

2016

UML-Grundlagen



Karl Steinam
Klara-Oppenheimer-Schule
20.11.2016

Einführung in UML

Überblick

Die Unified Modeling Language (vereinheitlichte Modellierungssprache), kurz UML, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen. Sie wird von der Object Management Group (OMG) entwickelt und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.4.1 standardisiert). Im Sinne einer Sprache definiert UML dabei Bezeichner für die meisten bei einer Modellierung wichtigen Begriffe und legt mögliche Beziehungen zwischen diesen Begriffen fest. UML definiert weiter grafische Notationen für diese Begriffe und für Modelle statischer Strukturen und dynamischer Abläufe, die man mit diesen Begriffen formulieren kann. (Quelle: Wikipedia)

Sie ist damit für die objektorientierte Programmierung das, was Struktogramme / Programmablaufpläne und ähnliche Konzepte für die strukturierte Programmierung waren,

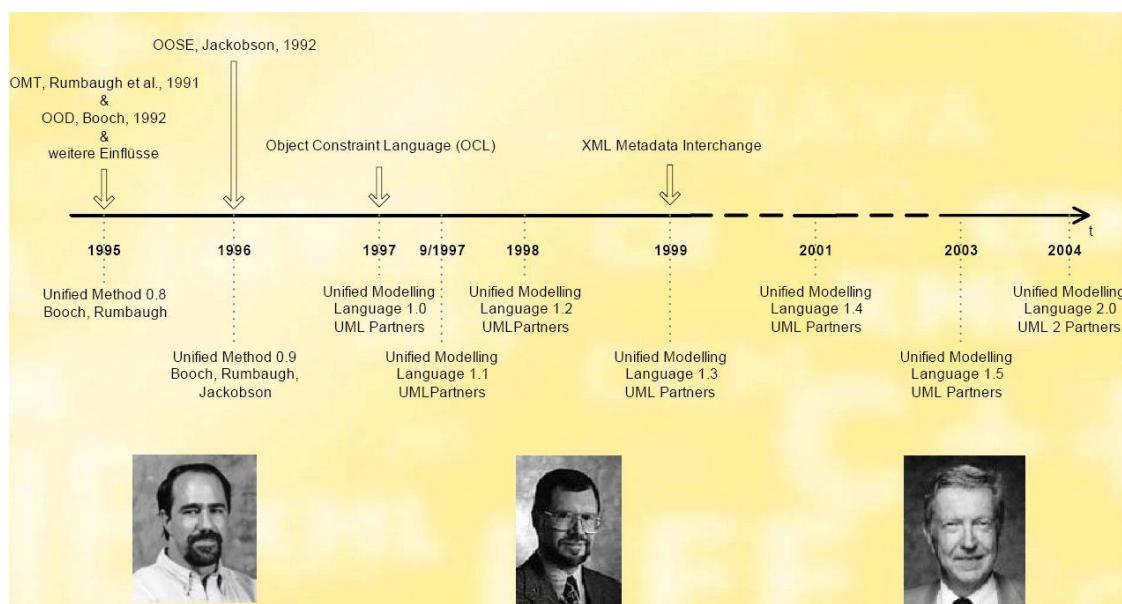
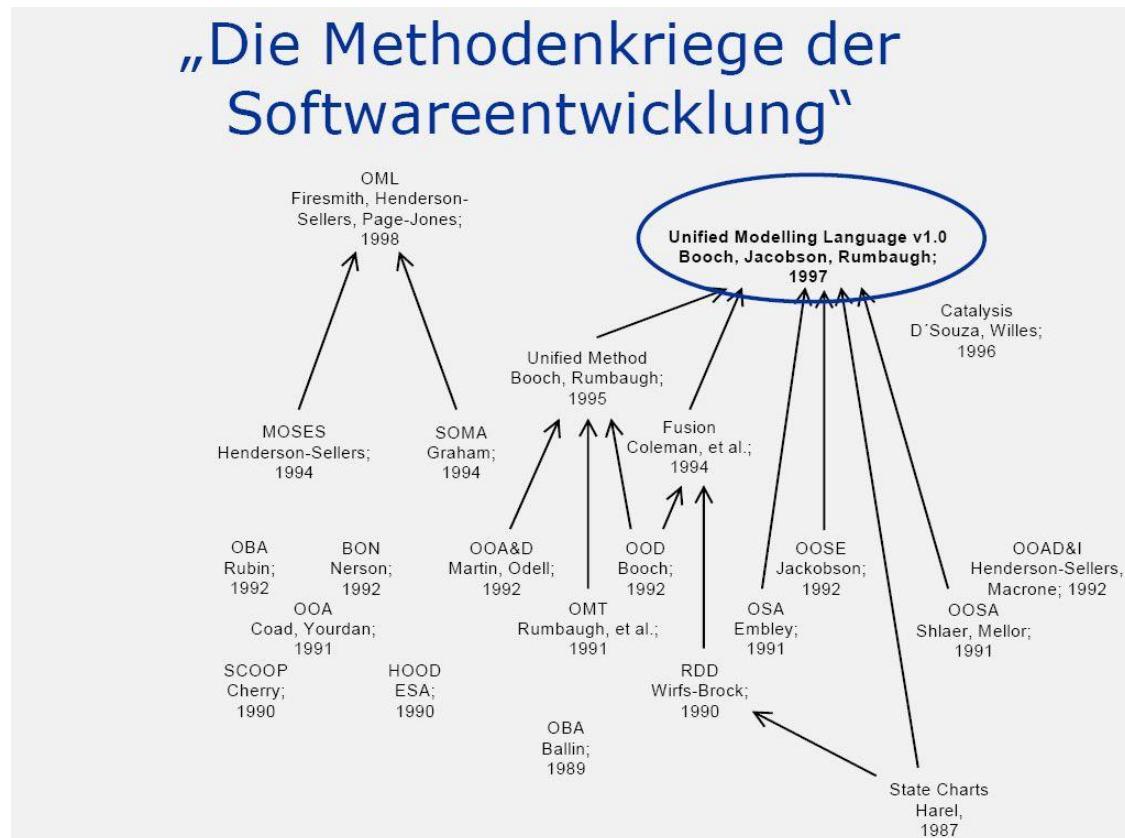
- Sie ist die verbreiteste Notation, um Softwaresysteme zu analysieren und zu entwerfen
- Sie dient zur Spezifizierung und Visualisierung komplexer Softwaresysteme unabhängig vom Fach- und Realisierungsbereich.
- Sie liefert die Notationselemente gleichermaßen für die statischen und dynamischen Modelle von Analyse, Design und Architektur und unterstützt objektorientierte Vorgehensweisen.

Die UML ist

- nicht perfekt
- nicht vollständig
- keine rein formale Sprache
- nicht spezialisiert auf ein Anwendungsgebiet
- kein vollständiger Ersatz für Textbeschreibung
- keine Methode oder ein Vorgehensmodell

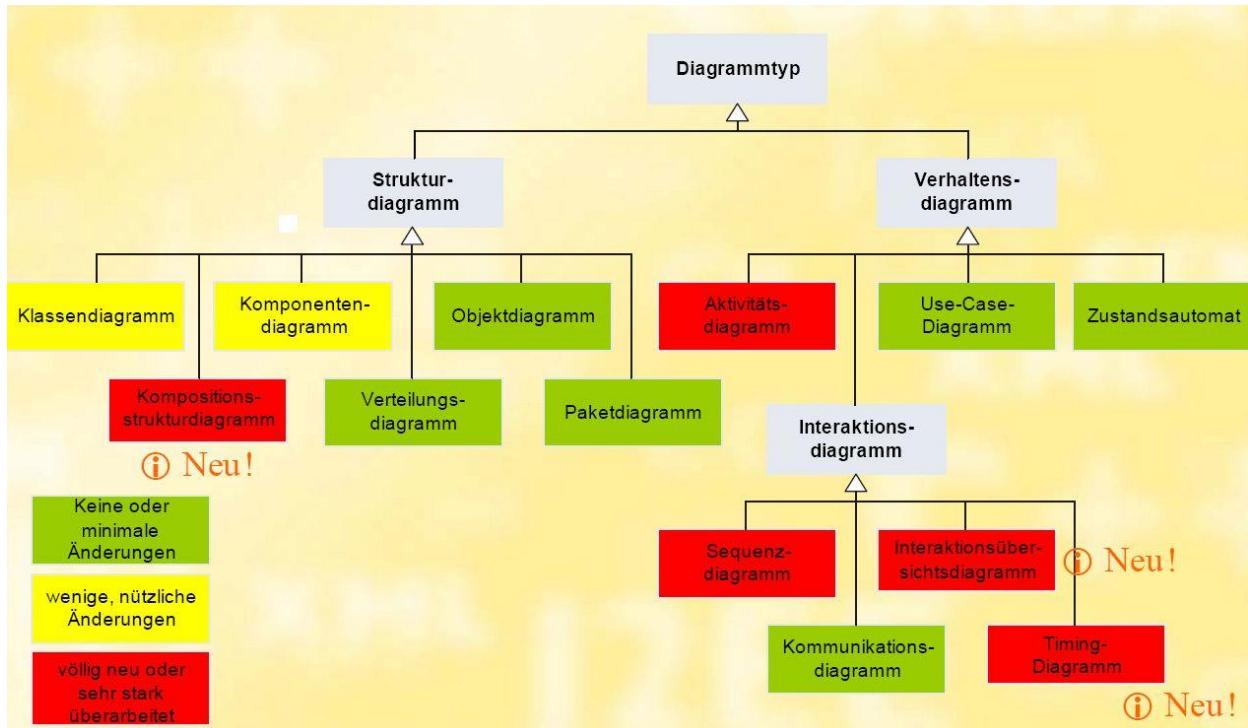
UML - historisch gesehen

Im Zuge der Entwicklung objektorientierter Programmiersprachen und deren Möglichkeiten, kam es besonders zum Anfang zu einer Vielzahl unterschiedlicher Notationsmodelle. Um diese zu vereinheitlichen, setzten sich zu Beginn der 90er Jahre mehrere Vertreter unter Führung der Firma RATIONAL zusammen, um eine gemeinsame Sichtweise (Unified) zu schaffen. Diese Arbeiten mündeten dann 1997 in die Verabschiedung der UML-Version 1.0.



UML-Diagramme im Überblick

Die bisher 7 Diagramme der UML 1.x - Notation wurden in der Version 2.x auf 13 Diagramme erweitert. Darüber hinaus wurden manche (alte) Diagramme mit neuen Inhalten versehen.

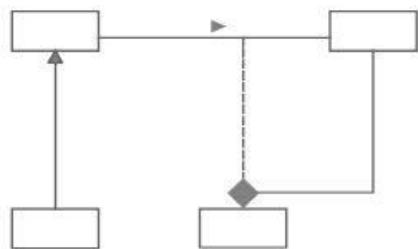


UML - Tabellarischer Überblick

Folgende Darstellung gibt einen Überblick über die Schwerpunkte der einzelnen Diagrammarten.

Klassendiagramm

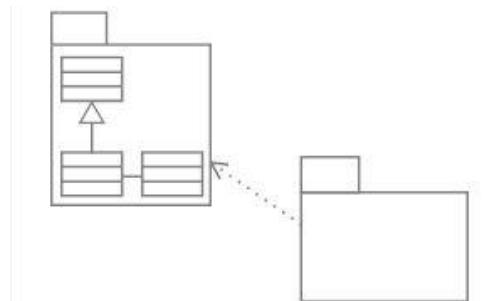
Es beschreibt die Klassen des System und wie sie miteinander in Beziehung stehen. Es ist das wichtigste Diagramm in der UML.



- Beschreibt die statische Struktur des Systems
- Enthält alle relevanten Strukturzusammenhänge/Datentypen
- Brücke zu dynamischen Diagrammen
- Normalerweise unverzichtbar

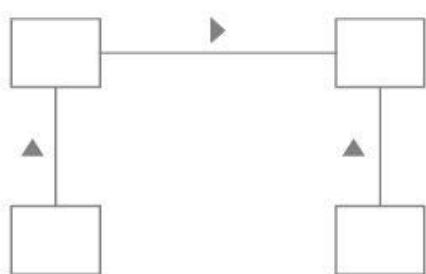
Paketdiagramm

Es bündelt die Klassen zu überschaubaren Paketen und beschreibt die Beziehungen zwischen den Paketen.



- Wie kann ich mein Paket so schneiden, dass ich den Überblick behalte
- Logische Zusammenfassung von Modellelementen
- Modellierung von Abhängigkeiten/Inklusion möglich

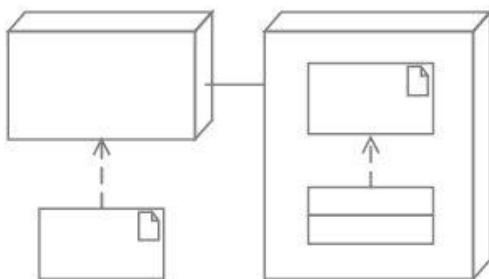
Objektdiagramm



- Welche innere Struktur besitzt mein System zu einem bestimmten Zeitpunkt zur Laufzeit (Klassendiagrammschnappschuss)
- Zeigt Objekte und Attributbelegungen zu einem bestimmten Zeitpunkt
- Verwendung beispielhaft zur Veranschaulichung
- Detailniveau wie im Klassendiagramm
- Sehr gute Darstellung von Mengenverhältnissen

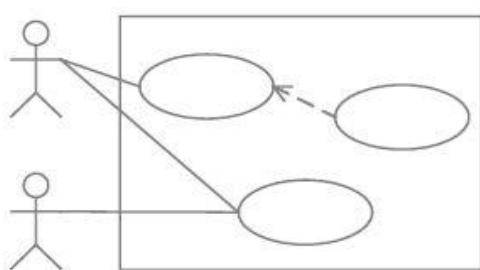


Verteilungsdiagramm



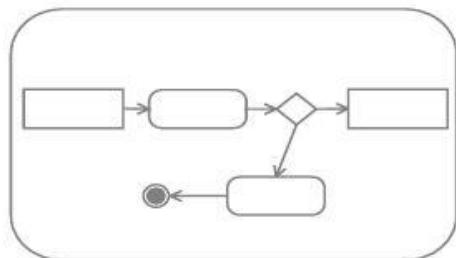
- Wie sieht das Einsatzumfeld (Hardware, Server, Datenbanken,...) des Systems aus. Wie werden die Komponenten zur Laufzeit wohin verteilt.
- Zeigt das Laufzeitumfeld des Systems mit den greifbaren Systemteilen
- Darstellung von Softwareservern möglich
- Hohes Abstraktionsniveau, kaum Notationselemente

UseCase-Diagramm



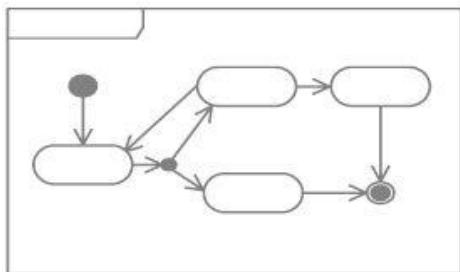
- Was leistet mein System für seine Umwelt
- Außensicht auf das System
- Geeignet zur Kontextabgrenzung

Aktivitätsdiagramm



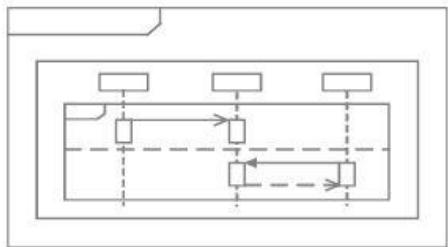
- Wie läuft ein bestimmter flussorientierter Prozess oder ein Algorithmus ab ?
- Sehr detaillierte Visualisierung von Abläufen mit Bedingungen , Schleifen, Verzweigungen.
- Parallelisierung und Synchronisation
- Darstellung von Datenflüssen

Zustandsdiagramm



- Welche Zustände kann ein Objekt, eine Schnittstelle, ein UseCase ... bei welchen Ereignissen annehmen ?
- Präzise Abbildung eines Zustandsmodells mit Zuständen, Ereignissen, Nebenläufigkeiten, Bedingungen, Ein- und Austritten.
- Schachtelung möglich

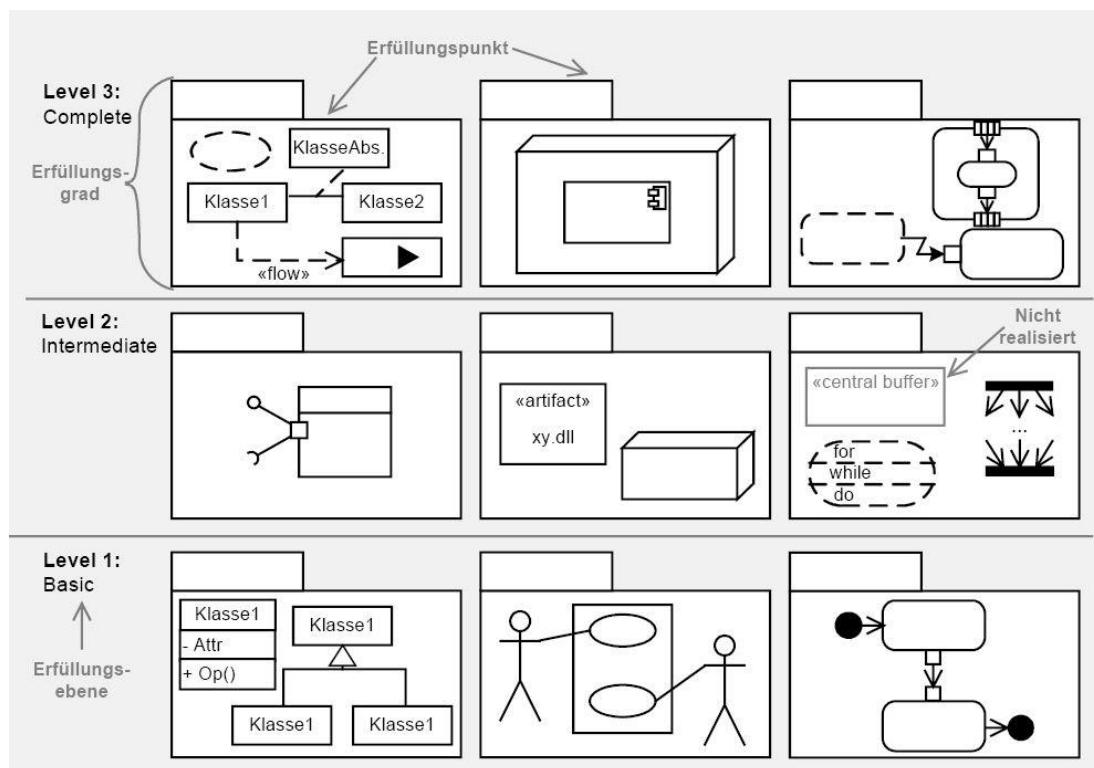
Sequenzdiagramm



- Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus.
- Darstellung des Informationsaustausches zwischen Kommunikationspartnern
- Sehr präzise Darstellung der zeitlichen Abfolge auch mit Nebenläufigkeiten

UML-Erfüllungsebenen

UML erlaubt durch seine Vielzahl seiner Diagramme eine sehr detaillierte Darstellung der Prozesse. Dies ist nicht immer notwendig und machbar. Folgende Grafik veranschaulicht Priorisierungen und Variationen des Detailierungsgrades.



Verständnisfragen

- Vor welchem Problem stand anfangs die Modellierungssprache UML ?
- Welche Bedeutung haben die Begriffe Statisch/Dynamisch im Zusammenhang mit Objekten und der UML?
- Gibt es neben der UML andere Modellierungssprachen? Welche Bereiche decken diese Sprachen ab ?

UseCase-Diagramm

Die Frage **Was soll mein geplantes System eigentlich leisten** sollte am Beginn jeder Systementwicklung stehen. **Das Use-Case-Diagramm zeigt das externe Verhalten eines Systems aus der Sicht der Nutzer.**

Das Ziel jedes Softwareentwicklungsprozesses ist es, eine Software zu entwickeln, die ganz bestimmte Anforderungen erfüllt. Die Entwicklung einer Software fängt mit der Zielsetzung an: Die Software soll, wenn fertiggestellt, die zu Beginn des Entwicklungsprozesses festgelegten Anforderungen/Ziele erfüllen.

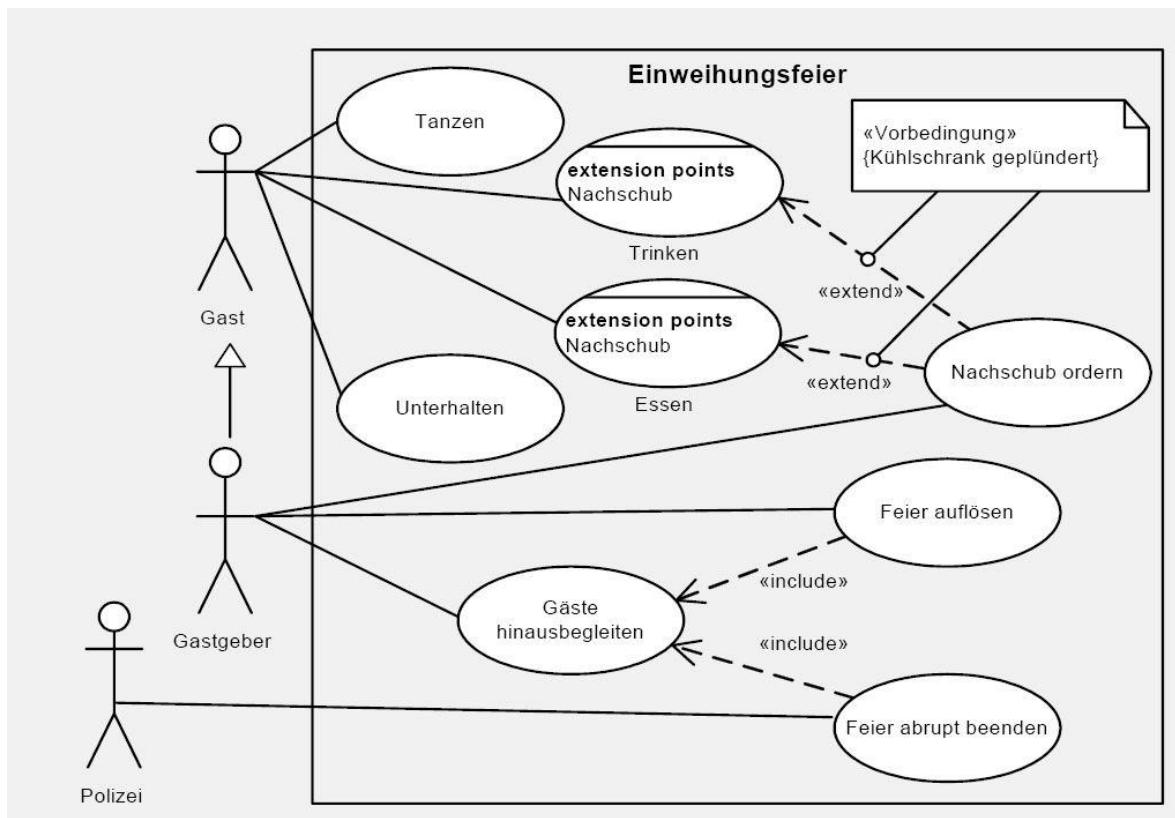
Leider sind diese Ziele nicht immer klar definiert. Man nimmt sich zum Beispiel vor, einen Online-Shop zu entwickeln, und stellt dann während des Entwicklungsprozesses fest, dass es unendlich viele unterschiedliche Funktionen in einem Online-Shop geben kann und man sich eigentlich nie klar gemacht hat, was man denn nun für Funktionen im Detail braucht. Man muss den Entwicklungsprozess daher wiederholt unterbrechen und inne halten, um sich zu überlegen, welche Funktionen, die einem bei der Entwicklung gerade eingefallen sind, notwendig sind und welche nicht.

Besonders schwierig wird die Situation, wenn der Auftraggeber des Entwicklungsprozesses nicht gleichzeitig der Entwickler ist. In diesem Fall kann der Entwickler nicht entscheiden, welche Funktionen notwendig sind - dies weiß nur der Auftraggeber. Dies führt zu einem ständigen Frage-Antwort-Spiel zwischen Entwickler und Auftraggeber, wenn der Entwickler nicht - noch schlimmer - die Entscheidungen selbst trifft und hofft, dies jeweils im Sinne des Auftraggebers zu tun.

Wenn Anforderungen an die zu entwickelnde Software nicht zu Beginn des Entwicklungsprozesses klipp und klar sind, wird der Entwicklungsprozess an sich unnötig erschwert. Denn das, was Sie entwickeln, richtet sich nach den bekannten Anforderungen. Jede Anforderung, die Ihnen oder Ihrem Auftraggeber später einfällt, führt dazu, dass Sie das, was Sie bisher entwickelt haben, ändern müssen. Denn die neuen Anforderungen hatten Sie logischerweise in Ihrer bisherigen Entwicklung nicht berücksichtigt. Grundsätzlich gilt, dass je später Anforderungen in einem Entwicklungsprozess bekannt werden, umso aufwändiger und daher teurer der Entwicklungsprozess wird. Anders gesagt: Wenn alle Anforderungen von Anfang an bekannt sind, bevor der Entwicklungsprozess gestartet wird, wäre das ideal.

Im Zusammenhang mit Anforderungen setzt man als erstes Hilfsmittel das Use-Case-Diagramm ein.

- Use-Case-Diagramme sind, auch bedingt durch die geringe Anzahl der Modellelemente, eingängig und übersichtlich.
- In der Projektrealität trifft man häufig eine sehr einfache skizzenhafte Verwendung von Use-Cases für erste Diskussionen.
- Sie enthalten die grafische Darstellung
 - des Systems
 - der Use-Cases
 - der Akteure außerhalb des Systems
 - der Beziehungen zwischen Akteur und Use-Case, der Akteure untereinander oder Use-Cases untereinander



Modellelemente

Aufgabe des Use-Case-Diagramms ist es, eine grobe Ordnung in die vielen zum Teil sehr detaillierten Anforderungen zu bringen. Das Use-Case-Diagramm soll uns einen Überblick über das verschaffen, was die zu entwickelnde Software eigentlich im Wesentlichen können muss. Es werden also wichtige Funktionen der Software herausgearbeitet und zueinander in Beziehung gesetzt. Die technische Implementierung spielt bei diesem Überblick keine Rolle - vergessen Sie alles, was irgendetwas mit Programmiersprachen zu tun hat. Es geht um das Was, nicht um das Wie.

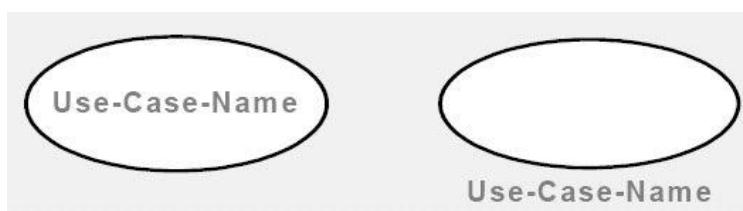
Das Use-Case-Diagramm ist ein recht einfacher Diagrammtyp, der schön anschaulich ist. Im Folgenden sehen Sie die grundlegenden Bausteine, mit denen alle Use-Case-Diagramme aufgebaut sind.

Das Use-Case-Diagramm verfügt über folgende Notationselemente:

- UseCase
- System
- Akteur
- includes-Beziehung
- extends-Beziehung

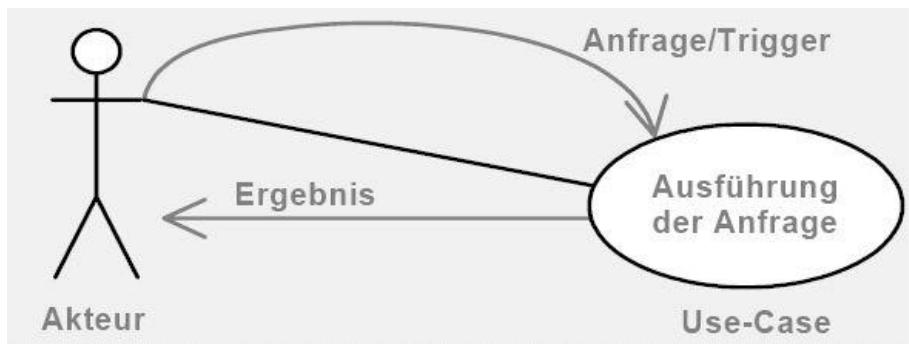
Use-Case = System + Anwendungsfall + Akteur

Im Use-Case-Diagramm gibt es den Akteur und das System. Der Akteur ist der Anwender, das System die zu entwickelnde Software. Im System, das als Rechteck dargestellt wird, werden verschiedene wesentliche Anforderungen platziert. Man schreibt hierzu sehr knappe Funktionsbeschreibungen in Ellipsen, wobei jede Ellipse genau eine Funktion darstellt. Die Ellipsen sind die Use-Cases - daher hat das Use-Case-Diagramm seinen Namen. Wenn Sie einen deutschen Begriff verwenden möchten: Use-Case wird für gewöhnlich mit Anwendungsfall übersetzt.



Da es beim Use-Case-Diagramm um einen ersten groben Überblick geht, beschränkt man sich auf die Darstellung wesentlicher Funktionen - alles, was nicht zum Überblick beiträgt, wird weggelassen. Beim Erstellen eines Use-Case-Diagramms geht es also nicht darum, möglichst viele Ellipsen in das Rechteck zu malen. Es geht darum, wesentliche Anforderungen zu finden und diese zusammenhängend als Use-Cases in das System einzuziehen. Die Zusammenhänge zwischen Use-Cases und dem Akteur als auch zwischen Use-Cases untereinander werden durch Verbindungslienien dargestellt.

- Beschreibt eine Reihe von Aktionen, die nacheinander ein Verhalten formen
- Wird immer von einem Akteur ausgelöst
- Hat immer ein Ergebnis
- Kann gleichzeitig mehrfach instanziert werden
- Spiegelt funktionales Verhalten wieder. Interne Abläufe sind irrelevant



System

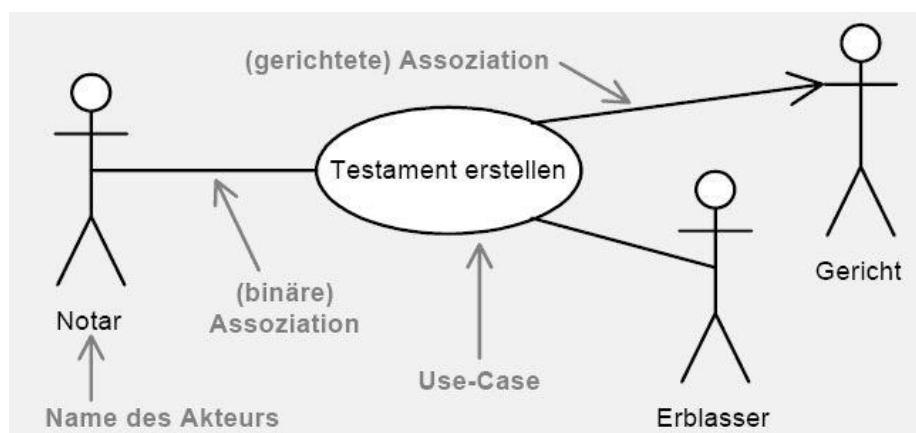


Die Darstellung erfolgt durch ein Rechteck, dessen Kanten die Systemgrenzen darstellen. Das System realisiert und etabliert das im Use Case beschriebene Verhalten. Sie kann selbst ein Subsystem sein.

Akteur



Er repräsentiert die Rolle, die von außen mit dem System interagiert. Er steht in ständigem Signal- und Datenaustausch mit dem UseCase; dies wird durch die Linie zwischen Akteur und UseCase dargestellt.



Beziehungen

Die Zusammenhänge zwischen Use-Cases und dem Akteur als auch zwischen Use-Cases untereinander werden durch Verbindungslien dargestellt.

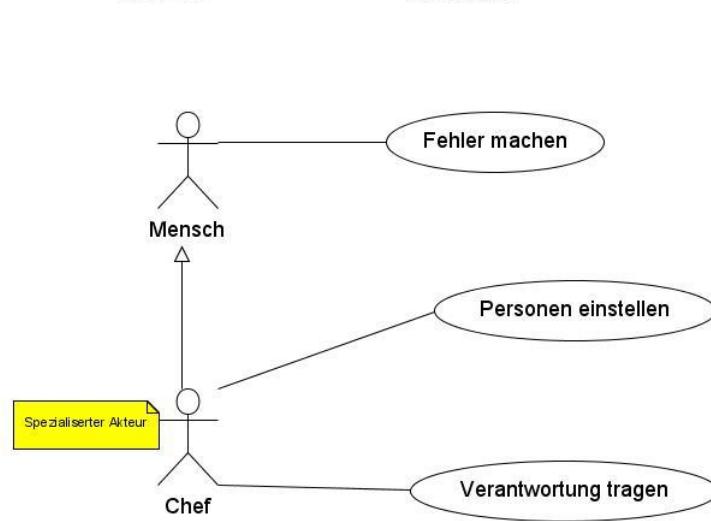
Verbindungslien in UML-Diagrammen werden Assoziationen genannt. Sie stellen Zusammenhänge zwischen Elementen an den Enden von Assoziationen dar. Welche Art von Assoziation ausgedrückt wird, hängt von der Darstellung der Verbindungslien und von Schlüsselwörtern ab, die an der Verbindungslien stehen können.

Im Use-Case-Diagramm gibt es zwei Arten von Verbindungslien: Die durchgezogene Verbindungslien stellt eine Assoziation zwischen dem Akteur und einem Use-Case dar. Sie bedeutet, dass der Akteur den Use-Case in irgendeiner Form anwendet. Der Akteur tauscht also Informationen mit dem Use-Case aus. Das kann zum Beispiel bedeuten, dass er eine Funktion des Systems startet oder ihm von einer Funktion des Systems Daten ausgegeben werden.

Die gestrichelte Verbindungslien stellt eine Assoziation zwischen zwei Use-Cases dar. Da es zwei verschiedene Arten von Assoziationen zwischen Use-Cases gibt, wird neben die gestrichelte Verbindungslien ein Schlüsselwort gesetzt. Schlüsselwörter in der UML, die zur Spezifizierung von Verbindungslien oder anderen geometrischen Formen verwendet werden, werden immer zwischen doppelte spitze Klammern gestellt. Diese Schlüsselwörter werden in der UML Stereotypen genannt.

Beziehungen zwischen Akteuren

Akteure dürfen untereinander in Beziehung stehen, es sind sogar Vererbungsbeziehungen erlaubt. Der spezialisierte Akteur ist dann an den gleichen UseCase-Abläufen beteiligt wie der vererbende Akteur.

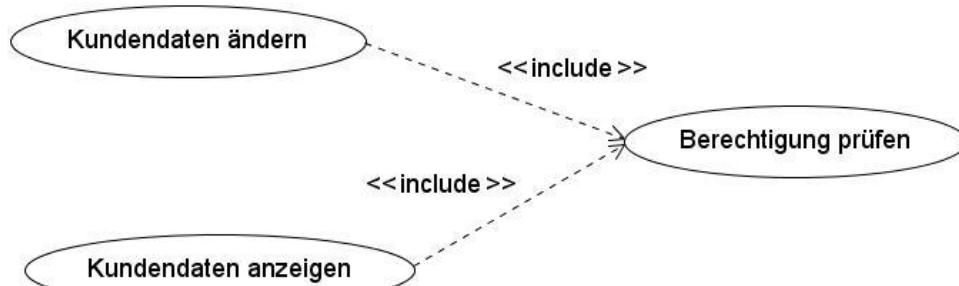


Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

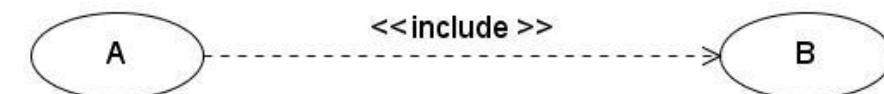
includes-Beziehung

Darstellung durch Abhängigkeitsbeziehung mit Stereotyp `include`. Der *linke*linke Use-Case importiert das Verhalten des *rechten Use-Case*.

Eine `include`-Beziehung ist nicht optional; das Verhalten wird immer eingeschlossen. Erst durch die Inklusion ergibt sich ein (sinnvolles) Gesamtverhalten. Rekursive `Include`-Beziehungen sind nicht erlaubt.



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.



