Analyse und Modellierung von Informationssystemen

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2014/2015

UML – Einführung

Klassendiagramme in der UML

Relationen zwischen Klassen

Einführung: Entwurfsmuster

Erzeugungsmuster

Fabrik-Muster

UML: Grundsätzliches

Die UML (*Unified Modeling Language*) ist ein grafisches Notations-Schema, das mittlerweile im objektorientierten Software-Design (und nicht nur dort) weit verbreitet ist.

Exakter: Die UML beschreibt ein System (daher auch *Language*), dieses Modell wird durch verschiedene Diagramm-Arten repräsentiert.

Hierbei wird im Wesentlichen zwischen zwei Arten von Diagrammen unterschieden:

- Strukturdiagramme, die (weitgehend statisch) die Architektur des Systems beschreiben, dies sind insbesondere Klassen- und Objektdiagramme,
- Verhaltensdiagramme, z. B. Aktivitäts-, Kommunikations- und Zustandsdiagramme.

Initiatoren der UML

Die UML und insbesondere die Vorgängerprojekte gehen zurück auf die *Three Amigos*:

- James Rumbaugh, Pionier in *Object-Modeling Technique* (OMT)
- Grady Booch, Pionier in *Object-Oriented Design* (OOD)
- Ivar Jacobson, Pionier in *Object-Oriented Software Engineering* (OOSE)

weitgehend bezahlt durch die Firma Rational Software Corporation, mittlerweile Teil von IBM.

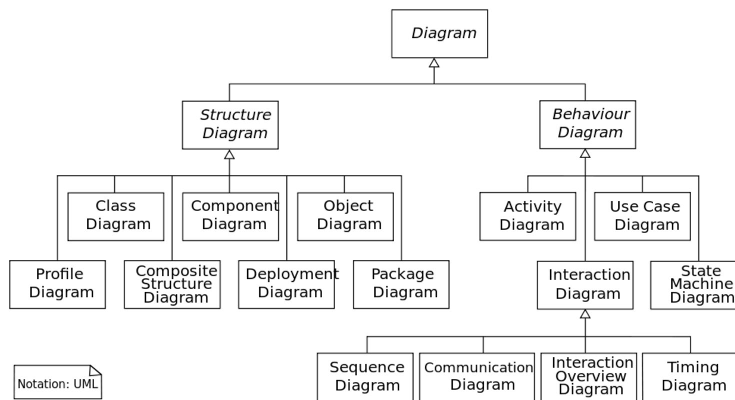
Geschichte der UML

- ab 1994 Vorläuferarbeiten durch die drei Amigos, schließlich eingereicht bei der *Object Management Group* (OMG).
- UML Version 1.1 wurde 1997 offiziell als OMG-Standard freigegeben.
- seit 2005 ist UML 2.x der offizielle OMG-Standard, seit 2011 gilt Version 2.4.1 (formal version), auch ISO-Standard. UML 2.5 liegt seit Ende 2013 in Version Beta 2 vor (adopted version).
- Mittlerweile ist UML auch ein ISO-Standard.

Diagrammtypen

Strukturdiagramm	Verhaltensdiagramm
Klassendiagramm	Aktivitätsdiagramm
Komponentendiagramm	Kommunikationsdiagramm
Kompositionsstruktur	Interaktionsübersicht
Verteilungsdiagramm	Sequenzdiagramm
Objektdiagramm	Zustandsdiagramm
Paketdiagramm	Zeitdiagramm
Profildiagramm	Anwendungsfalldiagr.

Schaubild der Diagrammtypen



Klassendiagramme

Eine Klasse wird durch ein Rechteck dargestellt (bei parametrisierten Klassen mit Angabe des Platzhalters für den parametrisierten Typ oben rechts). Dann werden, durch horizontale Linie getrennt, Attribute und Methoden der Klasse angegeben.

Bei Attributen wird der Name und Typ, optional Initialwert und Bedingungen (*constraints*) angegeben, bei Methoden neben dem Namen der Methode die Liste der Parameter und der Typ des Rückgabewertes.

Abstrakte Klassen und Methoden werden kursiv dargestellt, ggfs. (insbesondere bei Handschrift) noch explizit als „{abstrakt}“ markiert. So genannte «Stereotype» kennzeichnen Interfaces, aber ggfs. zur Verdeutlichung z. B. Control- oder Entity-Klassen.

