Inhaltsverzeichnis

1. UML

1.1. Ziele

☑ UML kennen lernen und anwenden können
☑ Einfache Probleme mit UML modellieren können
☑ Quelle
  ☐ Günter Wahl: UML Kompakt, In OBJEKTspektrum 2/1998
  ☐ Martin Glinz (http://www.ifi.uzh.ch/rerg/people/glinz/ 10.2010)
  ☐ http://www.toolboxx.de/uml/content.htm echt super
  ☐ http://www.websequencediagrams.com/ online Sequenzdiagramme erstellen

☑ Referenz-Cards:
  ☐ http://www.holub.com/goodies/uml/
  ☐ http://tnerual.eriogerg.free.fr/umlqrc.pdf

☑ Software
  ☐ Modelio: http://www.modelio.org/downloads/download-modelio.html
  ☐ http://java.dzone.com/articles/improve-your-java-development

1.2. Was ist die UML?

Die **Unified Modeling Language (UML)** ist eine Sprache zur

Beschreibung von Softwaresystemen.

Weitere Merkmale:
☑ Hersteller- und Softwareunabhängig

☑ OMG (Object Management Group) betreut UML

Alle **Softwaresysteme** (Datenbanken, Echtzeitsysteme, …) sollen mit der UML darstellbar sein.

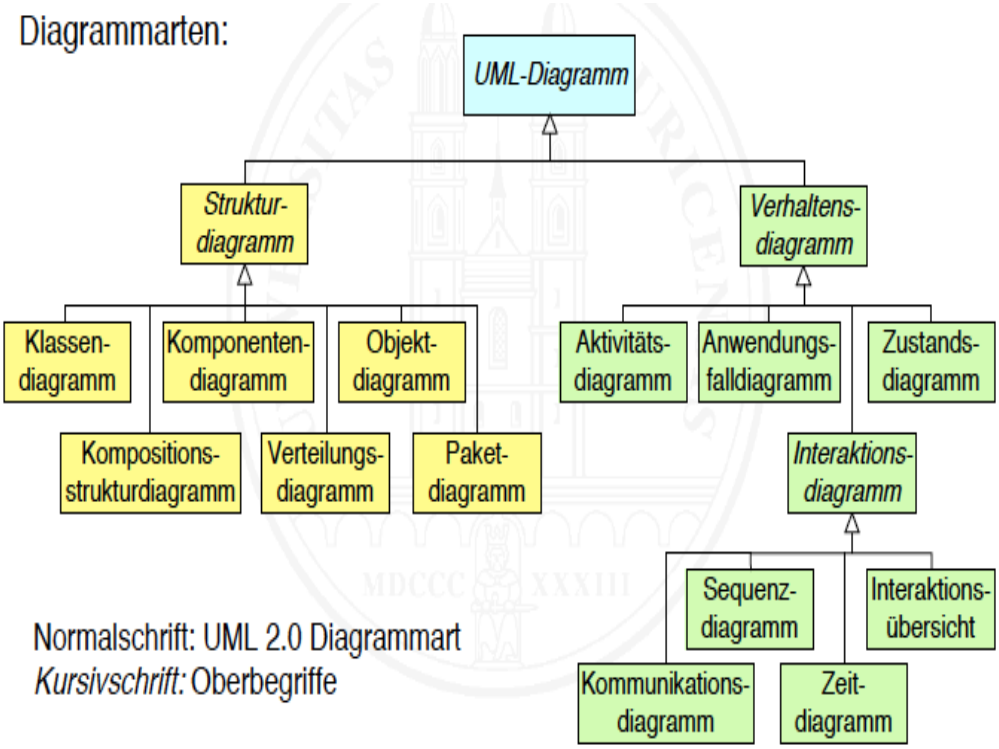
In **jeder Phase** der Softwareentwicklung (Analyse,Entwurf,Programmierung) werden **UML**-Diagramme eingesetzt.

1.3. Die Diagramme im Überblick

Die Notation der UML umfasst Diagramme für die

Darstellung der verschiedenen Ansichten auf das System,

vergleichbar mit Bauplänen für Häuser. Auch hier gibt es z. B. einen Grundriss, einen Lageplan, verschiedene Außenansichten und Werkpläne für die Handwerker. Für eine spezielle Aufgabe ist meist eine Diagrammart besser geeignet als andere.



☑ **Use-Case-Diagramm (Analysephase)**

☑ **Klassendiagramm (statisches Modell / Designphase)**

☑ **Interaktionsdiagramm (dynamisches Modell / Designphase)**

- ☐ **Sequenzdiagramm (Aufrufe zw. Objekten inkl. Zeitachse)**
- ☐ **Kollaborationsdiagramm (Aufrufe zw. Objekten ohne Zeitachse)**

☑ **Zustandsdiagramm** (dynamisches Modell / Designphase)

☑ **Aktivitätsdiagramm** (dynamisches Modell / Designphase)

☑ Package Diagramm

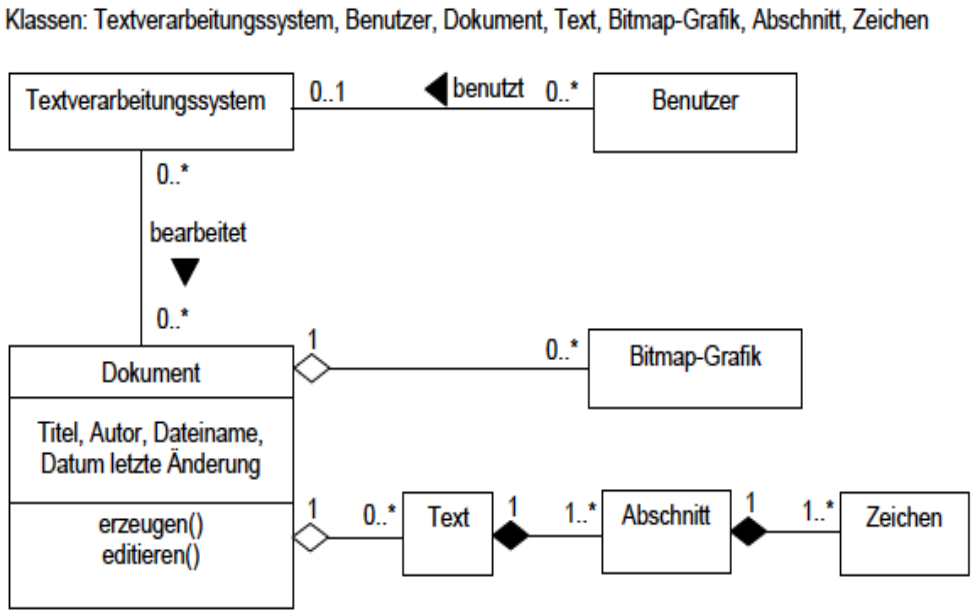
☑ Implementierungsdiagramm

- ☐ Komponentendiagramm
- ☐ Deploymentdiagramm

1.4. Beispiele für einen ersten Überblick

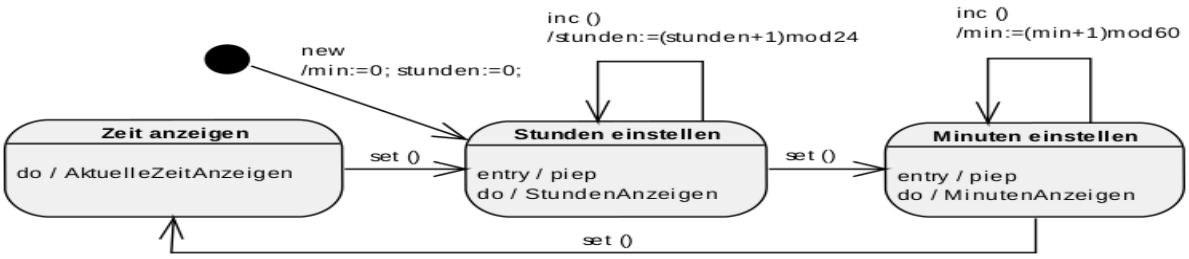
siehe auch: http://www.uml-diagrams.org/index-examples.html (SUPER)!!!