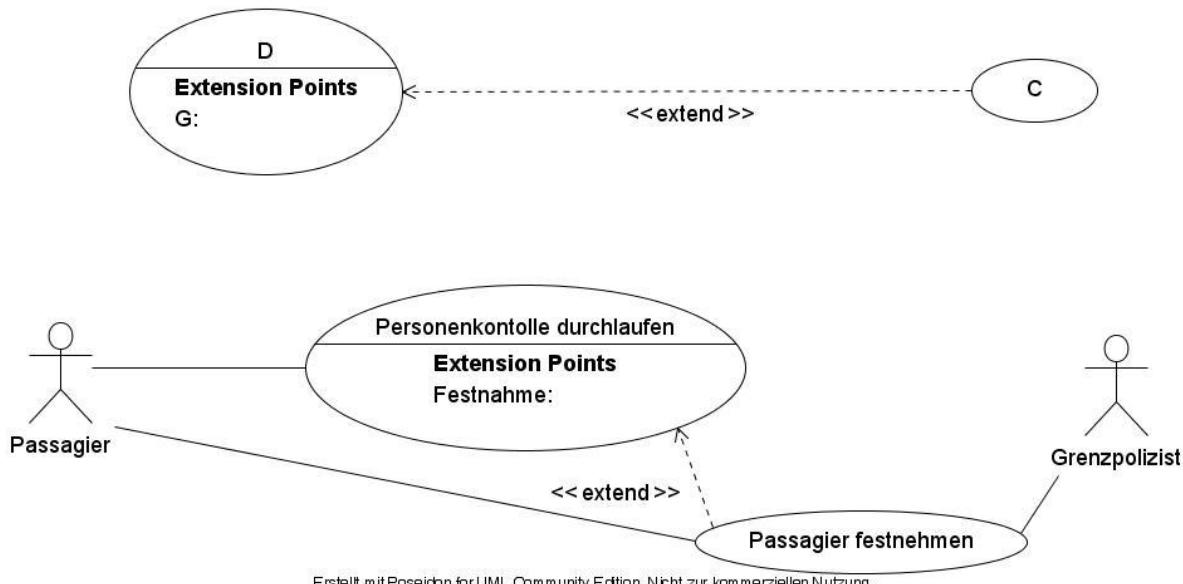
1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____

Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.



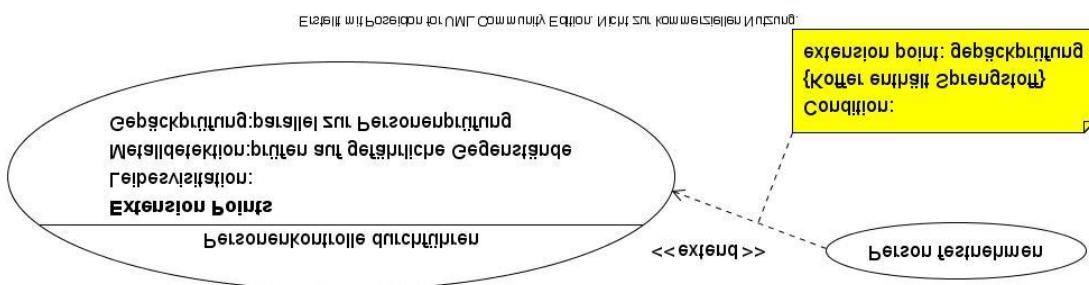
extends-Beziehung

Eine extend-Beziehung zeigt an, dass das Verhalten eines Use-Case (D) durch einen anderen Use-Case (C) erweitert werden kann, aber nicht muss.

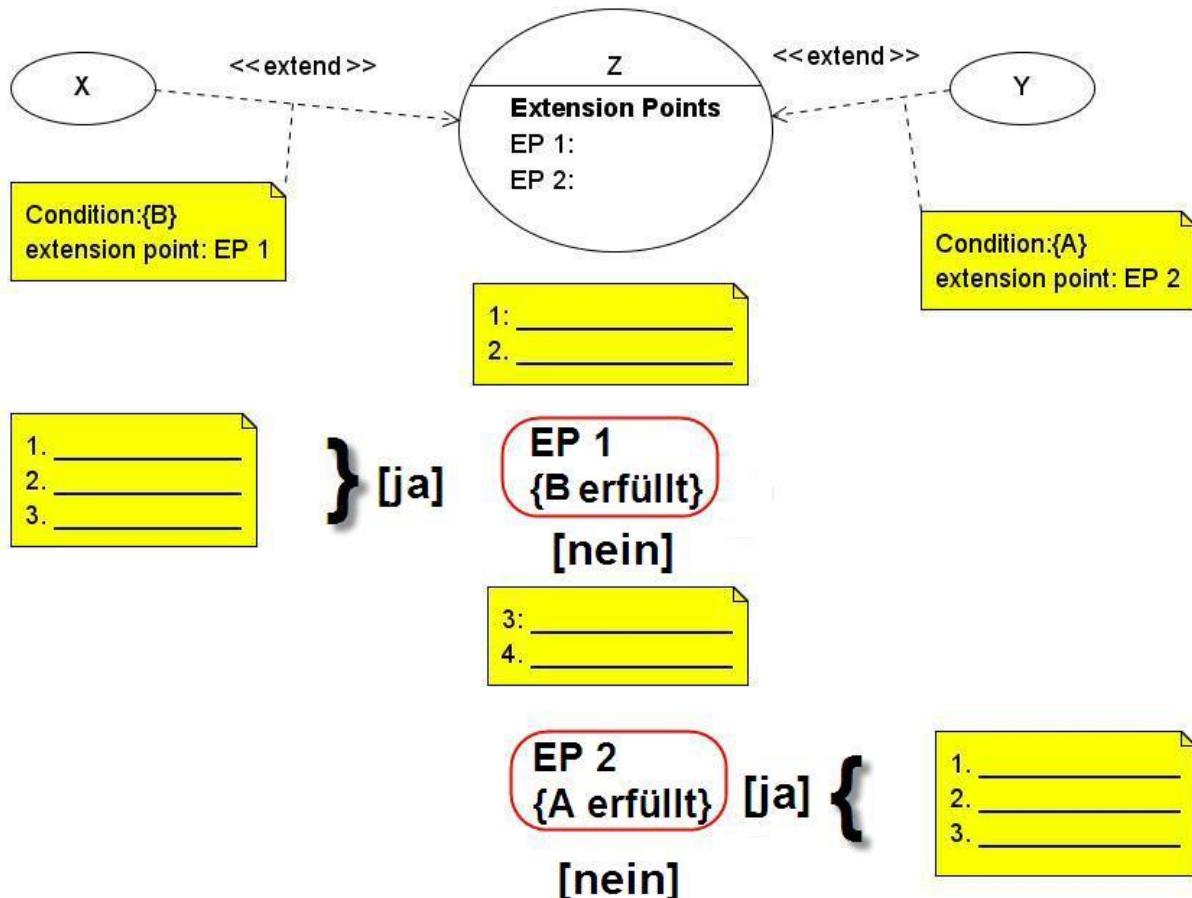


Obige Abbildung zeigt, dass der UseCase *Personenkontrolle durchlaufen* in bestimmten Fällen durch *Passagier festnehmen* erweitert wird.

Der Zeitpunkt, an dem ein Verhalten eines UseCases erweitert werden kann, wird als **Extension point** bezeichnet.

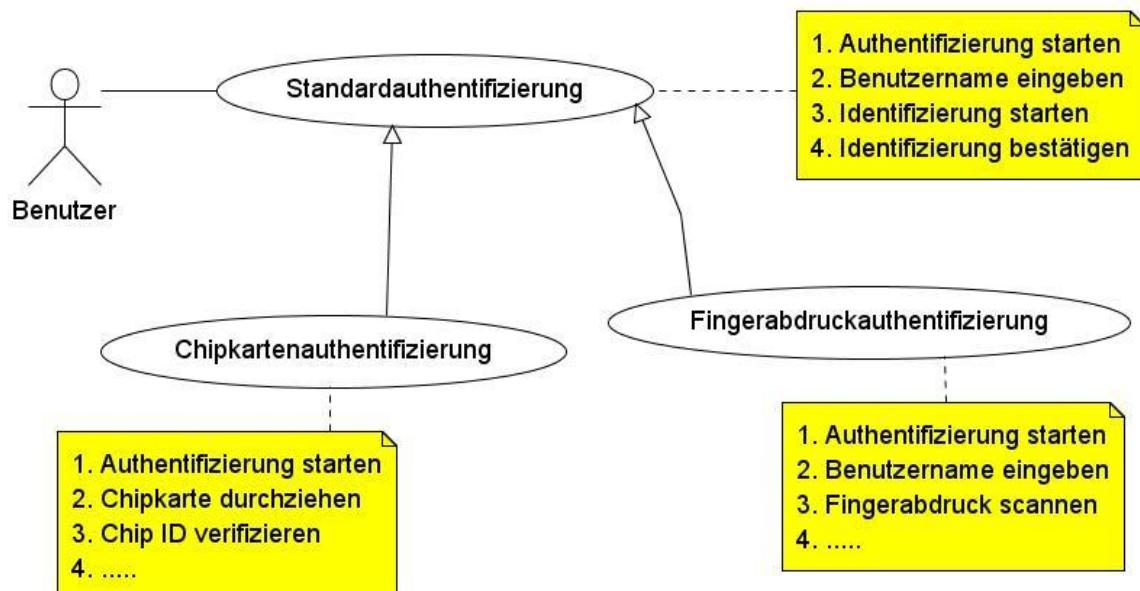


Neben dem Erweiterungspunkt kann eine Bedingung für die Erweiterung angegeben werden. Die Bedingung wird bei Erreichen des Erweiterungspunktes geprüft. Ist die Bedingung wahr, wird der Ablauf erweitert; ist die Bedingung nicht erfüllt, läuft der UseCase *normal* weiter.



Vererbung

Zwischen UseCases kann auch eine Vererbungsbeziehung modelliert werden. Sie entspricht von Konzept her der Vererbung im Klassendiagramm.



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

Unterschied Vererbung – Extends

In der Praxis werden Generalisierungs- und Extends-Beziehungen häufig falsch eingesetzt.

- Extends-Beziehung erweitert das Verhalten nur unter gewissen Bedingungen.

Die Generalisierungsbeziehung kopiert und überschreibt immer das Verhalten des Basis-UseCases.

- Nur die Extends-Beziehung erlaubt es, die Erweiterung des Verhaltens durch Bedingungen zu steuern

Unterschiede include-extend

<<include>>

```
includes
A -----> B
```

- Ablauf von A schließt immer Ablauf von B ein
- A ist meist unvollständig und wird erst durch Inklusion B zu einem vollständigen Ablauf.
- B ist häufig künstlich zur Redundanzvermeidung gebildet.
- Ablauf von B kann von verschiedenen Use-Cases genutzt werden
- B wird **unabhängig von A** modelliert, um die Nutzung durch weitere Use-Cases sicherzustellen (Wiederverwendbarkeit),
- B muss in sich **nicht vollständig** sein (B weiß nicht, durch wen er inkludiert wird)

<<extends>>

```
extends
A <----- B
```

- Ablauf von A kann, muss aber nicht durch Ablauf von B erweitert werden
- A besitzt neben Normalverhalten auslagerbare Sonderfälle
- A ist meist vollständig und kann durch B optional erweitert werden
- B ist meist in sich vollständig

Geschäftsprozessschablone

Wie werden Use Cases erstellt? Es werden für ein definiertes System alle Akteure festgestellt (hierin liegt das größte Problem, dass ein Akteur übersehen und/oder vergessen wird). Alle Anforderungen der jeweiligen Akteure werden festgehalten und der Reihe nach in Schablonen übertragen. Die Schablonen werden nummeriert. Nicht alle Werte einer Schablone müssen für jeden Vorgang (Aktion) ausgefüllt sein.

- **Übergeordneter elementarer Geschäftsprozess**

Hinweis auf übergeordneten Geschäftsprozess, wenn der betrachtete GP in einer <<include>> bzw. <<extends>> - Beziehung steht.

- **Beschreibung**

Der Name des UseCase.

- **Ziel**

Was ist das Ziel des UseCase. Hier erfolgt die genaue Darstellung des UseCases.

- **Vorbedingung**

Erwarteter Zustand, bevor der GP beginnt.

- **Nachbedingung bei erfolgreicher Ausführung**

Erwarteter Zustand nach erfolgreicher Ausführung des Geschäftsprozesses.

- **Nachbedingung bei fehlgeschlagener Ausführung**

Erwarteter Zustand, wenn das Ziel nicht erreicht werden kann.

- **Beteiligte Nutzer**

Rollen von Personen oder anderen Systemen, die den GP auslösen oder daran beteiligt sind.

- **Auslösendes Ereignis**

Wenn dieses Ereignis eintritt, dann wird der GP initiiert.

Geschäftsprozess-Schablone (Balzert)

Geschäftsprozess	Name des Anwendungsfalles
Ziel	Kurze Beschreibung des Zwecks
Kategorie	Primär, Sekundär oder Optional
Vorbedingung	Erwarteter Zustand vor Ausführung des Anwendungsfalles
Nachbedingung Erfolg	Erwarteter Zustand nach erfolgreicher Ausführung des Anwendungsfalles
Nachbedingung Fehlschlag	Erwarteter Zustand, wenn das Ziel nicht erreicht wird
Akteure	Wer ist an dem Anwendungsfall beteiligt?
Auslösendes Ereignis	Nach welchem Ereignis tritt dieser Anwendungsfall auf?
Beschreibung	Ablauf des Anwendungsfalles
Erweiterungen	Was läuft parallel, bzw. nach diesem Anwendungsfall ab und kann diesem zugerechnet werden.
Alternativen	Wie kann dieser Anwendungsfall noch ablaufen?

Zusammenfassung

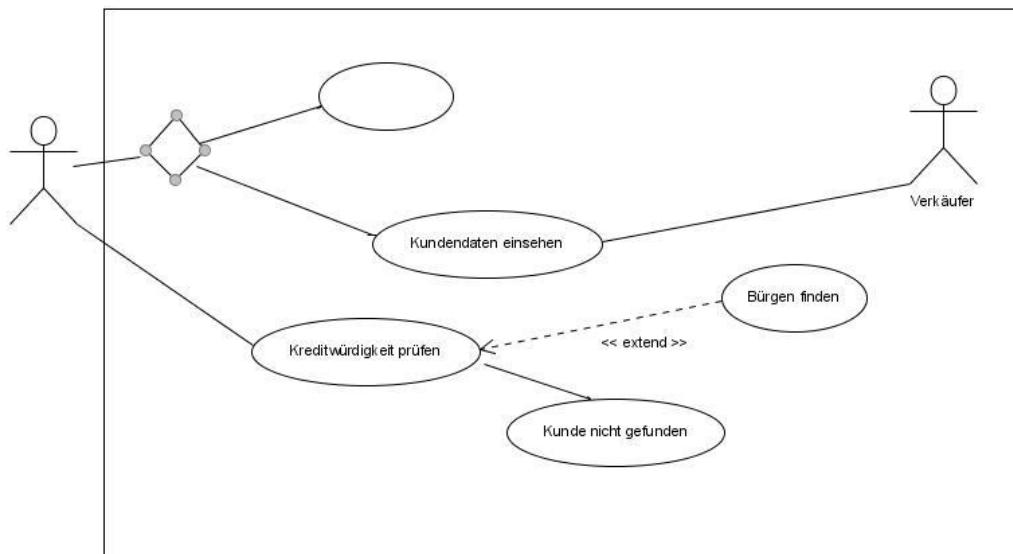
Fazit

Anforderungen und ihre Zusammenhänge ohne Berücksichtigung der Reihenfolge

Use-Case-Diagramme sind Verhaltensdiagramme: Sie beschreiben bestimmte Aspekte, wie sich ein System verhält. Wie Sie nun gesehen haben, können im Use-Case-Diagramm wesentliche Funktionen des Systems hervorgehoben und zueinander in Beziehung gesetzt werden. Diesen Diagrammen fehlt jedoch eine Möglichkeit, die Reihenfolge der Ausführung festzulegen. Das geht im eingeschränkten Maße mit include- und extend-Assoziationen. Da ein Use-Case in diesen Fällen aber jeweils vom anderen Use-Case eingeschlossen wird - bei include immer, bei extend in Abhängigkeit einer Bedingung - handelt es sich nicht um eine strikte Reihenfolge: Ein Use-Case, der einen anderen mit include oder extend einschließt, läuft nach Beendigung des eingeschlossenen Use-Case weiter. Dieser Nachteil des Use-Case-Diagramms ist an sich kein Nachteil, da Use-Case-Diagramme nicht für die Beschreibung einer Reihenfolge von Abläufen vorgesehen sind. Es ist also wichtig zu verstehen, dass jedes Diagramm bestimmte Aspekte eines Systems hervorhebt und beschreibt, aber nie alle. Es ist nicht die Aufgabe des Use-Case-Diagramms, die Reihenfolge einer Ausführung zu beschreiben. Zu diesem Zweck kann ein anderer Diagrammtyp verwendet werden, den Sie im nächsten Kapitel kennenlernen werden.

Fragen zu UML

- Was ist falsch an folgender Darstellung



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

- Die folgenden Anforderungen beschreiben eine einfache Artikelverwaltung, die objektorientiert zu modellieren ist.
1. Eine Firma will ihre Artikel und Bestellungen verwalten.
 2. Für jeden Artikel werden die Artikelnummer, die Artikelbezeichnung und der Verkaufspreis festgehalten. Jeder Artikel gehört zu einer oder mehreren Artikelgruppen.
 3. Jeder Artikel kann an mehreren Orten gelagert werden. Beispielsweise lagert die Firma Artikel in ihren Filialen Truchtelfingen, Adolzfurt und Ahrenviölfeld. Pro Lagerort werden für jeden Artikel dessen Maximalbestand, Mindestbestand, aktueller Bestand und Name des Lagers gespeichert. Eine Lagerliste soll Auskunft über die Bestände geben.
 4. Jeder Artikel kann von verschiedenen Lieferanten bezogen werden. Dabei gibt es je nach Lieferant unterschiedliche Einkaufspreise und Verpackungseinheiten. Bei jedem Lieferanten besitzt der Artikel eine eigene Bestellnummer.
 5. Für jeden Lieferanten müssen dessen Name, Adresse, Telefonnummer und der Ansprechpartner gespeichert werden. Jeder Lieferant kann mehrere Artikel liefern
 6. Jeden Tag werden die Lagerbestände kontrolliert. Wird der Mindestbestand unterschritten, so wird ein Bestellvorschlag erstellt. Er enthält außer den Daten eines Artikels auch die verschiedenen Lieferanten mit ihren Lieferkonditionen. Für jedes Lager wird errechnet, wie viele Artikel nachbestellt werden müssen. Die vorgeschlagene Anzahl errechnet sich aus dem Maximalbestand und dem aktuellen Bestand. Die Anzahl ist auf ein n-faches der Verpackungseinheit eines jeden Lieferanten abzurunden.
 7. Jeder Artikel kann in beliebiger Anzahl bestellt werden. Dabei wird vom System automatisch der günstigste Lieferant gewählt.
 8. Jeder Artikel wird einzeln bei einem Lieferanten bestellt. Für jede Lieferantenbestellung werden das Bestelldatum und das Lieferdatum festgehalten. Der Einfachheit halber werden Teillieferungen ausgeschlossen.
 9. Ein Kunde kann eine oder mehrere Bestellungen erteilen, die erfasst werden müssen. Jede Bestellung erhält eine eindeutige Bestellnummer. Außerdem werden das Bestelldatum, das Lieferdatum und die Portokosten festgehalten.
 10. Jede Kundenbestellung kann sich auf mehrere Artikel beziehen. Für jeden bestellten Artikel ist die gewünschte Anzahl festzuhalten. Außerdem kann der Gesamtpreis für mehrere Artikel ungleich Anzahl*Verkaufspreis sein.
 11. Jede Kundenbestellung wird von einem Sachbearbeiter bearbeitet. Für jeden Sachbearbeiter sollen dessen Personalnummer, ein Kürzel, der Name und die Telefonnummer gespeichert werden.
 12. Außerdem sollen folgende Statistiken erstellt werden:
 - Welchen Umsatz hat ein Kunde X im aktuellen Jahr erzielt?
 - Welchen Umsatz hat ein Sachbearbeiter X mit seinen verschiedenen Kunden erzielt.

Lösungen

Lösung zur Lagerverwaltung

Akteure:

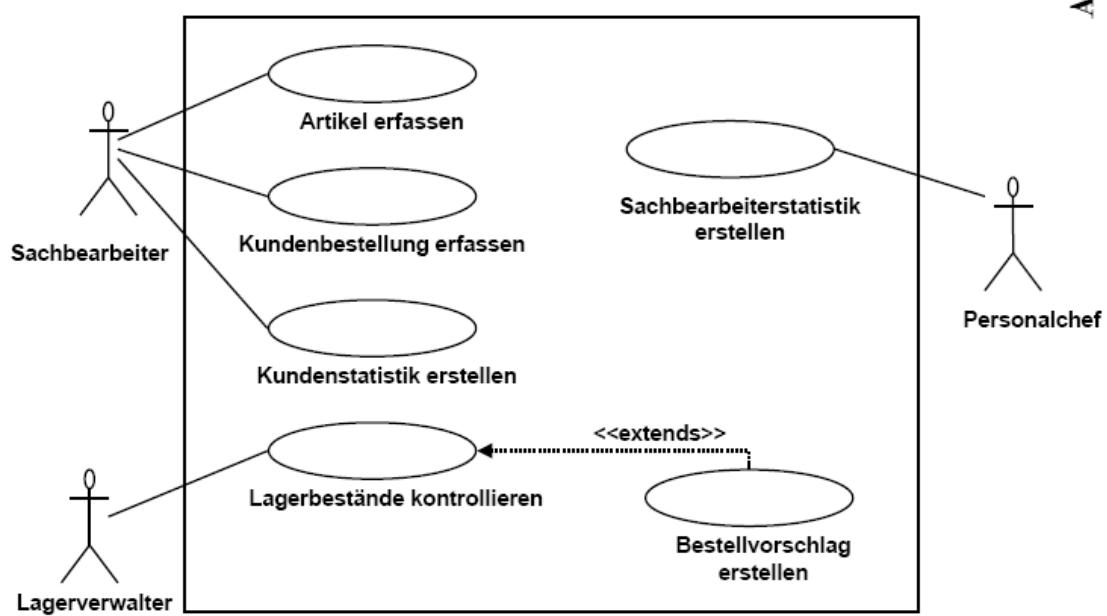
- Der Sachbearbeiter wird explizit erwähnt
- Es wird nichts darüber ausgesagt wer das Lager verwaltet und die Bestellungen bei den Lieferanten durchführt. Hier besteht Klärungsbedarf ! Die Aufgabe könnte vom Sachbearbeiter durchgeführt werden. Wahrscheinlicher ist aber ein Lagerverwalter.
- Wer erfaßt die Artikel? Der Sachbearbeiter, der Lagerverwalter, beide oder ein anderer ? Im folgenden werden Artikel vom Sachbearbeiter erfaßt.
- Wer erstellt die Statistiken bzw. wer darf diese erstellen ? Im folgenden wird die Sachbearbeiterstatistik von einem Aktor Personalchef und die Kundenstatistik von den Sachbearbeitern erstellt.
- Kunden und Lieferanten sind nur dann als Akteuren sinnvoll, wenn das zu erstellende System vorsieht, dass diese über das System ihre Angebote und Bestellungen unterbreiten können (z.B.: in einer Webbasierten Anwendung). Diese Möglichkeit wird im folgenden nicht weiterverfolgt.

Use-Cases

- Use Cases: Artikel erfassen.
- Kundenbestellung erfassen
- Lagerbestände kontrollieren
- Bestellvorschlag erstellen
- Kundenstatistik erstellen
- Sachbearbeiter Statistik erstellen

mögliche weitere Use-Cases (im Diagramm nicht angezeigt):

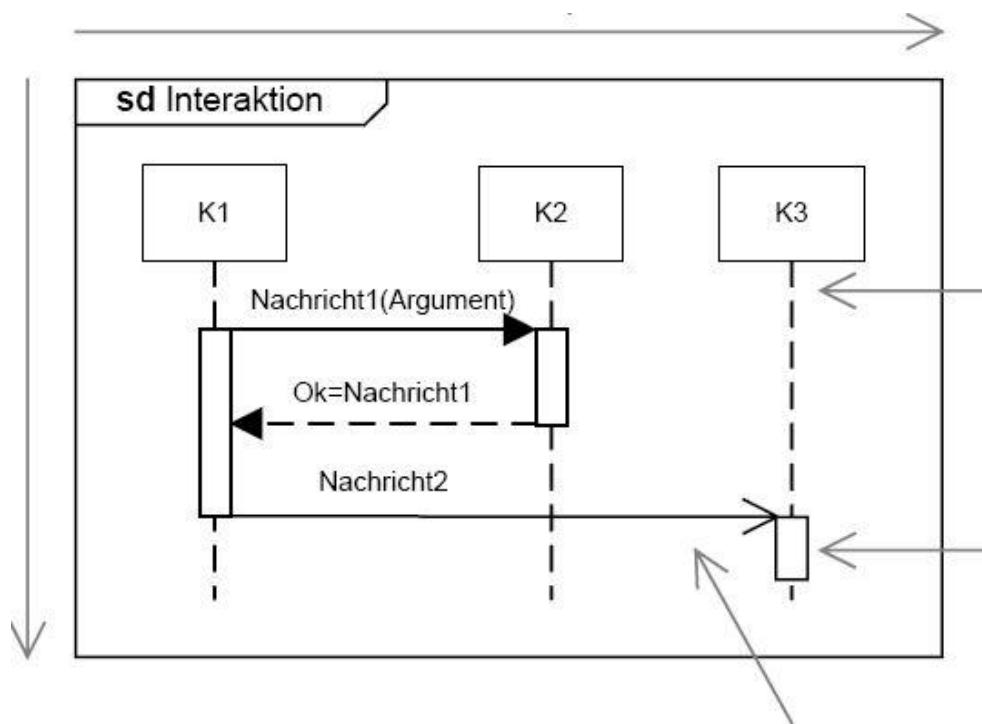
- Lieferung entgegennehmen (zumindest müssen die Bestände des gelieferten Artikels geändert werden)
- Auslieferung durchführen (zumindest müssen die Bestände des ausgelieferten Artikels geändert werden)
- Lieferanten (und Lieferkonditionen) erfassen



Sequenzdiagramm

“Erst die richtige Reihenfolge macht aus klugen Gedanken ein gutes Ergebnis.
A. van Rheyn”

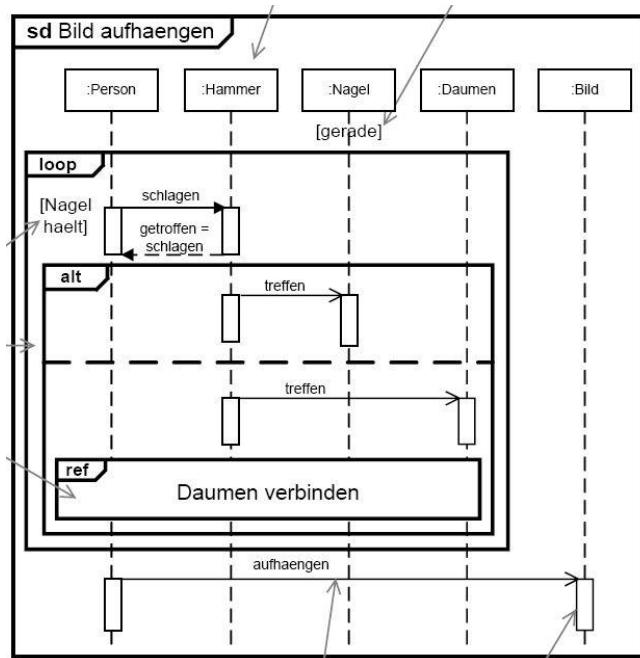
Sequenzdiagramme beschreiben die Kommunikation zwischen Objekten in einem bestimmten Szenario (UseCase). Es wird beschrieben **welche Objekte** an einem Szenario beteiligt sind, **welche Informationen** (Nachrichten) sie austauschen und in welcher **zeitlichen Reihenfolge** der Informationsaustausch stattfindet. Sequenzdiagramme enthalten eine implizite Zeitachse. Die Zeit schreitet in einem Diagramm von oben nach unten fort. Die Reihenfolge der Pfeile in einem Sequenzdiagramm gibt die zeitliche Reihenfolge der Nachrichten an.



Es wird vor allem benutzt zur

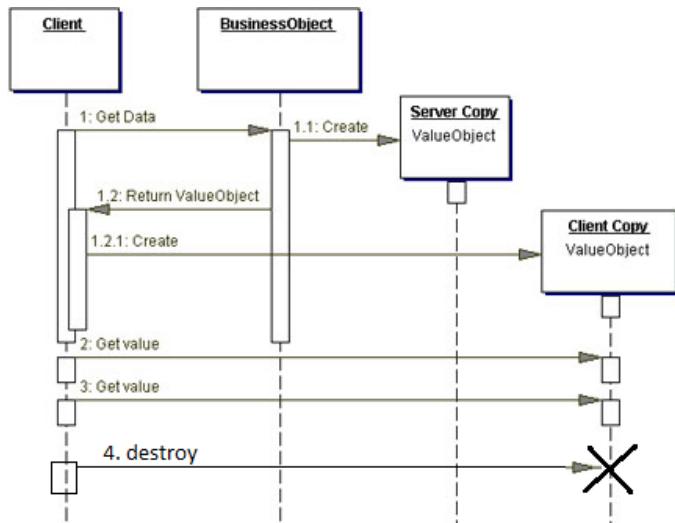
- Kommunikation und Beschreibung von UseCases
- Detailmodellierung im Feindesign
- Spezifikation von Schnittstellen, Tests und Simulation

Modellelemente



Das Sequenzdiagramm verfügt über relativ viele Notationselemente und kann leicht unübersichtlich werden. Es ist deshalb zu empfehlen, die betrachtete Sequenz zu reduzieren.

Klasse/Objekt/Lebenslinie



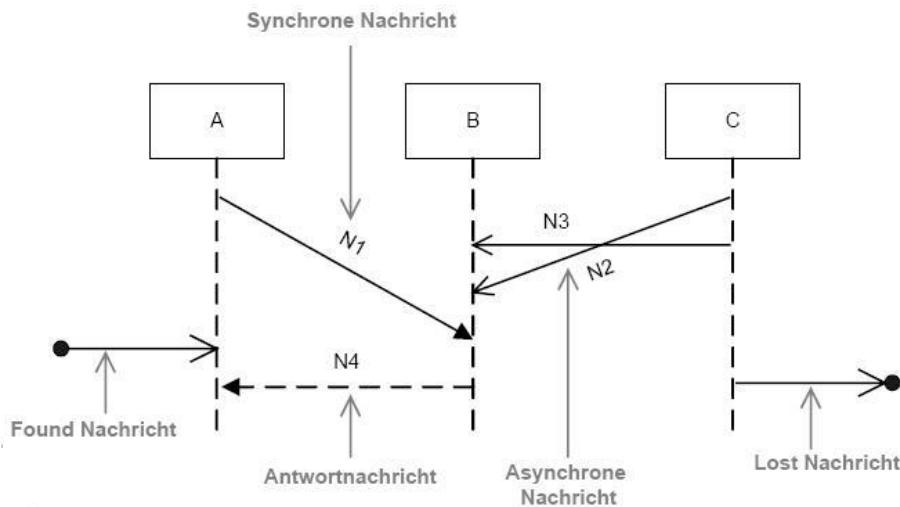
In dem Rechteck oberhalb der gestrichelten Linie wird der Objektname und der Klassenname angegeben. Der Name wird unterstrichen. Die senkrechte, gestrichelte Linie stellt die Lebenslinie (lifeline) eines Objekts dar. In diesem zeitlichen Bereich existiert das Objekt. Das schmale Rechteck auf der gestrichelten Linie stellt eine Aktivierung dar. Eine Aktivierung ist der Bereich, in dem eine Methode des Objektes aktiv ist (ausgeführt wird). Auf einer Lebenslinie können mehrere Aktivierungen enthalten sein.

Das Erzeugen eines Objektes wird durch eine Nachricht, die im Kopf des Objekts endet, dargestellt (gestrichelte Linie). Sie erhält häufig den Stereotyp <<create>>.

Das Zerstören (Löschen) eines Objektes wird durch ein X auf der Lebenslinie markiert. Das Objekt existiert anschließend nicht mehr innerhalb des Szenarios.

Nachricht

Objekte kommunizieren über Nachrichten. Nachrichten werden als Pfeile zwischen den Aktivierungen eingezeichnet. Der Name der Nachricht steht an dem Pfeil. Eine Nachricht liegt immer zwischen einem sendenden und einem empfangenden Objekt.



Der Pfeil mit der ausgefüllten Spitze bezeichnet einen synchronen Aufruf. Der Aufruf erfolgt von der Quelle zum Ziel, d. h. die Zielklasse muss eine entsprechende Methode implementieren. Die Quelle wartet mit der weiteren Verarbeitung bis die Zielklasse ihre Verarbeitung beendet hat und setzt die Verarbeitung dann fort. Ein Aufruf muss einen Namen haben; in runden Klammern können Aufrufparameter angegeben werden. Der gestrichelte Pfeil ist der Return. Die Bezeichnung des Return mit einem Namen ist optional.

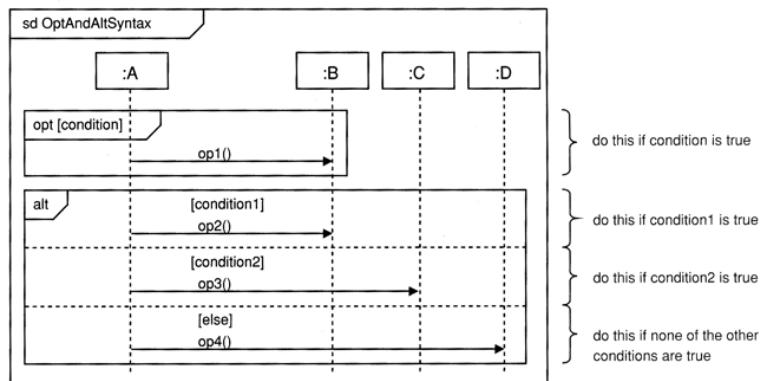
Mit einer offenen Pfeilspitze werden asynchrone Nachrichten gekennzeichnet. Der Aufruf erfolgt von der Quelle zum Ziel. Die Quelle wartet mit der Verarbeitung nicht auf die Zielklasse, sondern setzt ihre Arbeit nach dem Senden der Nachricht fort. Asynchrone Aufrufe werden verwendet, um parallele Threads zu modellieren. bei den sog. Lost-/Found-Nachrichten ist der Empfänger bzw. der Absender unbekannt.

Kombinierte Fragmente

Die kombinierten Fragmente (engl. combined fragments) bieten die Möglichkeit, in Sequenzdiagrammen bedingte Anweisungen und Verzweigungen, Schleifen und bestimmte Ausführungsarten zu modellieren. Dazu wird der Ablauf in ein oder mehrere Fragmente unterteilt. Eine Auswahl verschieden Interaktionsoperatoren bietet verschiedene Möglichkeiten, wie die erstellten Fragmente miteinander kombiniert und ausgeführt werden. Untenstehend werden einige wichtige Arten vorgestellt.

Kombinierte Fragmente – alt/opt

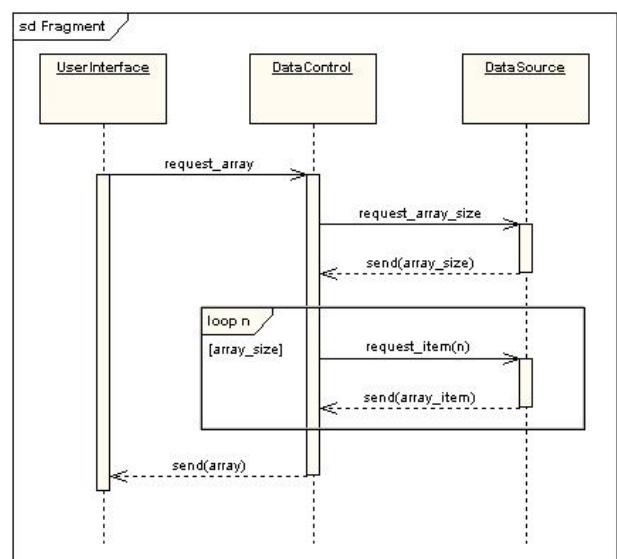
Das **alt**-Fragment ermöglicht die Existenz von mehr als einer Ablaufmöglichkeit und damit die



Modellierung von -If-else-Bedingungen-Switch/case -Anweisungen. Jeder Abschnitt kann eine eigene Bedingung enthalten. Das **opt**-Fragment steht eher für eine einzige Bedingung (if)

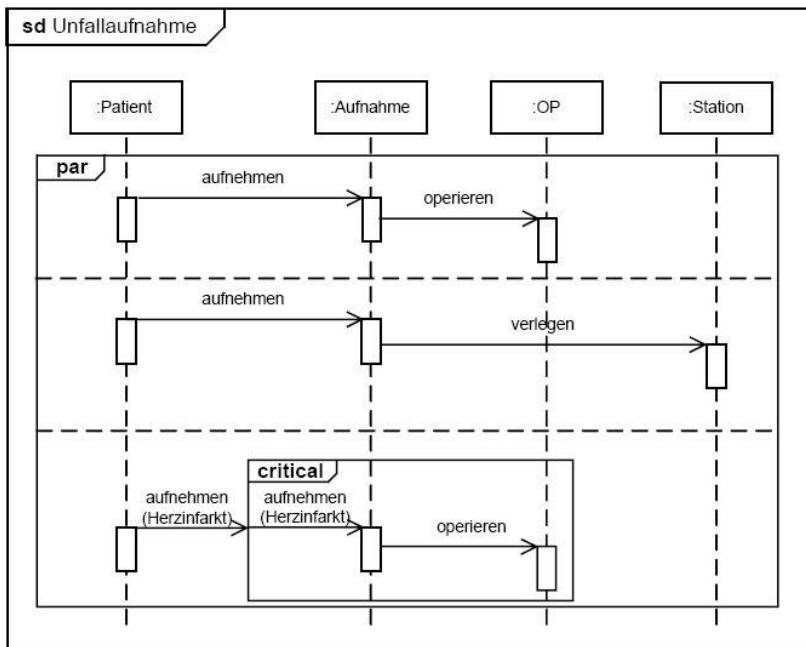
Kombinierte Fragmente – loop

Das loop-Fragment beschreibt die Anwendung von Schleifen. Es kann durch [min, max]-Informationen erweitert werden.



