Analyse und Modellierung von Informationssystemen

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2014/15

Formales

Die Vorlesung findet jeweils Montags um 18:00 Uhr statt.
Am Mo, den 24. November 2014, findet keine Vorlesung statt.
Die Klausur findet am letzten Semestertermin statt, Dauer 90 Minuten.
Erlaubte Hilfsmittel: Vorlesungsskript, eigene Aufzeichnngen, Bücher, kein Computer oder internetfähiges Handy (z. B. Smartphone).
Die Folien zur Vorlesung finden sich i. A. am Vorabend der Vorlesung auf der Webseite
http://www.klaus-hoeppner.de

Lerninhalt der Vorlesung

- Objektorientierte Konzepte und Modelle für Analyse & Design (OOAD)
  - Klassen, Objekte, Attribute, Methoden
  - Vererbung, Abstraktion, Polymorphie
  - Assoziationen und Kompositionen
- OOAD-Standards für Prozess und Notation mit der UML
  - Notationen in der UML
  - Klassendiagramme
  - Use-Case-Diagramme
  - Zustandsdiagramme
- Softwarearchitektur
  - Schichtenmodell, Multi-Tier-Modelle
  - Verwandte Entwurfsmuster (Model-View-Controller)

Lerninhalt der Vorlesung (Forts.)

- Entwurfsmuster
  - Erzeugungsmuster (Fabriken, Singleton usw.)
  - Strukturmuster (Adapter, Brücke, Dekorierer, Proxy usw.)
  - Verhaltensmuster (Besucher, Beobachter, Zustand usw.)
- Anwendung von OOAD
  - Beispiele
  - Anknüpfungspunkte von OOAD zu relationalen Datenbanken, Konzept des objekt-relationalen Mappings

Literatur

- Heide Balzert; Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2; Spektrum Akademischer Verlag (2004); ISBN 978-3-8274-2903-2; 19,95 EUR.
- Bernd Oestereich; Analyse und Design mit der UML 2.5: Objektorientierte Softwareentwicklung; Oldenbourg Wissenschaftsverlag (2013); ISBN 978-3-486-72140-9; 54,95 EUR.
- Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides; Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software; Addison-Wesley Verlag (2010); ISBN 978-3-8273-3043-7; 49,80 EUR.
- Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates; Entwurfsmuster von Kopf bis Fuß; O'Reilly (2005); ISBN 978-3-89721-421-7; 48,00 EUR.

Was ist eine Programmiersprache?

Eine Programmiersprache ist eine *formale Sprache* zur Darstellung von Computerprogrammen. Sie vermittelt dem Computer den vom Menschen implementierten Algorithmus und die dabei beteiligten Daten.

Besondere Ausprägungen:

- Maschinensprache
- Hochsprachen, dabei insbesondere
  - Prozedurale Sprachen
  - Funktionale Sprachen
  - Objektorientierte Sprachen

Maschinensprache

- Unter Maschinensprache versteht man die Sprache, die direkt auf einer bestimmten Rechnerarchitektur (z.B. i*x*86, PowerPC, Alpha, Motorola 68000) ausführbar ist.
- Für den Menschen nicht lesbar
- Typische Befehle:
  - Lade den Inhalt von Speicherzelle *a* in Register *x*
  - Addiere zu Register *x* den Inhalt von Speicherzelle *b*
  - Schreibe den Inhalt von Register *x* in Speicherzelle *c*
  - Falls das Register *y* den Wert 0 hat, setze die Abarbeitung des Programmcodes an Stelle *z* fort

Hochsprachen

Menschen schreiben ein Programm in einer Hochsprache in Form eines Quelltextes.

Damit der Computer dieses Programm versteht, muss es in Maschinensprache umgewandelt werden.
Dies kann geschehen mit:

- Interpreter
  Der Quelltext wird während der Ausführung *interpretiert*, also zur Laufzeit in Maschinencode umgewandelt. Dies geschieht bei jeder Auführung neu.
- Compiler
  Der Quelltext wird einmalig vor der Ausführung *kompiliert*, d. h. in Maschinensprache übersetzt (und zusammen mit den verwendeten Bibliotheks-Funktionen *gelinkt*).