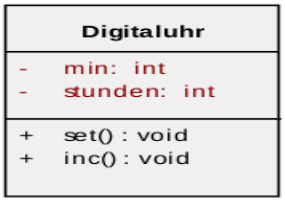
1.4.1. Klassendiagramm



1.4.2. Zustandsdiagramm

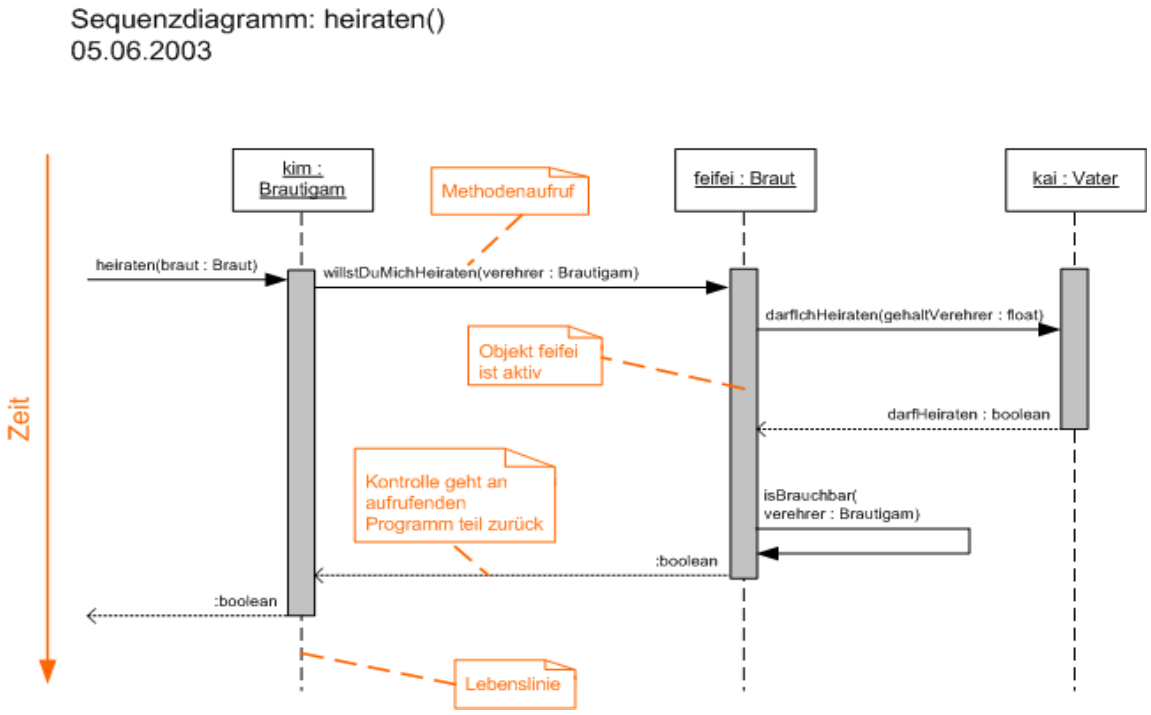
Event/Action und Activity



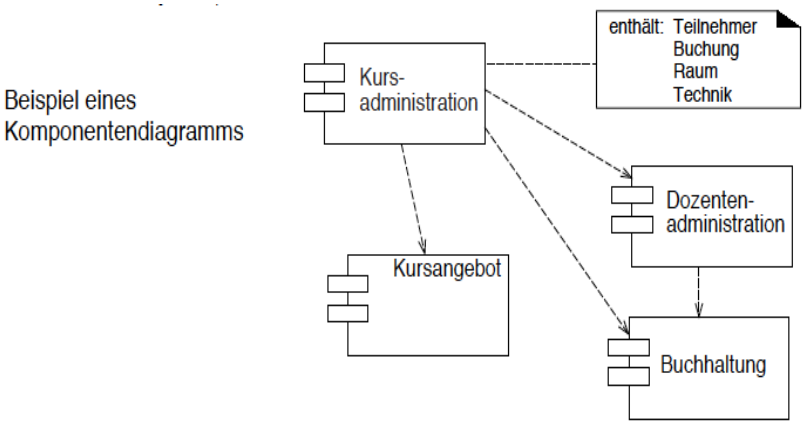


6

1.4.3. Sequenzdiagramm



1.4.4. Komponentendiagramm

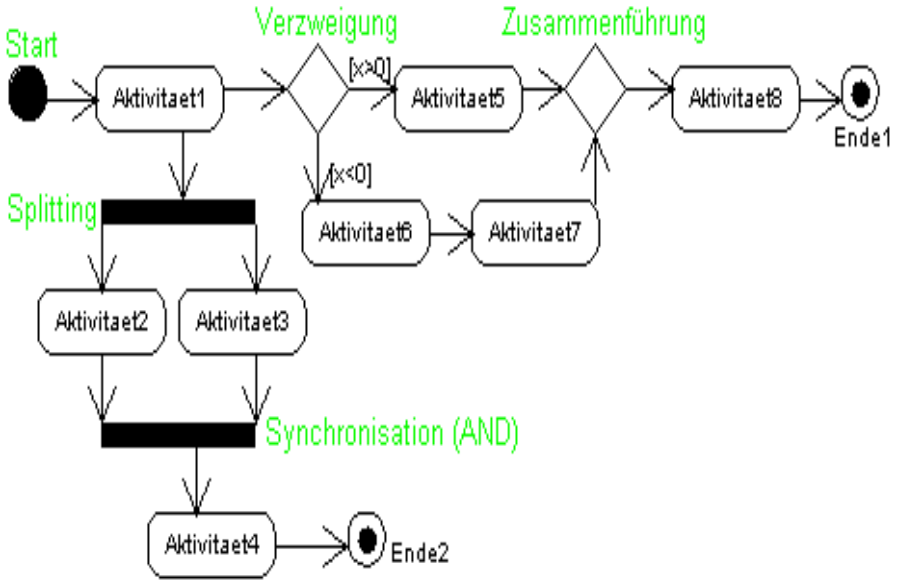


1.4.5. +Aufgabe: Komponentendiagramm f. DA

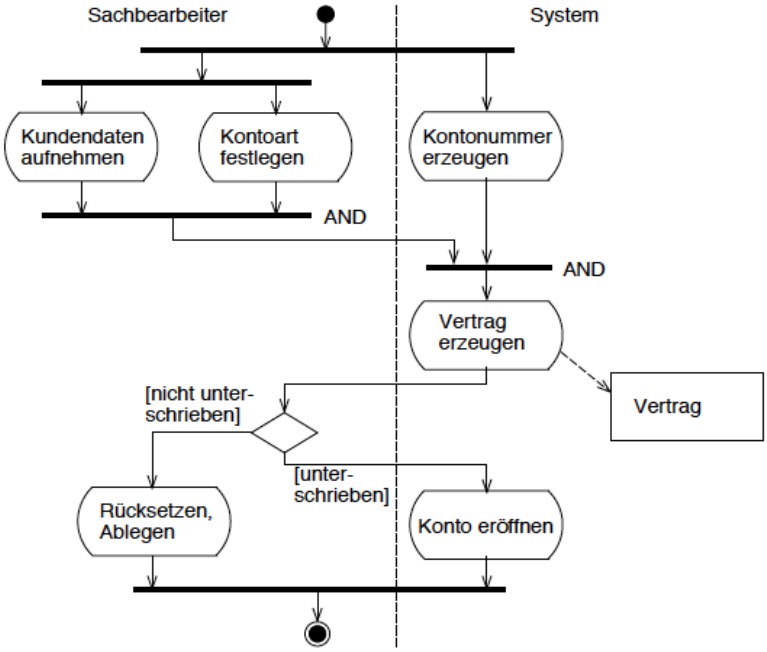
http://www.uml-diagrams.org/examples/online-shopping-uml-component-diagram-example.html?context=cmp-examples

Erstellen Sie auf der obigen Grundlage ein Komponentendiagramm Ihrer DA

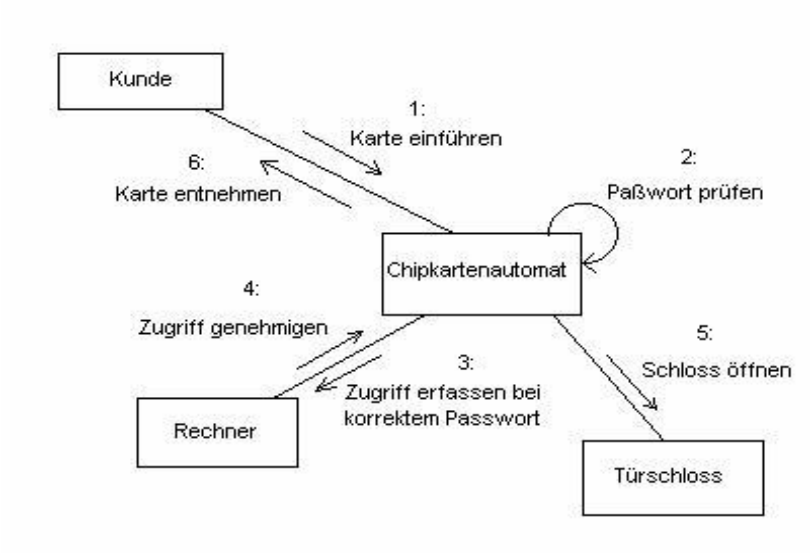
1.4.6. Aktivitätsdiagramm



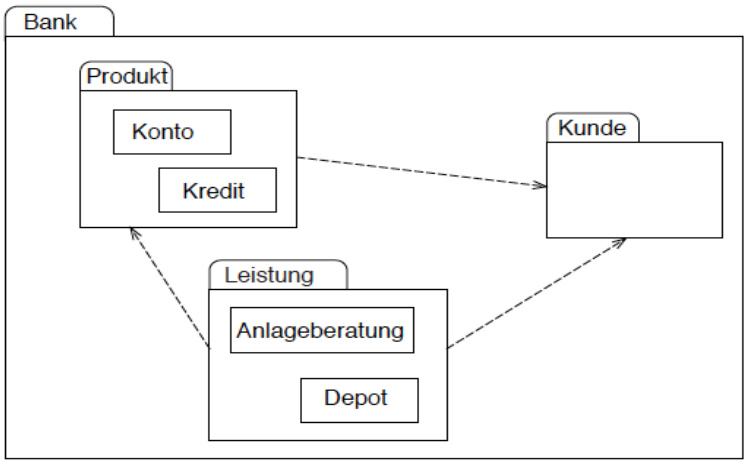
Beispiel eines Aktivitätsdiagramms:



1.4.7. Kommunikationsdiagramm / Kollaborationsdiagramm

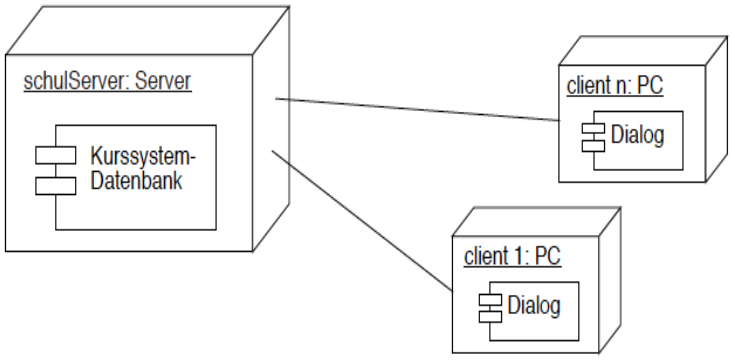


1.4.8. +Paketdiagramm



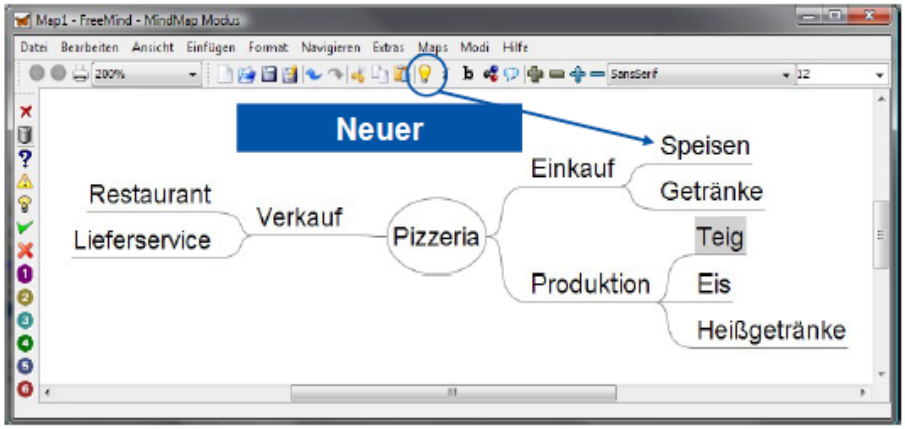
Beispiel eines Paketdiagramms

1.4.9. +Verteilungsdiagramm

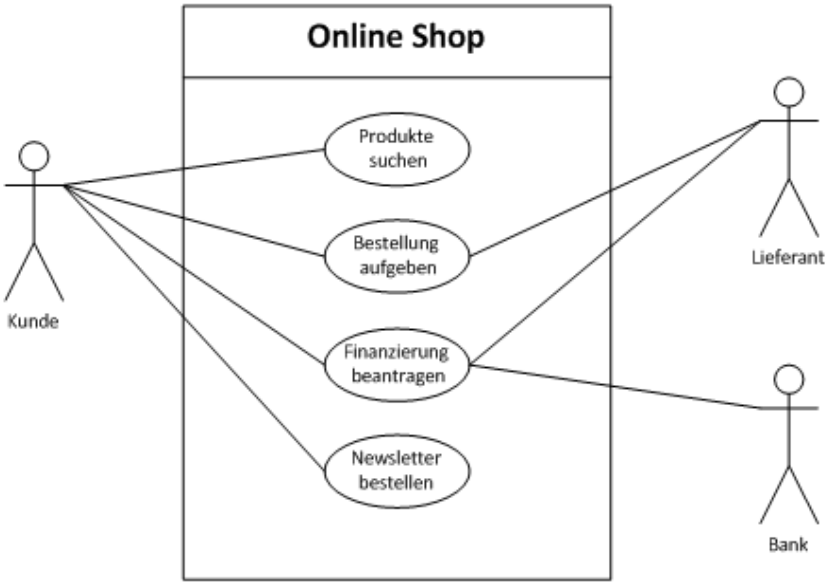


Beispiel eines Verteilungsdiagramms

1.4.10. Mindmap



1.4.11. Use-Case Diagramm



Zum Vorgehen ist soviel zu sagen, dass bei weitem **nicht alles modelliert werden soll**. Ein Modell ist eine Ansicht der Realität, die den Zweck hat, einen bestimmten Sachverhalt darzustellen.

1.5. Klassendiagramm (OOA,OOD)

Klassendiagramme sind der zentrale Bestandteil der UML und auch zahlreicher objektorientierter Methoden. Wie die Klassen ermittelt werden, darüber gibt die UML keine Auskunft; hierfür gibt es andere Techniken, z. B. CRC-Karten (Abk. für "Class, Responsibility and Collaboration") oder die Substantiv-Technik. Die UML beschreibt lediglich die Notation.

1.5.1. Elemente und Darstellung des Klassendiagramms

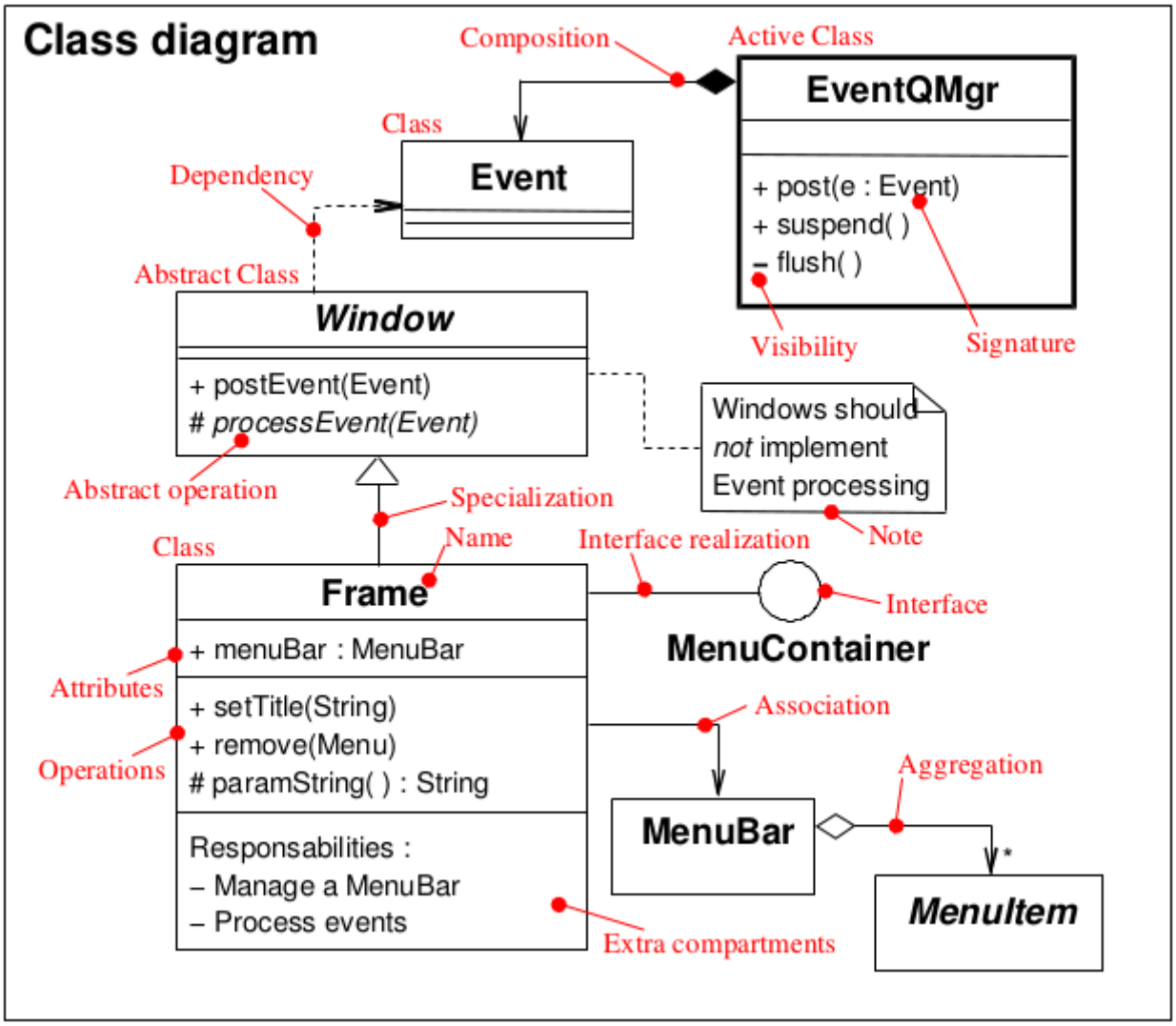
Das Klassendiagramm beschreibt die

statische Struktur

in einem System sowie ihre

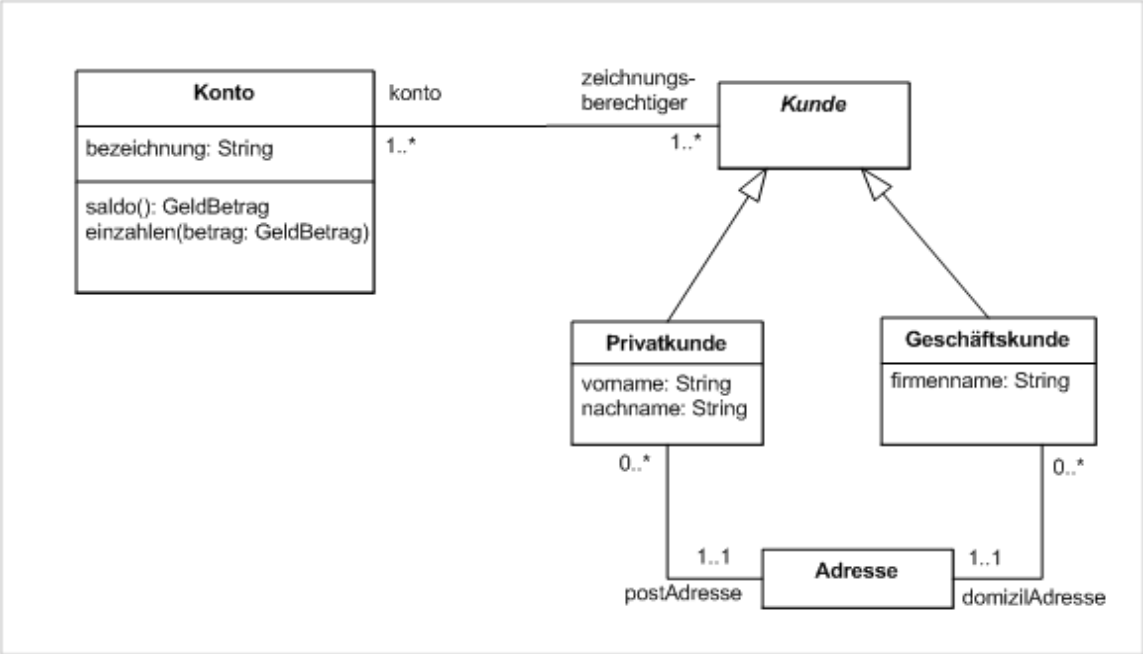
Beziehungen untereinander.

Sehen Sie hier eine Zusammenfassung der Bestandteile eines Klassendiagramms:



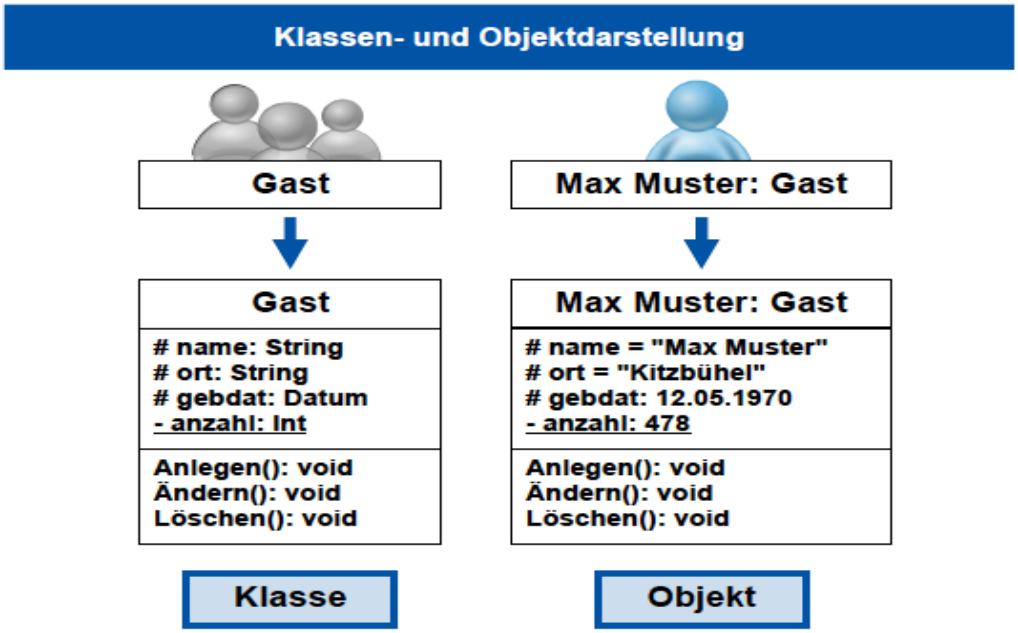
Die folg. Abbildung zeigt ein - zugegebenermaßen stark vereinfachtes - Klassendiagramm einer Bank mit einer automatischen Geldausgabe an einem Geldautomaten.