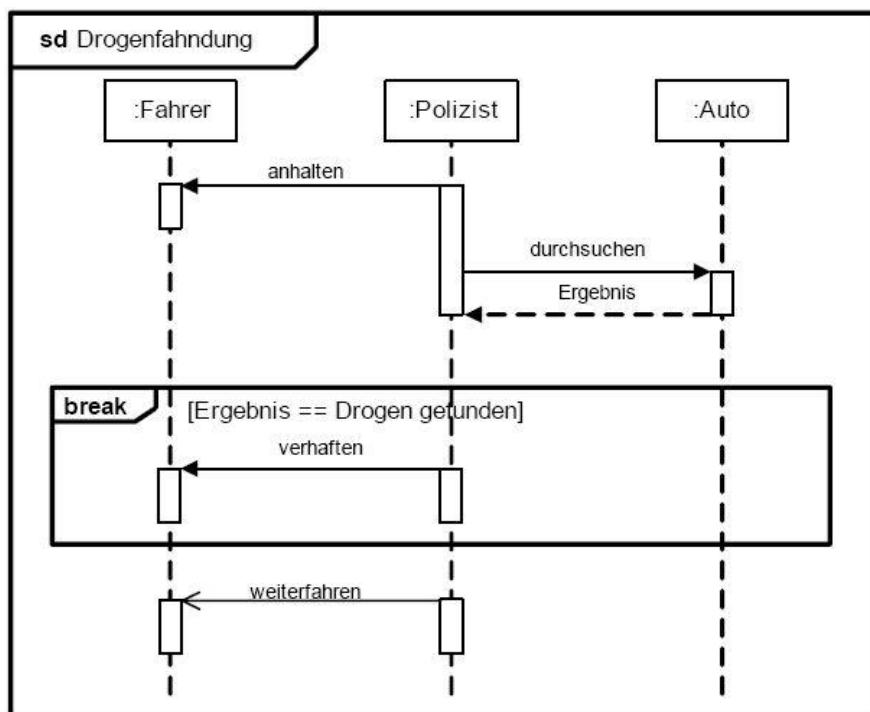
Kombinierte Fragmente – (par — critical)

Par beschreibt den gleichzeitigen Ablauf mehrerer Interaktionen, während **critical** einen nicht unterbrechbaren Ablauf beschreibt.



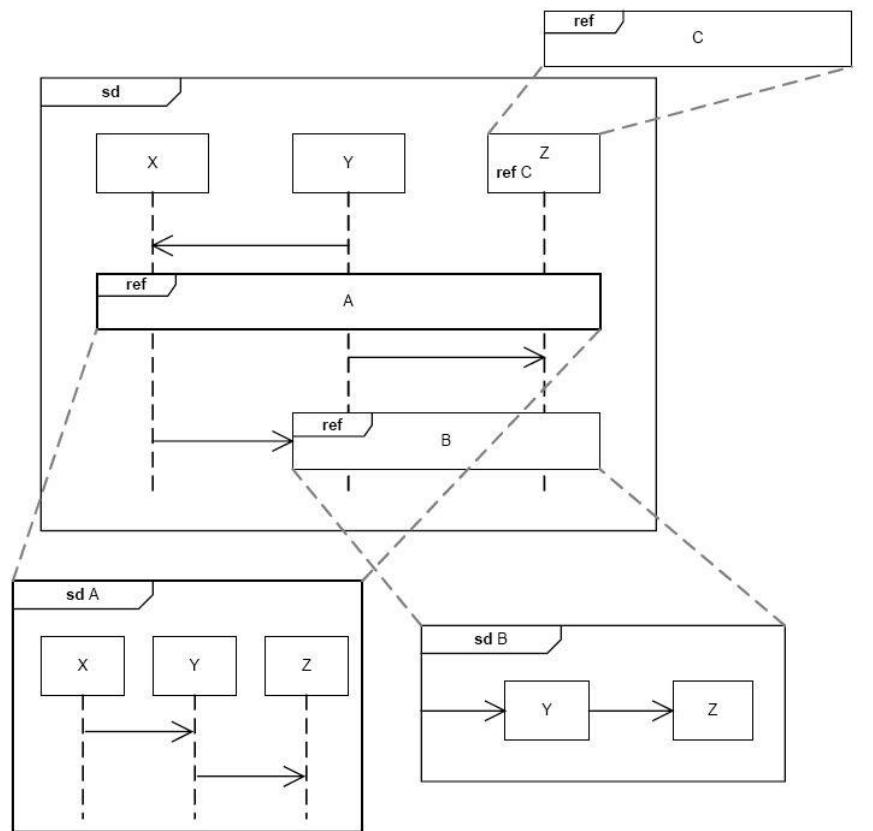
Kombinierte Fragmente – break

Das **break**-Fragment unterbricht den normalen Ablauf und ist geeignet zur Darstellung von Exceptions, Systemfehlern und kritischen asynchronen Ereignissen.



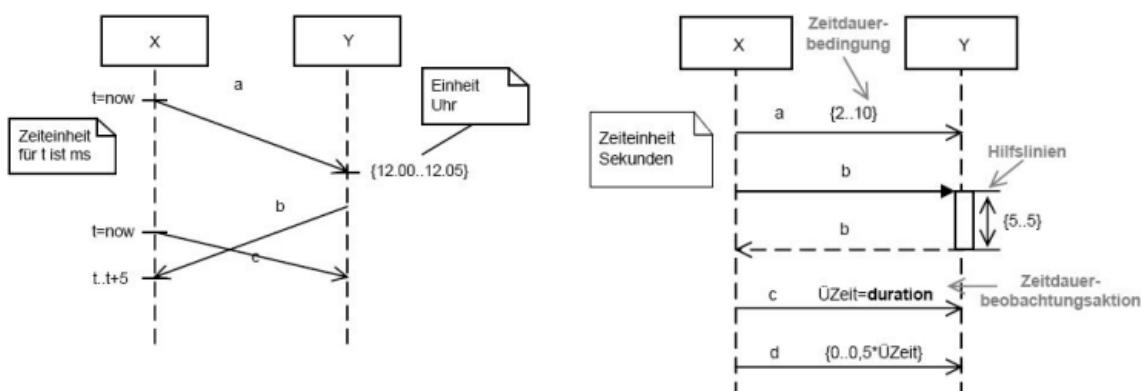
Kombinierte Fragmente – ref

Das **ref**-Feld verweist auf eine beliebige andere Interaktion und macht damit die mehrfache Verwendung von Sequenzdiagrammen möglich.



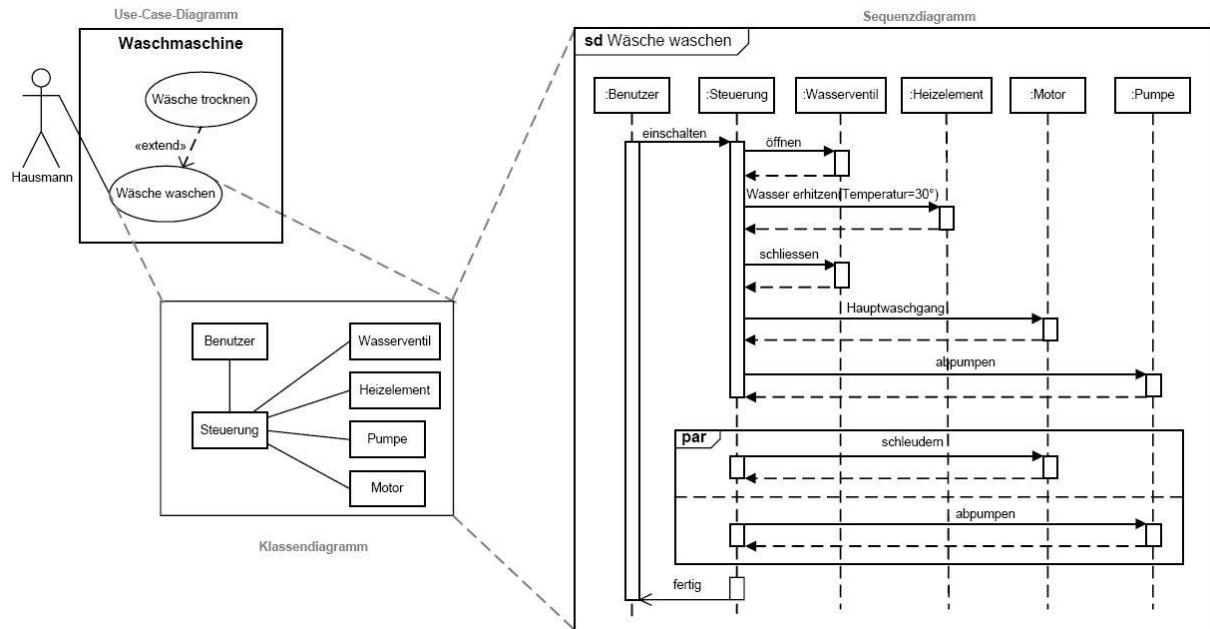
Zeitpunkt/Zeitdauer

Sie beschreiben, wann ein Ereignis eintritt bzw. die Dauer von Nachrichten oder Aktionssequenzen



Use Case und Sequenzdiagramme

Das folgende Bild beschreibt den Zusammenhang und das sich ergänzende Verhalten zwischen UseCase-, Sequenz- und Klassendiagramm.



Aufgaben

- Erstellen Sie für das Szenario des Abwickelns einer Autoreparatur ein Sequenzdiagramm.

Zunächst beauftragt der Kunde die Reparatur und übergibt danach dem Mechaniker das Auto. Dieser holt den Auftrag von der Auftragsannahme ein. Nachdem der Mechaniker die Reparatur (innerhalb von 24 Stunden) beendet hat, meldet er dies der Auftragsannahme, die daraufhin den Kunden benachrichtigt. Zum Schluss holt der Kunde sein Auto beim Mechaniker ab.

- Im Folgenden ist die Stimmenabgabe mittels eines E-Voting-Verfahrens beschrieben:



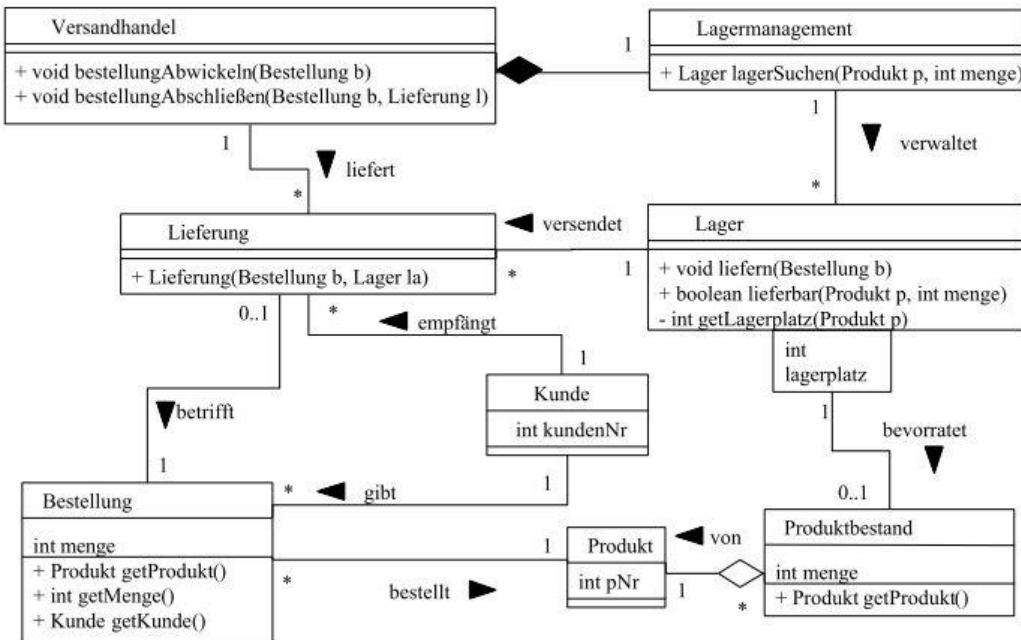
Abbildung 1: Der Wahlassistant von E-Vote - Anzeigen des Stimmzettels

Um seine Stimme bei einer Wahl abzugeben, muss der Wähler den Wahlassistanten auf seinem lokalen Rechner aufrufen. Der Wahlassistant zeigt eine Reihe von Menüoptionen an. Der Wähler wählt die Menüoption „Stimme abgeben“. Der Wahlassistant fordert den Wähler zur Eingabe des Namens der Datei auf, in der die Registrierung gespeichert ist. Nachdem der Wahlassistant die Datei mit der Registrierung eingelesen hat, fordert er den Stimmzettel beim Wahlamt an. Das Wahlamt überprüft die Registrierung des Wählers beim Wählerverzeichnis und im Anschluss, ob der Wähler noch keinen Stimmzettel abgegeben hat. Ist dies der Fall, überträgt das Wahlamt den Stimmzettel für die entsprechende Wahl an den Wahlassistanten. Der Wahlassistant zeigt dem Wähler den Stimmzettel an (siehe Abbildung 1). Der Stimmzettel besteht aus mehreren Stimmoptionen, von denen der Wähler (je nach Wahl) eine oder mehrere ankreuzen kann.

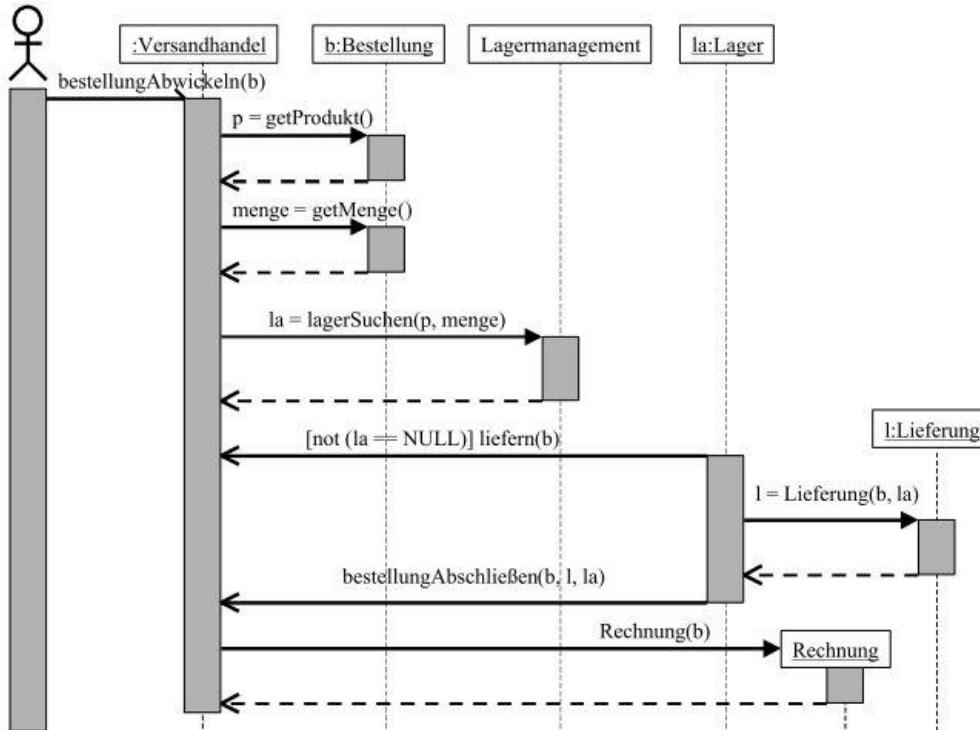
Nachdem der Wähler den Stimmzettel ausgefüllt hat, fordert der Wahlassistant ihn auf, den Wahlvorgang fortzusetzen und den Stimmzettel abzuschicken. Bestätigt der Wähler diese Aufforderung, verschlüsselt und signiert der Wahlassistant den Stimmzettel und sendet ihn an das Wahlamt, wo der Stimmzettel der Wahl zugeordnet und der Wähler als gewählt vermerkt wird. Danach wird eine Bestätigung der Stimmabgabe an den Wahlassistanten gesandt. Der Wähler hat nun seine Stimmabgabe für die Wahl abgeschlossen und kann den Wahlassistanten beenden.

Zeichnen Sie ein Sequenzdiagramm für den UseCase „Stimme abgeben“. Betrachten Sie nur den Standardfall und keine alternativen Abläufe.

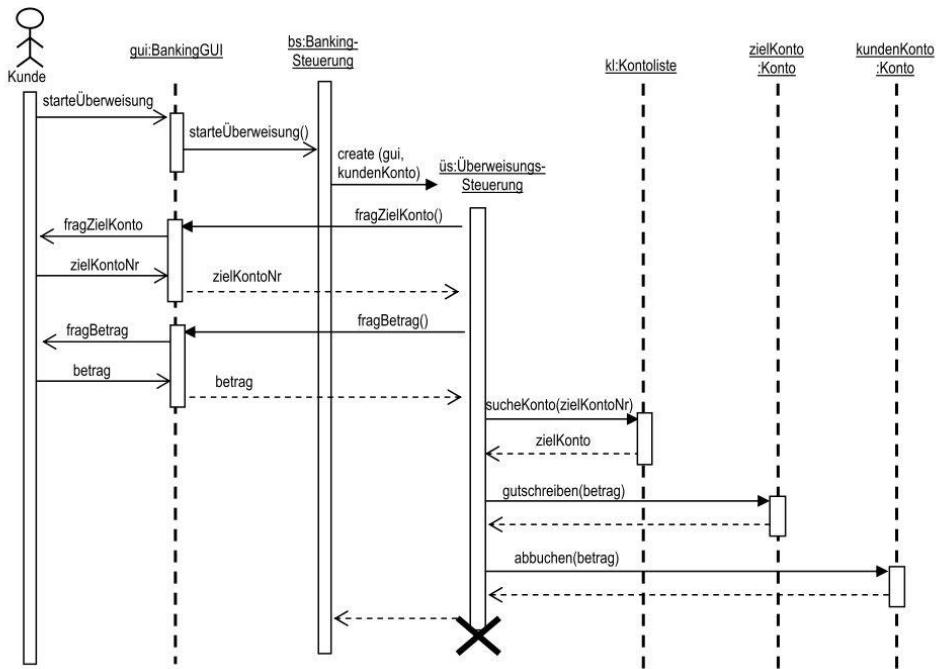
3. Gegeben sind folgendes Klassendiagramm:



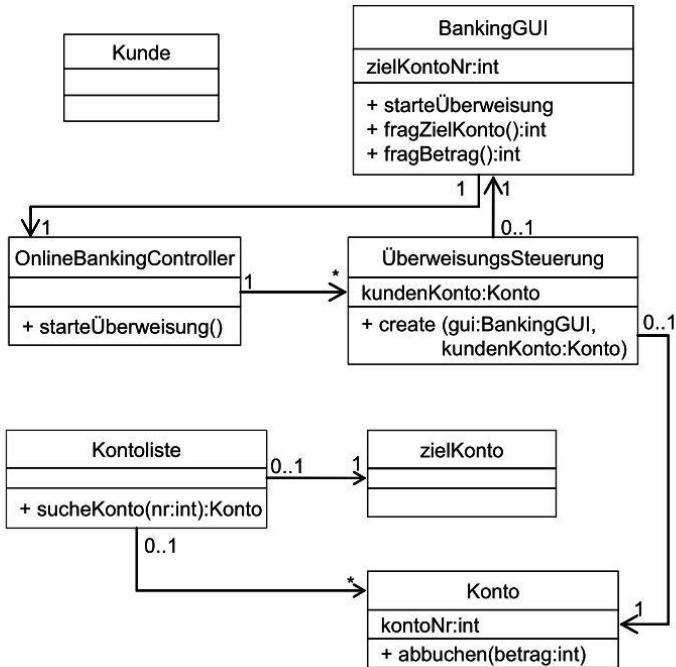
Überprüfen Sie, ob das unten dargestellte Sequenzdiagramm korrekte UML-Syntax verwendet und ob es konsistent zu dem Klassendiagramm ist. Markieren und korrigieren Sie erforderlichenfalls syntaktische Fehler und Inkonsistenzen im gegebenen Sequenzdiagramm. Begründen Sie Ihre Änderungen! (Hinweis: Sie müssen maximal 5 Korrekturen vornehmen.)



4. Gegeben sei das folgende Sequenzdiagramm aus der Feinanalyse der Produktfunktion "Überweisung ausführen":



Zu dem Sequenzdiagramm sei das folgende Analyseklassendiagramm gegeben.



Das Klassendiagramm soll nur die Dinge enthalten, die sich aus dem Sequenzdiagramm ableiten lassen. Tatsächlich enthält es aber zahlreiche Fehler und Inkonsistenzen zum Sequenzdiagramm. Markieren Sie jede fehlerhafte Stelle im Klassendiagramm und begründen Sie warum diese Stelle falsch ist.

5. Gegeben sei das folgende Szenario zur Beschreibung der Produktfunktion "Verkaufsabwicklung nach Versteigerung eines Artikels":

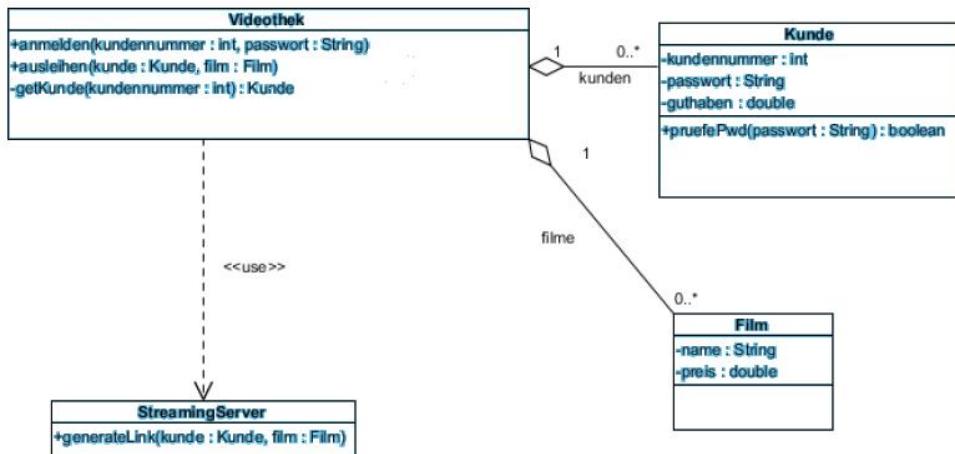
Schritt	Nutzer	Beschreibung der Aktivität
1		System teilt Verkäufer erzielten Preis mit.
2		System teilt Käufer zu zahlenden Preis mit.
3		System fordert Kontonummer vom Verkäufer an.
4	Verkäufer	Verkäufer teilt System seine Kontonummer mit.
5		System speichert Kontonummer des Verkäufers für spätere Auktionen ab.
6		System teilt Kontonummer dem Käufer zwecks Überweisung mit.

Erstellen Sie zu diesem Szenario ein Sequenzdiagramm mit den Interaktionen zwischen den Benutzern und dem System.

6. Sie sollen für die Online-Videothek den Vorgang des "Filmausleihens" modellieren.

Erstellen Sie dazu ein Sequenzdiagramm. Zur Vereinfachung können Sie davon ausgehen, dass sich das Mitglied bereits auf der Seite des gewünschten Films befindet. Wählen Sie geeignete Namen für die Elemente Ihres Diagramms.

Folgendes Klassendiagramm ist zur Zeit vorhanden und sollte von Ihnen genutzt bzw. erweitert werden.

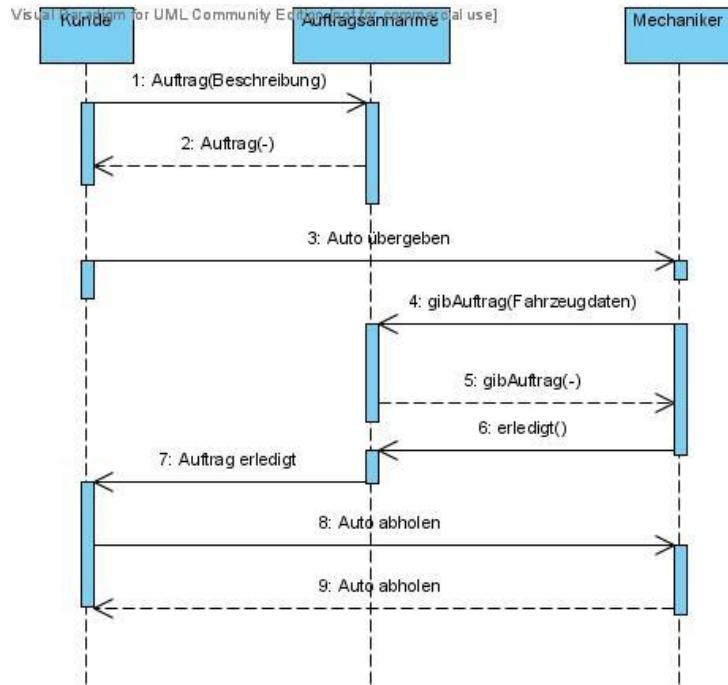


- Um den Film auszuleihen, muss das Mitglied sich zunächst erfolgreich anmelden. Dies wurde im Sequenzdiagramm SD_Login bereits beschrieben..
- War die Anmeldung erfolgreich, versucht die Person den Film auszuleihen. -
- Die Videothek berechnet zuerst, ob das Guthaben reicht um den Film zu bezahlen.
- Reicht das Guthaben nicht aus, wird stattdessen eine Aufforderung zum Auffüllen des Guthabens angezeigt.
- Falls das aktuelle Guthaben des Mitglieds ausreicht, veranlasst die Videothek einen Streaming-Server einen Link für den Film zu generieren.
- Die Videothek zeigt dem Benutzer den Link an, unter dem der Film zugreifbar ist.

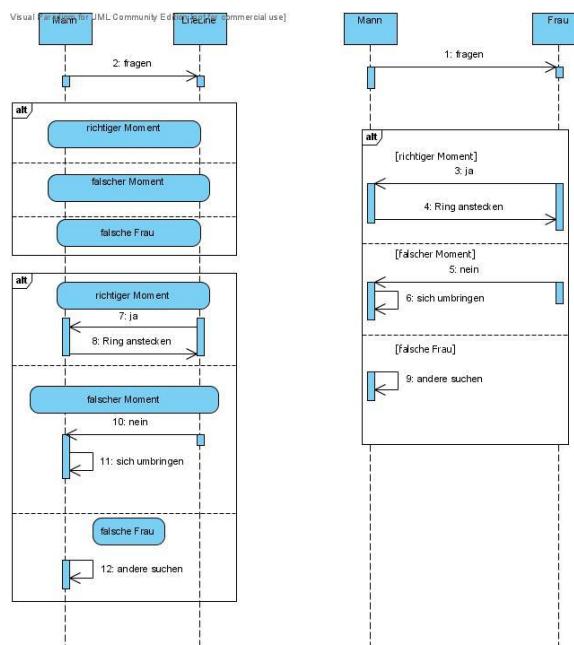


Lösung zu Sequenzdiagrammen

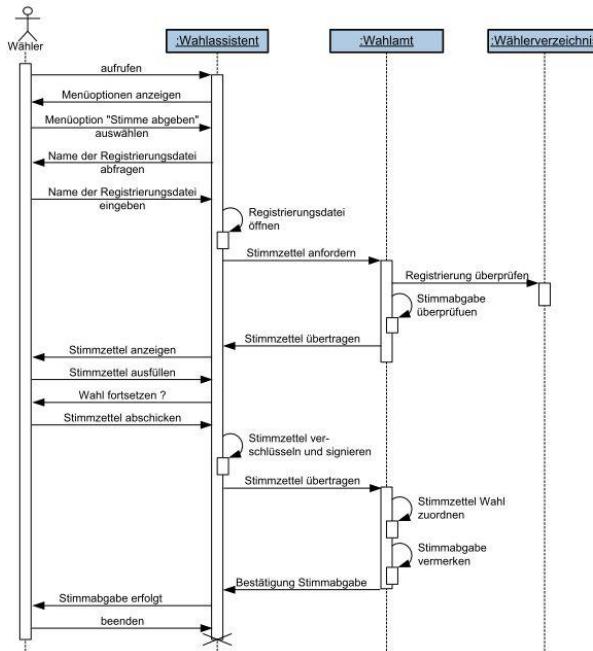
- Autoreparatur



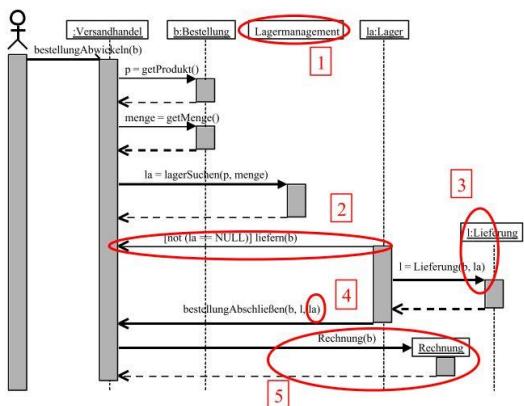
- Verlobung



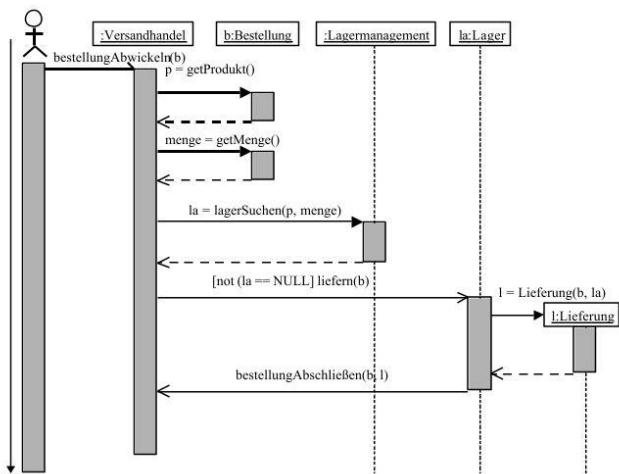
- E-Voting



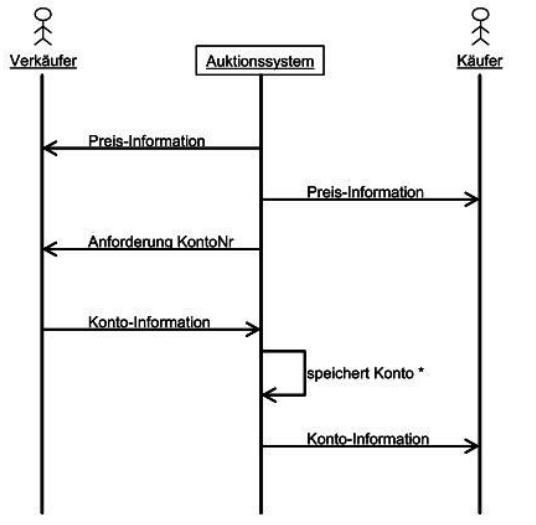
- kd_Sequenz



1. Lagermanagement muß als Objekt verwendet werden - Lagermanagement, im Sequenzdiagramm sind keine Klassen erlaubt (alternativ kann das Objekt auch benannt sein – l: Lagermanagement)
2. liefern(Bestellung b) ist Methode der Klasse Lager, nicht von Versandhandel, der Nachrichtenpfeil muß also in die andere Richtung zeigen
3. lieferung(b, la) ist ein parametrisierter Konstruktorauftrag, die Nachricht muß also direkt auf das Objektsymbol und nicht auf die Lebenslinie zeigen
4. bestellungAbschließen hat im Klassendiagramm nur zwei Parameter der Typen Bestellung und Lieferung, d.h. der dritte Parameter vom Typ Lager ist zu streichen
5. Eine Klasse Rechnung ist im Klassendiagramm nicht vorhanden (und somit ein entsprechender Konstruktorauftrag ebenfalls unmöglich)

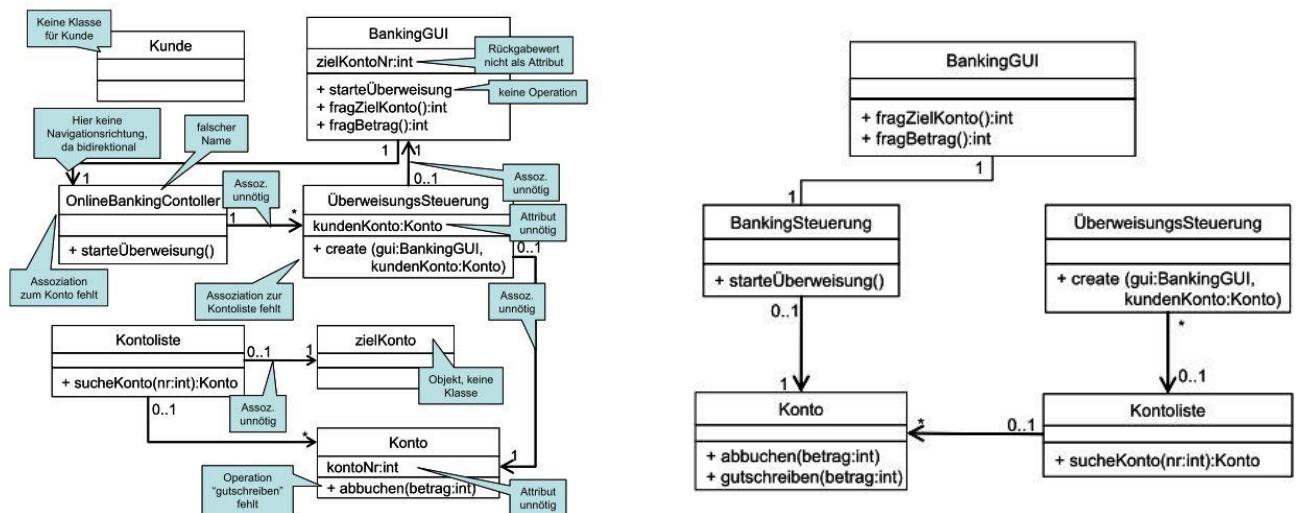


- TSE_Ebay

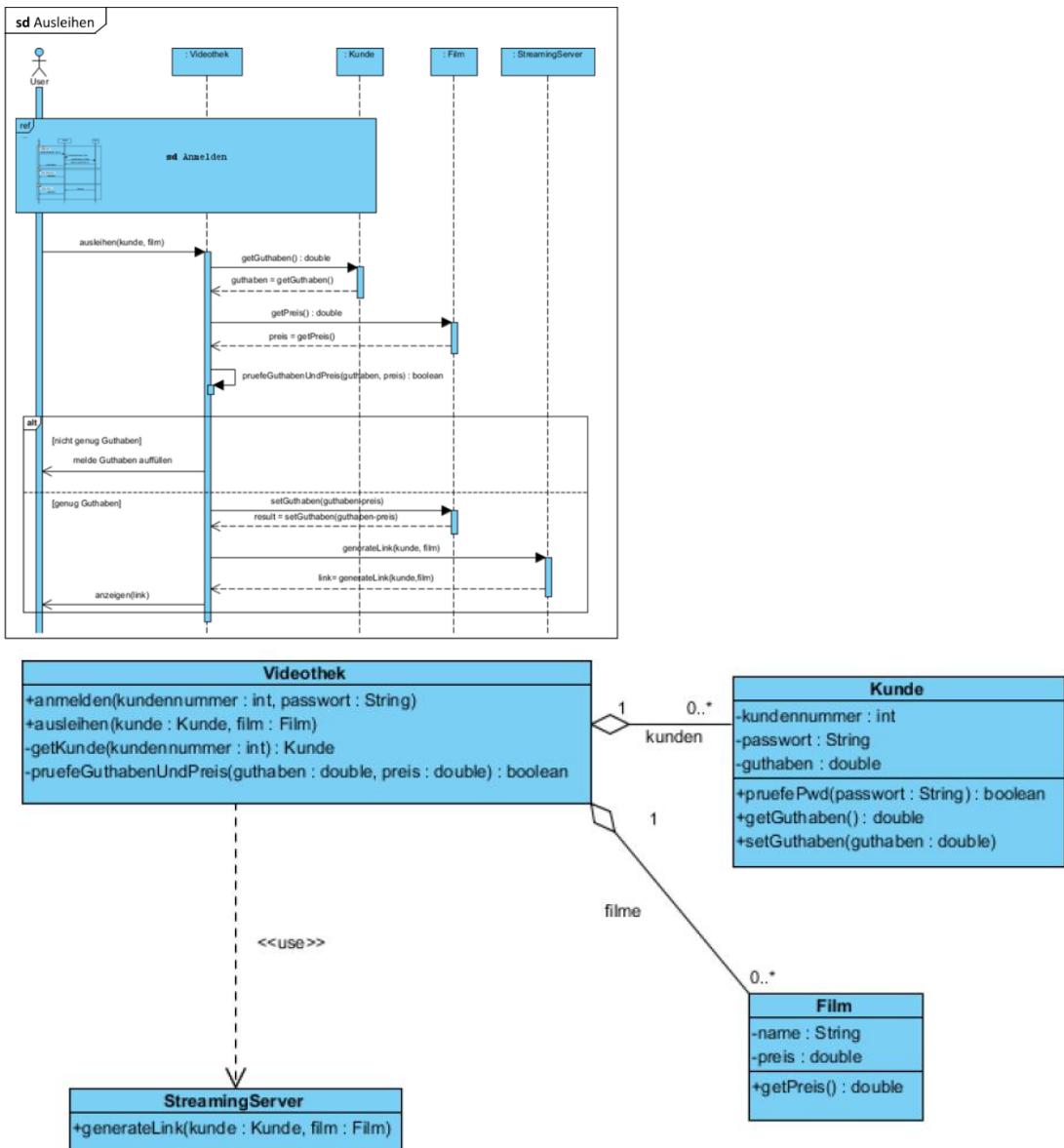


* optional

- Lösung Sequenz_Bank_Loesung

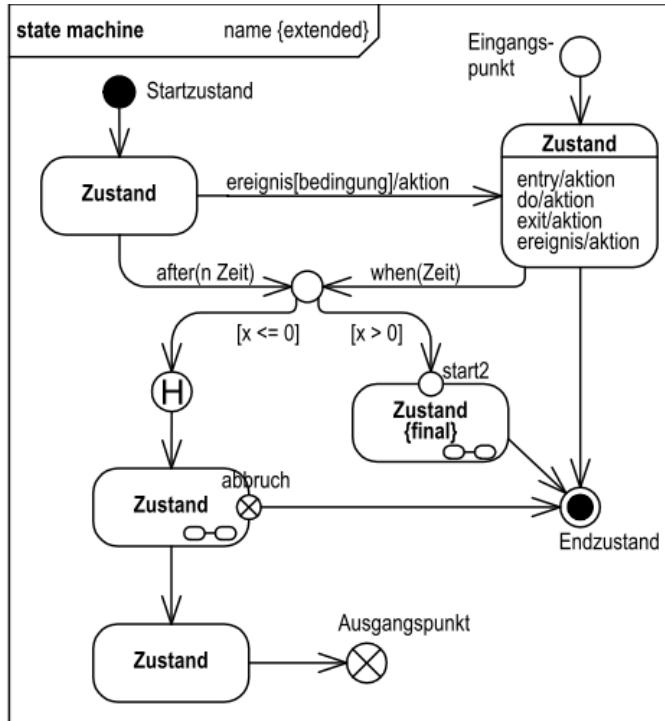


- Lösung Videothek

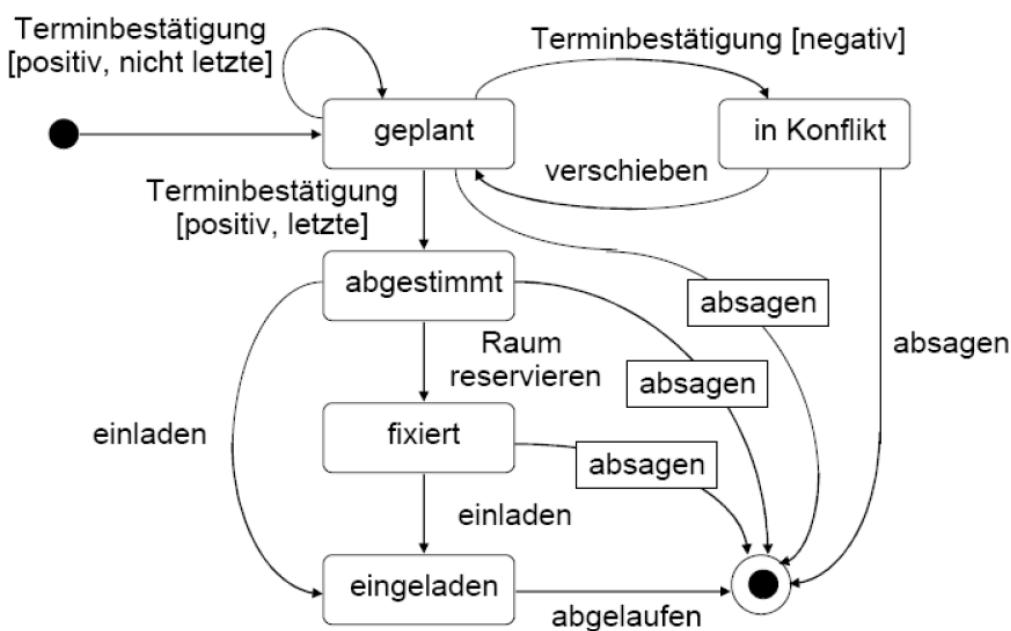


Zustandsdiagramm

Die OOP verwendet Zustandsdiagramme zur graphischen Darstellung des Zustandsautomaten (finite state machine). Es dient zur Darstellung der Zustände eines Objektes in seinem Lebenszyklus sowie der Abbildung von Zustandsänderungen und auslösenden Ereignissen.