Hochsprachen (Forts.)

- Ausführen von Bytecode in einer virtuellen Maschine
  Einige Programmiersprachen, insbesondere Java, verwenden zwar Compiler, die allerdings nicht direkt Maschinencode sondern einen Zwischencode erzeugen, den so genannten Bytecode.

  Dieser Bytecode wird dann in der Laufzeitumgebung (Virtuelle Maschine) ausgeführt.

  Ein wesentlicher Vorteil des Bytecodes besteht darin, dass dieser in der Regel von der Maschine und Plattform unabhängig ist. Nachteil liegt in einem geringfügigen Geschwindigkeitsnachteil gegenüber der direkten Ausführung von Maschinencode, da im Fall der Virtuellen Maschine dieser zur Laufzeit aus dem Bytecode erzeugt werden muss.

Kleine Zeittafel von Hochsprachen

Jahr	Sprache
1954	FORTRAN
1960	COBOL
1965	BASIC
1971	Pascal
1972	C
1983	C++
1987	Perl
1991	Python
1995	Java
1997	PHP

OOP: Motivation

In prozeduralen Sprachen sind die Daten und Funktionen (innerhalb ihres jeweiligen Geltungsbereiches) *öffentlich*, d. h. innerhalb des Programms kann frei auf die Daten und Funktionen zugegriffen werden.

Dies setzt bei der Programmierung besondere Sorgfalt voraus.

Beispiel: Geldautomat
Ein Geldautomat verwaltet die Kontostände der Kunden und zahlt Geld aus.
Hier ist der Programmierer selbst dafür verantwortlich, dass nur dann Geld ausgezahlt wird, wenn dieser Betrag gleichzeitig dem Konto des Kunden belastet wird (und umgekehrt).

OOP: Grundprinzip

- Objektorientierte Programmierung (OOP) ist eine Methode zur Strukturierung von Programmen, bei der Daten und Programmlogik eine Einheit bilden.
- Objektorientierte Sprachen nutzen als Organisationsstruktur *Klassen*
- Klassen beschreiben die gemeinsamen Eigenschaften *(Attribute)* und Fähigkeiten *(Methoden)* von gleichartigen *Objekten*.
- Ein einzelnes Objekt, das durch die Klasse *K* beschrieben wird, wird häufig auch als *Instanz* der Klasse *K* bezeichnet.

Beispiel

- Klasse: Mensch
- gemeinsame Eigenschaften: Haarfarbe, Augenfarbe, Hautfarbe, Größe, Gewicht, . . .
- gemeinsame Fähigkeiten: hören, riechen, sehen, sprechen, . . .

Hans Müller ist eine Instanz der Klasse Mensch mit Haarfarbe *blond*, Augenfarbe *blau*, Hautfarbe *weiß*.
Auch wenn alle Menschen Instanzen der gleichen Klasse Mensch sind, so sind sie doch eindeutig unterscheidbar.

Komplexes Beispiel

- Klasse: Geldautomat
- Daten:
  - Geldbestand im Automaten
  - Liste der Kontostände aller Kunden
- Methoden:
  - gebe Geld aus
  - belaste ein Konto
  - zeige Kontostand
  - zahle Geld vom Konto aus

Private Daten und Methoden

Als *privat* deklariert sind:

- Geldbestand
- Liste der Kontostände
- gebe Geld aus
- belaste Konto

Diese Daten und Methoden sind nur *innerhalb* der Klasse zugänglich.
Dem Kunden bleibt z. B. der Geldbestand im Automaten verborgen. Genauso ist es nicht möglich, die Methode *gebe Geld aus* direkt aufzurufen (und so eine Auszahlung zu erreichen, ohne dass diese einem Konto belastet wird).