☑ Ein erstes Klassendiagramm "Bank"



☑ **Klassen- und Objektdarstellung**

Die **Klasse** ist das zentrale Element; sie wird als **Rechteck** dargestellt (z. B. Konto).



Eine **Klasse** besteht aus **Attributen und Methoden**.

☑ Attribute stellen die Eigenschaften der Objekte einer Klasse dar und bilden den Datenbestand der Klasse.

☑ Was mit diesen Daten getan werden kann, legen die Methoden fest. Methoden sind die aus anderen Sprachen bekannten Funktionen.

Wer auf die Daten zugreifen kann, wird durch die Sichtbarkeit der Daten bestimmt. **Sichtbarkeiten** können frühestens nach der Festlegungsphase eingeführt werden.

☑ **private** (-) bedeutet, dass nur die Methoden dieser Klasse Zugriff haben, und

☑ **protected** (#) heißt, dass auch Objekte der abgeleiteten Klassen Zugriff haben.

☑ **public** (+) besagt, dass ein Zugriff jederzeit möglich ist.

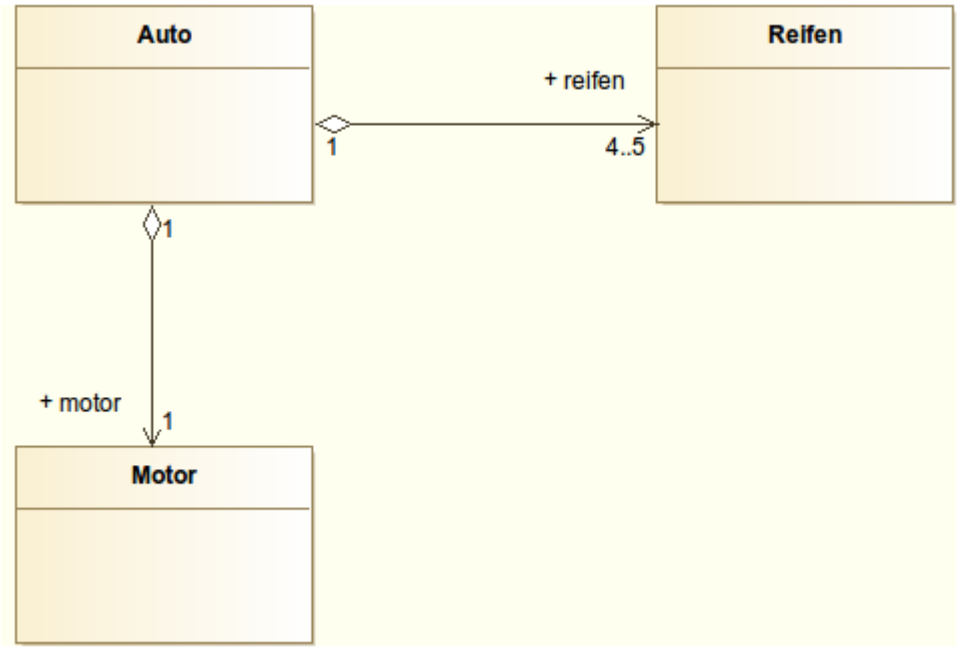
Die Sichtbarkeit kann sowohl für Attribute, als auch für Methoden angegeben werden.

Die gefundenen Klassen werden durch Linien miteinander verbunden. Diese Linien stellen die Elemente **Assoziation, Aggregation, Komposition** und Vererbung dar.

☑ **Beziehungen**

Die **Assoziation** stellt eine **allgemeine Beziehung** zwischen zwei Klassen dar - über die Realisierung wird dabei nichts ausgesagt.

Eine besondere Assoziation ist die **Aggregation**, die durch eine **Raute an der Linie** dargestellt wird (z. B. zwischen Kartenleser und Geldautomat). Sie gibt an, dass eine Klasse Kartenleser in der Klasse Geldautomat "enthalten" ist (Ist-Teil-von-Beziehung).



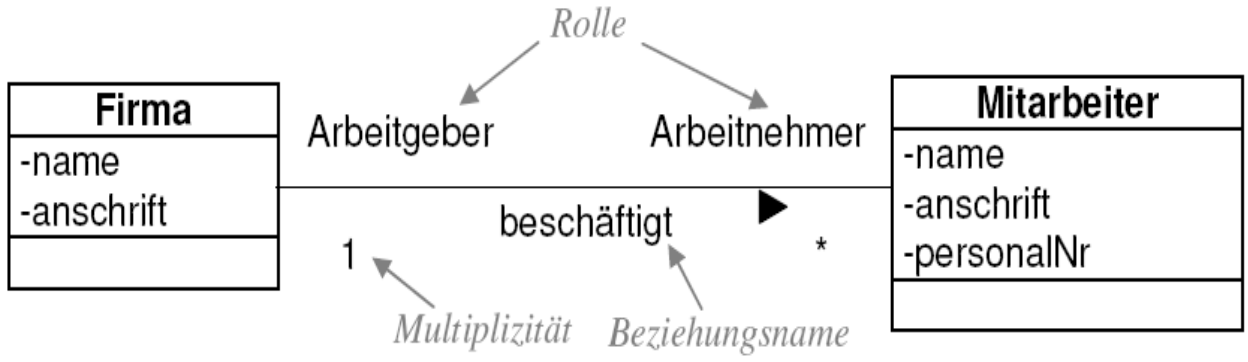
Die **Komposition ist eine stärkere Form der Aggregation**, die durch eine **ausgefüllte** Raute dargestellt wird (siehe Tresor). Bei der Beziehung Komposition handelt es sich um ein physikalisches Enthaltensein. In C++ können Aggregation und Komposition wie folgt realisiert werden:

```
class Mensch {
     // ...

     // Aggregation durch Zeiger
     Computer* pComputer;

     //Komposition: Kopf ist ein Datenmember
     Kopf kopf;

     //...
};
```

Jede Assoziation kann eine Richtung besitzen; hierfür wird ein Pfeil am Ende der Assoziation angebracht. Zugriffe können dann nur in Pfeilrichtung erfolgen. Man verwendet deshalb den Begriff der Navigationsfähigkeit (engl. Navigability). **Eine Assoziation mit Pfeil kann als Zeiger in einer Programmiersprache betrachtet werden.**

☑ Multiplizitäten

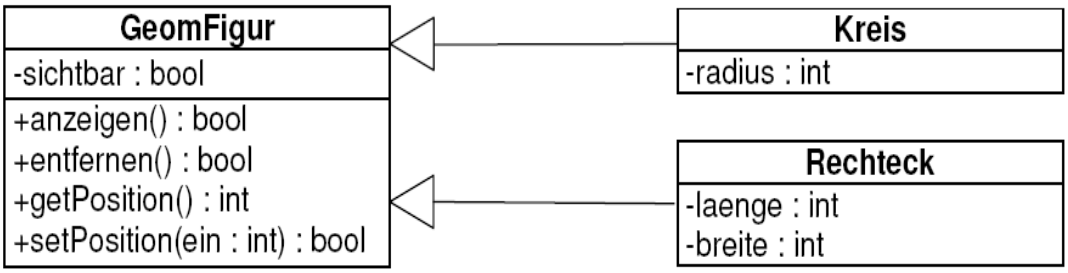
An einer Assoziation können noch **Multiplizitäten**, d. h. Zahlen oder Zahlbereiche, angegeben werden. Diese **bestimmen die Anzahl der Objekte, die miteinander in Beziehung** stehen. Beispielsweise arbeiten Mitarbeiter bei genau einer Firma.



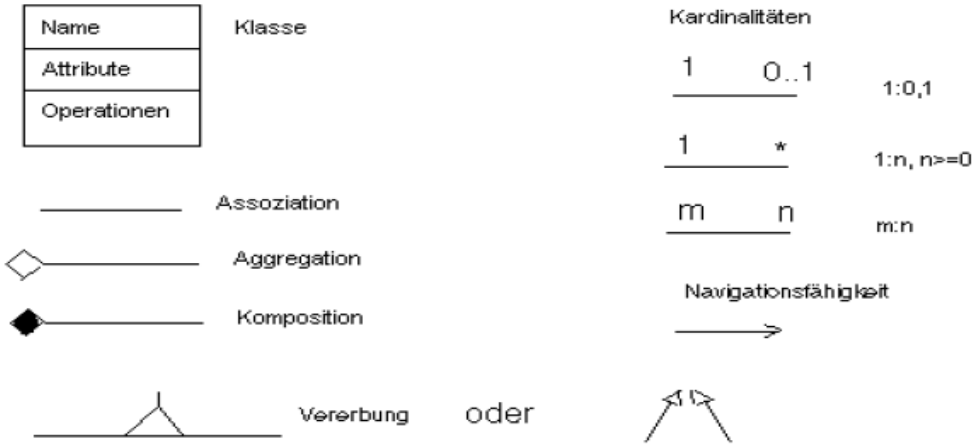
☑ Vererbung

Die Vererbung stellt eine Verallgemeinerung von Eigenschaften dar - sie wird auch als **Spezialisierung und Generalisierung** oder "**Ist-ein**"-Beziehung bezeichnet. Z. B. ist ein Auto ein Fahrzeug. Ein Fahrzeug hat generelle Eigenschaften eines Autos, und ein Auto spezialisiert die Eigenschaften von Fahrzeugen.