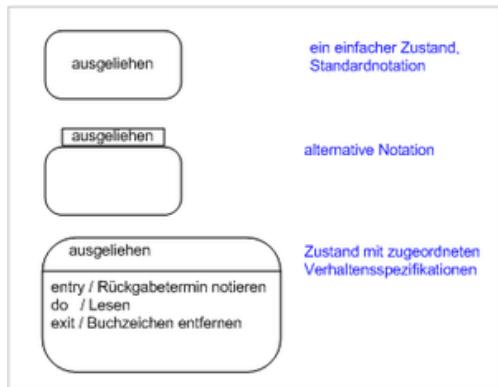


Ein Zustand ist eine Zeitspanne, in der ein Objekt auf ein Ereignis wartet, d.h. das Objekt verweilt eine bestimmte Zeit in diesem Zustand. In diesen Zustand gelangt das Objekt durch ein bestimmtes Ereignis. Das Ereignis selbst hat keine Dauer. Der Übergang zwischen zwei Zuständen ist die **Transition**. Ein Objekt kann - nacheinander - mehrere Zustände durchlaufen, es befindet sich zu einem bestimmten Zeitpunkt aber immer in genau einem Zustand. Tritt in einem beliebigen Zustand ein neues Ereignis ein, so ist der nächste Zustand sowohl vom aktuellen Zustand als auch vom Ereignis abhängig.



Modellelemente

Zustand



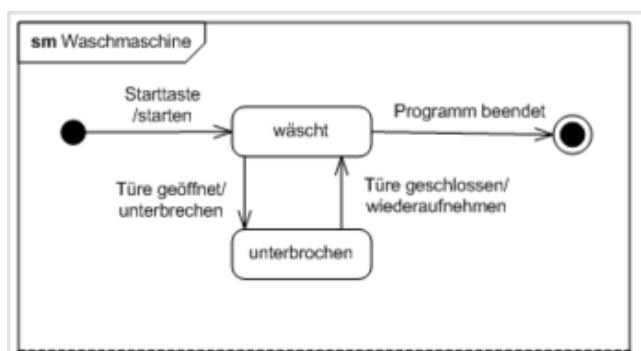
Ein Zustand wird im Diagramm als Rechteck mit abgerundeten Ecken dargestellt und mit dem Namen des Zustands beschriftet. Befindet sich ein Objekt in einem Zustand, so können alle sogenannten inneren Aktivitäten auf diesem Objekt ausgeführt werden, die in diesem Zustand spezifiziert sind. In diesem Fall ist die Darstellung des Zustands zweigeteilt. Im oberen Abschnitt des Rechtecks kann der Name des Zustands notiert werden. Im unteren Abschnitt können u.a. innere Aktivitäten angeführt werden, wobei eine Aktivität mehrere Aktionen beinhalten kann. Ein Zustand modelliert eine Situation, in der eine bestimmte, unveränderliche Bedingung gilt. Meistens ist diese Invariante nur implizit gegeben, will man sie explizit formulieren, kann man sie als Einschränkung dem Zustand zuordnen.

Dem Zustand können drei Verhaltensspezifikationen, zum Beispiel in Form einer Aktivität oder einer Interaktion, zugeordnet werden:

- ein Verhalten, das ausgeführt wird, wenn der Zustandsautomat in den Zustand eintritt (engl. entry behaviour)
- ein Verhalten, das ausgeführt wird, wenn der Zustandsautomat den Zustand verlässt (engl. exit behaviour)
- ein Verhalten, das ausgeführt wird, während sich der Zustandsautomat im Zustand befindet (engl. doActivity)
- ein Verhalten, das ausgeführt wird, wenn ein bestimmtes Event eintritt. (eng. event behaviour)

Im Gegensatz zu den ersten drei Verhalten wird beim event Verhalten ein tatsächliches Event im Zustand angegeben. (z.B.: mouseOver / peep)

Daneben gibt es noch sog. Pseudozustände, die vor allem den Beginn und das Ende des Objektlebenszyklus dokumentieren.



Transition

Eine Transition (Übergang) verbindet einen Quell- und einen Zielzustand. Der Transition kann eine Verhaltensspezifikation zugeordnet sein, die das Verhalten beschreibt, das ausgeführt wird, wenn die Transition durchlaufen wird. Dieses Verhalten heißt Effekt (engl. effect). Ein Wächterausdruck (engl. guard) kann die Transition schützen: die Transition kann nur durchlaufen werden, wenn der Wächterausdruck wahr ist. Häufig wird aufgrund eines Ereignisses ein Zustandsübergang erfolgen.

Aufgaben

Geldautomat

Ein Kartenautomat eines Parkhauses befindet sich im Zustand bereit. Nach dem Einziehen einer Karte wird diese auf Korrektheit geprüft. Ist die Karte falsch, wird sie wieder ausgeworfen. Ist die Karte in Ordnung, wird der zu zahlende Gesamtbetrag angezeigt. Sollte das eingeworfene Geld nicht ausreichen, wird der jeweilige Restbetrag angezeigt, ansonsten wird die Karte wieder ausgegeben. Anschließend hat der Kunde 5 Sekunden Zeit eine Quittung anzufordern. Während des Geldeinwerfens ist die prüft der Automat, ob der eingeworfene Betrag bereits ausreichend ist.

Bibliothek

Buch
erfassen()
ausleihen()
zurückgeben()
vorbestellen()
entfernen()

Wenn in einer Bibliothek ein Buch beschafft wird, dann werden seine Daten erfasst und ein neues Objekt der Klasse Buch erzeugt. Der Einfachheit halber gebe es von jedem Buch nur ein einziges Exemplar. Jedes Buch kann ausgeliehen werden. Wird ein ausgeliehenes Buch von einem anderen Leser gewünscht, dann muss es vorbestellt werden. Nicht vorbestellte Bücher stehen nach der Rückgabe sofort für eine erneute Ausleihe bereit. Vorbestellte Bücher werden nach der Ausleihe für die entsprechenden Leser zur Abholung bereitgelegt und der Leser wird informiert. Wird das Buch nicht fristgemäß abgeholt, dann steht es für eine erneute Ausleihe bereit. Defekte Bücher oder Bücher, die nicht zurückgegeben wurden, werden aus dem Bestand entfernt. Wenn ein neues Buch im System gespeichert wird, dann befindet es sich zunächst im Zustand präsent.

Mietwagen

Spezifizieren Sie für die Klasse Mietwagen den Lebenszyklus. Die Wirkung der Operationen ist wie folgt definiert:

- **vermieten**

Ein neues Objekt von Mietvertrag erzeugen und entweder vorhandenem Kunden zuordnen oder neuen Kunden erfassen

- **zurücknehmen**

Gibt der Kunde den Mietwagen vertragsgemäß zurück, so wird das Auto gewaschen und überprüft

- **bereitstellen**

Ergibt sich bei der Prüfung die einwandfreie Funktion, so wird der Wagen wieder in den Fuhrpark überstellt

- **anmeldenWerkstatt**

Ein defekter Mietwagen wird zur Reparatur angemeldet. Sobald ein Mietwagen repariert ist, wird er wieder bereitgestellt.

- **ausmustern**

Ergibt sich bei der Prüfung, dass der Kilometerstand größer als 80000 km ist, dann wird er aus dem Fuhrpark des Mietwagenverleihs entfernt und zum Verkauf bereitgestellt. Ist der Wagen defekt, dann wird er vorher noch repariert, um einen höheren Verkaufspreis zu erzielen.

Sommer 2012

Die B&G GmbH verkauft Immobilien.

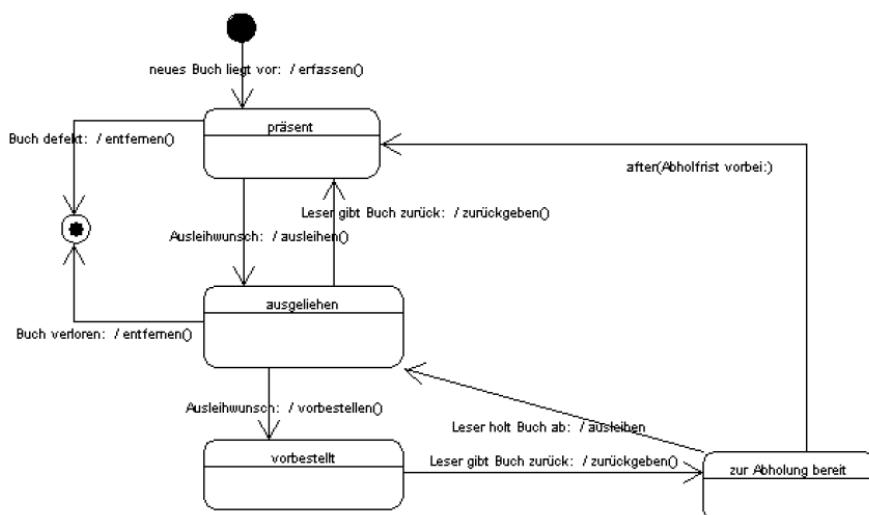
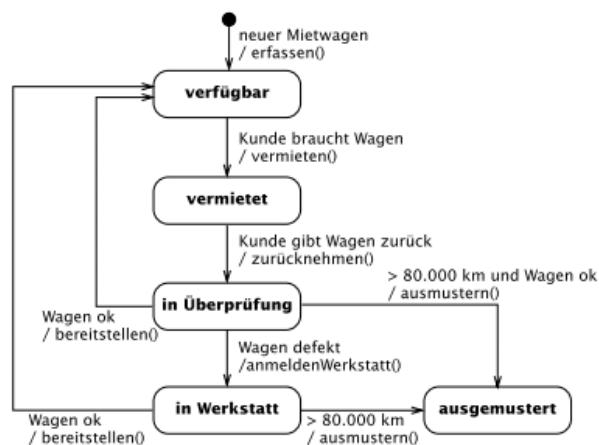
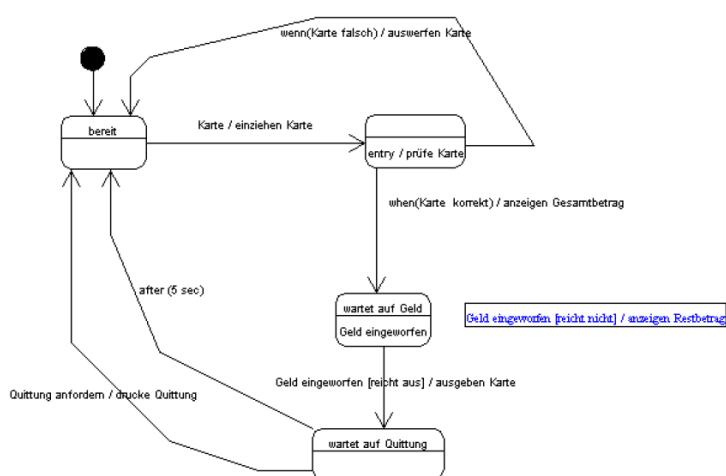
Eine Verkaufsimmobilie kann die folgenden Zustände haben.

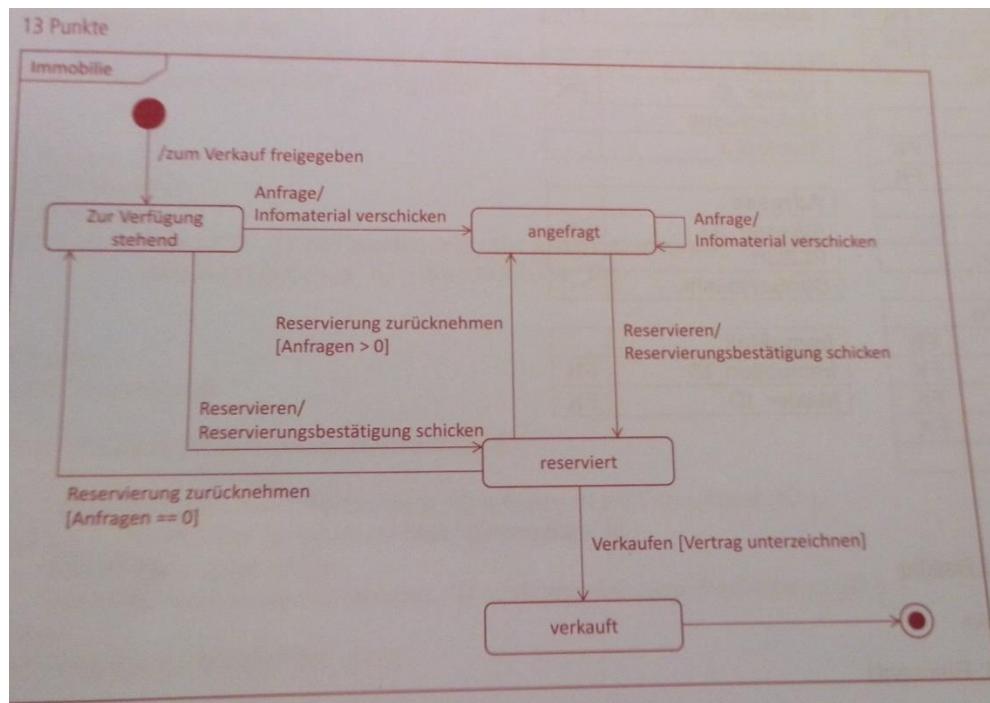
Wenn Sie zum Verkauf freigegeben wurde, steht sie zur Verfügung. Für zur Verfügung stehende Immobilien können Anfragen entgegengenommen werden. Die Immobilie ist damit angefragt. Für angefragte Immobilien können weitere Anfragen erfolgen. Zur Verfügung stehende oder angefragte Immobilien können reserviert werden (nur eine Reservierung ist möglich). Die Immobilie ist dann reserviert. Reservierte Immobilien können verkauft werden. Mit dem Unterschreiben des Verkaufsantrages ist er Endzustand erreicht.

Hinweis: - Auf jede Anfrage wird Informationsmaterial verschickt. - Anfragen bleiben bei einer Reservierung bestehen. - Bei einer Reservierung wird eine Reservierungsbestätigung verschickt. - Reservierungen können zurückgenommen werden.

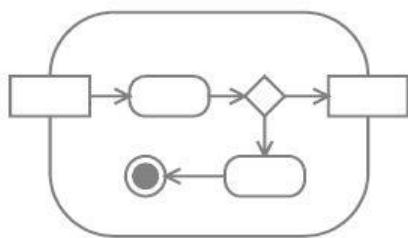
Erstellen Sie ein UML-Zustandsdiagramm. (13 P)

Lösung





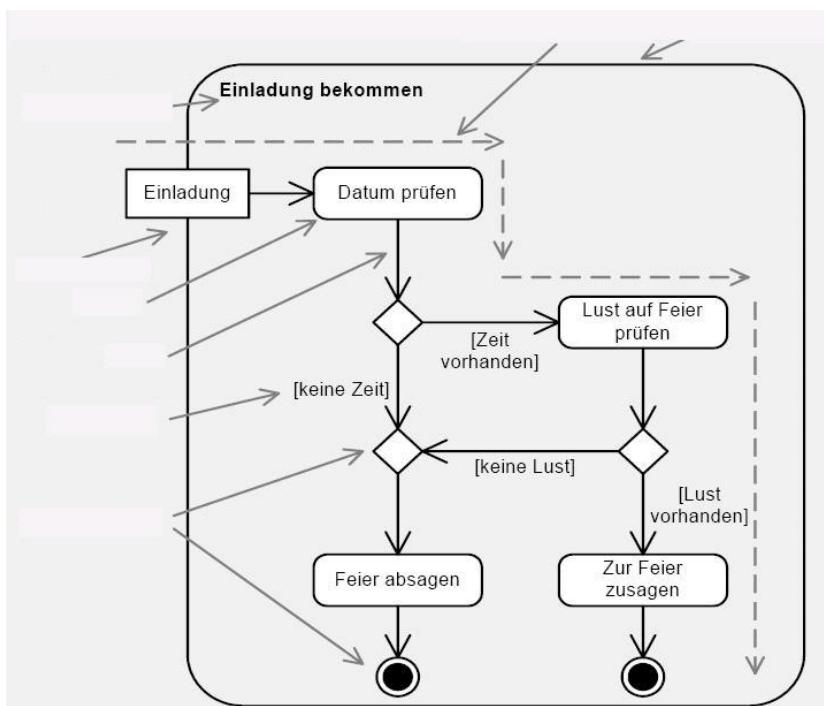
Aktivitätsdiagramm



Aktivitätsdiagramme visualisieren Abläufe, die zu unterschiedlichen Projektzeitpunkten mit stark variierendem Detaillierungsgrad modelliert werden. Sie werden zur Geschäftsprozessmodellierung, zur Beschreibung von Use-Cases oder Implementierung einer Operation eingesetzt.

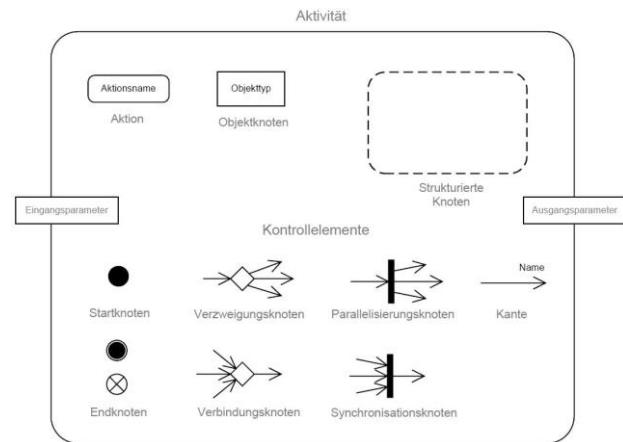
Das Aktivitätsdiagramm geht auf die Frage ein: **Wie realisiert mein System ein bestimmtes Verhalten?** Es steht also eine vom System zu bewältigende Aufgabe im Vordergrund, die es in Einzelschritte zu zerlegen gilt. Ein Aktivitätsdiagramm zeigt dabei im Wesentlichen folgende Elemente:

- eine oder mehrere Aktivitäten
- Aktionen
- Objektknoten
- Kontrollelemente zur Ablaufsteuerung
- verbindende Kanten

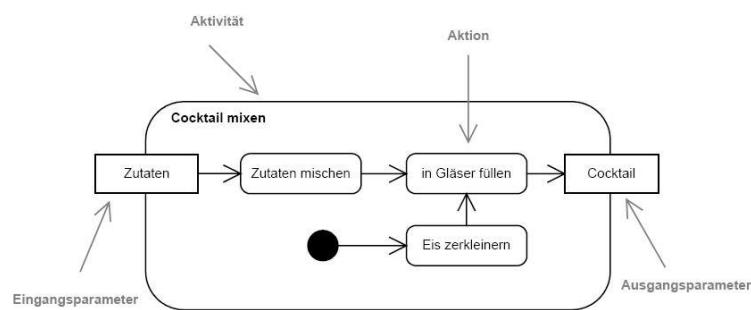


Modellelemente

Das Aktivitätsdiagramm verfügt über folgende Notationselemente:



Aktivität

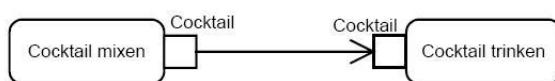


Sie wird durch ein großes Rechteck mit abgerundeten Ecken dargestellt. Es symbolisiert beispielsweise einen UseCase und beschreibt das komplette Diagramm. Es kann über Ein- und Ausgangsparameter verfügen.

Im Gegensatz dazu sind **Aktionen** Verhaltensaaufrufe. Die Summe der Aktionen realisiert die Aktivität.

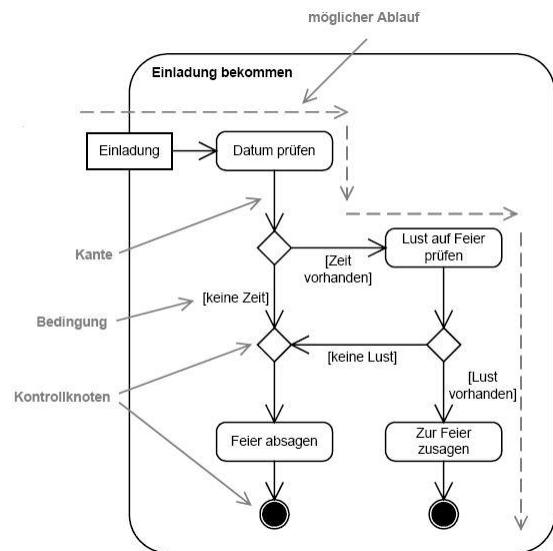
Objektknoten

- Objektknoten repräsentieren nicht das Objekt selber, sondern den Typ des Objekts
- Darstellung als Objektknoten oder durch Pin-Notation
- Objektknoten als Datenspeicher



Kontrollelemente

Kontrollelemente steuern den Ablauf der Aktivität.



Startknoten



- Starten den Ablauf
- Mehrere Startknoten initiieren parallele Abläufe

Endknoten



- Beenden die Aktivität oder einen einzigen Ablauf

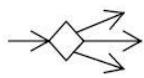


Kanten



- Übergänge zwischen zwei Knoten
- Beschreiben die Ablaufrichtung

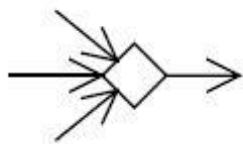
Verzweigungsknoten



- Ermöglichen alternative Abläufe
- Bedingungen geben Ablaufrichtung vor

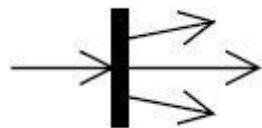


Verbindungsknoten



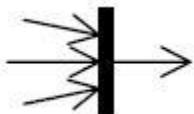
- Vereinigen mehrere Abläufe

Parallelisierungsknoten



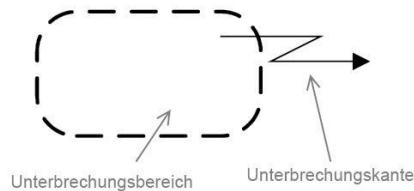
- Aufteilen in parallele Abläufe

Synchronisationsknoten



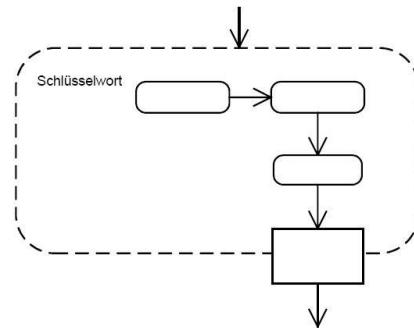
- Zusammenführen paralleler Abläufe

Unterbrechungsbereich



- Umfasst ein oder mehrere Aktionen
- Kann über Unterbrechungskante verlassen werden
- Alle Aktionen im Bereich werden dann gestoppt

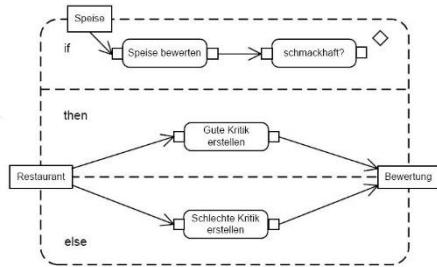
Strukturierte Knoten



- Umfassen mehrere Elemente einer Aktivität
- Aktivitätsablauf startet erst dann, wenn an allen eingehenden Kanten Token anliegen
- Objektknoten als Ein- und Ausgangsparameter möglich

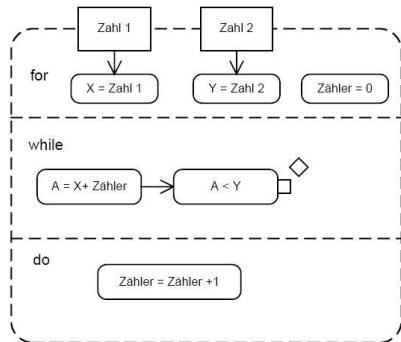


Entscheidungsknoten



- Visualisierung komplexer Entscheidungen
- if: prüfen der Bedingung
- then: auszuführende Elemente:
- else: möglicher Ablauf, wenn kein if-Bereich zutrifft
- else if: wie if-Bereich nur mit vorgegebener Prüfreihenfolge

Schleifenknoten

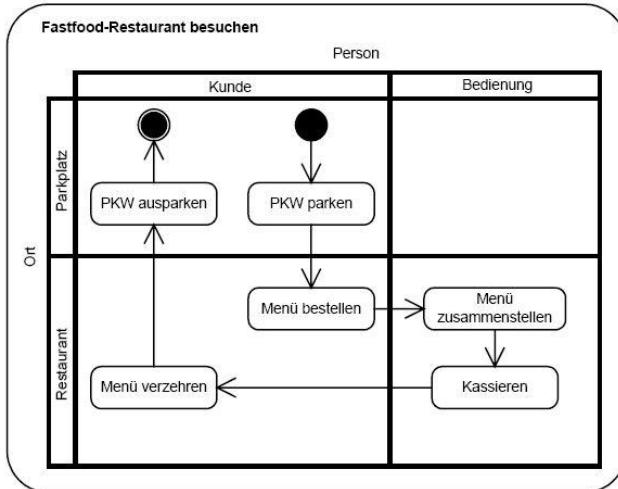


- Visualisieren den Ablauf komplexer Schleifen
- for: initiale Aufgaben
- while: prüfen der Bedingung für einen Durchlauf
- do: Schleifenrumpf
- Reihenfolge do ?while ist optional

Aktivitätsbereiche (Swimlanes)

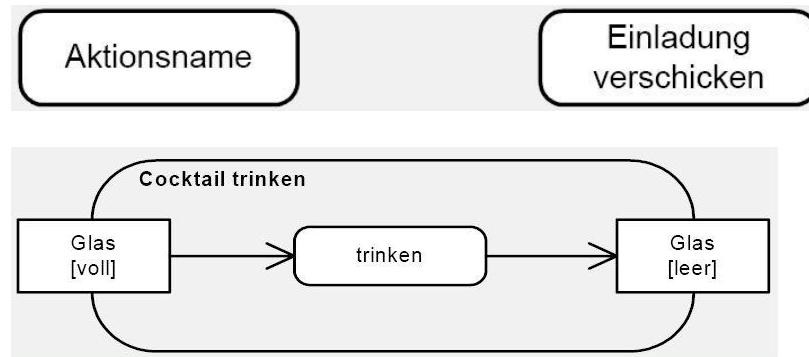
Aktivitätsbereiche dienen dazu, den Aktivitäten die verantwortlichen Objekte zuzuordnen.

- Einteilen des Diagramms in Bereiche mit gemeinsamen Eigenschaften
- Hierarchische und mehrdimensionale Aufteilung möglich



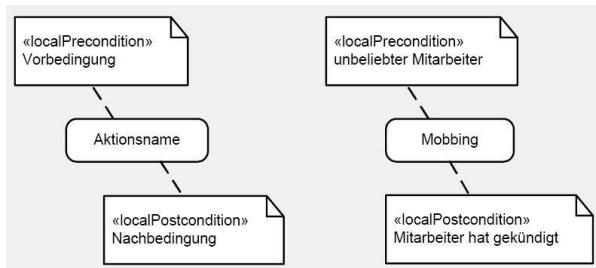
Aktion

Darstellung durch ein Rechteck mit abgerundeten Ecken mit Namen oder Text zur Operation



- Aufruf von Verhalten oder Bearbeitung von Daten
- Einzelschritt der Aktivität
- Zentrales Element des Aktivitätsdiagramms
- Über Kanten mit anderen Elementen verbunden

Vor- und Nachbedingung



Sind bei Aktionsstart oder -ende Bedingungen zu beachten, werden als Notizzettel notiert.

Signale und Ereignisse

Sonderformen der Aktion; SendSignalAction und

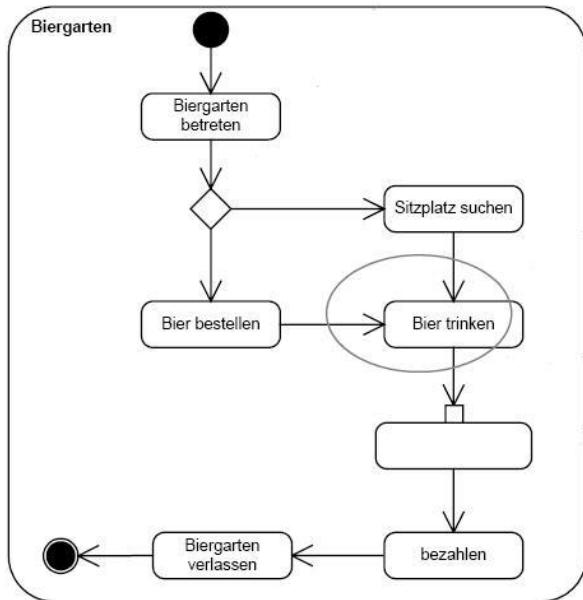


AcceptEventAction



Fragen zu Aktivitätsdiagramm

1. Was ist formal an folgendem Aktivitätsdiagramm falsch



2. Die Aktivität **Vergessen** soll einen normalen Kneipenabend eines Fachinformatikers abilden.

Nachdem der Guest die Kneipe betreten hat, wird der Inhalt der Geldbörse mit dem Bierpreis verglichen. Ist noch Geld vorhanden, wird ein Bier bestellt, bezahlt und getrunken.

Solange noch Geld vorhanden ist, wird dieser Vorgang wiederholt. Wenn das Geld nicht mehr ausreicht, wird die Kneipe verlassen.

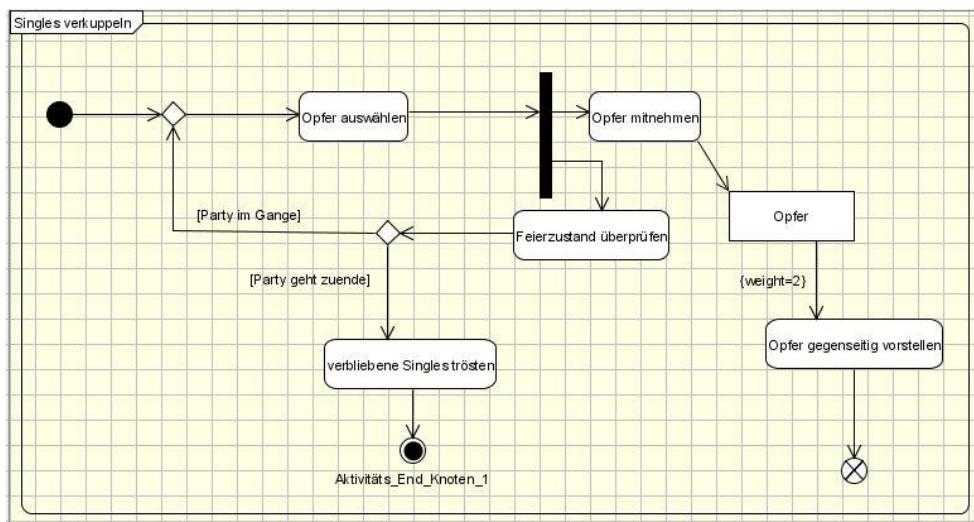
3. Sie sollen durch ein Aktivitätsdiagramm das typische Verhalten eines Gourmettesters abbilden

Der Tester testet die Speise und bewertet sie. Falls das Essen schmackhaft ist, erhält das Restaurant eine gute Bewertung, ansonsten eine schlechte.