Öffentliche Methoden

Als *öffentlich* deklariert sind:

- zeige Kontostand
- zahle Geld vom Konto aus
  Implementation:
  - Falls Geldbestand ausreichend und Konto gedeckt:
    - gebe Geld aus
    - belaste Konto

Hierdurch ist sicher gestellt, dass Geld nur dann ausgezahlt wird, wenn das entsprechende Konto belastet wird (und umgekehrt).

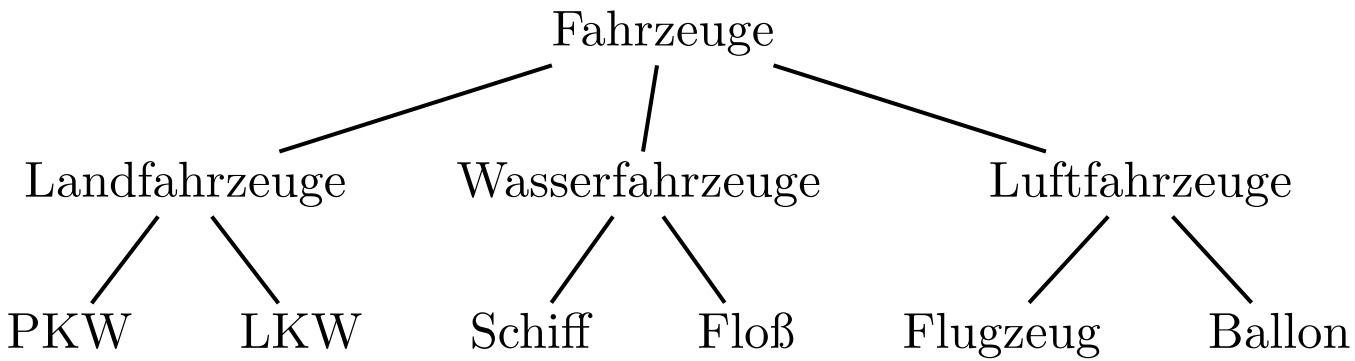
OOP: Grundlegende Eigenschaften

Abstraktion Jedes Objekt „kommuniziert“ mit andern Objekten durch die in der Klassendefinition definierten Schnittstellen. Die eigentliche Implementation der Methoden bleibt verborgen.

Kapselung Die Kommunikation über öffentliche Schnittstellen sorgt dafür, dass der interne Zustand der Objekte nicht in unerwarteter Weise geändert werden kann.

OOP: Grundlegende Eigenschaften

Vererbung Klassen können von anderen Klassen erben. Dies bedeutet, dass eine neue Klasse die Eigenschaften und Methoden der Ursprungsklasse *erbt*, d. h. übernimmt, ohne dass sie noch einmal neu implementiert werden müssen.
In der neuen Klasse können dann neue Methoden und Eigenschaften definiert werden, weiterhin können z. B. Methoden der Ursprungsklasse überschrieben oder erweitert werden.
Bei der Vererbung entsteht in der Regel aus einer allgemeinen Klasse eine spezialisiertere Klasse



Die Klasse *Fahrzeug* besitzt die Eigenschaft *Geschwindigkeit* und die Methode *(horizontal) bewegen*.
Die Klasse *Luftfahrzeug* besitzt zusätzlich die Eigenschaft *Höhe* und die Methoden *steigen* und *sinken*.

Vererbung

Eine Klasse kann eine speziellere Version einer allgemeineren Klasse sein, z. B.