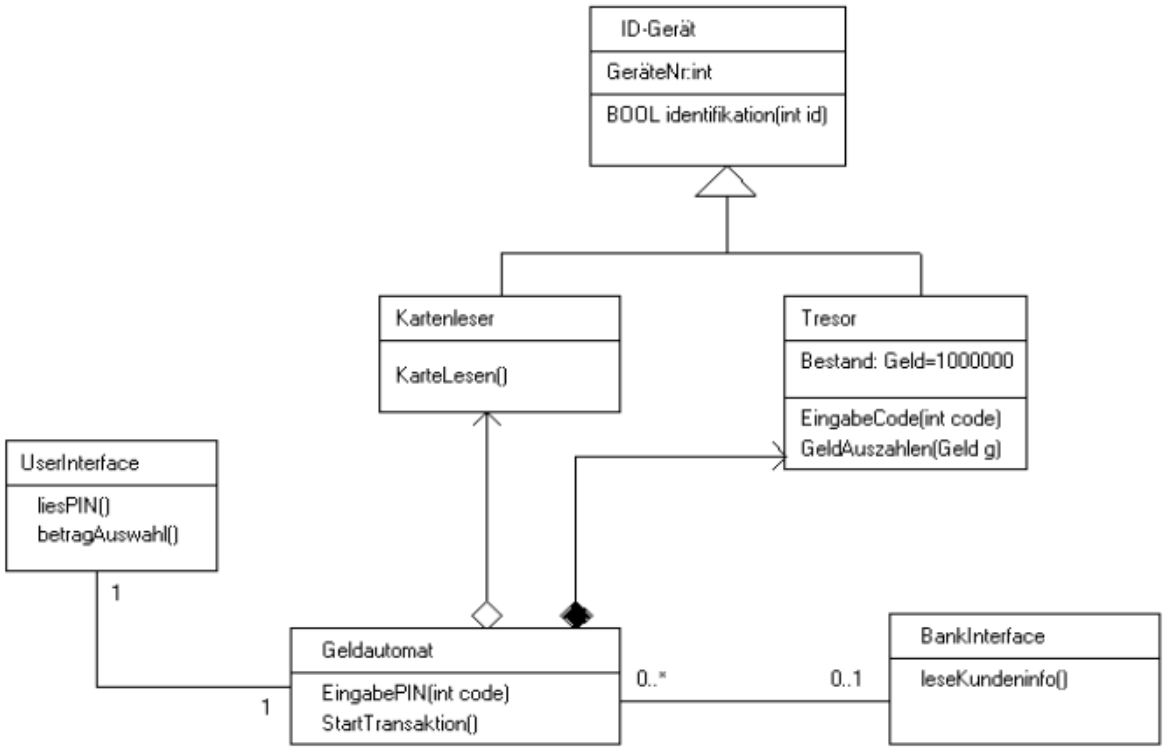
Die wichtigsten Elemente eines Klassendiagramms sind nochmals in der folg. Abbildung dargestellt.



☑ **Elemente eines Klassendiagramms**



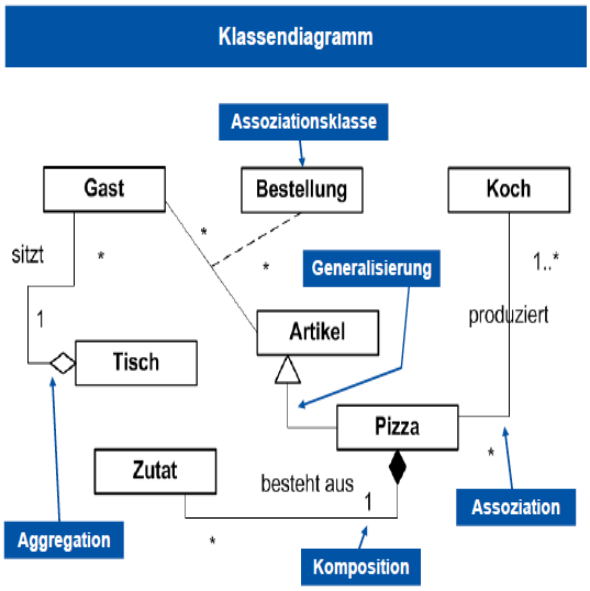
☑ Hier noch ein Klassendiagramm: „Geldautomat“



1.5.2. Übung: Klassendiagramm: Pizza (L)

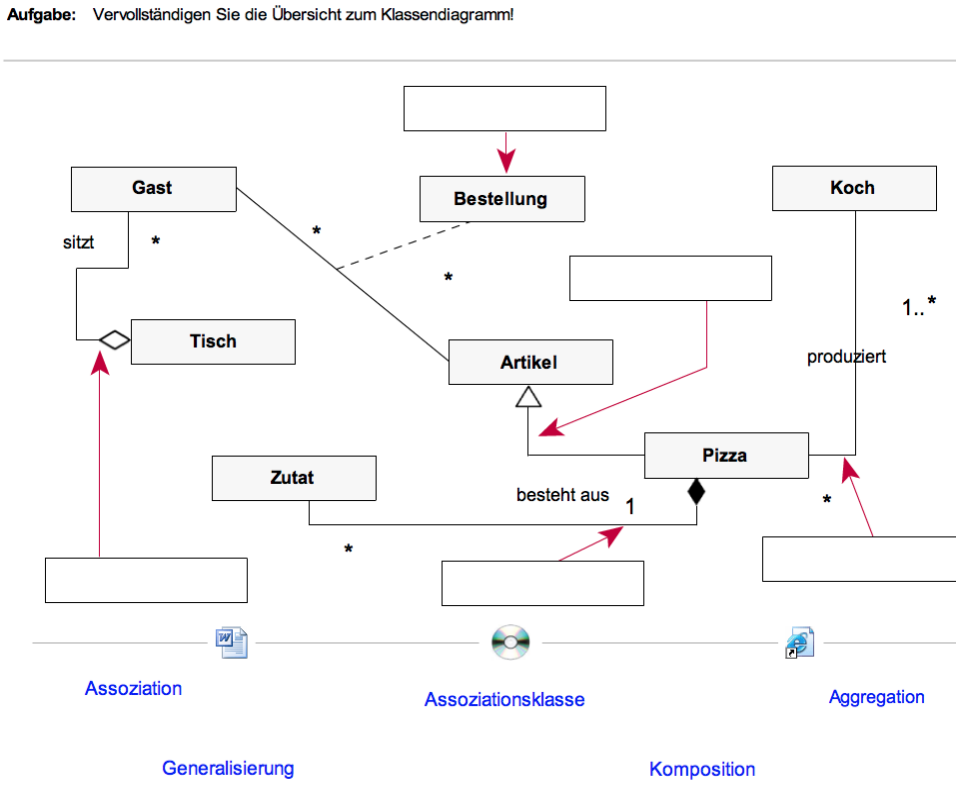
Erstellen Sie das Klassendiagramm Pizza. Folgendes sollte berücksichtigt werden:
Gast, Tisch,Bestellung,Zutat,Pizza,Artikel,Koch
Verzichten Sie hier noch auf die Verwendung von Attributen und Methoden.

Lösung: Klassendiagramm: Pizza



1.5.3. Übung: Klassendiagramm: Pizza (Begriffe) (m)

Vervollständigen Sie die Begriffe im Klassendiagramm:



1.5.4. +Aufgabe: Klassendiagramm DA

Erstellen Sie ein Klassendiagramm, das die wichtigsten Klassen Ihrer DA enthält.
Fügen Sie dieses Klassendiagramm in Ihre DA ein.

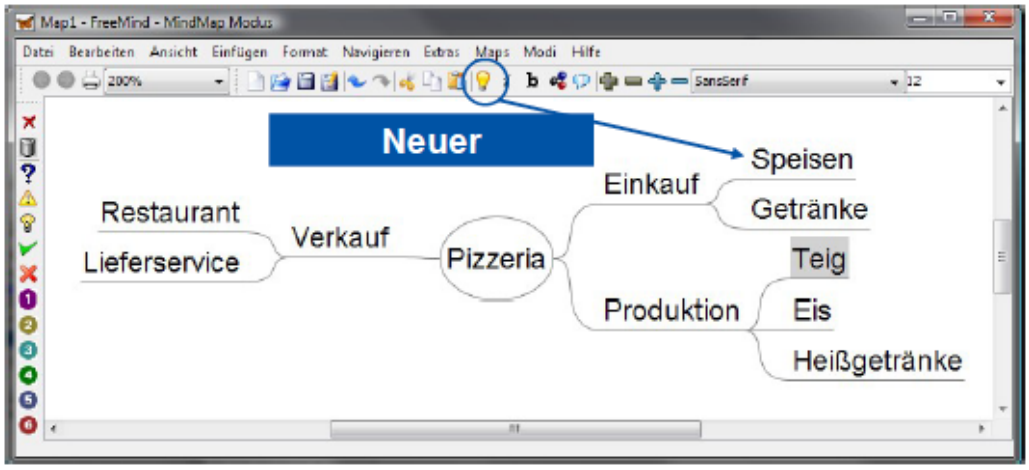
1.6. +Mindmaps

Bevor wir uns die Diagramme im Detail ansehen, wollen wir noch ein Produkt kennenlernen, das sehr oft in der ERST-Analyse eingesetzt wird.

In einer Mindmap werden Informationen gesammelt, geordnet und strukturiert.

Freemind ist ein freies Werkzeug http://freemind.sourceforge.net/wiki/index.php/Main_Page

Hier nun ein Erstentwurf einer Pizzeria mit Restaurant und Lieferservice



Häufig werden Mindmaps auch in Kleingruppen auf Flipcharts gezeichnet.

1.6.1. Aufgabe: Mindmap DA

Erstellen Sie ein Mindmap für ihre DA und fügen Sie diese inklusive einer kurzen Beschreibung in ihre DA ein.

1.7. +Use-Case (OOA)

Soviel gibt es über die Use-Cases gar nicht zu erzählen - sie sind schnell erklärt, aber ungemein nützlich. Die Use-Cases wurden von Ivar Jacobson eingeführt.

Use-Case Diagramme beschreiben das

Zusammenwirken von Personen mit einem System.

Also

DAS TUN der BETEILIGTEN

Unter einem Use-Case wird **eine typische Handlung** verstanden, die ein Benutzer mit dem System ausführt, z. B. "Aktienkauf".