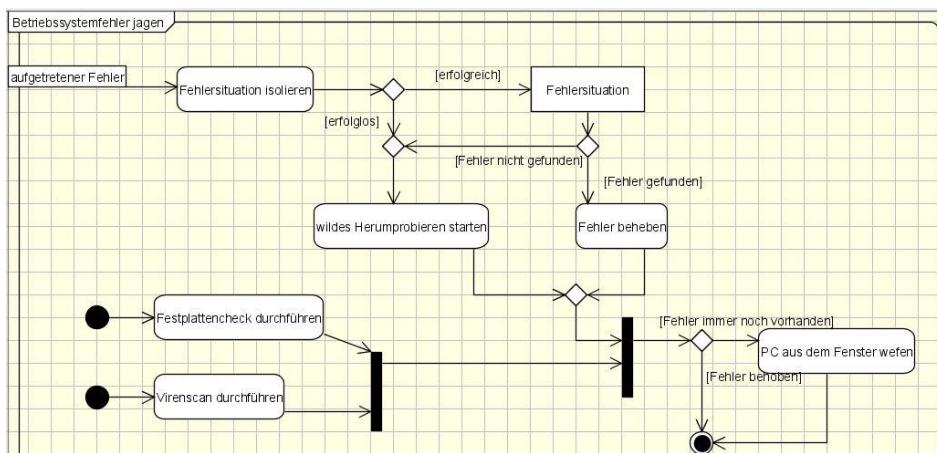
4. Bilden Sie folgendes kleines Programm in einem Aktivitätsdiagramm ab.

```
double getAverageSpeed(double distance, double time) throws DivisionByZeroException
{
    if(time == 0) throw new DivisionByZeroException();
    else
        return (distance/time)
}
```

5. Erläutern Sie das folgende Aktivitätsdiagramm

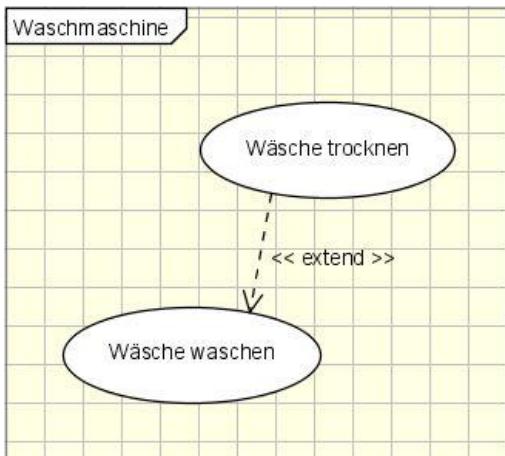


6. Erläutern Sie die untenstehende Abbildung. Betriebssystemfehler_jagen



Wieviel parallelisierte Abläufe sind entstanden ?

7. Formulieren Sie aus folgendem UseCase ein Aktivitätsdiagramm. Benutzen Sie dabei ihre eigenen Kenntnisse oder die ihres Lebensabschnittsgefährten.



8. Im Rahmen eines Softwareentwicklungsprojekts soll ein terminalgestütztes Check-In-System entwickelt werden, das es einem Gast erlaubt, sich selbst in ein Hotel einzuchecken. Hierbei können die Daten einer Reservierung (Zimmerkategorie und Zeitraum) für die Belegung übernommen werden, oder es können neue Belegungsdaten ausgewählt werden.

Betrachten Sie die folgenden Szenarien zu der Produktfunktion “Zimmer belegen” aus Sicht der Benutzerrolle Gast. Erstellen Sie ein Aktivitätsdiagramm, das den allgemeinen Ablauf zu dieser Produktfunktion beschreibt und die gegebenen Szenarien erlaubt. Gehen Sie dabei in den folgenden Schritten vor:

- Erstellen Sie ein Aktivitätsdiagramm für den Standardablauf

Szenario für den Standardablauf (Erfolg)

Schritt	Nutzer	Beschreibung der Aktivität
1	Gast	bei Check-In-System anmelden
2		Check-In-System zeigt Reservierungsdaten an
3	Gast	Reservierungsdaten für Belegung übernehmen
4		Check-In-System zeigt verfügbare Zimmer der reservierten Zimmerkategorie an
5	Gast	Zimmer auswählen
6		Check-In-System zeigt Belegungsinformation an

- Ergänzen Sie in dem obigen Aktivitätsdiagramm die Informationen für die gegebenen alternativen Abläufe.

Szenarien für alternative Abläufe (Misserfolg oder Umwege zum Erfolg)

Schritt	Bedingung, unter der Alternative eintritt	Beschreibung der Aktivität
2	es liegen keine Reservierungsdaten vor	Auswahl dialog für Zimmerkategorie und Belegungszeitraum anzeigen
3	es liegen keine Reservierungsdaten vor	Zimmerkategorie auswählen
3	es liegen keine Reservierungsdaten vor	Belegungszeitraum auswählen
3	ein Zimmer der ausgewählten Kategorie ist für den ausgewählten Zeitraum nicht verfügbar	Auswahl dialog für Zimmerkategorie und Belegungszeitraum anzeigen
3	Belegung soll von Reservierungsdaten abweichen	Auswahl dialog für Zimmerkategorie und Belegungszeitraum anzeigen

Ergänzen Sie das Aktivitätendiagramm an geeigneter Stelle um interne Aktivitäten

- für die Prüfung, ob eine Reservierung für den angemeldeten Guest vorliegt,
 - die Prüfung, ob ein Zimmer der ausgewählten Kategorie im ausgewählten Zeitraum verfügbar ist,
 - das Löschen der Reservierung, sobald ihre Daten nicht mehr benötigt werden,
 - das Belegen des Zimmers.
- Gibt es in Ihren obigen Lösungen Möglichkeiten zur Parallelisierung von Aktivitäten, die bisher sequentiell ausgeführt werden? Falls ja, geben Sie eine solche Parallelisierungen an.



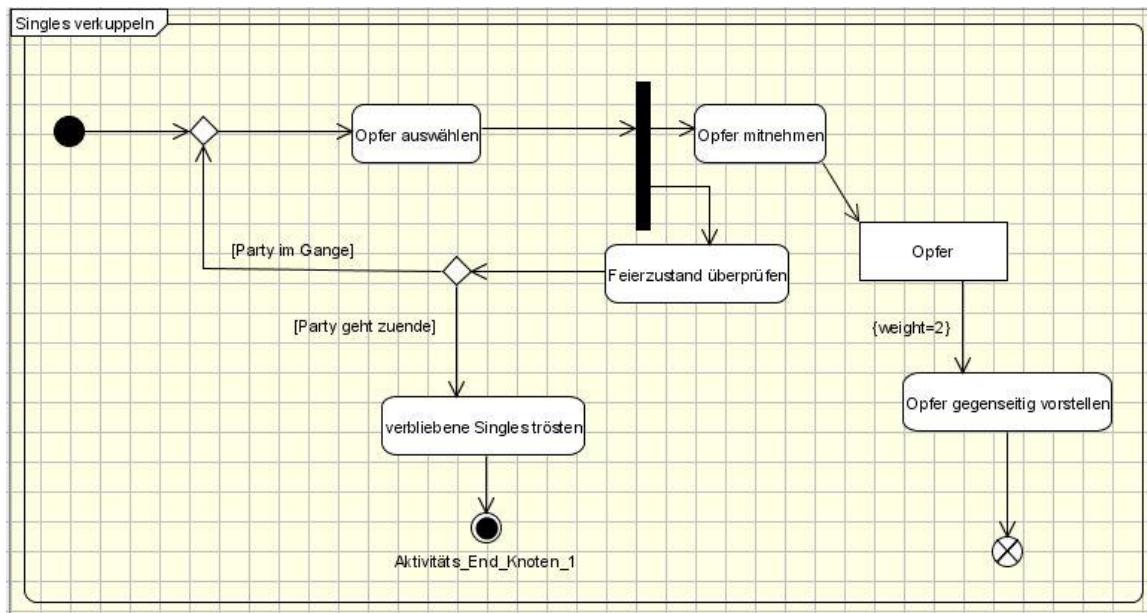
9. Erstellen Sie für folgenden Vertriebsprozess ein Aktivitätsdiagramm. Bewerten Sie dann den Prozess im Hinblick auf mögliche Schwachstellen und Sie dann den Prozess im Hinblick auf mögliche Schwachstellen und Abhängigkeiten im Prozessablauf. Treffen Sie dafür ggf. geeignete Annahmen.
- Der Vertrieb klärt mit dem Kunden Art und Umfang der Produkte bzw. Dienstleistungen welche der Kunde in Anspruch nehmen möchte
 - Weiterhin werden die Lieferbedingungen vereinbart.
 - Die Vereinbarungen münden in ein Angebot, welches der Vertrieb dem Kunden unterbreitet
 - Sollte der Kunde das Angebot akzeptieren, wird auf Grundlage des Angebotes ein Auftrag erstellt.
 - Nachdem die Produkte geliefert bzw. der Service erbracht wurde, erstellt der Vertrieb die Rechnung und verbucht diese im ERP-System.
 - Sollte der Kunde das Angebot nicht akzeptieren, muss der Vertrieb versuchen, diese Unstimmigkeit zu klären und erneut Art und Umfang der Produkte/Dienstleistungen anpassen.
 - Wenn keine Übereinkunft mit dem Kunden getroffen werden kann, wird der Vertriebsprozess mit einer entsprechenden Meldung an das Marketing abgebrochen/beendet.

Weitere Aufgaben

- [Aufgaben zu UseCase, Klassendiagramm, Sequenzdiagramm und Activitiydiagramm](#)
[Aufgaben zu UseCase, Klassendiagramm, Sequenzdiagramm und Activitiydiagramm](#)
- [Aufgaben zu Klassendiagramm und Activitiydiagramm](#)
[Lösung zu Activitiydiagramm](#)
[Lösung zu Klassendiagramm](#)
- [Aufgaben zu Usecase und Activitiydiagramm](#)
[Lösung zu Usecase und Activitiydiagramm](#)
- [Aufgaben zu Usecase und Activitiydiagramm](#)
[Lösung zu Usecase und Activitiydiagramm](#)
- [Aufgaben zu Usecase, Activitiydiagramm, Sequenzdiagramm](#)
[Lösung](#)
- [Aufgaben zu Usecase, Activitiydiagramm, Sequenzdiagramm](#)
[Lösung](#)
[Lösung](#)

Lösungen

Lösung zu Aktivitätsdiagramm Singles_Verkuppen



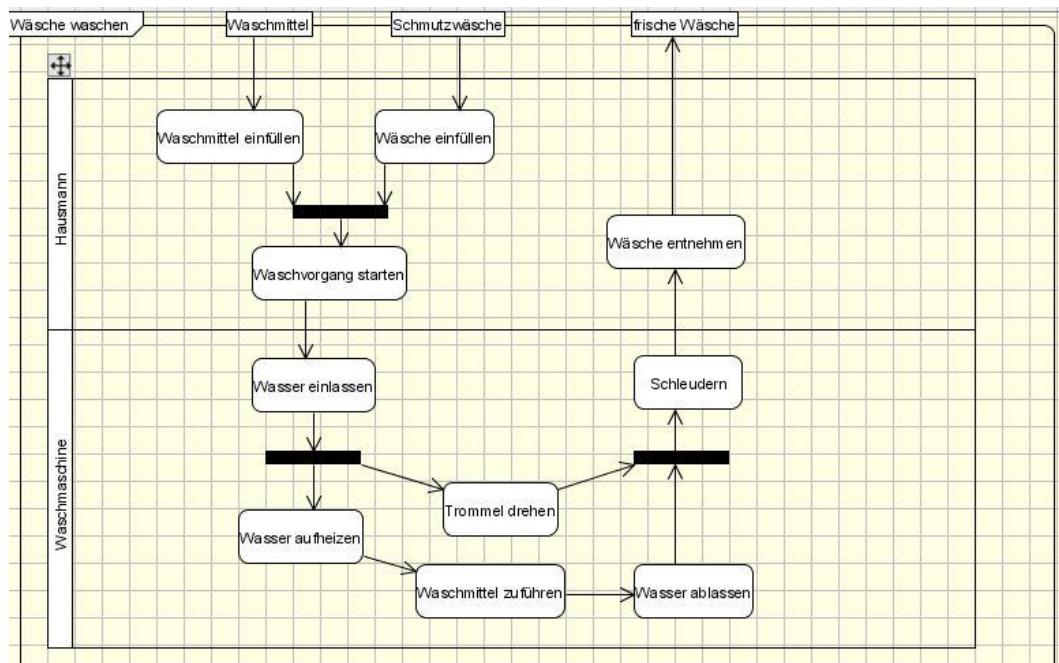
Der Kuppler sucht sich zunächst ein Opfer (Opfer auswählen), woraufhin zwei Dinge gleichzeitig geschehen:

- Zum einen wird das Opfer mitgenommen (die Aktion Opfer mitnehmen liefert das Opfer als Objektknoten), dann bleibt der Fluss jedoch stehen, weil zwei Datentoken notwendig wären {weight = 2}, um den Ablauf fortzusetzen.
- Parallel zum Mitnehmen des Opfers wird geprüft, ob die Feier noch länger andauert. Wenn ja, sucht sich der Kuppler das neue Opfer , woraufhin die Bedingung nach 2 Datentoken erfüllt ist und der Kuppler den Opfern miteinander bekannt macht.

Mit dem Zusammenbringen zweier Personen ist der rechte Strang des Diagramms beendet. Immer wenn zwei Gäste als Opfer gefunden wurden, bringt der Kuppler diese zusammen und beginnt von neuem.

Sollte die Feier zu Ende gehen, werden die verbliebenen Singles im Raum getröstet und die Aktivität ist komplett beendet.

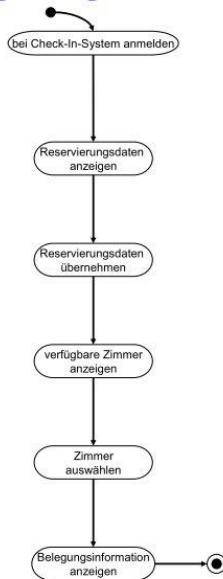
Lösung zu UseCase Wäsche



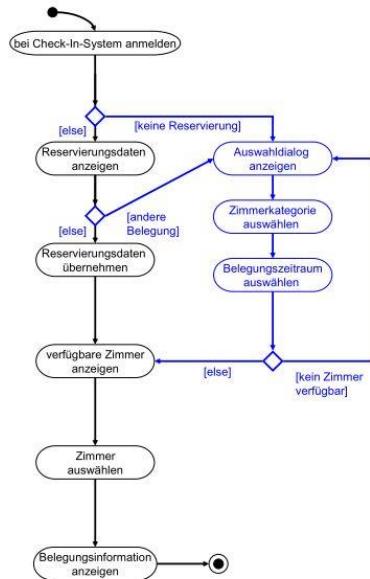
Lösung zu checkin

Lösung zu Aufgabe 2

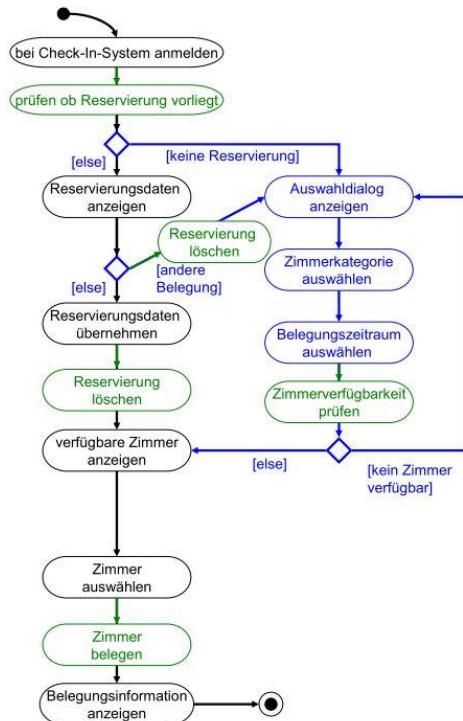
a)

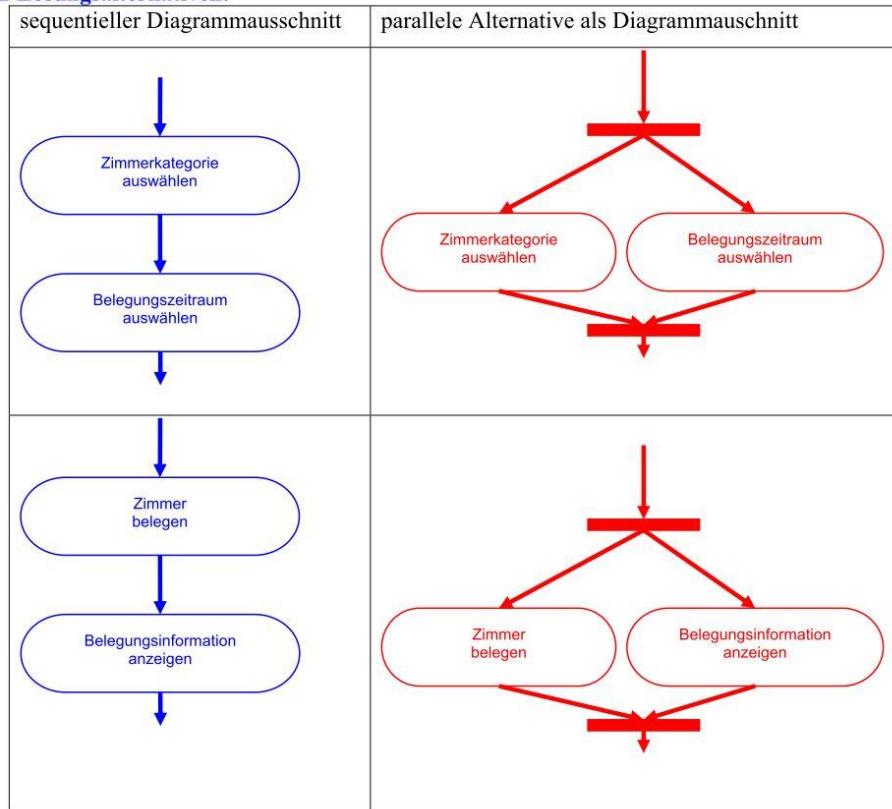


b)

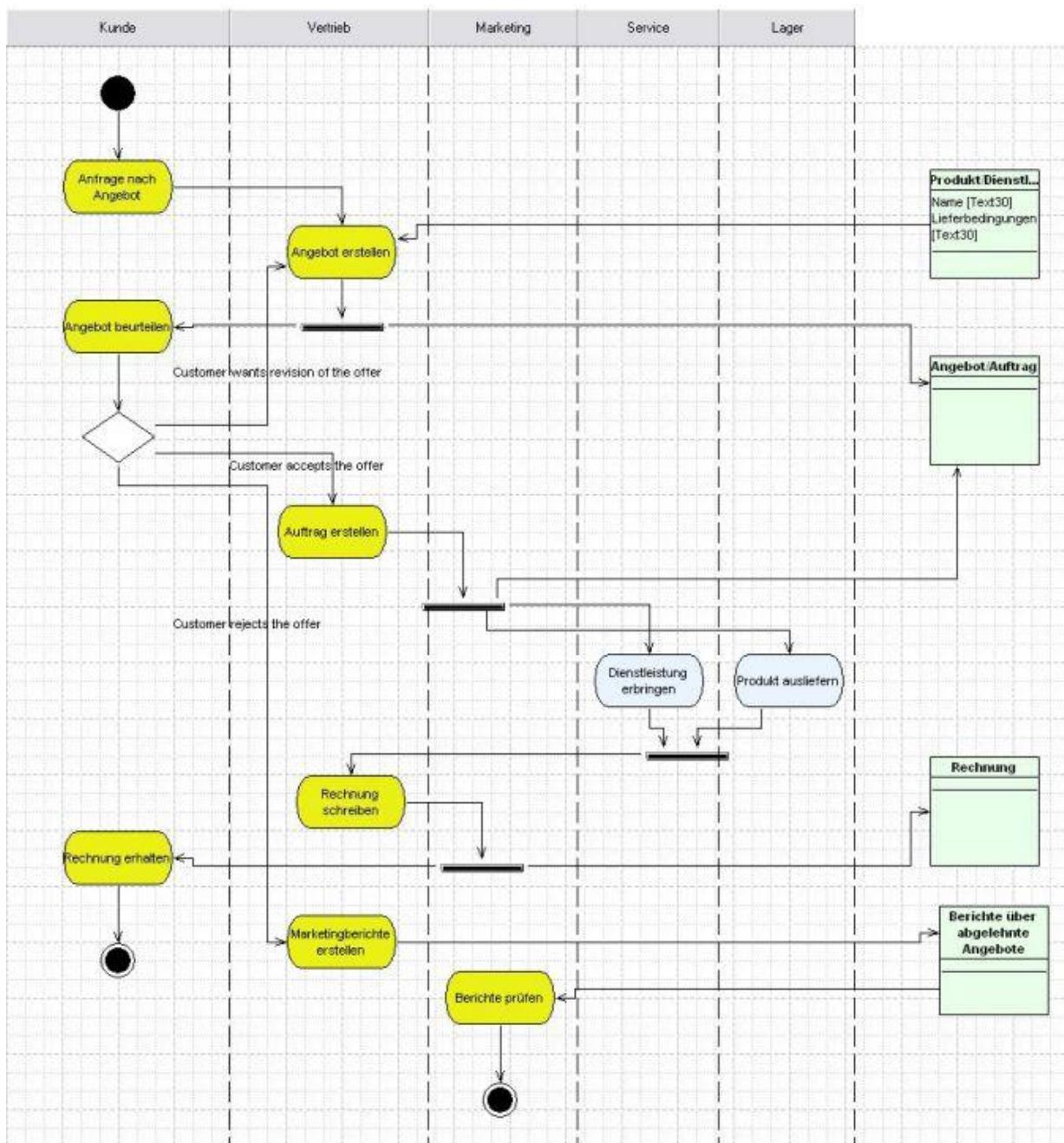


c)

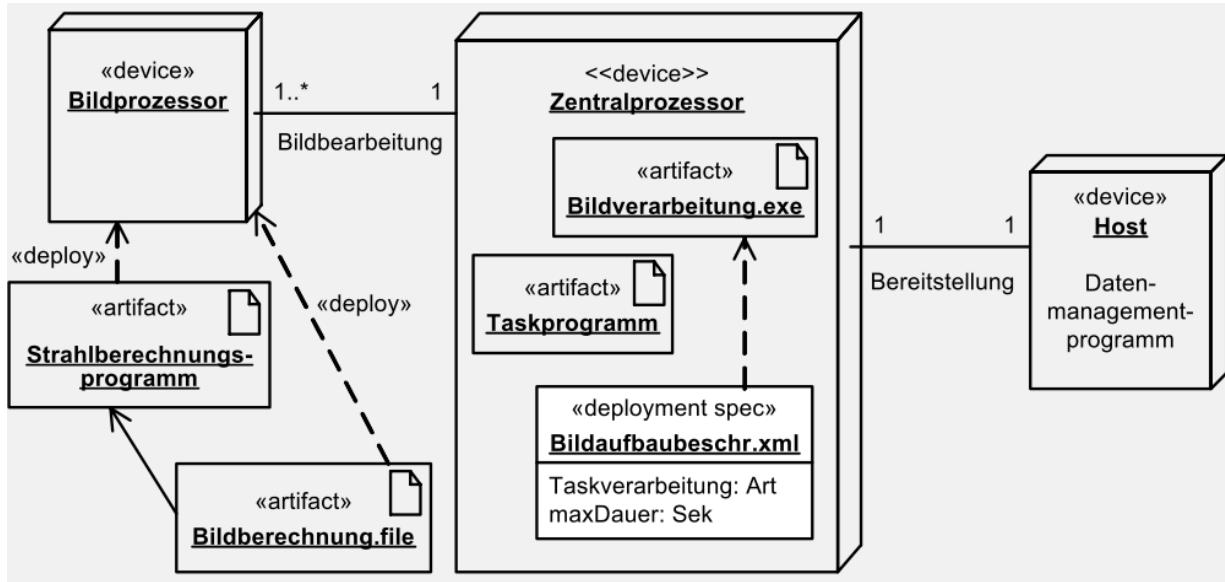


2 Lösungsalternativen:

Lösung zu Vertrieb



Verteilungsdiagramm



Das Verteilungsdiagramm dient

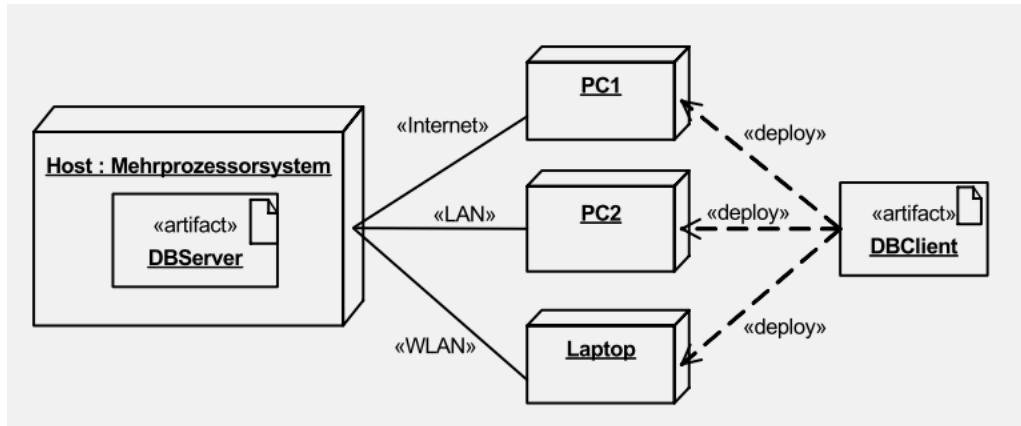
- dem Aufzeigen der Zuordnung von Softwarekomponenten auf Hardwareknoten
- der Darstellung von Kommunikation und Abhängigkeit zwischen den Knoten

Sie gibt Antwort auf die Frage: Wie werden Systemkomponenten zur Laufzeit wohin verteilt?

Sie werden für folgende Situationen eingesetzt:

- Visualisierung von Systemtopologie
- Darstellung von Client-Server-Systemen
- Dokumentation von Hardwarevorgaben in visualisierter Form (besser als textuelle Darstellung)

Modellelemente



- Knoten
- Kommunikationspfad
- Verteilungsbeziehung
- Einsatzspezifikation

Knoten



- Repräsentiert Ressource, die genutzt wird für Installation, Konfiguration, Bereitstellung, Ausführung von Artefakten
- Lassen sich schachteln

Knoten <device>



Repräsentiert ein Gerät und damit ein Stück Hardware

Ausführungsumgebung (Execution Environment)



Softwareumgebung, unter der die Softwarekomponenten ausgeführt werden (z.B. Tomcat)

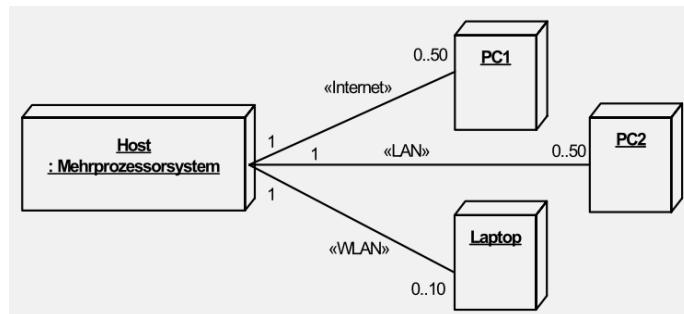
Kommunikationspfad

Ungerichteter Kommunikationspfad



Verbindet Knoten um Nachrichten auszutauschen
(physische Verbindung)

Gerichteter Kommunikationspfad



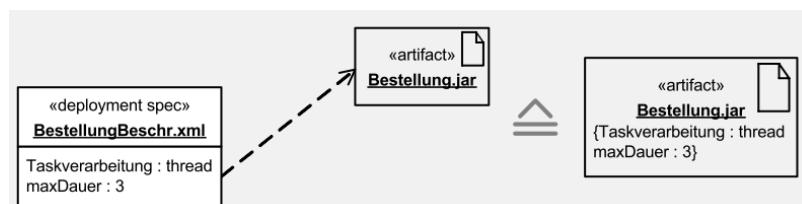
Verteilungsbeziehung



Darstellung durch gestrichelten Pfeil und <deploy>

Beziehung zwischen Artefakt und Knoten, auf dem es verteilt ist

Einsatzspezifikation



Ist durch Abhängigkeitsbeziehung mit Artefakt verbunden

- Beinhaltet Parameter zur Artefaktverteilung auf Knoten
- Durch Parameterfestlegung ist Artefaktausführung steuerbar
- Beziehung zwischen Artefakt und Knoten, auf dem es verteilt ist
- alternative Notation: {Parameter der Einsatzspezifikation} in Artefakt

Aufgaben

Siehe GH1 2009 und 2006 FI SYS

Lösung

Noch keine Lösung

Klassendiagramm

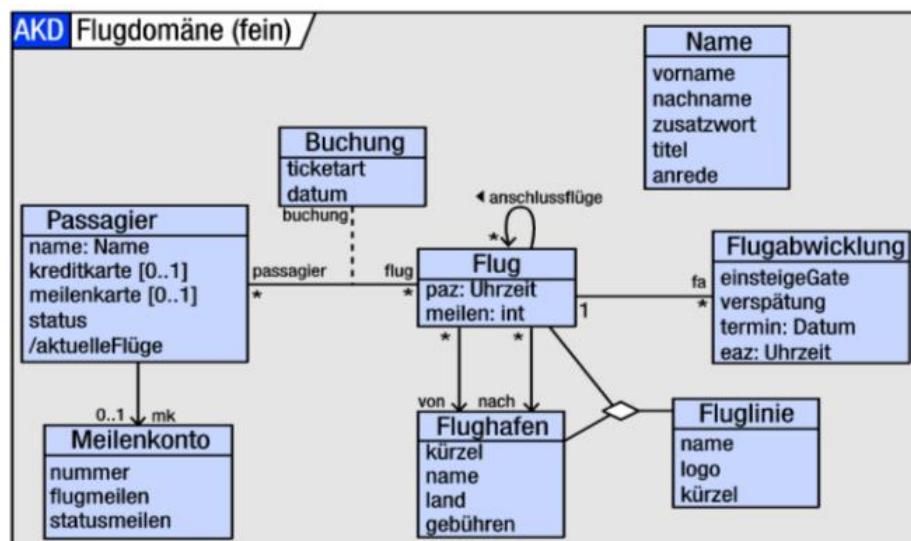
Das Klassendiagramm wurde aus dem Object Model der Object Modeling Technique (OMT) entwickelt und dient:

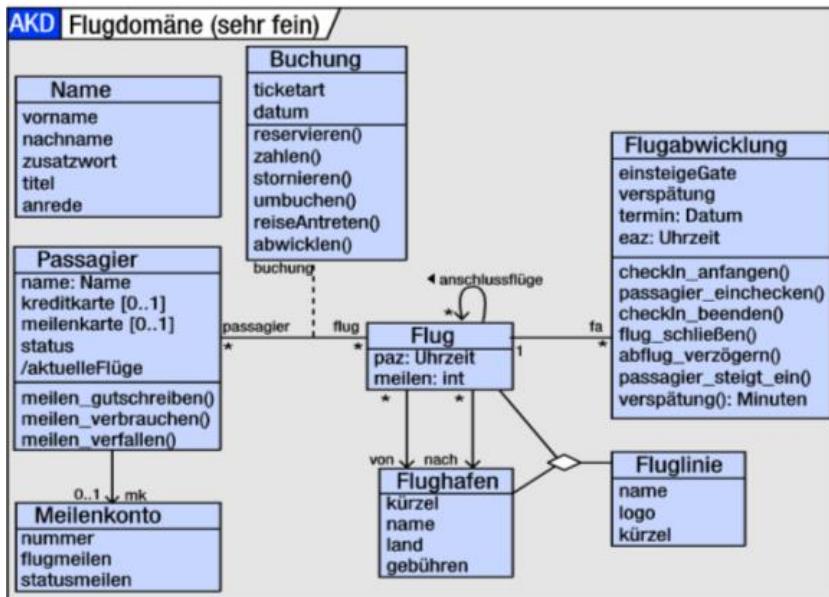
- Der Darstellung der Struktur eines Systems
- Dem Aufzeigen von statischen Eigenschaften und Beziehungen
- Der Sammlung grundlegender Modellierungskonstrukte
- Ist Antwort auf die Frage? Wie sind Daten und Verhalten meines Systems strukturiert??

Je nach Absicht kann das Klassendiagramm in 3 verschiedenen Stufen eingesetzt werden.

Analyse-Klassendiagramm

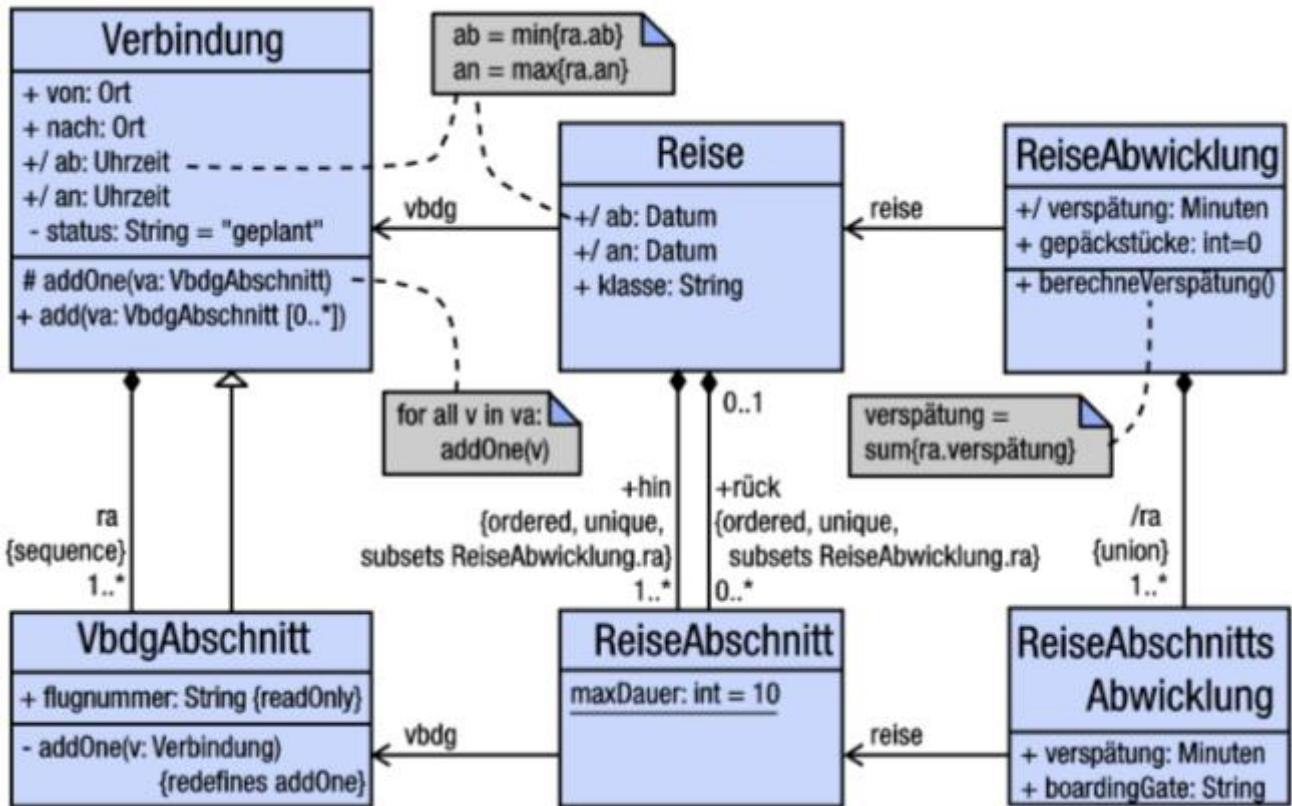
Auf der Analyseebene sind Klassen Konzepte. Es geht darum zu verstehen und zu dokumentieren, wie der Problembereich aufgebaut ist.





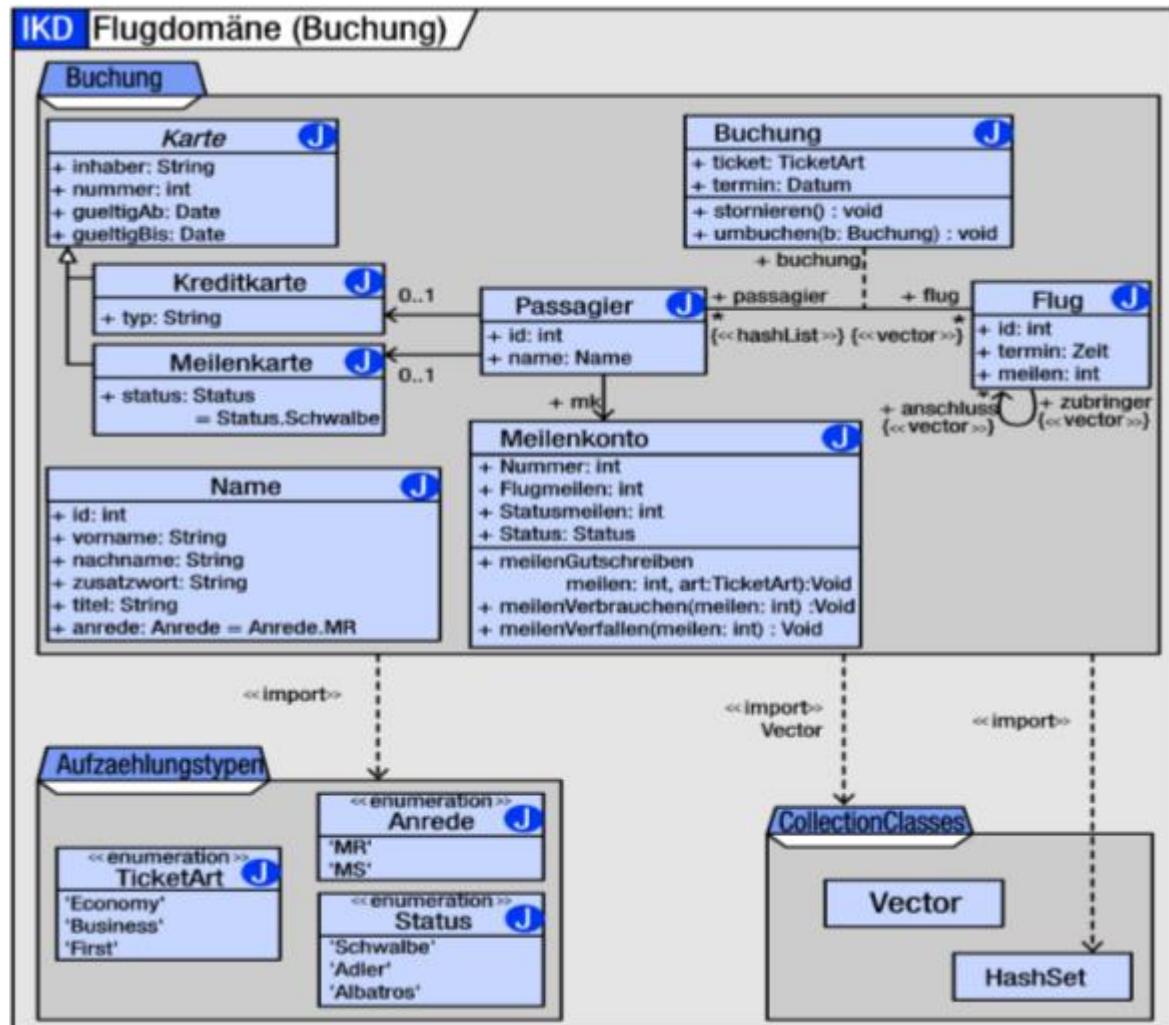
Entwurfs-Klassendiagramm

Auf der Entwurfsebene geht es um die technische Umsetzung der fachlichen Strukturen, also darum wie der Lösungsbereich strukturiert sein soll.



