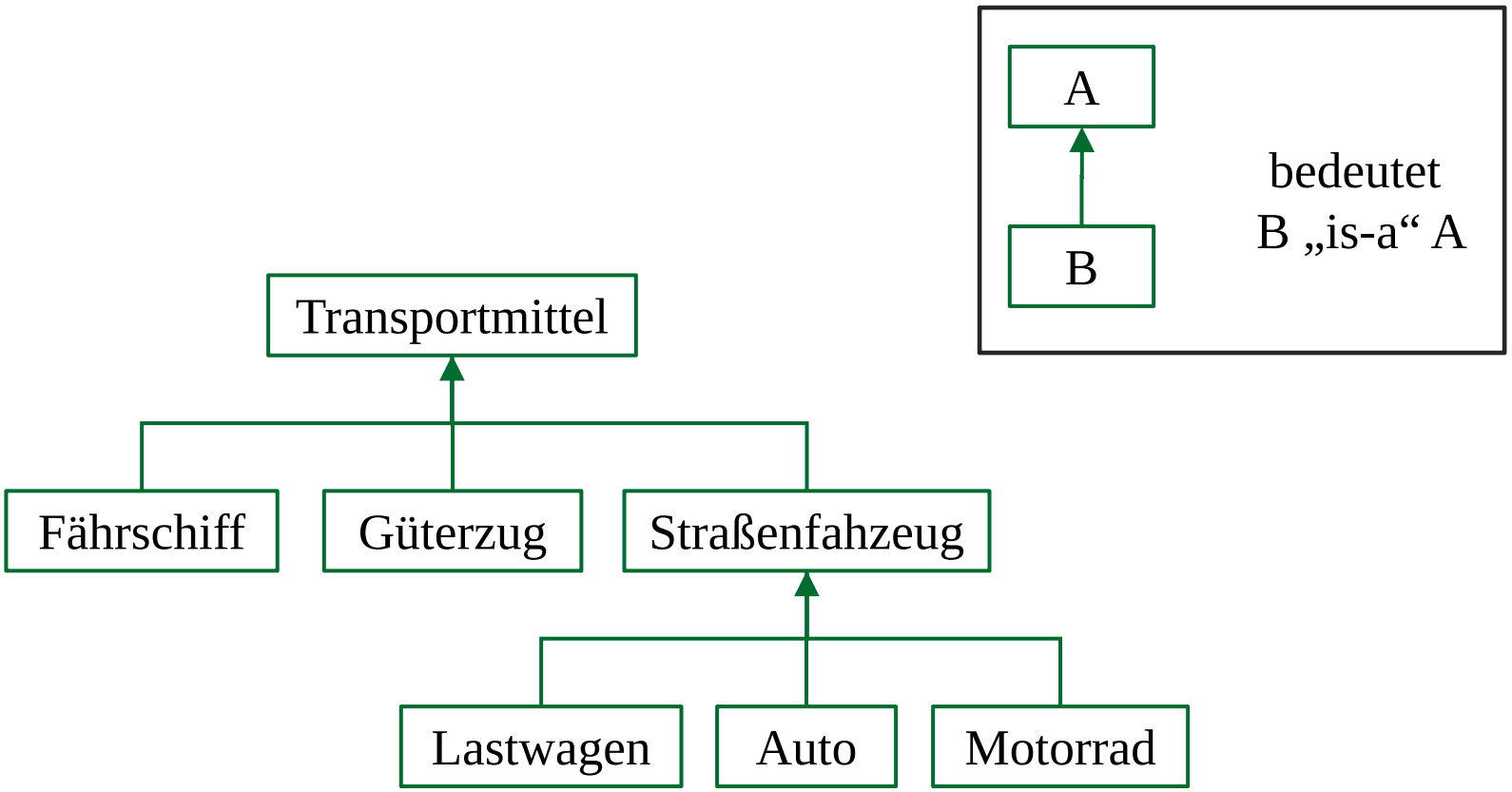
Beziehungen

- Klassen und Objekte existieren typischerweise nicht isoliert, sondern stehen in Beziehung zueinander.
- Beispiel:
  - Sowohl ein Motorrad, als auch ein Auto und ein Lastwagen sind Straßenfahrzeuge.
  - Ein Lastwagen kann möglicherweise Motorräder aufnehmen.
  - Ein Fährschiff ist ebenfalls ein Transportmittel und kann viele Autos, Lastwagen und Motorräder aufnehmen.
- Grundsätzlich gibt es in der OO Programmierung drei Arten von Beziehungen:
  - Generalisierung und Spezialisierung (“is-a”-Beziehungen),
  - Aggregation und Komposition (“part-of”-Beziehungen),
  - Verwendungs- und Aufrufbeziehungen.