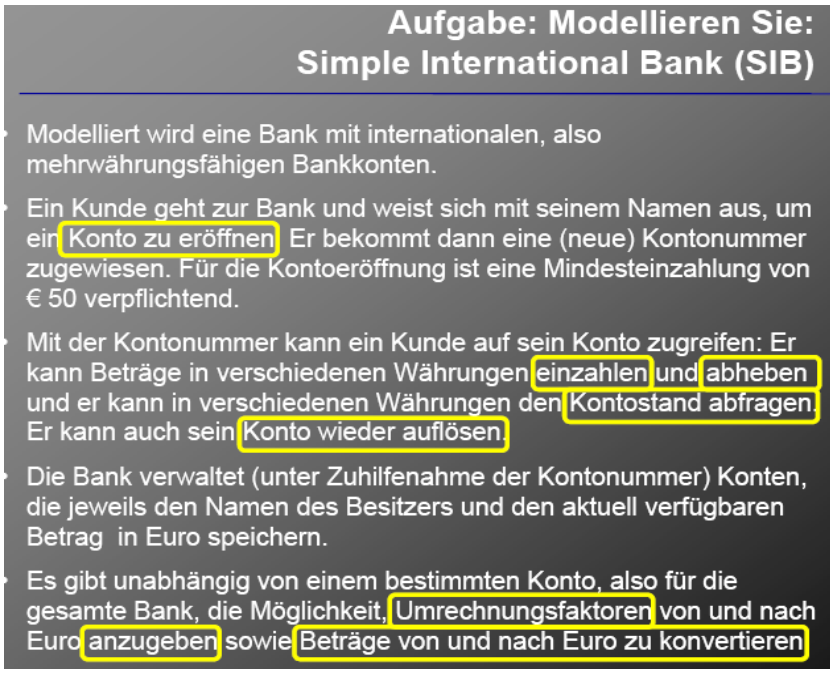
☑ Vorgehen: 3 Schritte

1. Schritt 1: Typisches Scenario erstellen:

   Tipp: **Einfache Sätze** beschreiben **DAS TUN der BETEILIGTEN**
   Tipp: Weniger als 9 Sätze

2. Schritt 2: Glossar erstellen
3. Schritt 3: Use-Case Diagramm erstellen

1.7.1. Beispiel: TYPISCHES SCENARIO für Simple International Bank



1.7.2. Das GLOSSAR erstellen

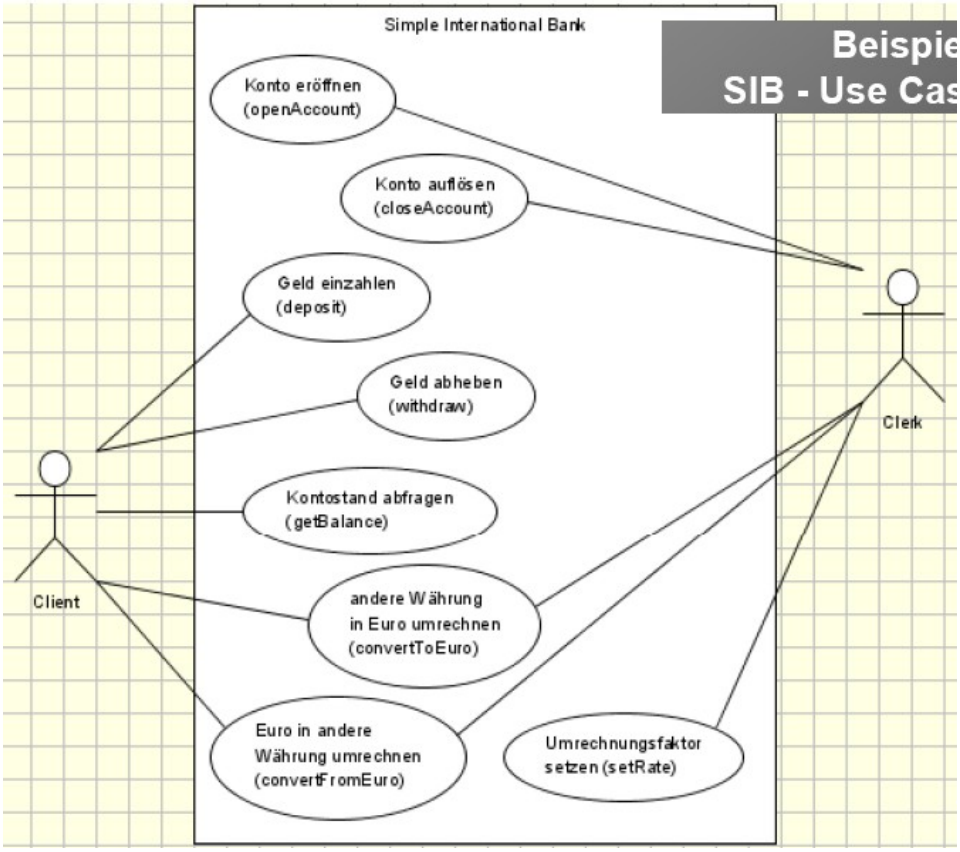
Während der Beschreibung der Anforderungen ist es sinnvoll, ein Glossar mit **Definitionen** und **Erläuterungen** zu den wichtigsten, in den Use-Case-Scenarien verwendeten Begriffen aufzustellen. Solch ein Glossar hilft allen Projektbeteiligten beim Verstehen der Use-Cases und vermeidet von Anfang an Missverständnisse der Art: "Ich meinte damit aber eigentlich ..."

Zwei Beispieleinträge im Glossar für ein Bibliothekssystem:

- **Benutzer-ID**
  ist eine ganze Zahl, die der eindeutigen Identifizierung eines Bibliotheksbenutzers dient. Das System vergibt die Benutzer-ID für neue Benutzer automatisch.

- **Rückgabedatum**
  ist das Datum, bis zu dem ein Bibliotheksbenutzer ein ausgeliehenes Buch spätestens zurückgeben muss. Eine Verlängerung der Ausleihe ist nicht möglich.

1.7.3. Beispiel: USE-CASE-DIAGRAMM für Simple International Bank



Personen werden als **Aktoren** bezeichnet. Aktoren können aber auch andere Systeme sein.

In das Use-Case-Diagramm werden die **Use-Cases als Ellipsen** eingezeichnet.

Verbindungen zwischen den Use-Cases und den Aktoren werden durch Linien hergestellt. Damit wird angezeigt, welche Aktoren an dem entsprechenden Use-Case beteiligt sind.

1.7.4. Weitere Beispiele

☑ Ein Scenario

Use Case Name:	Schedule Customer Appointment
Iteration:	Finished
Summary:	Define an appointment between agent and customer
Basic course of events:	1. The office administrator, using date/time parameters supplied by the customer, requests a list of available times when the agent and customer can have a consultation. 2. The system responds with a list of available appointment times. 3. The office administrator picks one of the times based on the customer's preference. 4. The system records the appointment time and later notifies the agent of the upcoming appointment.
Alternative Paths:	In step 1, if the office administrator has more than one appointment to record, the system allows multiple appointment entry.

☑ Ein use-case-diagramm: online-shop

1.7.5. Aufgabe: use-case: DA

Erstellen Sie zumindest

☑ ein typisches Szenario und

☑ ein entsprechendes Use-Case Diagramm

aus Ihrer DA und fügen Sie dieses Ergebnis in ihre DA ein.

1.8. +Interaktionsdiagramme

Es gibt zwei Arten von Interaktionsdiagrammen:

☑ **Sequenzdiagramme und**

☑ **Kollaborationsdiagramme**.

Beide beschreiben die zeitlichen Abläufe, d. h. **Aufrufsequenzen**.

Ausgehend von Fallbeispielen - den Use-cases - können Interaktionsdiagramme erstellt werden. Beim Erstellen der Diagramme konzentriert man sich auf die "**wichtigsten" Anwendungsfälle.**

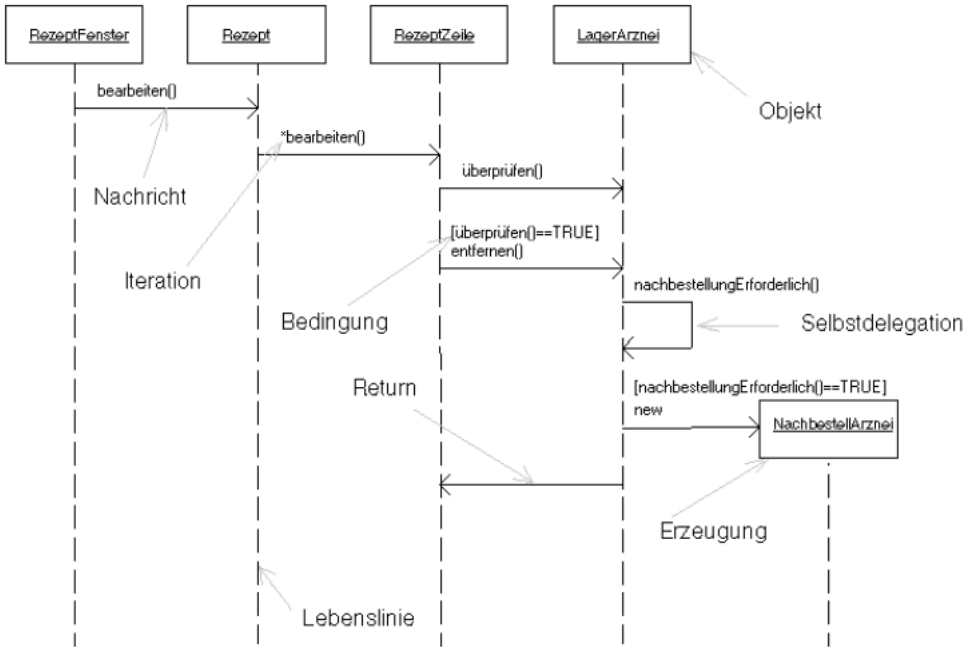
1.8.1. Sequenzdiagramme

Als Beispiel für die Erklärung der Interaktionsdiagramme soll die Adler-Apotheke dienen. In dieser Apotheke werden die Rezepte der Kranken in einen Computer eingegeben. Die Software ist wie folgt aufgebaut:

- In einem Eingabefenster RezeptEingabe werden die Rezepte eingegeben. Dieses Fenster sendet die **Nachricht bearbeiten an das Rezept**.
- Das Rezept sendet die **Nachricht bearbeiten an jede RezeptZeile** von Rezept.
- Jede RezeptZeile überprüft das Objekt LagerArznei wie folgt: Der Rückgabewert TRUE bedeutet, dass die RezeptZeile die entsprechende Menge der LagerArznei aus dem Lager entfernt hat. Anderenfalls ist die Menge der LagerArznei unter den Mindestbestand für eine Nachlieferung gefallen, und eine Nachlieferung dieser LagerArznei wurde veranlasst.