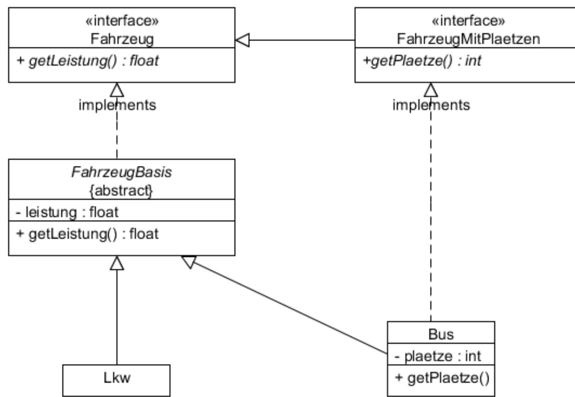
Notation der Sichtbarkeit

Notation	Bedeutung
`-attr`	privat
`#attr`	protected
`~attr`	package
`+attr`	öffentlich
<u>`attr`</u>	statisch
`/attr`	abgeleitetes Attribut

Die gleichen Notationen (außer abgeleitet) gelten auch für Methoden.

Vorlesung 02

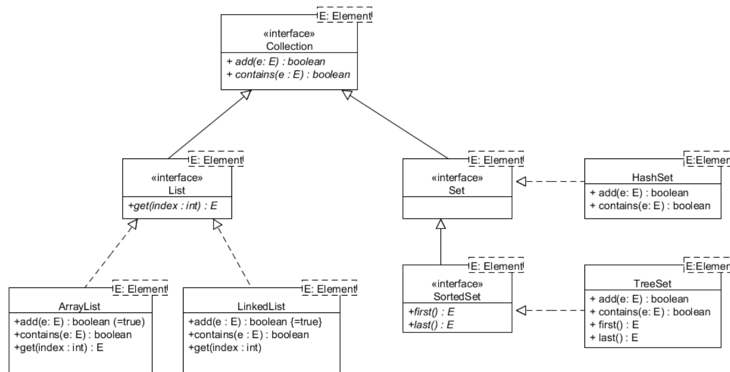
Beispiel



Analyse

- Das UML-Diagramm stellt zwei Interfaces dar, wobei das eine das andere erweitert.
- Die Klasse `FahrzeugBasis` implementiert ein Interface, ist aber abstrakt. Es dürfen also keine Instanzen der Klasse gebildet werden.
- Die konkreten Klassen `Lkw` und `Bus` sind konkrete Klassen, die jeweils eines der beiden Interfaces implementieren und von der abstrakten Klasse erben.

Praxis-Beispiel: Java-Collections, parametrisiert



Assoziationen

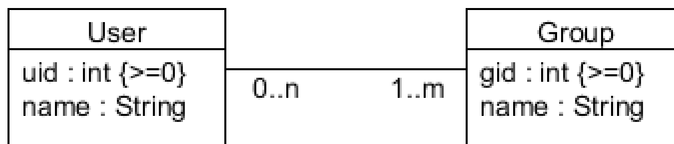
Ein Klassendiagramm kann auch Assoziationen zwischen Klassen darstellen. Hierbei wird unterschieden zwischen Assoziationen vom Typ:

- 1:n, bei denen jeweils ein Objekt der ersten Klasse mit n Objekten der zweiten Klasse assoziiert sein kann.
- n:1 für die Gegenrichtung.
- m:n in einem Fall, wo jeweils ein Objekt der ersten Klasse mit verschiedenen Objekten der zweiten, in Gegenrichtung aber ebenfalls jedes Objekt der zweiten mit verschiednen Objekten der ersten Klasse assoziiert sein darf.

UML – Einführung Klassendiagramme in der UML Relationen zwischen Klassen Einführung: Entwurfsmuster Erzeugungsmuster

Beispiel

Unter Unix/Linux kann ein User einer oder mehr Gruppen angehören, in jeder Gruppe können sich beliebig viele (auch kein) User befinden.



UML – Einführung Klassendiagramme in der UML Relationen zwischen Klassen Einführung: Entwurfsmuster Erzeugungsmuster

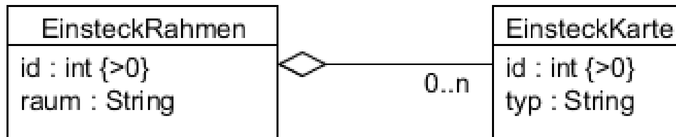
Aggregation

Ein Sonderfall einer Assoziation ist die Aggregation, in der die Hauptgestalt eines Objekt durch die Objekte ausgemacht wird, die mit ihm assoziiert sind.