Implementierungs-Klassendiagramm

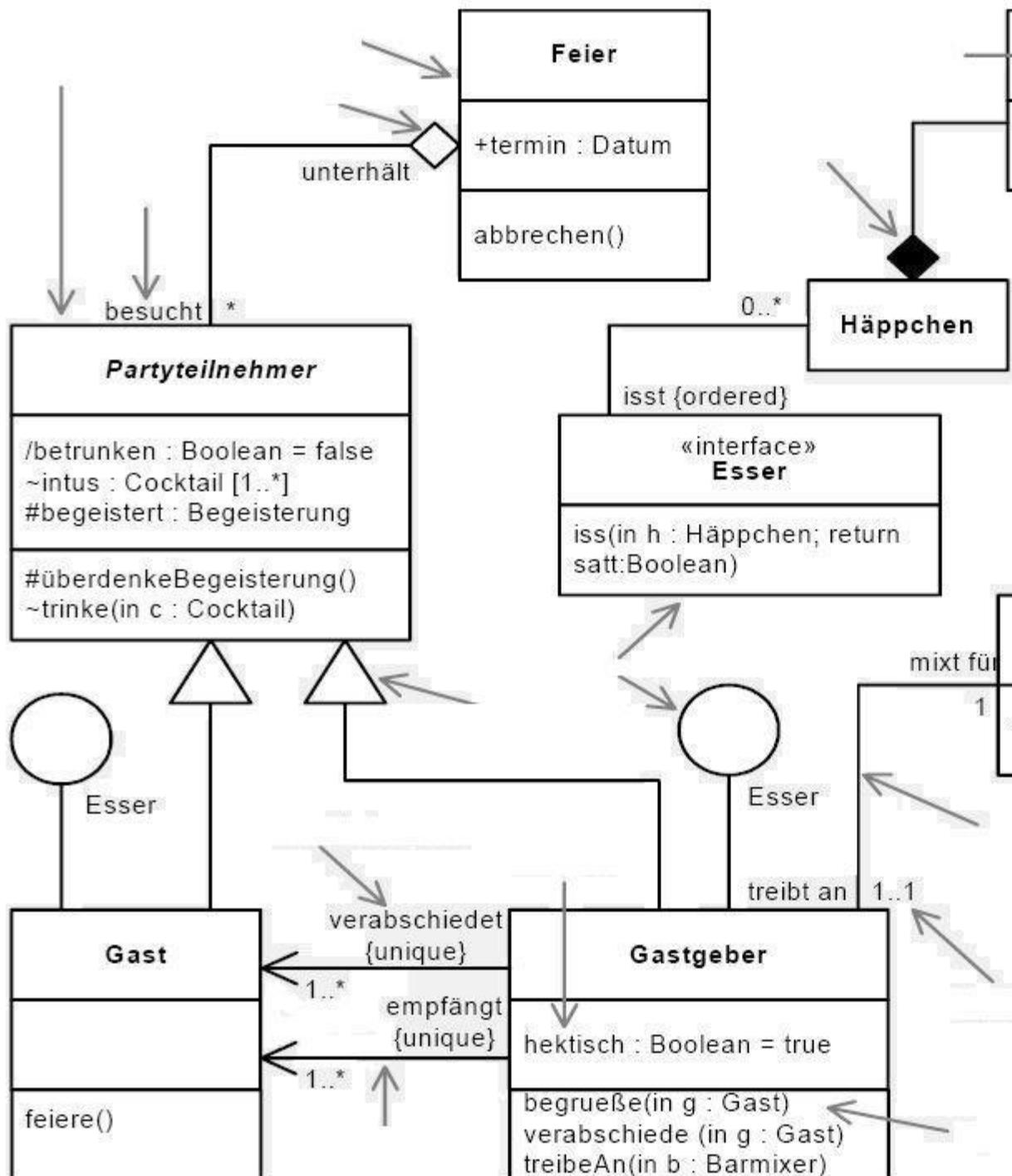
Auf der Implementierungsebene sind Klassen unmittelbare Äquivalente zu Konstrukten einer Programmiersprache.



Elemente des Klassendiagramms

Das Klassendiagramm besteht aus den Elementen:

- Klassen
- Schnittstellen
- Attributen
- Operationen
- Assoziationen mit Sonderformen Aggregation und Komposition
- Generalisierungsbeziehungen
- Abhängigkeitsbeziehungen



Notationselemente im Einzelnen

Im Klassendiagramm finden graphische Primitive Anwendung:

- **Klassen**
- **Attribute**
- **Operationen**

- **Schnittstellen**
- **Parametrisierte Klassen**
- **Generalisierung**
- **Assoziationen**
- **Assoziationsklassen**
- **Kommentar**
- **Stereotype**
- **Eigenschaftswerte**

Klasse

Arbeitsblatt zu Klasse

Unter **Klasse** versteht die UML eine Sammlung von Exemplaren, die über **gemeinsame Eigenschaften und Operationen** verfügen. Jede Klasse ist somit ein abstrakter Sammelbegriff für eine Menge gleichartiger, in der physisch fassbaren Realität oder der gedanklichen Anschauung existierender Dinge. Damit macht der Klassenbegriff nichts anderes als wir im täglichen Leben, wenn wir von *unseren Freunden* reden anstelle sie einzeln aufzuzählen.

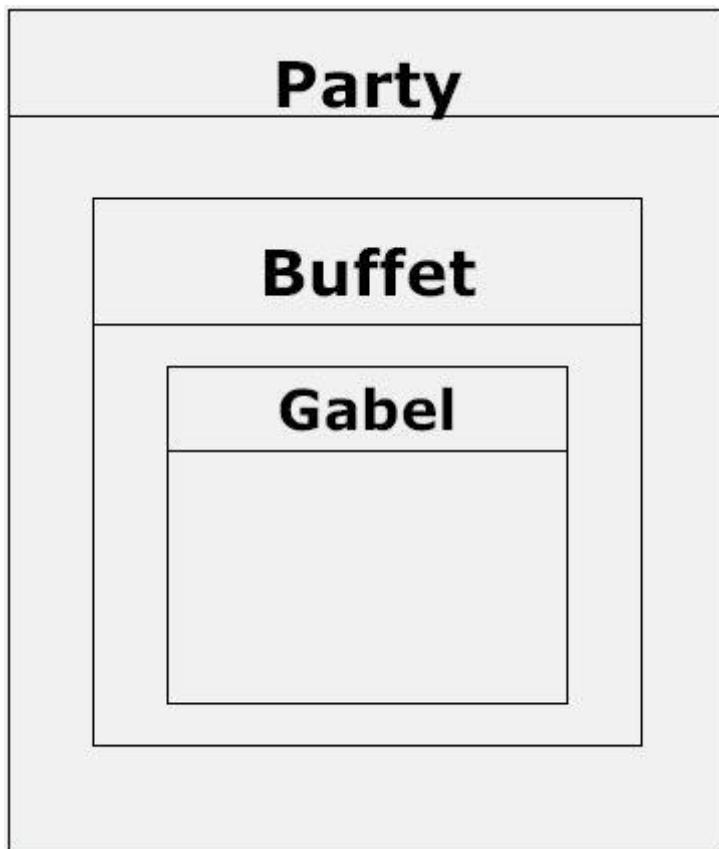
Die UML interpretiert eine Klasse als **Typ**, dessen Ausprägungen **Objekte** heißen.

Klassenname

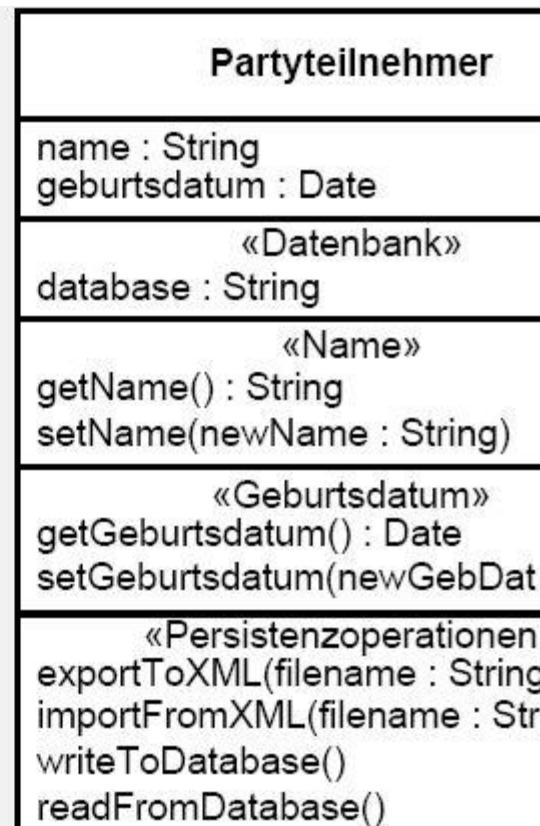
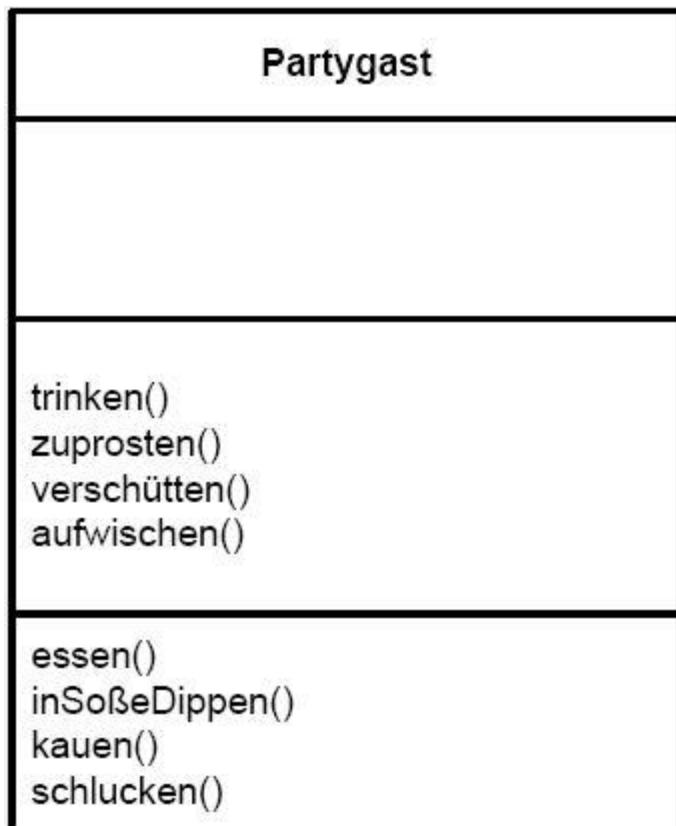
Klassenname in Rechteck fett und mittig dargestellt

Klassenname
attribut
operation ()

Sichtbare Darstellung von Attributen und/oder Operationen



Die Schachtelung von Klassen ist beliebig möglich Eingebettete Klasse: hat keine Außenbeziehung hier: Gabel nur Beziehung zu Buffet



Anzahl und Inhalt der Rechteckelemente NICHT beschränkt Operationen und Attribute können gruppiert und in eigenen Darstellungselementen untergebracht werden

Attribut

Attribute definieren die Eigenschaften von **Objekten**, auch wenn sie auf Klassenebene definiert werden.



Das Attribut linksbündig in Rechteckselement



Das Attribut wird durch einen klassenweit eindeutigen Namen spezifiziert!

- public Jede andere Systemkomponente hat uneingeschränkten Zugriff
- private Nur Ausprägungen der das Attribut oder die Operation beherbergenden Klasse dürfen zugreifen
- protected Das Attribut, die Operation, ist nur in der definierenden und denjenigen Klassen sicht- und zugreifbar, die als Spezialisierungen von ihr definiert sind
- package Das Attribut, die Operation, ist für alle Klassen zugreifbar, die sich im selben Paket finden, wie die definierende Klasse

Sichtbarkeit: Die UML sieht symbolische Schreibweisen für die Sichtbarkeit von Attributen und Operationen vor:

```
+ public
- private
# protected
~ package
```

Attributname, frei wählbar aus Zeichensatz

Name Der Datentyp eines Attributes. Die UML kennt keine Einschränkungen hinsichtlich der zur Verfügung stehenden Typen

Typ legt Unter- und Obergrenze der Anzahl der Ausprägungen fest, die unter einem Attributnamen abgelegt werden können

Multiplizität Erlaubt die Angabe eines Wertes, der für das Attribut automatisch gesetzt wird.

Vorgabewert Mit dieser in geschweiften Klammern eingeschlossenen Angaben können besondere Charakteristiken der Attribute benannt werden. Es sind zunächst folgende Angaben möglich:

- readOnly nicht veränderbare Attribute
- ordered Inhalte eines Attributes liegen in **geordneter Reihenfolge** und **ohne Duplikate** vor. Ist dementsprechend nur sinnvoll, wenn die Multiplizitätsobergrenze > 1 ist
- bag Inhalte eines Attributes können **ungeordnet** und müssen **nicht duplikatfrei** sein.
- seq oder sequence Die Inhalte eines Attributes müssen **geordnet** aber **nicht zwingend duplikatfrei** sein.
- composite definiert, dass das Attribut die Verantwortung für die Zerstörung der in ihm enthaltenen Werte übernimmt. Im Kern ist dieses Attribut damit eine Referenz auf ein anderes Objekt (Komposition)

Eigenschaftswert

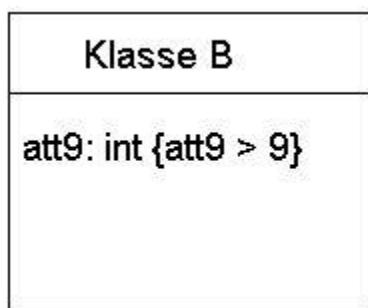
Attributdeklaration	
public zähler : int	Umlaut erlaubt
/alter	Datentyp nicht klar
private adressen: String [1...*]	Menge von Strings
protected bruder: Person	Datentyp kann Schnittstelle sein

Attributdeklaration	
String	Attributname
private, public name: String	Mehr als ein Attribut
public / int	Attributnamen
private dateiname: String=„temp“ lock=exclusive	Eigenschaften
public int zähler	Name und Typ

Beispiel für mögliche Attribute

AttMix

```
att1 : int
att6 : KlasseB [0..1] {composite}
att7 : String [0..*] {ordered}
/ att8
public:
    att2 : int
    pi : double = 3.1415
private:
    att3 : boolean
protected:
    att4 : short
package:
    att5 : String = "Test" {readOnly}
```



- att1

Attribut vom Datentyp INT mit unspezifizierter Sichtbarkeit

- att2

Wie Att1, jedoch mit Sichtbarkeit **public** (+)

- pi

Statisches Attribut (Unterstrich) mit Vorgabewert und Sichtbarkeit

- att3

private Sichtbarkeit und Datentyp boolean

- att4

protected Sichtbarkeit und Datentyp short

- att5

package-Sichtbarkeit und readOnly-Eigenschaft.
Der Attributwert kann durch kein Programm verändert werden

- att6/att7

Attribut als Reference auf ein Objekt der Klasse KlasseB; Multiplizität ist 0 oder 1. Attribut sorgt für die Zerstörung des Objektes (composite)

att7 hält Strings, geordnet aber nicht unbedingt duplikatfrei

- att8 abgeleitetes Attribut
- att9 selbst definierte Einschränkung, die nur Werte > 10 zulässt

Umsetzung in Programmiersprachen

Operation

Operationen einer Klasse werden mindestens durch Namen und wahlfreie weitere Angaben dargestellt:

Abstrakte Attributspezifikation:



```
[Sichtbarkeit] Name (Parameterliste) : Rückgabetyp
[ {Eigenschaftswert} ]
```

Parameterliste:

```
[Übergaberichtung] Name : Typ [ ` [ `Multiplizität` ] ` ]
[ = Vorgabewert ] [ ` { `Eigenschaftswert` } ` ]
halloWelt()
```

Eine Operation ohne Übergabe- und Rückgabeparameter

```
+add(summand1, summand2) : int
```

Öffentlich zugreifbare Operation mit zwei Übergabe- und einem Rückgabeparameter; der Typ der Übergabeparameter ist nicht festgelegt

```
-div(divident : int, divisor : int) : double
```

Private Operation mit zwei Übergabeparametern vom Typ *int* und einem Rückgabeparameter

```
sub(in minuend : double, in subtrahend : double, out resultat : double)
```

Operation mit drei Übergabeparametern, wobei zwei lesend als Eingabe verarbeitet werden.
Der dritte (mit *out* deklarierte) hält das Ergebnis

```
inc(inout wert)
```

Operation, deren Parameter im Rahmen der Ausführung verändert und an den Aufrufer zurückgegeben wird.

```
mult(summand : Matrix[2..*]{ordered}) : Matrix
```

Operation, die einen Übergabeparameter besitzt, der eine Menge von mindestens zwei Werten des Typs *Matrix* beherbergt. Die Angabe der Eigenschaft *ordered* bestimmt, dass die Elemente der Menge in der übergebenen Reihenfolge verarbeitet werden müssen. Die Rückgabe ist ebenfalls vom Typ *Matrix*

```
setLstKl(Lohnsteuerklasse : int = 1)
```

Die Operation definiert, dass, wenn der Parameter Lohnsteuerklasse keinen Wert besitzt, automatisch die Vorgabe 1 angenommen wird. **Tabelle 3.4. Beispiele für Notation von Operationen**

```
add(int i, int j)
```

Der Parametername muss vor dem Typ stehen und von ihm durch einen Doppelpunkt getrennt sein.

```
Int : sub(i:int, j : int)
```

Der Rückgabetyp muss nach dem Operationsnamen platziert werden

```
clear(foo) : void
```

Das Schlüsselwort *void* gibt es in der UML nicht

```
summe(i : int, ...)
```

Variable Parameterlisten existieren nicht.

Umsetzung in verschiedenen Programmiersprachen

OpMix

+ op1()

- op2(in param1 : int=5) : int {readOnly}

op3(inout param2 : KlasseC)

~ op4(out param3 : String[1..*] {sequence}



Aufgabe Implementieren Sie das Klassendiagramm in einer Ihnen bekannten Programmiersprache. Versuchen Sie dabei die Vorgaben des Klassendiagramms in der jeweiligen Programmiersprache so genau als möglich umzusetzen.

Lösung: Siehe jeckle, Seite 59 ff.

C++

```
class OpMix {
    KlasseB op4 (vector<string> param3) {
        // Implementierung
        return wert;
    }
}
```

public:

```
static void op1() {  
    //Implementierung  
}  
  
private:  
  
    int op2(const int param1=5) const {  
        //Implementierung  
        return wert;  
    }  
  
protected:  
  
    void op3(KlasseC* param2) {  
        //Implementierung  
    }  
  
};  
  
:::  
  
java  
  
public class OpMix {  
    public static void op1() {  
        //Implementierung  
    }
```

```
}

private int op2(final int param1) {

    //Implementierung

    return wert;

}

protected void op3(KlasseC param3) {

    //Implementierung

}

KlasseB op4(String param3[]) {

    //Implementierung

    return wert;

}

}

:::

C#
```

```
class OpMix {

    public static void op1() {

        //Implementierung

    }

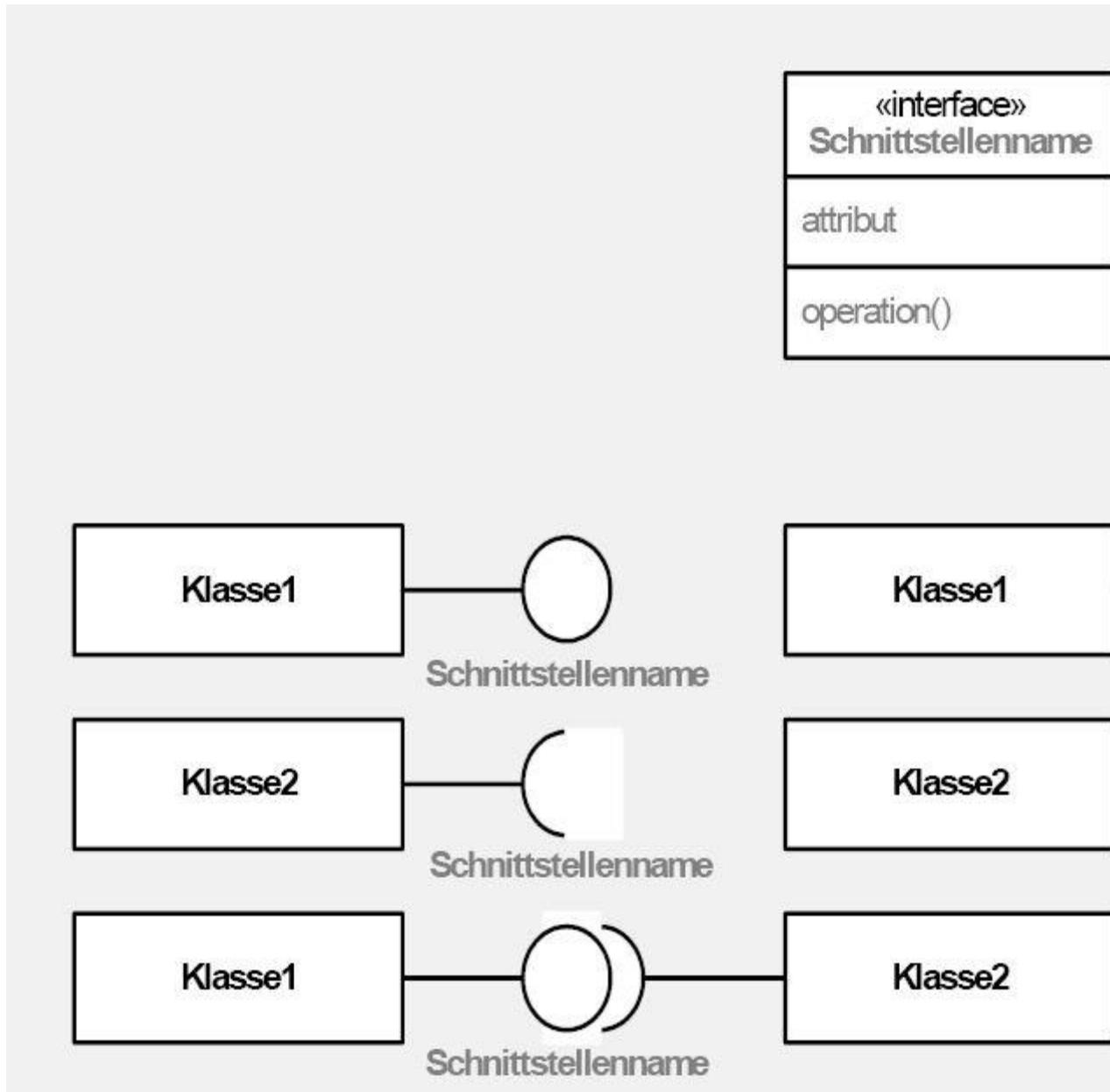
}
```

```
private int op21(int param1) {  
    //Implementierung  
  
    return wert;  
}  
  
private int op22(params object[] list) {  
    //Implementierung  
  
    return wert;  
}  
  
protected void op3(ref ClassC param3) {  
    //Implementierung  
}  
  
internal int op4(out StringCollection param3) {  
    //Implementierung  
  
    return wert;  
}  
}
```

Schnittstelle

Darstellung durch das Klassensymbol mit Zusatz «interface». Schnittstelle kann weitere Schnittstellen

- beinhalten
- spezialisieren
- generalisieren



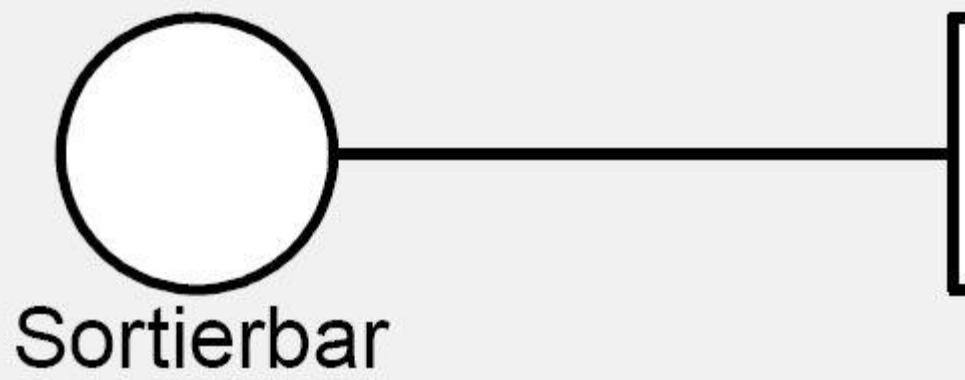
Anmerkung NICHT möglich: Implementationsbeziehung zwischen zwei Schnittstellen

Die Schnittstelle wird nicht instanziert, sondern durch Klassen realisiert:

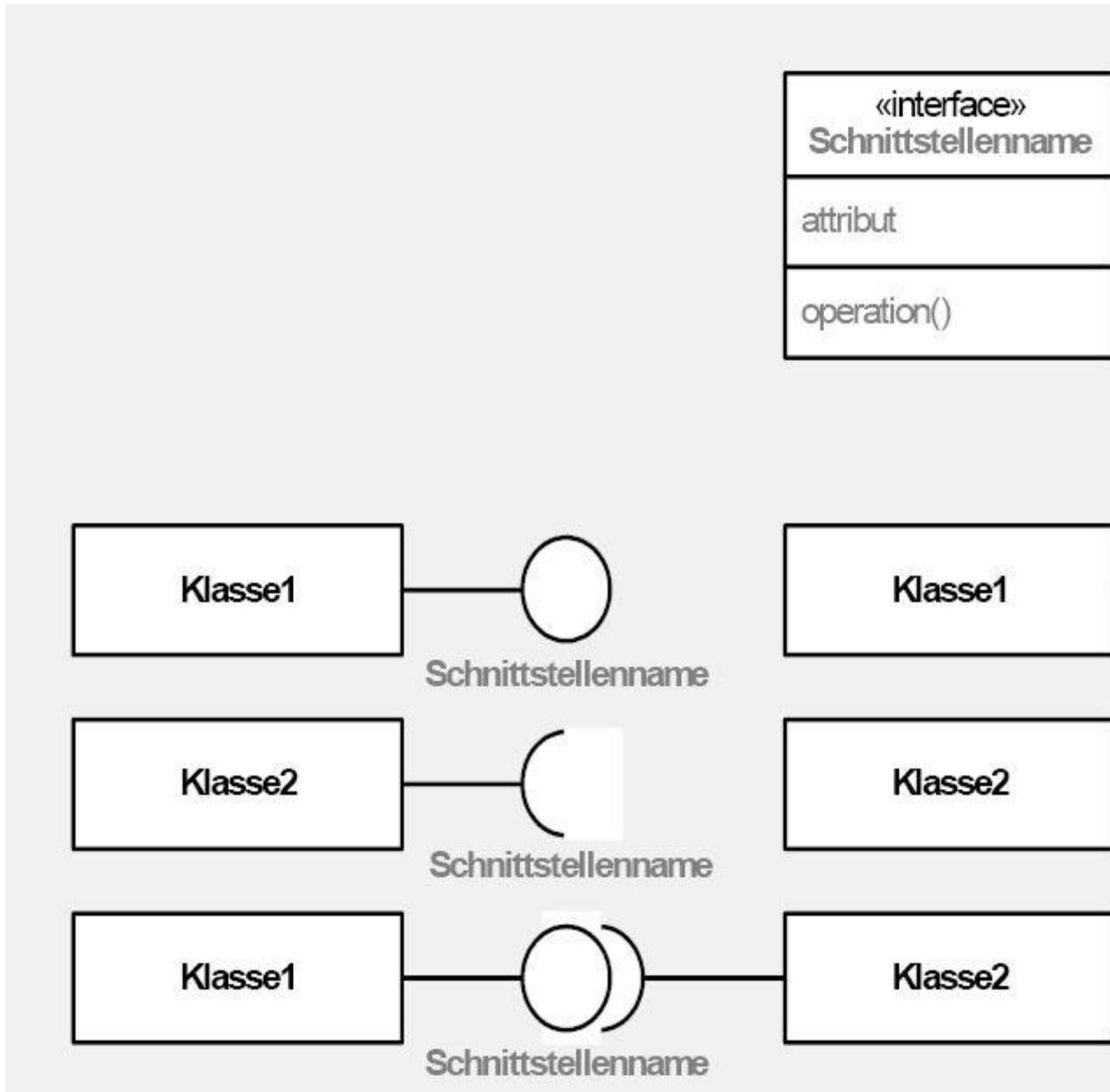
- Klasse ist konform zur Schnittstelle
- Eine Schnittstelle kann von mehreren Classifiern umgesetzt werden
- Eine Klasse kann mehrere Schnittstellen umsetzen



«interface» **Sortierbar**



Beispiel zu Interface



- Klasse 1 (Ball): **Implementiert** eine Schnittstelle
- Klasse 2 (Socket): **Benötigt** diese Schnittstelle zur Erfüllung seiner Aufgabe
- Klasse 1 unten (Lollipop): Verdeutlicht durch die Kombination von Socket und Ball das Inaneindergreifen von Schnittstellenbereitsteller und -nutzer

In Java:

```
interface SortierteListe {
```

```
int sortierReihenfolge=0; //Konstante

public void einfügen(Eintrag e);

public void löschen(Eintrag e);

}

class Datenbank implements SortierteListe {

    public void einfügen(Eintrag e) {
        //...
    }

    public void löschen(Eintrag e) {
        //...
    }
}

#####
#
```

In C#

```
interface SortierteListe {

    void einfuegen(Eintrag e);

    void loeschen(Eintrag e);

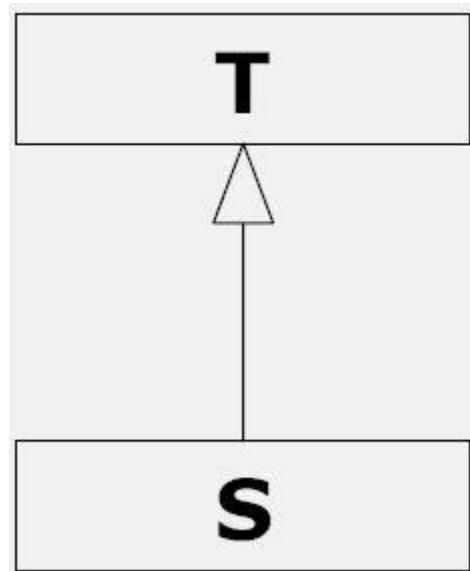
}
```

```
class Datenbank : SortierteListe {  
  
    public void einfuegen(Eintrag e) {}  
  
    public void loeschen(Eintrag e) {}  
  
}
```

Parametrisierte Klasse

Generalisierung

Darstellung durch gerichtete Kante mit leerer Spitze am Pfeilende



Anmerkung Generalisierung beschränkt sich nicht nur auf Klassen, sondern kann auch auf andere Modellelemente angewendet werden, z.B. Assoziationen

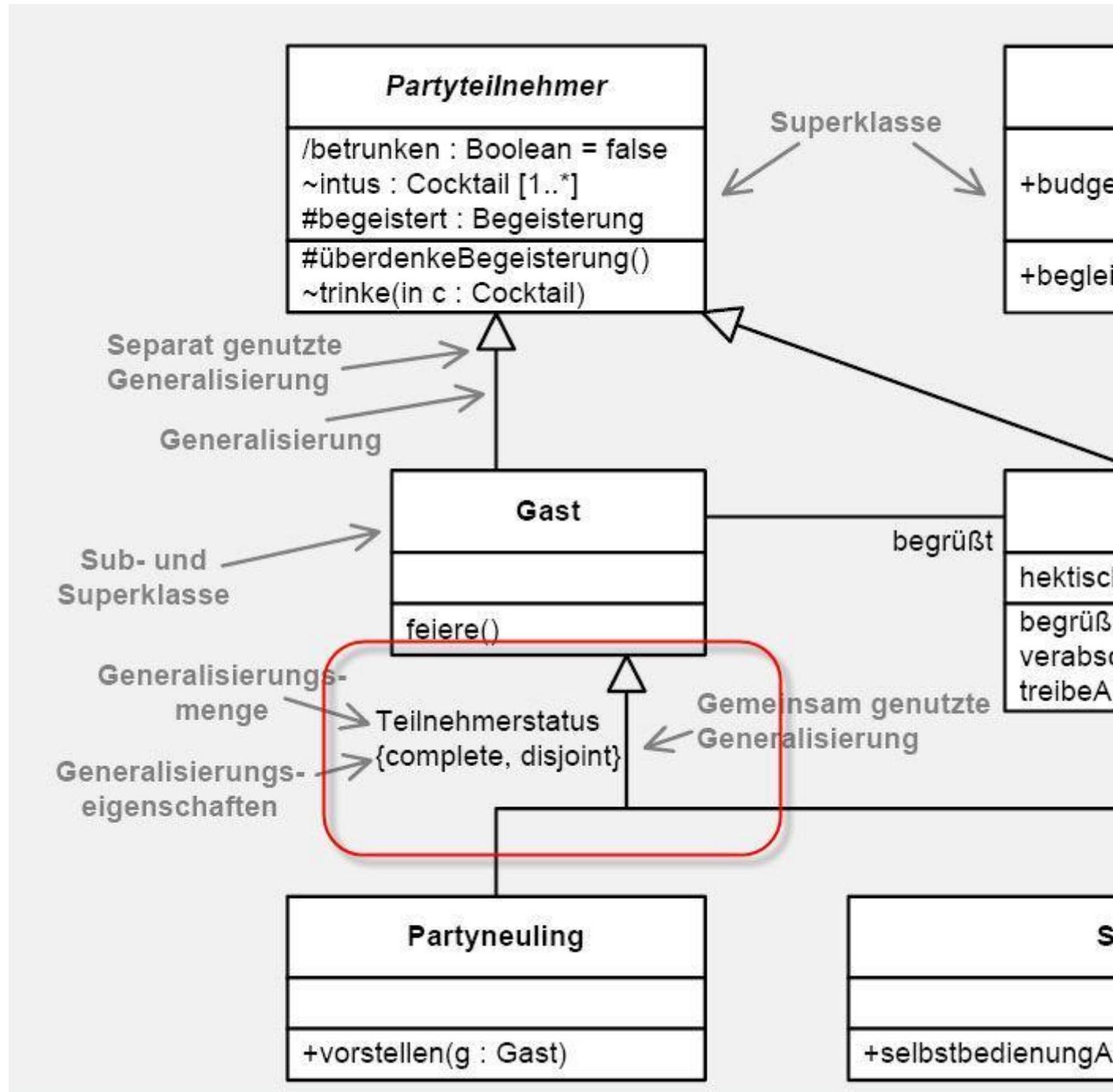
Durch die Benennung einer **Generalisierungsmenge** kann der Grund für eine Generalisierung ausgedrückt werden. Daneben können auch noch Generalisierungseigenschaften angegeben werden. Folgende Schlüsselworte stehen zur Verfügung:

- complete
- incomplete
- disjoint

- overlapping

Die Vereinigung aller Partitionen versammelt alle im Modellkontext sinnvoll denkbaren Spezialisierungen eines Typs

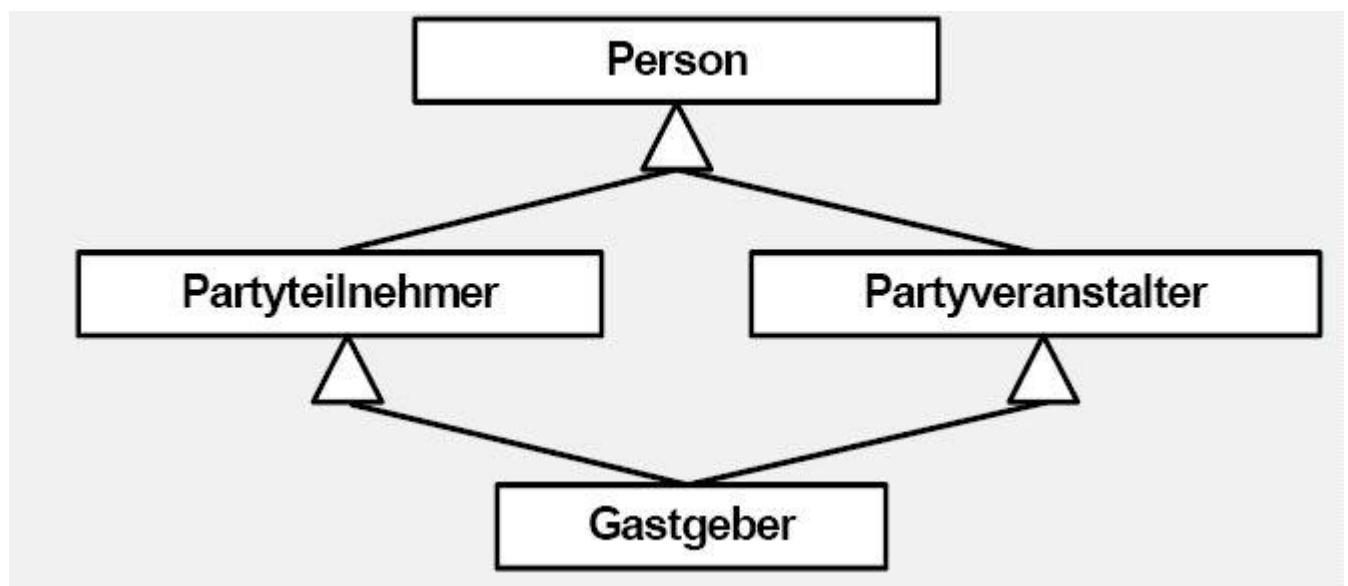
Gegensatz zu complete Keine Instanz eines Subtyps ist gleichzeitig Instanz eines anderen Subtyps Gegensatz zu disjoint **Tabelle 3.6. Generalisierungseigenschaften**



[Video zur Vererbung](#)

[Video zum MethodOverloading](#)

Mehrfachvererbung



- Syntaktisch möglich
- In verschiedenen Programmersprachen problematisch
- In einigen Programmiersprachen nicht umsetzbar

[Video zur MehrfachVererbung](#)

Substitution

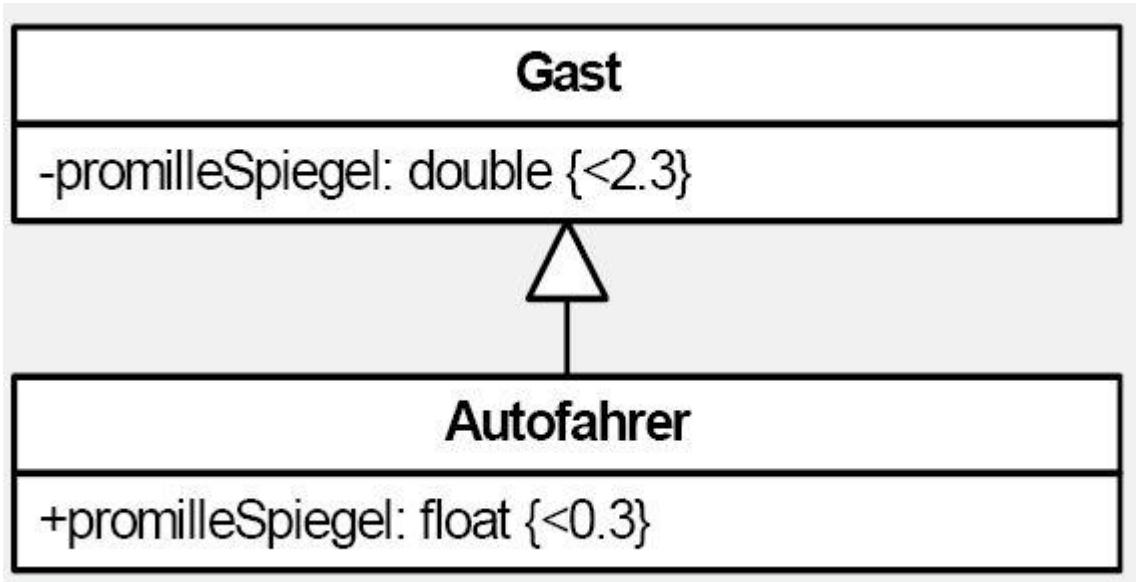
1988: Liskovsches Substitutionsprinzip (Liskov Substitution Principle, LSP)

substitution property: If for each object o1 of type S there is an object of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.