Eine Darstellung als Sequenzdiagramm

Sequenzdiagramm "Adler Apotheke"



☑ Die **Zeitachse** verläuft von oben nach unten.

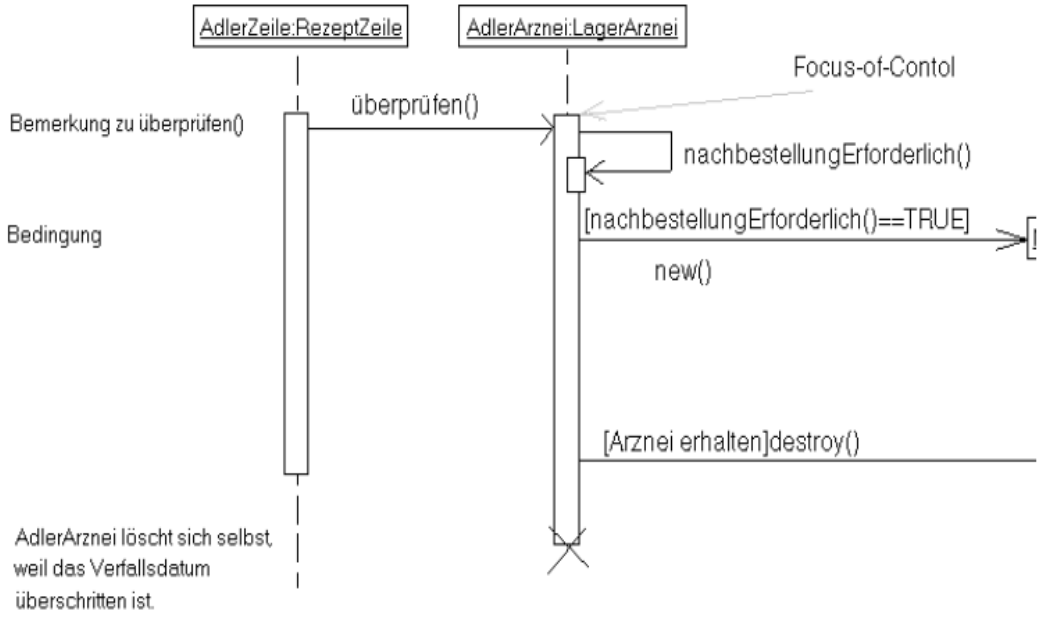
☑ **Objekte sind als Rechtecke** am oberen Ende von gestrichelten Linien - den "Lebenslinien" - dargestellt.

☑ Zwischen den Lebenslinien wird der **Austausch von Nachrichten durch Pfeile** dargestellt.

Eine Nachricht kann an eine **Bedingung** geknüpft sein, die **in eckigen Klammern** angegeben wird. Optional können noch Rücksprünge eingetragen werden - im allgemeinen verschlechtern diese aber die Lesbarkeit des Diagramms. Einige andere Details des Diagramms, wie etwa die Erzeugung von Objekten und die Iteration, erklären sich von selbst.

Die Sequenzdiagramme wurden - um ihre Aussagekraft zu erhöhen - um einige nützliche Notationselemente erweitert;

☑ + Sequenzdiagramm mit Focus-of-Control, Erzeugen/Löschen und Bedingungen



Der **Focus-of-Control (das langezogene Rechteck)** gibt an, wo eine Aktivität ausgeführt wird. Ist ein Objekt ohne Aktivität vorhanden, wird dies durch eine gestrichelte Linie angezeigt.

Objekte werden durch einen Pfeil auf das Rechteck erzeugt, ein Löschen wird durch ein Kreuz dargestellt.

Über den Pfeilen stehen die **Operationen, die ausgeführt werden**.

In eckigen Klammern können **Bedingungen** angegeben werden, ob die Operation ausgeführt wird. Ruft das Objekt eine Operation von sich selbst auf, etwa nachbestellungErforderlich(), dann zeigt der Pfeil auf die Lebenslinie zurück. Zusätzlich können am linken Rand noch Bemerkungen zu den einzelnen Operationen angegeben werden.

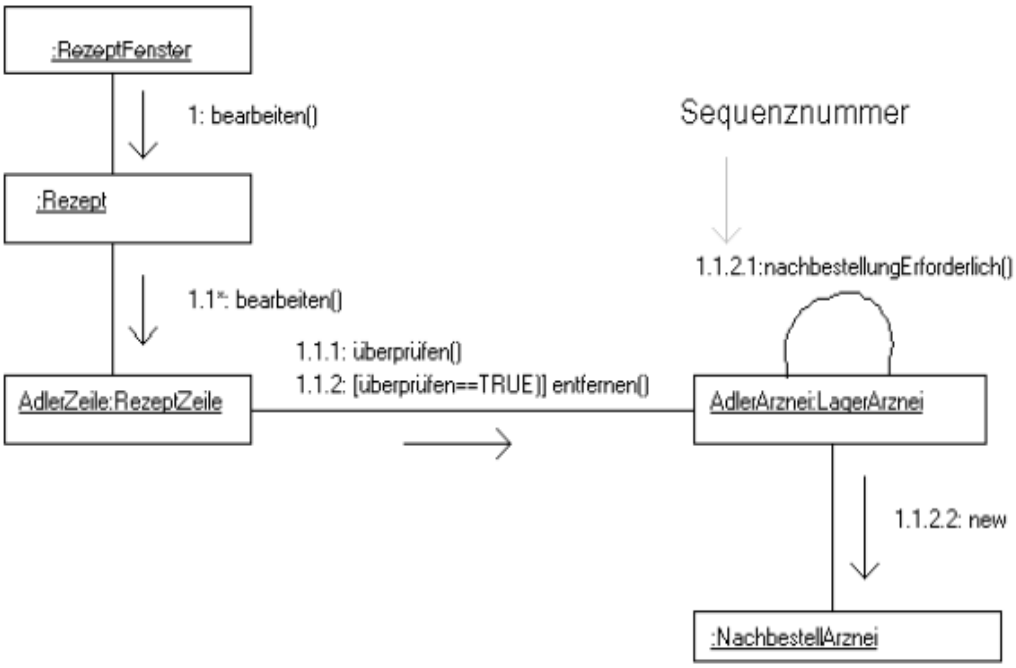
1.8.2. Kollaborationsdiagramm

Das Kollaborationsdiagramm und das Sequenzdiagramm beinhalten die gleichen Informationen und unterscheiden sich lediglich in der Darstellung. Die automatische Umwandlung eines Sequenzdiagramms in ein Kollaborationsdiagramm und umgekehrt ist möglich.

Bei vielen Klassen und wenigen Nachrichten sind Kollaborationsdiagramme übersichtlicher als Sequenzdiagramme sind. Sind wenige Klassen und viele Nachrichten vorhanden, so ist das Sequenzdiagramm besser geeignet.

Ausgehend von unserem Beispiel, der Adler-Apotheke, ergibt sich für das vorher vorgestellte Beispiel der Bearbeitung von Rezepten das Kollaborationsdiagramm.

Kollobarationsdiagramm für "Adler Apotheken Rezepte"



Das Kollaborationsdiagramm ist nicht nur ein wichtiges Hilfsmittel beim Entwurf und der Analyse, es hat auch schon oft bei der Fehlersuche geholfen. Während der Fehlersuche zeichnet man ein Kollaborationsdiagramm - damit läßt sich die Aufrufstruktur und das Zustandekommen eines Fehlers gut verfolgen.

1.9. Zustandsdiagramm

Aus den Interaktionsdiagrammen können Zustandsdiagramme entwickelt werden.

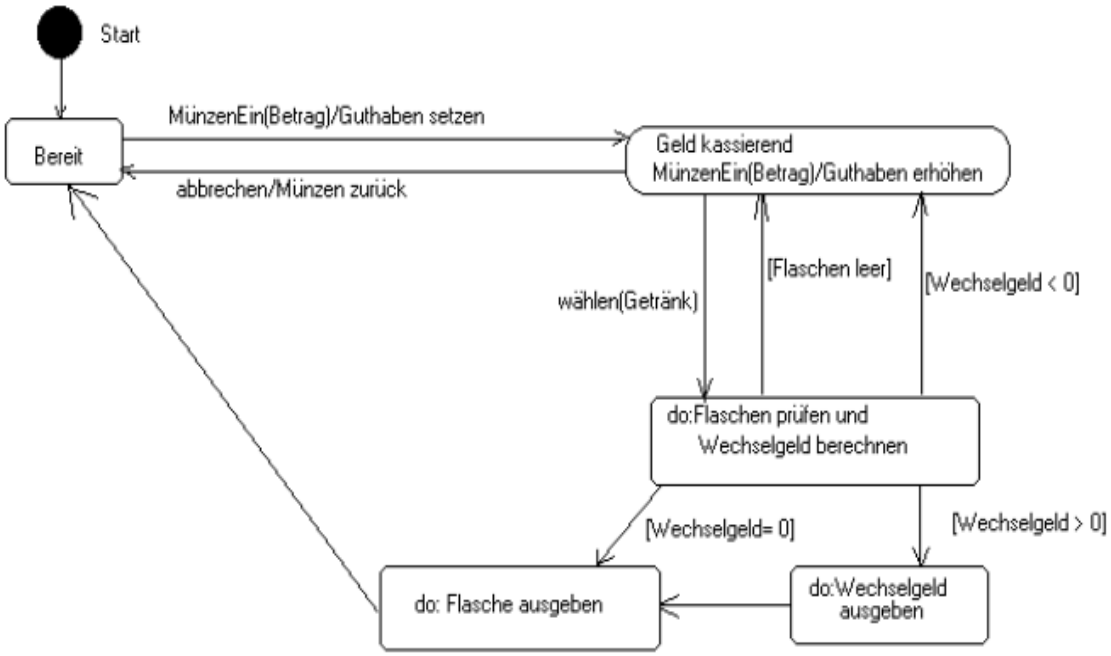
Nur für Klassen mit interessantem Verhalten werden Zustandsdiagramme angefertigt.

Ein interessantes Verhalten weisen unter anderem all diejenigen Klassen auf, deren reale Objekte in der Umgangssprache mit "**Automat**" bezeichnet werden, wie z. B. Geldautomaten, Getränkeautomaten, automatische Aufzüge oder automatische Garagenöffner.

Weitere Klassen mit interessantem Verhalten sind Klassen, die **Übertragungsprotokolle** realisieren, sowie Klassen, die **Benutzeraktionen** abarbeiten (z. B. Kennworteingabe).