Beispiel:
Die Eigenschaften eines Elektronikrahmens mit Einschüben für Einsteckkarten wird im Wesentlichen dadurch bestimmt, welche Einsteckkarten in ihm verbaut sind.
Die Einsteckkarten sind jedoch von dem Einsteckrahmen unabhängig, eine Einsteckkarte kann sich also auch uneingebaut im Lager befinden und auf den Einbau warten.

Diagramm

Die Aggregation wird mit einer offenen Raute in der UML-Notation dargestellt.



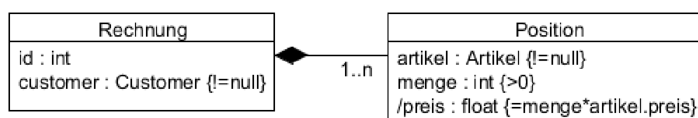
Komposition

Die Komposition unterscheidet von der Aggregation dadurch, dass die assozierten Objekte von dem Objekt abhängen, also ohne dieses nicht lebensfähig sind.

Beispiel:
Eine Rechnung besteht aus einer oder mehreren Rechnungspositionen. Die Positionen sind ohne die Rechnung, zu der sie gehören, sinnlos.
Wird also z. B. eine Rechnung nach Ablauf der buchhalterischen Aufbewahrungsfrist aus der Datenbank gelöscht, so werden auch die zugehörigen Rechnungspositionen entfernt. Bei Datenbanken wird dies i. A. als Kaskadierung bezeichnet.

Diagramm

Die Komposition wird mit einer gefüllten Raute in der UML-Notation dargestellt.



Entwurfsmuster: Die Gang of Four

1994 veröffentlichten Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides das Buch

> *Design Patterns, Elements of Reusable Object-Oriented Software*

das eine Sammlung von wiederverwendbaren Entwurfsmustern für die objektorientierte Programmierung enthält. Das Vorwort zum Buch schrieb Grady Booch, einer der Erfinder der UML. Das Buch führt die Entwurfsmuster anhand eines konkreten Beispiels, der Entwicklung eines Editors, ein.

Das Buch gehört noch heute zu einem der Standardwerke der Softwareentwicklung, die Autoren werden oft kurz als *Gang of Four* (GoF oder Go4) bezeichnet.

UML – Einführung Klassendiagramme in der UML Relationen zwischen Klassen Einführung: Entwurfsmuster Erzeugungsmuster

Die Autoren

Erich Gamma ist ein Schweizer Software-Entwickler (geb. 1961), der lange für IBM arbeite. Er ist Koautor von JUnit, war Eclipse-Entwickler, mittlerweile arbeitet er für Microsoft (Visual Studio).

Richard Helm arbeitete früher für IBM, heute für die Boston Consulting Group in Sidney.

Ralph Johnson (geb. 1955) arbeitet an der University of Illinois und war Pionier der Softwareentwicklung in Smalltalk.

John Vlissides (geb. 1961, gest. 2005) arbeitete bei IBM.

UML – Einführung Klassendiagramme in der UML Relationen zwischen Klassen Einführung: Entwurfsmuster Erzeugungsmuster

Klassifizierung der Entwurfsmuster

Die im Buch vorgestellten Entwurfsmuster werden in drei Kategorien eingeteilt:

Erzeugungsmuster die der Erzeugung von Instanzen einer Klasse dienen. Hierzu gehören insbesondere in vielen Frameworks verwendete Entwurfsmuster wie (abstrakte) Fabriken oder das Singleton.

Strukturmuster die Objekte oder Klassen (z. B. durch Komposition) zu einer Struktur zusammenfassen. Beispiele hierfür sind Adapter, Dekorierer oder Brücken.

Verhaltensmuster bestimmen das Verhalten von Objekten bei der Kommunikation untereinander. Eines der Beispiele ist die Auswahl einer Teilfunktionalität eines Objektes zur Laufzeit (Strategie-Muster) oder ein per Beobachter-Muster realisierter Publish-Subscribe-Mechanismus.