LSP verdeutlicht des Zusammenhang zwischen Vererbung und Substitution:

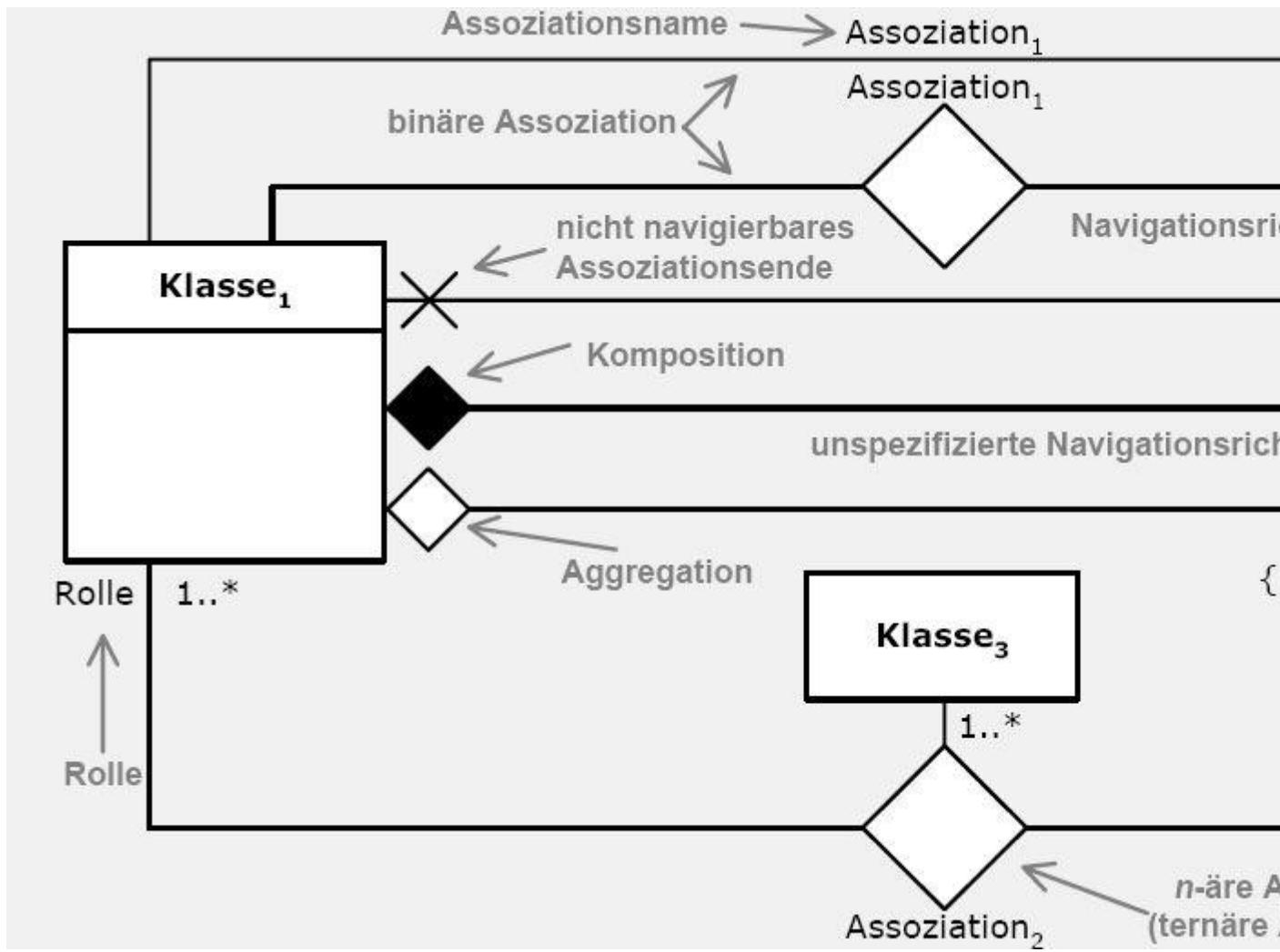
- Forderung an alle Unterklassen (S) einer Klasse(T) dieselbe Semantik für eine Schnittstelle anzubieten
- Sicherstellung, dass sich Programm bei Verwendung einer Schnittstelle gleich verhält, egal ob Instanz einer Klasse oder der Unterklasse verwendet wird
- Besonders wichtig bei Redefinition von Methoden

Geht das ???



Assoziation

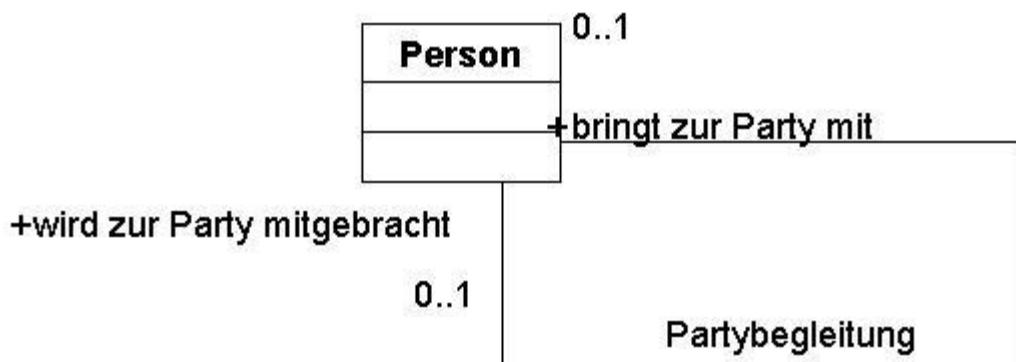
Assoziation = Menge semantisch gleichartiger Beziehungen zwischen Classifiern



Die Ausprägung einer Assoziation ist eine Beziehungsinstanz (Link) und hat verschiedene Ausprägungen:

- Binäre / zweiwertige Assoziation: hat zwei Enden
- n-äre / höherwertige Assoziation: mehr als zwei Enden
- Zirkulär-reflexive Assoziation: kehrt zum Ausgangsclassifier zurück
- Abgeleitete Assoziation:

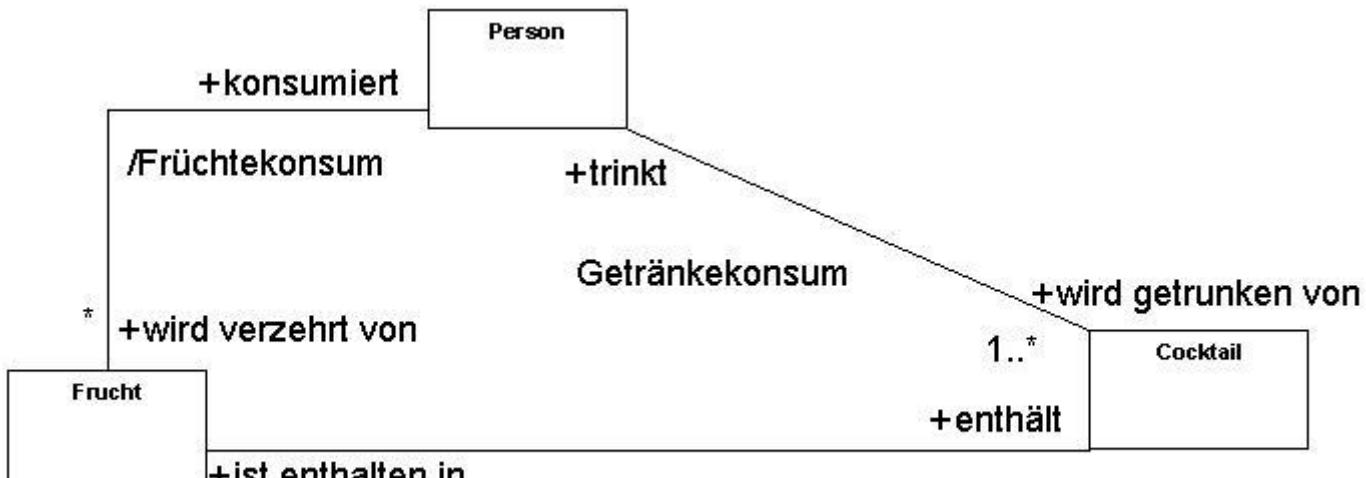
Binäre Assoziation



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

Abgeleitete Assoziation

gekennzeichnet durch / und kann aus anderen Zusammenhängen abgeleitet werden



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

In obigem Beispiel fügt die Assoziation Fruchteksumsum dem Modell keine neue Aussage hinzu, sondern kann jeweils aus den beiden anderen Assoziationen abgeleitet werden.

Rollen, Multiplizität, Einschränkung und Eigenschaften

Die Enden einer Assoziation können zusätzlich mit beschreibenden Angaben versehen werden. Sie erläutern die Bedeutung einer an der Assoziation teilnehmenden Klasse und die Form der Teilnahme näher.



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

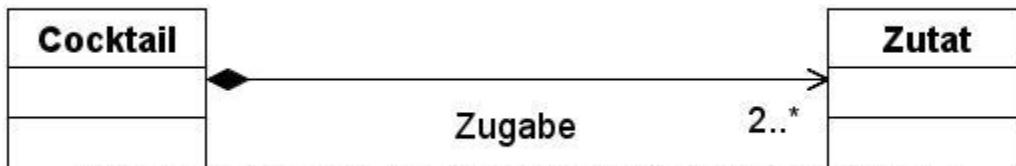
Die die beiden Assoziationen verbindende gestrichelte Linie bedeutet eine Einschränkung. Für jede Ausprägung der beiden durch die Assoziation verbundenen Klassen darf nur eine der beiden Beziehungen bestehen. Ein mit der Getränkeerzeugung beschäftigter Barmixer darf sich nicht gleichzeitig als Partygast am Getränkekonsum beteiligen und umgekehrt.

Navigierbarkeit

Wurde schon an anderer Stelle angesprochen.

[Ein kleines Video zu Beziehungen](#)

Aggregation und Komposition



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

Aggregation

Drückt eine **Teil-Ganzes-Beziehung** aus. Weitere Einschränkungen werden durch die Aggregation nicht getroffen, insbesondere zu wie vielen verschiedenen Ganzten ein Teil gleichzeitig

gehören kann oder in welcher Reihenfolge die so verbundenen Objekte erzeugt und zerstört werden müssen.

Komposition

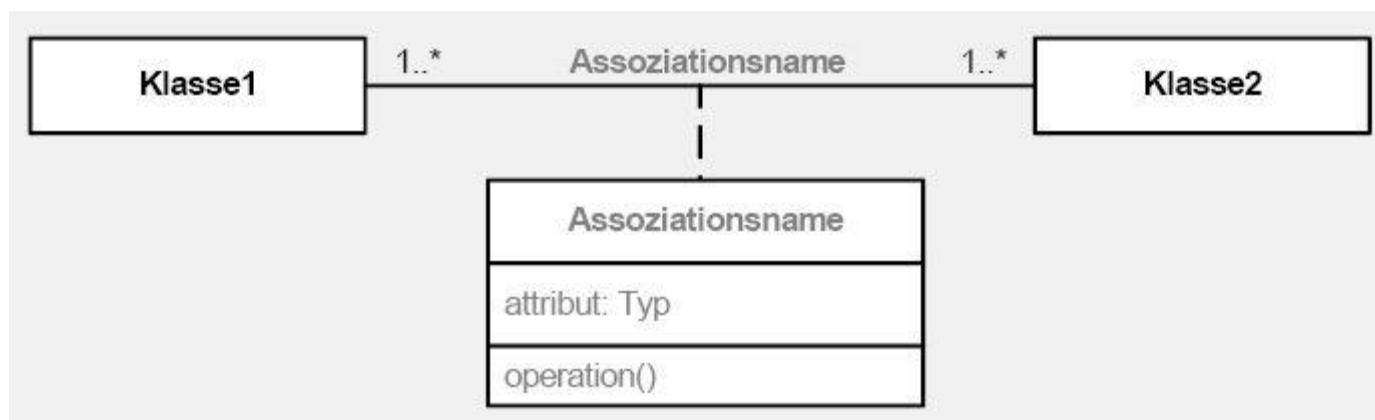
Sie drückt **physische Inklusion der Teile im Ganzen** aus. Deshalb gilt hier die verschärzte Einschränkung, dass ein Teil zu einem Zeitpunkt nur genau einem Ganzen zugeordnet sein darf.

Die Lebensdauer des Teils hängt vom Ganzen ab.

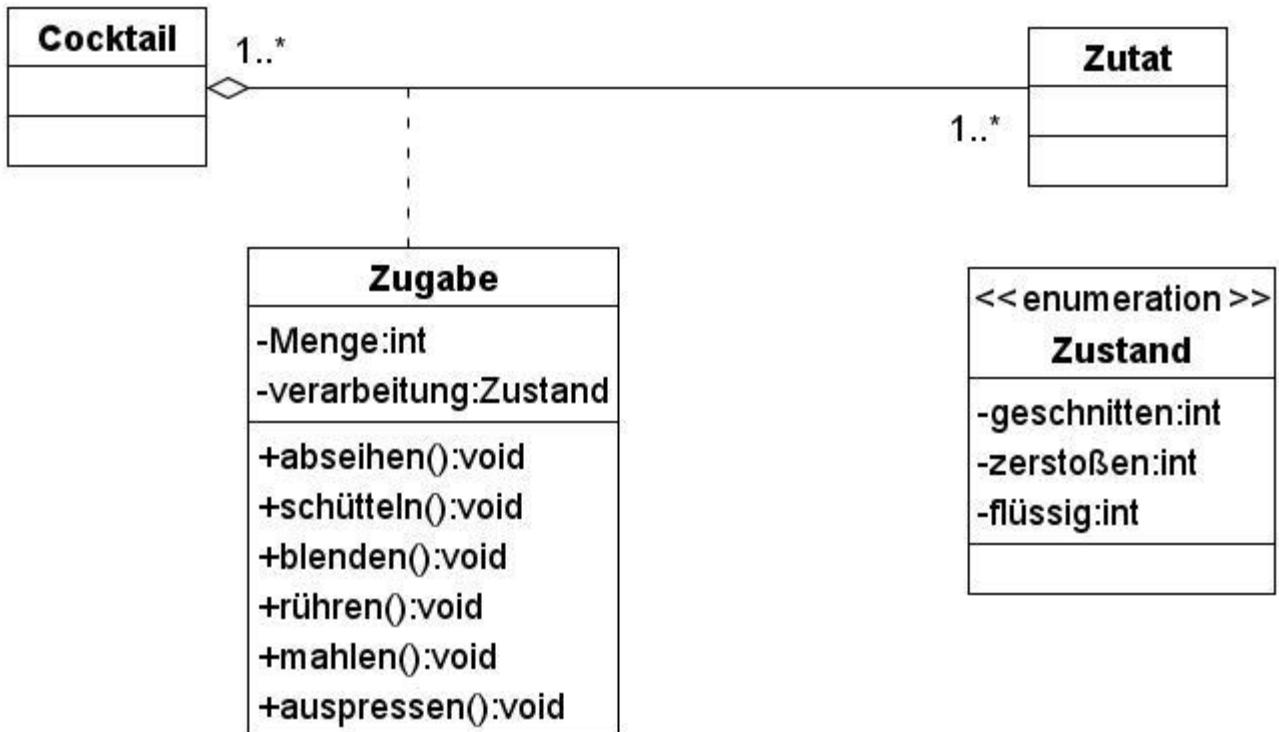
Im oben abgebildeten Beispiel wird durch die Komposition ausgesagt, dass ein Cocktail aus mindestens 2 Zutaten bestehen muss und erst dadurch zum Cocktail wird. Umgekehrt ist jede Zutatsausprägung in genau einem Cocktail physisch vorhanden. Verlässt eine Zutat den Cocktail, so ist dieser kein Cocktail mehr.

Ein weiteres Beispiel für den Unterschied zwischen Komposition und Aggregation finden Sie [hier](#)

Assoziationsklasse



Die Assoziationsklasse wird als gewöhnliche Klasse modelliert, die über eine unterbrochene Linie mit einer Assoziation verbunden ist. Sie dient dazu, Eigenschaften näher zu beschreiben, die keiner der zur Assoziation beitragenden Klassen sinnvoll zugeordnet werden können.



Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

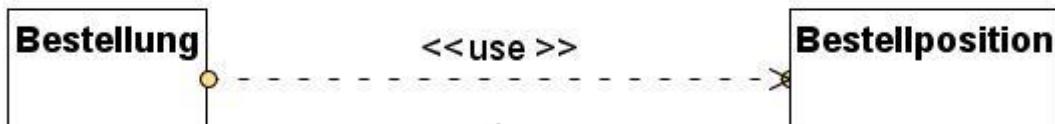
Die obige Assoziationsklasse dient dazu, die Zutaten eines Cocktails hinsichtlich Menge und Verarbeitung näher zu beschreiben. Da die Ausprägung der Assoziation (d.h. der Link) immer genau eine Ausprägung von Cocktail mit genau einem Exemplar von Zutat verbindet, bildet diese Beziehungsinstanz den idealen Ort zur Definition der konkreten Mischeigenschaften, da sie weder eindeutig der Zutat noch dem Cocktail selbst zugeordnet werden können.

Abhängigkeit

Eine Abhängigkeitsbeziehung wird in der UML durch einen gestrichelten Pfeil dargestellt, der auf das Modellelement zeigt, von dem das andere abhängig ist.

Sie dient zur Dokumentation des Designs. Das abhängige Modellelement ist ohne das Element, von dem es abhängt, unvollständig. Die Beziehung wird auch als **Supplier-Client (Anbieter-Verbraucher)** - Beziehung interpretiert, in der das abhängige Element als Verbraucher, das andere Element als Anbieter fungiert.

Das obige Beispiel illustriert, dass die Klasse Bestellung zur Erfüllung ihrer Aufgaben die Klasse Bestellposition **verwendet**.



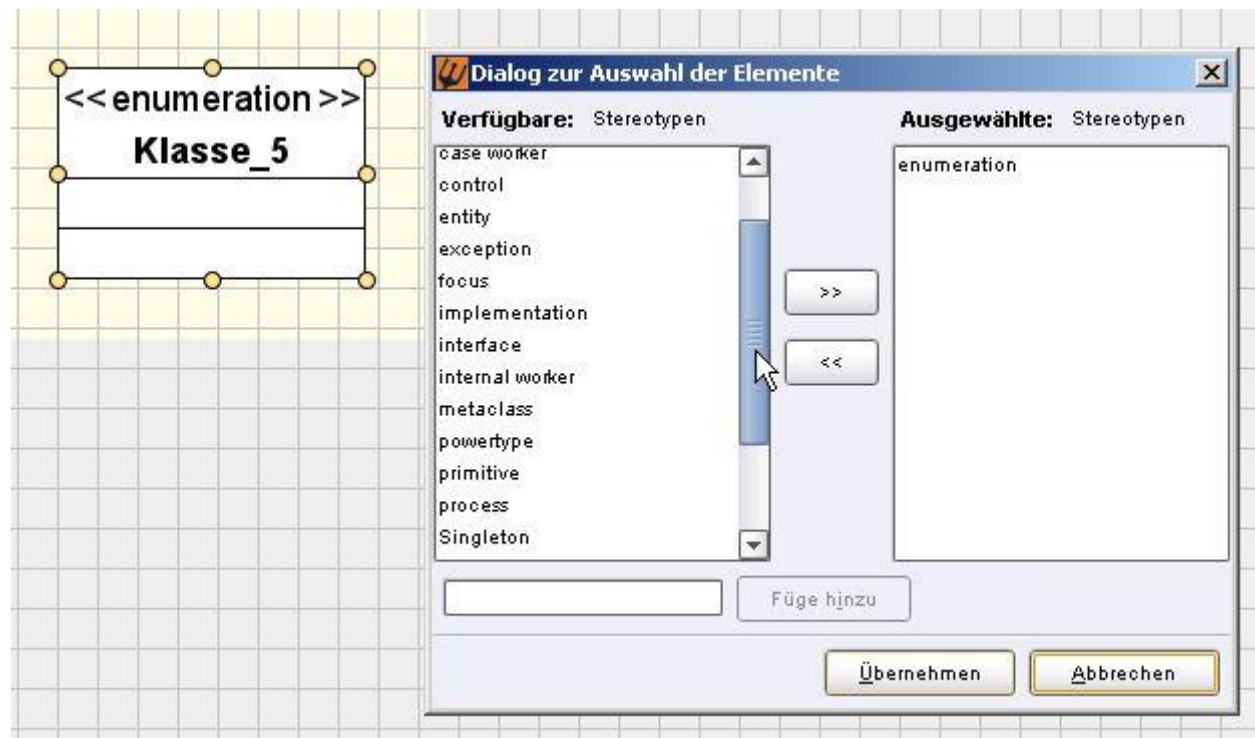
Erstellt mit Poseidon for UML Community Edition. Nicht zur kommerziellen Nutzung.

Kommentar

Nichts besonderes zu sagen

Stereotype

Stereotypes bilden die Möglichkeit, die UML mit benutzerspezifischer Terminologie anzureichern.



Folgende Tabelle listet einige wichtige Stereotypen und deren Bedeutung auf.

Stereotyp Beschreibung auxiliary call create derive destroy instantiate interface enumeration refine responsibility substitute utility

Eigenschaftswerte

dxcx

Fragen zum Klassendiagramm

- Sie haben den Auftrag erhalten, ein Computersystem für eine Universitätsbibliothek zu entwickeln. Derzeit verwendet die Bibliothek ein Programm aus den Sechzigern, geschrieben in einer überholten Sprache. Damit erledigt sie eine Reihe von einfachen Verwaltungsaufgaben und führt einen Schlagwortkatalog auf Kartenbasis.
- Erstellen Sie aus der folgenden Beschreibung ein Klassendiagramm Gegeben seien folgende Begriffe aus der zu modellierenden Realität: PKW, LKW, LKWmitAnhänger, Taxi, Autobus, Containerschiff, Fähre, Floss, Yacht.
- Verfeinern Sie das folgende Klassendiagramm der Bibliothek um eine nähere Beschreibung der Assoziationen (Multiplizität, Typ der Assoziation). Passen Sie auch die Rückgabewerte und Parameter der Operationen an. Jeder Benutzer kann höchstens fünf Dinge entleihen.
- Gegeben seien folgende Anforderungen an ein Textverarbeitungssystem: Das Textverarbeitungssystem erlaubt es Peter Müller und anderen Benutzern Dokumente anzulegen und zu editieren. Ein Dokument kann Text und Bitmap-Grafik enthalten. Text besteht aus Abschnitten, jeder Abschnitt aus Zeichen. Ein Dokument enthält ausserdem verschiedene administrative Informationen wie seinen Titel, seinen Autor, den Dateinamen, in dem es abgelegt ist, sowie das Datum der letzten Änderung. Welche Objekte/Klassen lassen sich im obigen Problemfall identifizieren? Welche Beziehungen (Assoziation, Aggregation, Komposition) bestehen zwischen den Klassen? Zeichnen Sie das entsprechende Klassendiagramm in der UML-Notation. Welche Attribute hat die Klasse Dokument?

Ergänzen Sie das gezeichnete Klassendiagramm entsprechend. Achten Sie dabei auf Assoziationsnamen, Assoziationsenden (Rollen), Navigierbarkeit und Multiplizitäten.

- Beim Bäckermeister Krause kann man verschiedene Backwaren kaufen. Herr Krause steht hinter einer Theke, auf der eine Kasse steht. Herr Krause ist ein moderner Bäcker und hat bereits eine Scannerkasse. Das Warenangebot wird den Kunden in der Theke präsentiert. Die Kunden kaufen Brötchen, Brot, Kuchen und kleine Teilchen. Herr Krause ist so nett und verpackt die gewünschten Waren passend in verschiedenen großen Tüten. Jeder Kunde bekommt neben der Tüte einen Kassenbon, auf dem die Waren und ihr Preis verzeichnet sind. Der Bäckerladen von Herrn Krause öffnet morgens schon um 6 Uhr, um auch hungrige Schüler, die von einer Shuttleparty kommen, mit warmen Brötchen versorgen zu können.
- Die 9 Planeten unseres Sonnensystems (außer der Erde) werden in innere und äußere Planeten unterteilt. Sie unterscheiden sich hauptsächlich durch ihre mittlere Entfernung zur Sonne und ihren Durchmesser. Den Tag, an dem Sonne, innerer Planet und Erde eine Linie bilden, nennt man Konjunktionsdatum, den Tag, an dem Sonne, Erde und äußerer Planet eine Linie bilden, Oppositionsdatum. Neben den bekannten großen Planeten schwirren noch tausende Kleinplaneten zwischen Mars und Jupiter umher. Ihre Bahnen sind teilweise so lang gestreckt, dass sie die Bahnen von bis zu 5 anderen Planeten kreuzen können. Generell haben Planeten einen eindeutigen Namen, bis zu 30 Monde und bestehen aus fester oder gasförmiger Oberfläche, Atmosphäre (jeweils bestehend aus mehreren chemischen Elementen) und evtl. einem oder mehreren Ringen (charakterisiert anhand des Durchmessers)



- Es soll eine Datenbank angelegt werden, die Information über Orchester enthält. Ihr Kollege hat begonnen, ein entsprechendes UML-Klassendiagramm zu entwerfen und ist dann in Urlaub gefahren (siehe nächste Seite) ? Der Entwurf muss fertig werden. Sie erhalten das begonnene Diagramm und die folgende Beschreibung der Anwendungsdomäne. Ergänzen Sie das UML-Diagramm so, dass es die Beschreibung möglichst genau widerspiegelt! Verwenden Sie u.a. Generalisierung, Aggregation, (mehrstellige) Beziehungen. Vergessen Sie nicht Attribute und Kardinalitäten passend zu ergänzen!
- Beurteilen Sie, welche Informationen in der Analyse und welche

im Entwurf dokumentiert werden müssen.

Der Systemanalytiker Mayer führt bei einer Videothek eine Systemanalyse durch, wobei er folgende Informationen aufnimmt:

1. Für jeden Videofilm sind Titel, Laufzeit und Jahr zu speichern.
2. Die Videofilme sind nach Titeln aufsteigend sortiert in der Datenbankxy zu speichern.
3. Jede Ausleihe von Videofilmen wird im System gespeichert.
4. Defekte Videofilme werden aus der Videothek entfernt und in der Datei mit einem »L« markiert.
5. Das System soll jederzeit einen Überblick über die Ausleihhäufigkeit der einzelnen Filme erlauben.
6. Für die Realisierung der Benutzeroberfläche wird die Klassenbibliothek abc verwendet.
7. Da es sich um eine große Videothek handelt, ist eine Client-Server-Anwendung notwendig, wobei alle

zentralen Daten auf dem Server liegen.

Ordnen Sie die einzelnen Informationen zu den Entwicklungsphasen Analyse und Entwurf zu. Benutzen Sie dazu folgende Tabelle.



Information	Relevant für
1. Für jeden Videofilm sind Titel, Laufzeit und Jahr zu speichern.	
2. Die Videofilme sind nach Titeln aufsteigend sortiert in der Datenbank xy zu speichern.	
3. Jede Ausleihe von Videofilmen wird im System gespeichert.	
4. Defekte Videofilme werden aus der Videothek entfernt und in der Datei mit einem »L« markiert.	
5. Das System soll jederzeit einen Überblick über die Ausleihhäufigkeit der einzelnen Filme erlauben.	
6. Für die Realisierung der Benutzeroberfläche wird die Klassenbibliothek abc verwendet.	
7. Da es sich um eine große Videothek handelt, ist eine Client-Server-Anwendung notwendig, wobei alle zentralen Daten auf dem Server liegen.	

- Sie haben den Auftrag erhalten, ein Computersystem für eine Universitätsbibliothek zu entwickeln. Derzeit verwendet die Bibliothek ein Programm

aus den Sechzigern, geschrieben in einer überholten Sprache. Damit erledigt sie eine Reihe von einfachen Verwaltungsaufgaben und führt einen Schlagwortkatalog auf Kartenbasis.

Von Ihnen wird ein interaktives System erwartet, mit dem beide Tätigkeiten online erledigt werden können. Nach sorgfältigen Untersuchungen kristallisieren sich die folgenden Anforderungen heraus, wie sie durch ein ideales System abgedeckt wären:

- Bücher und Zeitschriften:

In der Bibliothek gibt es Bücher und Zeitschriften. Von Büchern kann es auch mehrere Exemplare geben. Einige Bücher sind nur für Kurzzeitentleihe gedacht. Alle anderen Bücher können von jedem Bibliotheksmitglied für drei Wochen ausgeliehen werden. Zeitschriften dürfen nur von Mitarbeitern ausgeliehen werden. Mitglieder der Bibliothek können normalerweise bis zu sechs Einheiten entleihen, während Mitarbeiter der Bibliothek bis zu 12 Einheiten zur gleichen Zeit entleihen können.

- Entleihe

Wichtig ist, dass das System verfolgen kann, wann Bücher und Zeitschriften ausgeliehen und zurückgegeben werden, weil dies auch das aktuelle System schon kann. Das neue System sollte darüber hinaus Mahnungen auswerfen, wenn ein Entleihzeitraum überzogen ist. Eine zukünftige Anforderung kann sein, dass Benutzer die Entleihdauer eines Buches verlängern können, wenn es nicht anderweitig vorbestellt ist.

- Suche

Vom System sollte ermöglicht werden, dass Anwender nach einem Buch zu einem bestimmten Thema oder von einem bestimmten Autor etc. suchen und prüfen können, ob das Buch zur Entleihe verfügbar ist und, falls nicht, es auch gleich reservieren können. Die Suche ist für jedermann zugänglich.



Aufgabe

1. Sind diese Anforderungen vollständig? Welche weiteren Fragen müssten Sie noch stellen?
2. Identifizieren Sie im ersten Schritt alle vermutlichen Klassen des Systems durch Hauptwort-Identifikation aus der Anforderungsbeschreibung.
3. Sortieren Sie alle Hauptwörter aus, die aus offensichtlichen Gründen weniger gut als Klassen geeignet sind, um eine sinnvolle Liste der Klassen des Systems zu erhalten. Begründen Sie Ihre Entscheidung.
4. (Diskussionsfrage) Sind Sie zu der vorgestellten Lösung anderer Meinung? Warum?

A: Lösungen gibt es [hier](#)

- An einer Hochschule sind studentische Hilfskräfte und Angestellte zu verwalten. Für alle Personen sind der Name, bestehend aus Vor- und Nachname, und die Adresse, bestehend aus PLZ, Ort und Straße, zu speichern. Für studentische Hilfskräfte sind außer der Matrikelnummer auch Beginn und Ende aller Arbeitsverträge sowie die jeweilige wöchentliche Stundenzahl einzutragen. Alle studentischen Hilfskräfte erhalten den gleichen Studentenlohn. Für jeden



Angestellten wird das Eintrittsdatum gespeichert.