Generalisierung und Spezialisierung

- Eine “is-a”-Beziehung zwischen zwei Klassen *A* und *B* sagt aus, dass “*B* ein *A* ist”, also *B* alle Eigenschaften von *A* besitzt (und darüberhinaus vermutlich noch ein paar mehr).
- *B* ist eine *Spezialisierung* von *A*.
- *A* ist eine *Generalisierung* von *B*.
- Beispiel:
  - Motorrad, Auto und Lastwagen sind zwar paarweise unterschiedlich, aber alle sind Straßenfahrzeuge.
  - Straßenfahrzeuge sind, genau wie Fährschiffe (und Güterzüge etc.), Transportmittel.
  - Die ensprechende “is-a”-Beziehung ist in der *Hierarchie* auf der folgenden Folie veranschaulicht.

Generalisierung und Spezialisierung



Generalisierung und Spezialisierung

- “is-a”-Beziehungen werden in der OO Programmierung durch *Vererbung* modelliert.
- Die speziellere Klasse wird dabei nicht komplett neu definiert, sondern von der allgemeineren Klasse *abgeleitet*.
- Die speziellere Klasse erbt alle Eigenschaften der allgemeineren Klasse (auch: *Vaterklasse*). Eigene Eigenschaften können nach Belieben hinzugefügt werden.
- Vererbungen können mehrstufig sein, d.h. eine abgeleitete Klasse kann wiederum Vaterklasse für andere abgeleitete Klassen sein (siehe *Vererbungshierarchie* auf der vorherigen Folie).
- Grundsätzlich kann eine Klasse auch von mehreren Vaterklassen abgeleitet sein. Beispielsweise ist ein Amphibienfahrzeug sowohl ein Wasserfahrzeug als auch ein Landfahrzeug. In diesem Fall spricht man von *Mehrfachvererbung*.

Aggregation und Komposition

- Eine “part-of”-Beziehung beschreibt die *Zusammensetzung* eines Objekts aus anderen Objekten.
- Ist diese Zusammensetzung essentiell für die Existenz des zusammengesetzten Objekts, spricht man von *Komposition*. Ein Güterzug besteht z.B. aus einer Lokomotive und mehreren Anhängern.
- Ohne diese Teile gibt es keine Güterzüge.
- Ist die Zusammensetzung nicht essentiell für die Existenz des zusammengesetzten Objekts, spricht man von *Aggregation*. Zwischen Straßenfahrzeug und Fährschiff besteht eine “part-of”-Beziehung, aber das Fährschiff kann auch leer fahren.
- **Bemerkung**: In den meisten OO Programmiersprachen werden Komposition und Aggregation gleich behandelt.

Verwendungs- und Aufrufbeziehungen

- Verwendungs- und Aufrufbeziehungen (*Assoziationen*) kommen dadurch zustande, dass z.B.
  - eine Methode einer Klasse *A* während ihrer Ausführung ein temporäres Objekt der Klasse *B* benutzt,
  - in der Parameterliste einer Methode einer Klasse *A* eine Variable definiert wird, die als Typ die Klasse *B* hat,
  - etc.

Polymorphismus

- Betrachten wir noch einmal Fährschiffe: Sie können Autos, Lastwägen und Motorräder – oder ganz allgemein Straßenfahrzeuge – aufnehmen.
- Um diese Aggregation zu beschreiben, genügt es also zu definieren, dass Fährschiffe Straßenfahrzeuge aufnehmen.
- *Polymorphismus* besagt, dass nicht nur Objekte der Klasse Straßenfahrzeuge aufgenommen werden können, sondern auch Objekte aller abgeleiteten Klassen von der Vaterklasse Straßenfahrzeuge, also auch Autos, Lastwägen und Motorräder.
- Die zusätzlichen Eigenschaften der abgeleiteten Klassen sind für das Fährschiff irrelevant.
- Andersherum funktioniert Polymorphismus nicht!

Polymorphismus

- Ein weiterer Aspekt ist, dass Objekte unterschiedlicher Klassen die gleiche Funktionalität besitzen können (die allerdings in jeder Klasse unterschiedlich realisiert ist).
- Beispiel:
  - Bei allen Straßenfahrzeugen gibt es die Funktionalität, die Anzahl der Reifen abzufragen (realisiert durch eine Methode “anzahlReifen”).
  - Diese Methode kann in allen drei Klassen (Auto, Lastwagen, Motorrad) denselben Namen haben. In allen drei Fällen steckt aber ein unterschiedlicher Algorithmus dahinter.
  - Diese Funktionalität kann bereits in der Vaterklasse zur Verfügung stehen und in den abgeleiteten Klassen *überschrieben* / *redefiniert* werden.
  - Wird im Zusammenhang eines Fährschiffes die Anzahl der Reifen eines Straßenfahrzeugs benötigt, das auf diesem Schiff gerade transportiert wird, wird dynamisch ausgewählt, welche Methode ausgeführt wird, ohne dass sich das Fährschiff darum kümmern muss (ist das Straßenfahrzeug z.B. ein Auto, wird “anzahlReifen” der Klasse Auto ausgeführt).