Erzeugungsmuster

Allgemein bekannt sind die zwei folgenden Erzeugungsmuster:

- Fabrikmuster dienen der Erzeugung von Instanzen einer Klasse. Eine solche Fabrik kann einerseits eine eigene Klasse sein, oder eine statische Methode der Klasse, von der Instanzen erzeugt werden sollen.
- Das Singleton ist ein Spezialfall, in dem von einer Klasse nur eine einzige Instanz erzeugt werden kann/darf. Hierbei ist bei nebenläufiger Programmierung besondere Vorsicht angesagt, so dass hier im Code besondere Maßnahmen zu treffen sind (und teilweise Änderungen im Speichermanagement von Java hierdurch motiviert wurden).

Schaubild 1

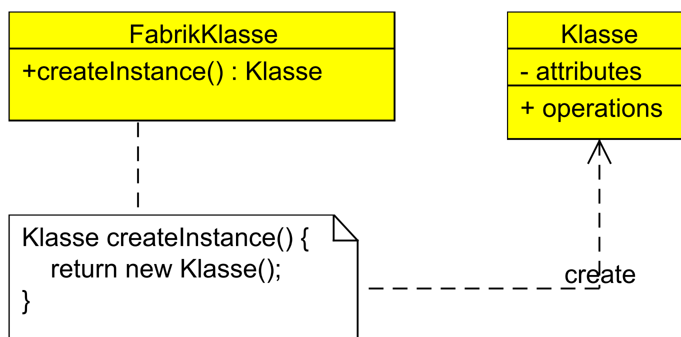
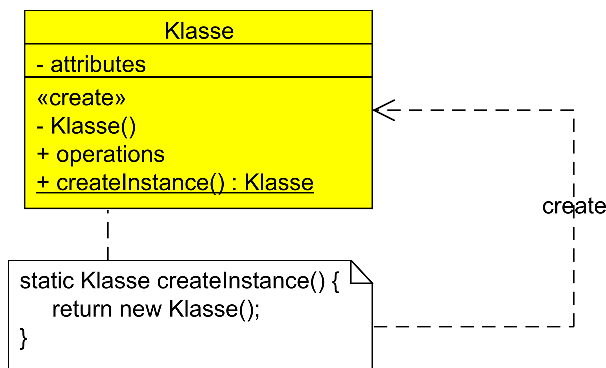


Schaubild 2



Gründe für den Einsatz einer Fabrik

- Eine Instanz kann auf verschiedene Arten definiert werden, die durch unterschiedliche Fabrikmethoden repräsentiert werden.
- Die Fabrikmethode erzeugt Instanzen einer Kindklasse/einer konkreten Klasse, liefert aber als Rückgabe eine Referenz vom Typ der Elternklasse/eines Interfaces zurück.
- Die Instanzen werden nur zurück geliefert, wenn bestimmte Kriterien erfüllt sind.
- Die Instanzen werden anhand der Informationen aus einer externen Datenquelle (Datenbank, XML-Datei) erzeugt.

Beispiel: Kredit

```
public class Kredit {
   final static float MINDEST_TILGUNG = 0.01F;

   float zinssatz;
   float kreditsumme;
   float rate;

   public static Kredit createKredit(float zinssatz,
         float kreditsumme, float rate) {
      Kredit kredit = new Kredit(zinssatz, kreditsumme, rate);
      if (kredit.getTilgung()/kredit.getKreditsumme()>=MINDEST_TILGUNG) {
         return kredit;
      } else {
         return null;
      }
   }

   private Kredit(float zinssatz, float kreditsumme, float rate) {
      this.zinssatz = 0.01F*zinssatz;
      this.kreditsumme = kreditsumme;
      this.rate = rate;
   }
```


Beispiel: Kredit (Forts.)

```
   public float getZinssatz() {
      return zinssatz;
   }

   public float getKreditsumme() {
      return kreditsumme;
   }

   public float getRate() {
      return rate;
   }

   public float getZinsen() {
      return zinssatz*kreditsumme;
   }

   public float getTilgung() {
      return rate-getZinsen();
   }
}
```

Beispiel: Kreis

```
public class Kreis {
   private double radius;

   public static Kreis createFromRadius(double radius) {
      if (radius<0) throw new RuntimeException();
      return new Kreis(radius);
   }

   public static Kreis createFromFlaeche(double flaeche) {
      if (flaeche<0) throw new RuntimeException();
      return new Kreis(Math.sqrt(flaeche/Math.PI));
   }

   private Kreis(double radius) {
      this.radius = radius;
   }
```


Beispiel: Kreis (Forts.)

```
   public double getRadius() {
      return radius;
   }

   public double getFlaeche() {
      return radius*radius*Math.PI;
   }
}
```