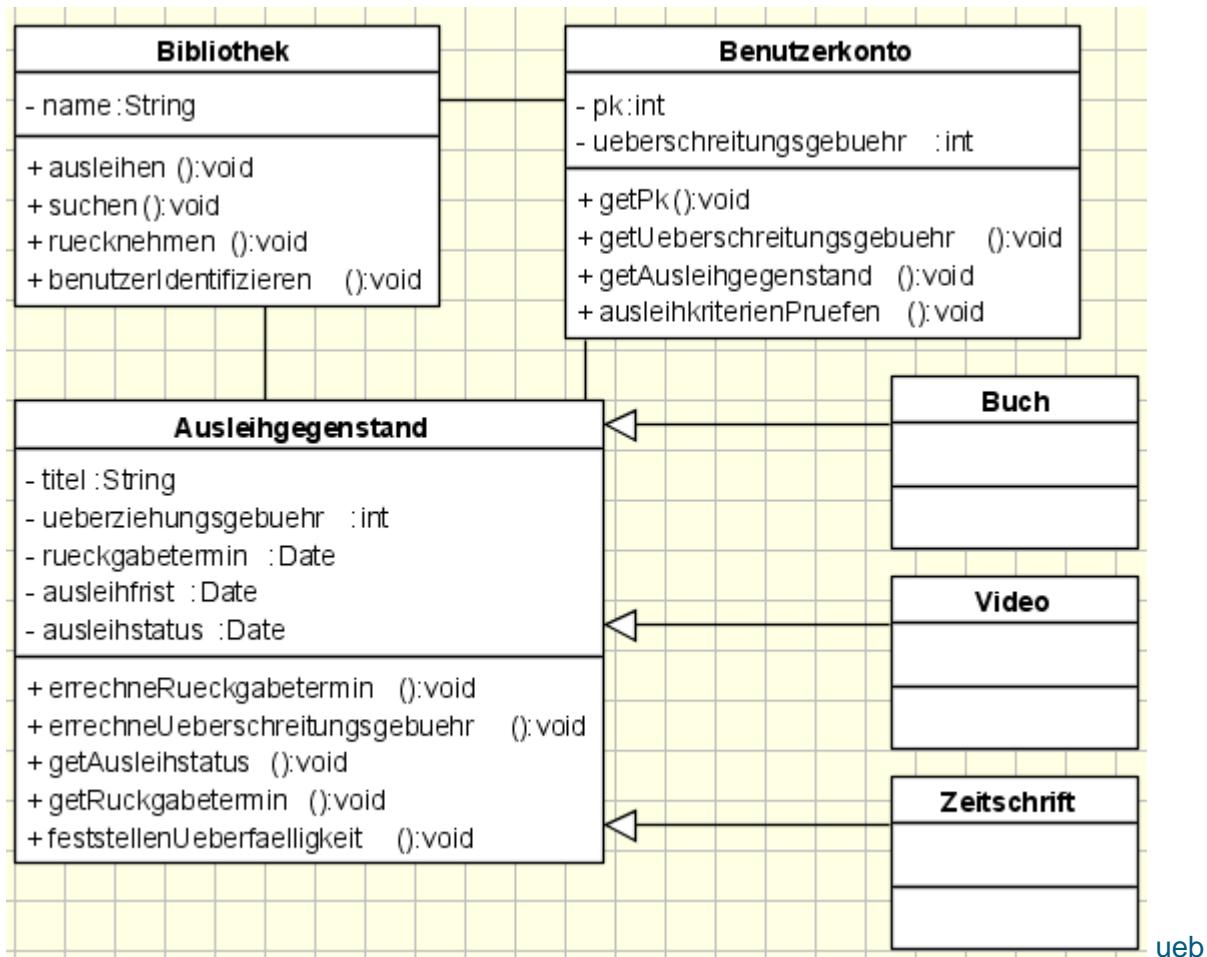
Identifizieren Sie Klassen und Attribute und stellen sie als Diagramm dar. Für jede Klasse ist eine Klassenbeschreibung zu erstellen. Alle Attribute sind zu spezifizieren. Gibt es elementare Klassen?

- Erstellen Sie aus der folgenden Beschreibung ein Klassendiagramm

Wir betrachten eine Bank und ihre Kunden. Eine Person wird Kunde, wenn sie ein Konto eröffnet. Ein Kunde kann beliebig viele Konten eröffnen. Für jeden neuen Kunden werden dessen (nicht notwendigerweise eindeutiger) Namen, Adresse und das Datum der erste Kontoeröffnung erfasst. Bei der Kontoeröffnung muss der Kunde gleich eine Einzahlung vornehmen. Wir unterscheiden Giro- und Sparkonten. Girokonten dürfen bis zu einem gewissen Betrag überzogen werden. Für jedes Konto wird ein individueller Habenzins, für Girokonten auch ein individueller Sollzins festgelegt; außerdem besitzt jedes Konto eine eindeutige Kontonummer. Für jedes Sparkonto wird die Art des Spars - z.B. Festgeld - gespeichert. Ein Kunde kann Beträge einzahlen und abheben. Des Weiteren werden Zinsen gutgeschrieben und bei Girokonten Überziehungszinsen abgebucht. Um die Zinsen zu berechnen, muss für jede Kontobewegung das Datum und der Betrag notiert werden. Die Gutschrift/Abbuchung der Zinsen erfolgt bei den Sparkonten jährlich und bei den Girokonten quartalsweise. Ein Kunde kann jedes seiner Konten wieder auflösen. Bei der Auflösung des letzten Kontos hört er auf, Kunde zu sein.

- Gegeben seien folgende Begriffe aus der zu modellierenden Realität: PKW, LKW, LKWmitAnhänger, Taxi, Autobus, Containerschiff, Fähre, Floss, Yacht.
 1. Identifizieren Sie die Klassen und entwerfen Sie eine Vererbungshierarchie.

2. Finden Sie Verallgemeinerungen der Klassen und ergänzen Sie die Vererbungshierarchie (zuerst in der Vererbungshierarchie soll nur eine Klasse stehen).
 3. Teilen Sie Ihre gefundenen Klassen in konkrete und abstrakte Klassen ein. Bezeichnen Sie konkrete Klassen mit K, abstrakte Klassen mit A.
- Verfeinern Sie das folgende Klassendiagramm der Bibliothek um eine nähere Beschreibung der Assoziationen (Multiplizität, Typ der Assoziation). Passen Sie auch die Rückgabewerte und Parameter der Operationen an. Jeder Benutzer kann höchstens fünf Dinge entleihen.



1.zuml A:

F: Gegeben seien folgende Anforderungen an ein Textverarbeitungssystem: Das Textverarbeitungssystem erlaubt es Peter Müller und anderen Benutzern Dokumente anzulegen und zu editieren. Ein Dokument kann Text und Bitmap-Grafik enthalten. Text besteht aus Abschnitten,

jeder Abschnitt aus Zeichen. Ein Dokument enthält außerdem verschiedene administrative Informationen wie seinen Titel, seinen Autor, den Dateinamen, in dem es abgelegt ist, sowie das Datum der letzten Änderung.

1. Welche Objekte/Klassen lassen sich im obigen Problemfall identifizieren?
2. Welche Beziehungen (Assoziation, Aggregation, Komposition) bestehen zwischen den Klassen? Zeichnen Sie das entsprechende Klassendiagramm in der UML-Notation.
3. Welche Attribute hat die Klasse Dokument? Ergänzen Sie das gezeichnete Klassendiagramm entsprechend. Achten Sie dabei auf Assoziationsnamen, Assoziationsenden (Rollen), Navigierbarkeit und Multiplizitäten.

A:

F: Spezifikation:

Eine Bahnlinie hat eine Nummer. An einer Bahnlinie liegen Bahnhöfe, die einen Namen tragen. Eine Bahnlinie hat einen Startbahnhof, einen Endbahnhof und keinen, einen oder mehrere Haltebahnhöfe. Einer Bahnlinie sind Fahrten zugeordnet, die nummeriert sind (Zugnummer) und die fahrplanmäßig auf dieser Bahnlinie verkehren. Die Fahrten werden von Zügen durchgeführt und haben einen Zustand (z.B. pünktlich oder unpünktlich). Ein Zug absolviert eine oder mehrere Fahrten der Bahnlinie. An den Haltebahnhöfen der Bahnlinie wird eine Fahrt jeweils durch einen Stop mit festgelegter Ankunftszeit und Abfahrtszeit unterbrochen. Jeder Zug wird durch eine Lok geführt und hat darüber hinaus Gepäckwagen, Schlafwagen und Personenwagen. Lok und alle Wagen tragen eine individuelle Herstellungsnummer.

Stellen Sie die oben beschriebene Spezifikation in einem statischen Klassendiagramm dar. Achten Sie dabei besonders auf Vererbung, Assoziationen, Aggregationen und Kompositionen, Rollen- und Assoziationsnamen (gegebenenfalls mit Richtung) und Multiplizitäten.

A: [hier](#) **F:** Beim Bäckermeister Krause kann man verschiedene Backwaren kaufen. Herr Krause steht hinter einer Theke, auf der eine Kasse steht. Herr Krause ist ein moderner Bäcker und hat bereits eine Scannerkasse. Das Warenangebot wird den Kunden in der Theke präsentiert. Die Kunden kaufen Brötchen, Brot, Kuchen und kleine Teilchen. Herr Krause ist so nett und verpackt die gewünschten Waren passend in verschiedenen großen Tüten. Jeder Kunde bekommt neben der Tüte einen Kassenbon, auf dem die Waren und ihr Preis verzeichnet sind.

Der Bäckerladen von Herrn Krause öffnet morgens schon um 6 Uhr, um auch hungrige Schüler, die von einer Shuttleparty kommen, mit warmen Brötchen versorgen zu können.

1. Identifizieren Sie Objekte und begründen Sie Ihre Wahl kurz anhand der in der Vorlesung genannten Prinzipien.
2. Gruppieren Sie die Objekte zu Klassen und stellen Sie sie in einem UML-Klassendiagramm dar.
3. Geben Sie für die Klassen die Schnittstellen an. Identifizieren Sie dabei Attribute und Methoden.
4. Beschreiben Sie die vielfältigen Beziehungen zwischen den gefundenen Klassen mit Hilfe der in der Vorlesung vorgestellten Modellierungstechniken.

A: [hier](#) **F:** Die 9 Planeten unseres Sonnensystems (außer der Erde) werden in innere und äußere Planeten unterteilt. Sie unterscheiden sich hauptsächlich durch ihre mittlere Entfernung zur Sonne und ihren Durchmesser. Den Tag, an dem Sonne, innerer Planet und Erde eine Linie bilden, nennt man Konjunktionsdatum, den Tag, an dem Sonne, Erde und äußerer Planet eine Linie bilden, Oppositionsdatum. Neben den bekannten großen Planeten schwirren noch tausende Kleinplaneten zwischen Mars und Jupiter umher. Ihre Bahnen sind teilweise so lang gestreckt, dass sie die Bahnen von bis zu 5 anderen Planeten kreuzen können. Generell haben Planeten einen eindeutigen Namen, bis zu 30 Monde und bestehen aus fester oder gasförmiger Oberfläche, Atmosphäre (jeweils bestehend aus mehreren chemischen Elementen) und evtl. einem oder mehreren Ringen (charakterisiert anhand des Durchmessers)

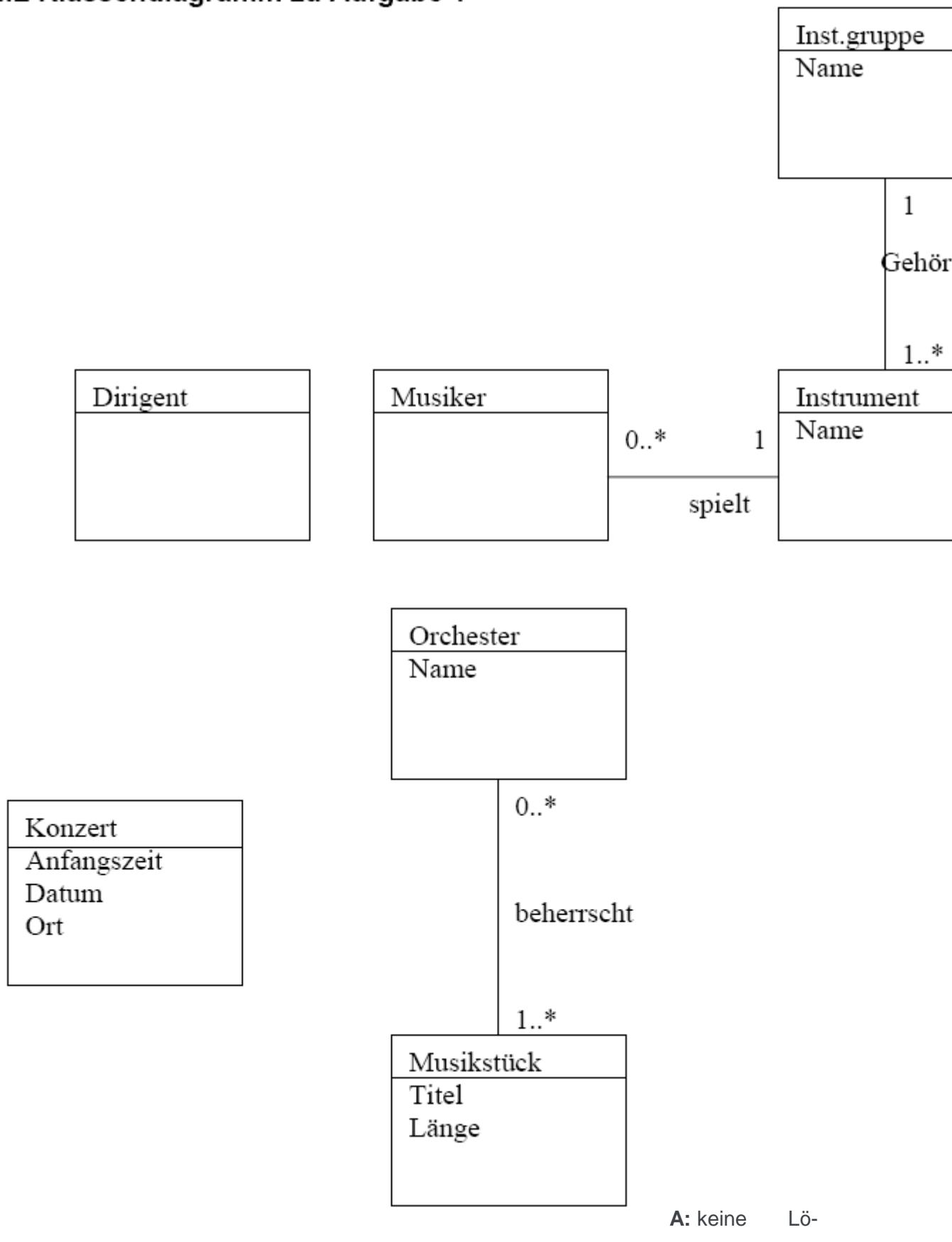
A: [hier](#) **F:** Es soll eine Datenbank angelegt werden, die Information über Orchester enthält. Ihr Kollege hat begonnen, ein entsprechendes UML-Klassendiagramm zu entwerfen und ist dann in Urlaub gefahren (siehe nächste Seite) ? Der Entwurf muss fertig werden. Sie erhalten das begonnene Diagramm und die folgende Beschreibung der Anwendungsdomäne. Ergänzen Sie das UML-Diagramm so, dass es die Beschreibung möglichst genau widerspiegelt! Verwenden Sie u.a. Generalisierung, Aggregierung, (mehrstellige) Beziehungen. Vergessen Sie nicht Attribute und Kardinalitäten passend zu ergänzen!

Orchester haben einen Namen und einen Heimatort. Sie bestehen aus einem Dirigenten und zwischen zehn und 50 Musikern. Sowohl Musiker als auch Dirigenten verfügen über einen Namen, ein Geburtsdatum und eine Adresse. Jeder Musiker spielt ein Instrument. Jedes Instrument gehört zu einer Instrumentengruppe (Streichinstrument, Holzbläser, Blechbläser,?). Orchester beherrschen Musikstücke (beschrieben durch ihren Titel und ihre Länge). Jedes Musikstück wurde von einem Komponisten (Name, Geburts- und ggf. Todesdatum) geschrieben. Musikstücke benötigen Instrumente in bestimmter Anzahl (Konzert No. 5 braucht z.B. 3

Flöten, 2 Geigen, ?). Orchester bringen ein oder mehrere Musikstücke bei Konzerten zur Aufführung. Konzerte sind beschrieben durch Datum, Anfangszeit und Ort. Sie werden von Personen (Name, Adresse) besucht.



UML Klassendiagramm zu Aufgabe 1



sung vorhanden

A: keine Lö-

Lösungen

Lösung zu Analyse/Entwurf

Aufgabe 1.2 Informationen für Analyse und Entwurf

Information	Relevant für
1. Für jeden Videofilm sind Titel, Laufzeit und Jahr zu speichern.	Daten aus Benutzersicht sind festzuhalten im <ul style="list-style-type: none"> • Pflichtenheft • OOA-Modell
2. Die Videofilme sind nach Titeln aufsteigend sortiert in der Datenbank xy zu speichern.	Pflichtenheft: gewünschte Datenbank als Information eintragen, wenn vom Auftraggeber vorgegeben Entwurf und Implementierung: festlegen, wie Informationen in der Datenbank gespeichert werden
3. Jede Ausleihe von Videofilmen wird im System gespeichert.	= Benutzerfunktion: wird im Pflichtenheft und im OOA-Modell dokumentiert

4. Defekte Videofilme werden aus der Videothek entfernt und in der Datei mit einem »L« markiert.	Analyse: = Benutzerfunktion: wird im Pflichtenheft und im OOA-Modell dokumentiert Entwurf und Implementierung: festgelegen, wie das logische Löschen der Sätze realisiert wird.
5. Das System soll jederzeit einen Überblick über die Ausleihhäufigkeit der einzelnen Filme erlauben.	Zielsetzung: im Pflichtenheft dokumentiert
6. Für die Realisierung der Benutzungsoberfläche wird die Klassenbibliothek abc verwendet.	Wie die Benutzungsschnittstelle realisiert wird, ist Gegenstand des Entwurfs
7. Da es sich um eine große Videothek handelt, ist eine Client-Server-Anwendung notwendig, wobei alle zentralen Daten auf dem Server liegen.	Systemanalyse: Nur die Forderung einer Client-Server-Verteilung spezifizieren Entwurf: Spezifizieren, wie die Verteilung durchzuführen ist.

Lösung zu Klassenidentifikation

a) Mögliche weitere Fragen an den Auftraggeber sind zum Beispiel:

- Welche Informationen über Bücher, Zeitschriften und Benutzer müssen erfasst werden?
- Wie werden Bücher, Zeitschriften und Benutzer verwaltet, neu angelegt, gelöscht usw.?
- Werden Benutzer direkt mit dem System interagieren oder wird es nur von den Bibliothekaren benutzt werden? Wird die Benutzerinteraktion nur auf die Suche beschränkt sein oder dürfen sie auch Daten ändern?
- Wie läuft die Ausleihe ab? Wie wird ein Buchexemplar dabei identifiziert? (Es könnte z.B. über einen aufgedruckten Barcode von einem Scanner identifiziert werden).
- Werden Anfragen durch die Bibliothekare oder nur elektronisch durchgeführt?
- Ist eine E-Mail oder WWW-Schnittstelle erforderlich und wenn ja für welche Funktionen?
- Welche Funktionen haben die höchste Priorität?
- Welche Erweiterungen des Systems könnten in Zukunft erforderlich sein?

Bücher und Zeitschriften In der **Bibliothek** gibt es **Bücher** und **Zeitschriften**. Von Büchern kann es **mehrere Exemplare** geben. Einige Bücher sind nur für **Kurzzeitentleihe** gedacht. Alle anderen Bücher können von jedem **Bibliotheksmitglied** für drei **Wochen** entliehen werden. Zeitschriften dürfen nur von **Mitarbeitern** ausgeliehen werden. **Mitglieder der Bibliothek** können normalerweise bis zu 6 **Einheiten** entleihen, während Mitarbeiter der Bibliothek bis zu 12 Einheiten zur gleichen **Zeit** entleihen können.

Entleihe Wichtig ist, dass das **System** verfolgen kann, wann Bücher und Zeitschriften ausgeliehen und zurückgegeben werden, weil dies auch das aktuelle System schon kann. Das neue System sollte darüber hinaus **Mahnungen** auswerfen, wenn ein **Entleihzeitraum** überzogen ist. Eine zukünftige **Anforderung** kann sein, dass **Benutzer** die **Entleihdauer** eines Buches verlängern können, wenn es nicht anderweitig vorbestellt ist.

Suche Vom System sollte ermöglicht werden, dass **Anwender** nach einem Buch zu einem bestimmten **Thema** oder von einem bestimmten **Autor** etc. suchen und prüfen können, ob das Buch zur **Entleihe** verfügbar ist und, falls nicht, es auch gleich reservieren können. Die **Suche** ist für jedermann zugänglich.

c) Aussortierte Hauptwörter:

- **Bibliothek**, weil es außerhalb des Bereichs des Systems liegt
- **Entleihe** und **Kurzzeitentleihe**, weil eine Entleihe genau genommen ein Ereignis ist, eben die Ausgabe eines Buches an einen Benutzer, was nach derzeitigem Ermessen kein brauchbares Objekt für das System sein kann (siehe d))
- **Woche**, weil es eine Zeiteinheit ist, nicht ein Ding
- **Mitglied der Bibliothek**, weil es redundant ist und das Gleiche bedeutet wie Bibliotheksmitglied, das wir beibehalten und nicht aussondern
- **Einheit**, weil es zu unklar ist; bei näherer Untersuchung sehen wir, dass es für Buch oder Zeitschrift steht
- **Zeit**, weil es außerhalb des Bereichs unseres Systems liegt
- **System** und **Anforderung**, weil es zur Metasprache der Anforderungsbeschreibung gehört und nicht Bestandteil des beschriebenen Problembereichs ist
- **Benutzer** und **Anwender** aus dem gleichen Grund
- **Mahnung**, weil hier ein Ereignis/Vorgang gemeint ist ("Mahnung auswerfen")
- **Entleihzeitraum** bzw. **Entleihdauer** ist ein Attribut
- **Suche**, weil es sich um einen Vorgang handelt
- **Thema** und **Autor**, weil dies Attribute von Buch sind

Danach bleibt eine Liste der relevanten Klassen übrig:

- **Buch**
- **Zeitschrift**
- **Exemplar (von Buch)**
- **Bibliotheksmitglied**
- **Mitarbeiter**

Diese Liste ist ein erster Ansatz zur weiteren Modellierung des Systems.

d) Bei der objektorientierten Analyse gibt es keine eindeutigen Lösungen. Man kann sicherlich darüber diskutieren, ob der eine oder andere Begriff beibehalten werden sollte. Eine Möglichkeit ist die „Entleihe“ im Sinne eines komplexen Vorganges (Entleihe, Storno, Rückgabe, Mahnung,...), mit deren Hilfe man den Entleihvorgang handhaben und für statistische Zwecke auswerten könnte. Diese Begriffe sind im Hinterkopf zu behalten und ggf. bei der weiteren Analyse (Klassendiagramm, Use Cases, CRC-Karten) zu ergänzen. Hier müsste man unter anderem abwägen zwischen der Speicherung redundanter Informationen und Effizienz. Ein weiterer Kandidat in dieser Kategorie ist „Suche“.

Lösung zur Spezifikation von Klassen und Attributen

Angestellter	Studentische Hilfskraft
Name	Matrikelnummer
Adresse	Name
Eintrittsdatum	Adresse
	Beschäftigung
	Stundenlohn

Klassenbeschreibung:

Studentische Hilfskraft: Verwaltet Studenten, die zeitlich begrenzt als Hilfskraft an der Hochschule tätig sind
 Angestellter: Verwaltet alle Angestellten einer Hochschule

Studentische Hilfskraft:

Matrikelnummer:	String
Name:	NameT
Adresse:	AdresseT
Beschäftigung:	list of ArbeitsvertragT
Stundenlohn:	Fixed (2,2)

Angestellter:

Name:	NameT
Adresse:	AdresseT
Eintrittsdatum:	Date

Hierbei wurden die elementaren Klassen NameT, AdresseT und ArbeitsvertragT verwendet.

NameT	AdresseT	ArbeitsvertragT
Vorname: String	Strasse: String	Beginn: Date
Nachname: String	PLZ: String	Ende: Date
	Ort: String	Stundenzahl: UInt

Abbildung 2: Elementare Klassen

Anmerkungen:

- Weil der Stundenlohn für alle studentischen Hilfskräfte gleich ist, wird er als Klassenattribut modelliert.
- Als Typ wurde für den Stundenlohn Fixed (2,2) gewählt. Damit wird eine genaue Darstellung auf den Pfennig genau ohne Rundungsfehler gefordert.

- Für die Stundenzahl wurde der Typ `UInt` verwendet, weil davon ausgegangen wird, dass nur ganze Stunden vereinbart werden.

Lösungen zur Bank

[Einfach](#)

[Attribute und Methoden](#)

[Detailliert](#)

Lösungen zu Wedel Aufgabe 1

[loes_ueb01.zuml](#)

Lösungen zu Wedel Aufgabe 2

[loes_ueb02.zuml](#)

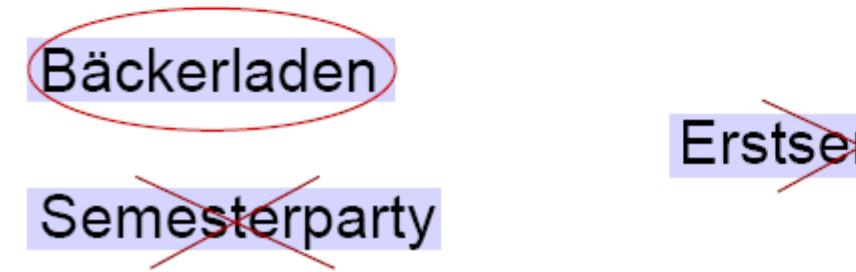
Lösungen zu Bäckereimeister Krause

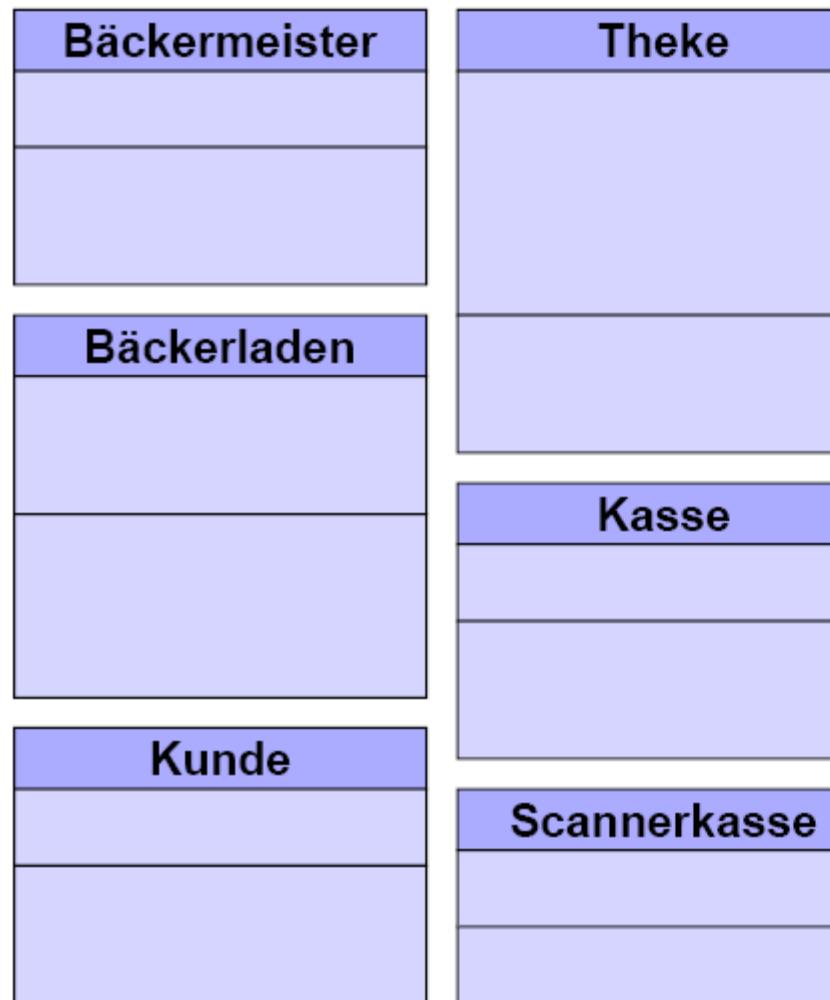
- Aufgabe 1:



- Aufgabe 2:





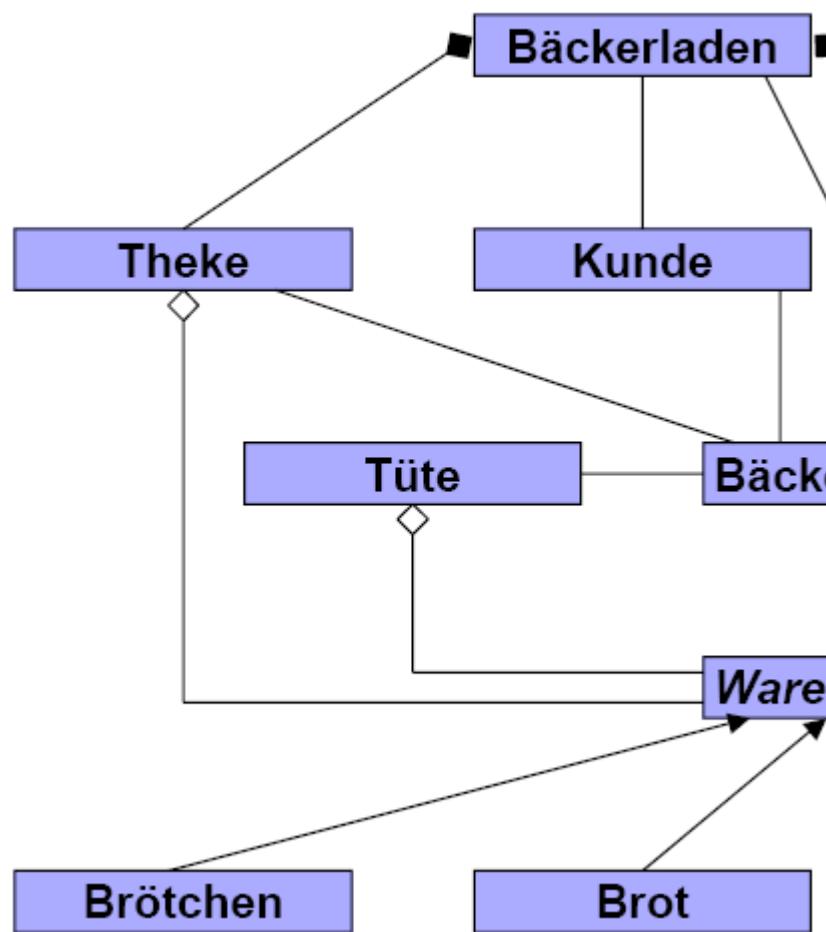


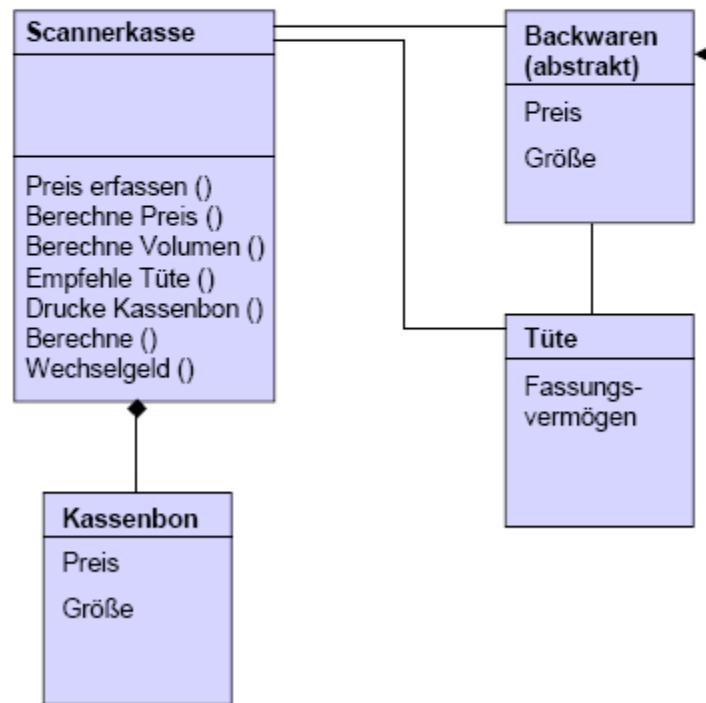
- Aufgabe 3:



Bäckermeister	Theke
Name	Brötchenvorrat Brotvorrat Kuchenvorrat Teilchenvorrat
backen bedienen	
Bäckerladen	
Firmenname Öffnungszeiten	Ware hineinlegen Ware entnehmen
öffnen schließen betreten	
Kunde	Kasse
Name	Zwischensumme
bestellen bezahlen	Artikel eingeben Summe ausgeben
	Scannerkasse
	Artikel eingeben

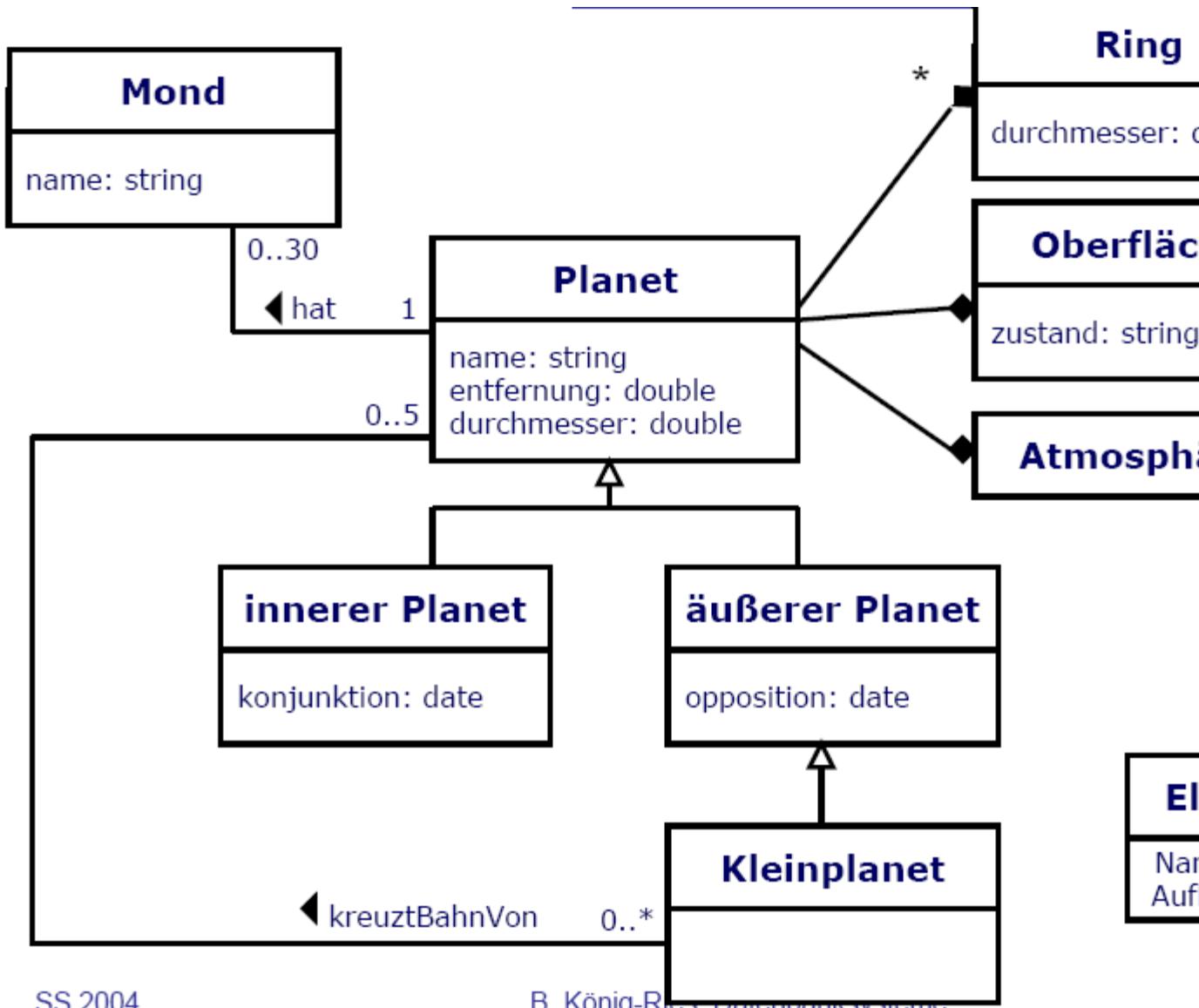
- Aufgabe 4:





Aufgabe zu Sonnensystem

Die 9 Planeten unseres Sonnensystems (außer der Erde) innere und äußere Planeten unterteilt. Sie unterscheiden hauptsächlich durch ihre mittlere Entfernung zur Sonne und Durchmesser. Den Tag, an dem Sonne, innerer Planet und Linie bilden, nennt man Konjunktionsdatum, den Tag, an dem Erde und äußerer Planet eine Linie bilden, Oppositionsdatum. Die bekannten großen Planeten schwirren noch tausende Kleinplaneten zwischen Mars und Jupiter umher. Ihre Bahnen sind teilweise so lang gestreckt, dass sie die Bahnen von bis zu 20 Planeten kreuzen können. Generell haben Planeten einen Namen, bis zu 30 Monde und bestehen aus fester oder gasiger Oberfläche, Atmosphäre (jeweils bestehend aus mehreren Elementen) und evtl. einem oder mehreren Ringen (charakterisiert anhand des Durchmessers).

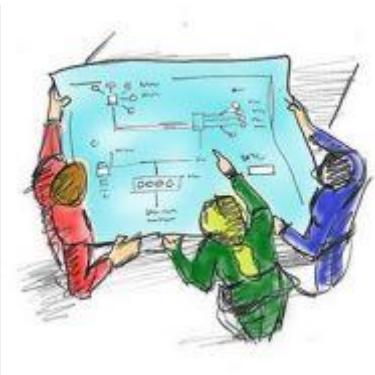


SS 2004

B. König-Ries, Universität Regensburg

Index

- Stichwortverzeichnis
- Modulindex
- Suche



Inhalt

- [UML »](#)

© Copyright 2014, ste. Mit [Sphinx](#) 1.5b1 erstellt.