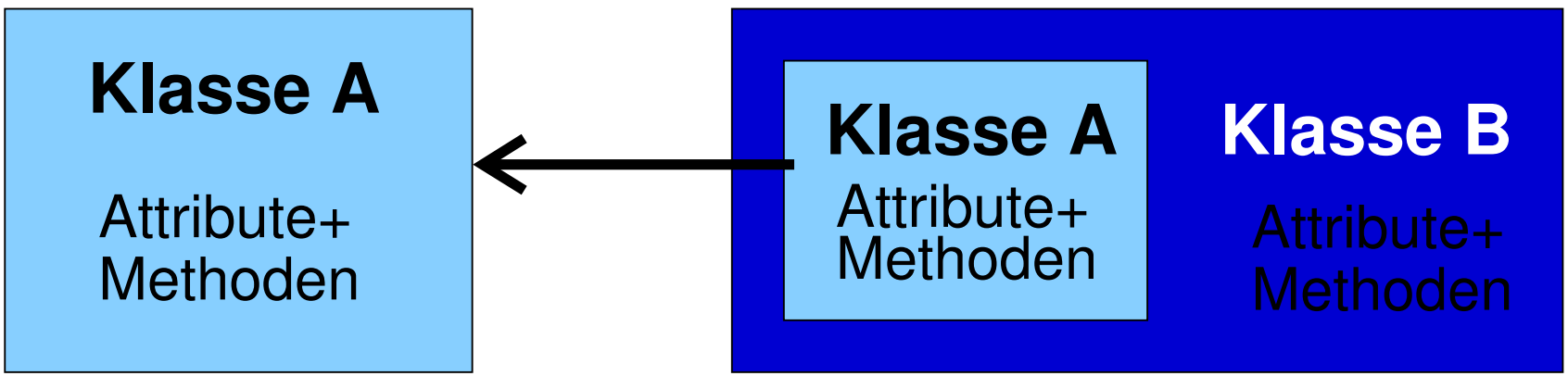
- Kfz (allgemein) – Pkw, Lkw (speziell)
- Konto (allgemein) – Girokonto, Sparkonto (speziell)

Diesen Mechanismus nennt man *Vererbung*, d. h. die spezielle Klasse *erbt* die Attribute und Methoden der allgemeinen Klasse.
In der abgeleiteten Klasse können neue Attribute/Methoden hinzu kommen, aber auch Attribute/Methoden der Basisklasse redefiniert werden.

Beispiel: In einer Leihbibliothek gibt es Waren in Form von Büchern, DVDs, CDs etc.:

Inventar – Buch, DVD, CD, . . .

Schaubild


Klasse A
Attribute+
Methoden
Klasse A
Attribute+
Methoden
Klasse B
Attribute+
Methoden

Definition: Polymorphie

Der Begriff *Polymorphie* (manchmal auch exakter: Laufzeit-Polymorphie) bei der Ausführung von Methoden bedeutet:

- Es wird erst zur Laufzeit bestimmt wird, welche Implementierung für eine bestimmte Methode aufzurufen ist.
- Es kann also vom Programmablauf abhängig sein, aus welcher Klasse die Methode verwendet wird.
- Polymorphie ist ein wichtiger Bestandteil der objektorientierten Programmierung.

Häufig wird Polymorphie als *späte Bindung* bezeichnet.

Abstrakte Klassen

Klassen in Java können Methoden besitzen, die *abstrakt* sind.

Abstrakte Methoden besitzen nur einen (als abstrakt gekennzeichneten) Deklarationskopf, aber keinen Code.

Klassen mit mindestens einer abstrakten Methode *müssen* als abstrakt markiert sein.

Von abstrakten Klassen können keine Instanzen erzeugt werden.

Interfaces

Manchmal implementieren abstrakte Klassen keinerlei Code, sondern *deklarieren* nur nach außen sichtbare Methoden.

Für solche Zwecke gibt es in Java ein spezielles Sprachelement, die so genannten *Interfaces*.

Ein Interface wird in Java wie eine Klasse definiert (nur mit dem Schlüsselwort `interface` statt `class`).

Namen von Interfaces enden häufig auf `-able`.

Programmieren gegen Interfaces

Interfaces in Java dienen dazu, dem Programmierer die Möglichkeit zur Trennung von *Schnittstelle* und *Implementierung* zu geben.

Hierbei bietet es sich an, im Code dann Objektreferenzen vom Typ des *Interfaces* zu benutzen.
Vorteile:

- Der Programmierer besitzt die Garantie, dass die referenzierten Objekte die im Interface deklarierten Methoden unterstüzten,
- aber die eigentliche Implementierung befindet sich in der Klasse, von der das referenzierte Objekt instanziert wurde, und bleibt so gekapselt.