Im folgenden soll das Beispiel eines Getränkeautomaten, aus dem verschiedene Getränke entnommen werden können, behandelt werden

1.9.1. Zustandsdiagramm für einen Getränkeautomaten



Rechtecke mit abgerundeten Ecken stellen die **Zustände** dar; zwischen jedem Nachrichtenaustausch im Sequenzdiagramm eines Objekts kann ein Zustand liegen.

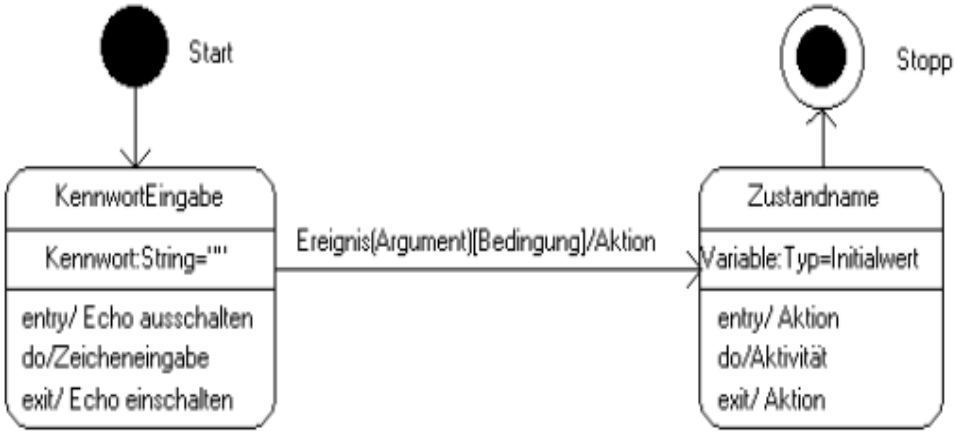
Durch das Eintreffen von Ereignissen kann ein anderer Zustand erreicht werden, was durch die **Pfeile** angedeutet wird, die eine Beschriftung für das auslösende Ereignis haben. Die Pfeile stellen die Übergänge dar.

In den Zuständen werden **Operationen** mit einer gewissen Zeitdauer ausgeführt, die **Aktivitäten** genannt werden.

Im Gegensatz dazu wird die Zeitdauer von Aktionen mit Null angenommen. **Aktionen** sind Operationen, die an den Übergängen ausgeführt werden. Angezeigt werden die Aktionen durch einen **Schrägstrich vor dem Namen**. Selbstverständlich können Übergänge auch an Bedingungen, die wieder in eckigen Klammern stehen, geknüpft werden.

Aus meiner Sicht (und auch nach [You96]) ist die Unterscheidung zwischen Aktivitäten und Aktionen akademisch. Jede reale Operation hat eine Zeitdauer - ob diese für eine Aufgabe relevant ist, hängt vom Einzelfall und nicht von der Darstellung in einem Diagramm ab. Den allgemeinen Aufbau eines Zustandsdiagramms sehen Sie hier.

1.9.2. Zusammenfassung: Zustandsdiagramm Symbole



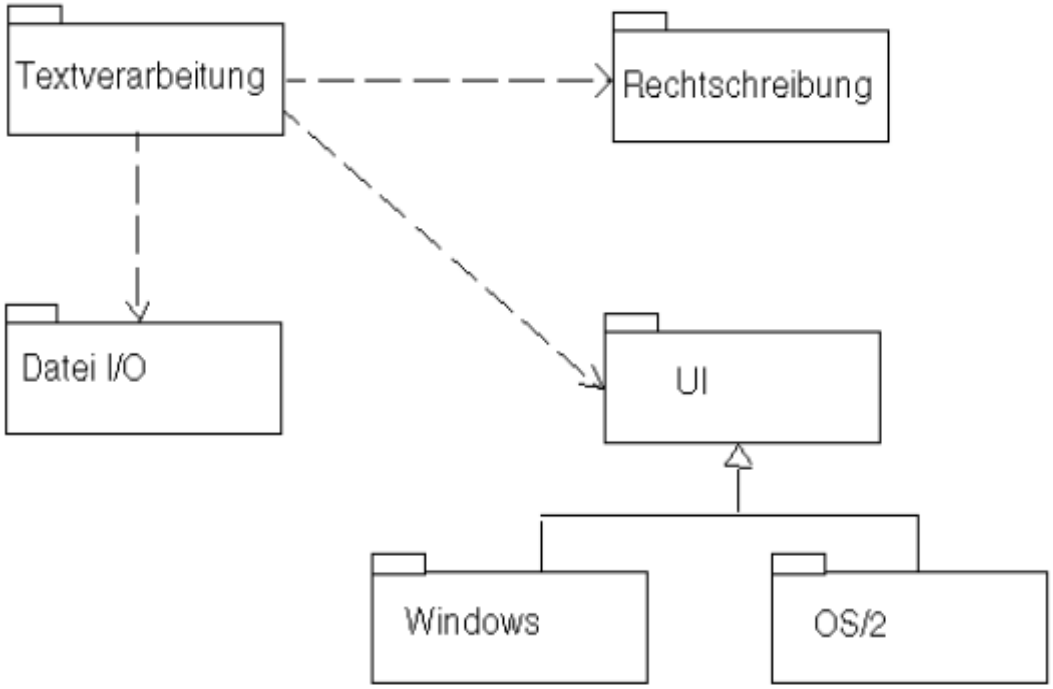
Die Wörter entry, do und exit sind reserviert und können als Bezeichnungen für Ereignisse nicht verwendet werden. entry gibt an, welche Aktion beim Eintritt in einen Zustand ausgeführt wird. Ähnlich bezeichnet exit die Aktion, die beim Verlassen des Zustandes ausgeführt wird. Und schließlich beschreibt do den Aufruf eines verschachtelten Zustandsdiagramms. Allgemein bezeichnet do eine andauernde Aktivität, die unendlich lange andauern kann.

Zustandsdiagramme für die Formatprüfung von Benutzereingaben, z. B. die Uhrzeit, können sehr gut direkt gezeichnet werden. Aber auch Übertragungsprotokolle werden häufig über Zustandsdiagramme definiert. Nicht immer läßt sich ein Zustandsdiagramm direkt zeichnen, dann sind vorher geeignete Szenarios und gegebenenfalls Interaktionsdiagramme zu erstellen.

1.10. +Package-Diagramm

Package-Diagramme dienen der **Strukturierung** der verschiedenen Darstellungen. Damit werden **Gruppen von Diagrammen oder Elementen zusammengefaßt**. Dies ist insbesondere wichtig für den Überblick über das System und die Aufteilung in verschiedene Übersetzungseinheiten.

1.10.1. Beispiel für ein Package-Diagramm



Ein Package-Diagramm besteht im wesentlichen aus **Packages** (dargestellt durch große Rechtecke mit kleinem Rechteck links oben) und **Abhängigkeiten** (den gestrichelten Pfeilen).

Eine Abhängigkeit gibt an, dass bei einer Änderung des Packages an der Pfeilspitze das Package am anderen Ende der gestrichelten Linie eventuell geändert bzw. neu übersetzt werden muß. Ob eine Änderung bzw. Neuübersetzung erforderlich ist, muß im Einzelfall geprüft werden. Packages, die nicht durch eine Abhängigkeit verbunden sind, können bei der Änderung außer acht gelassen werden. Packages können weitere Packages enthalten.

Packages geben zum einen einen guten Überblick über die Auswirkung von Änderungen und ermöglichen zum anderen einen Überblick über das Gesamtsystem. Insbesondere für größere Systeme ist eine solche Strukturierung wichtig.

In der obigen Abbildung muß die Applikation Textverarbeitung auf Änderungen überprüft werden, wenn sich das Package Datei I/O, UI oder Rechtschreibung ändert. Das Package UI soll hier eine abstrakte Klasse darstellen, die lediglich als Schnittstelle dient. Damit ist es möglich, daß Änderungen in den Packages Windows und OS/2 keine Auswirkungen auf das Package Textverarbeitung haben. Aus diesem Grund ist hier eine Generalisierung eingezeichnet (vgl. Klassendiagramme).

1.11. +Aktivitätsdiagramm

Grundelemente dieser Diagrammart sind:

- Aktivitäten,
- Transitionen,
- Synchronisationslinien und
- optionale "Swimlanes" (Schwimmbahnen).

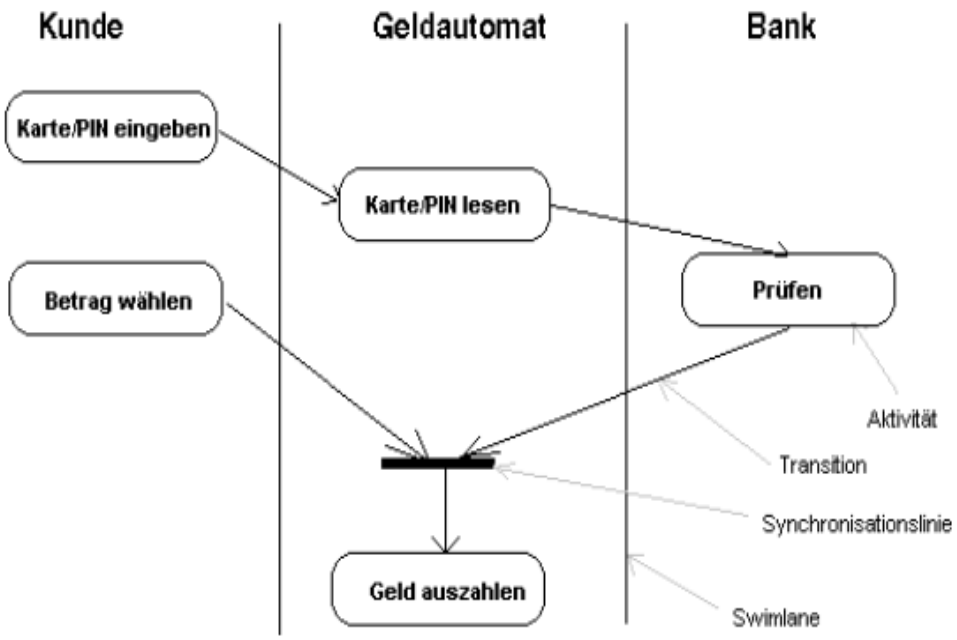
Aktivitäten sind Zustände, in denen Vorgänge ablaufen.

Transitionen erfolgen automatisch **am Ende der Aktivitäten**; sie werden durch Pfeile dargestellt.

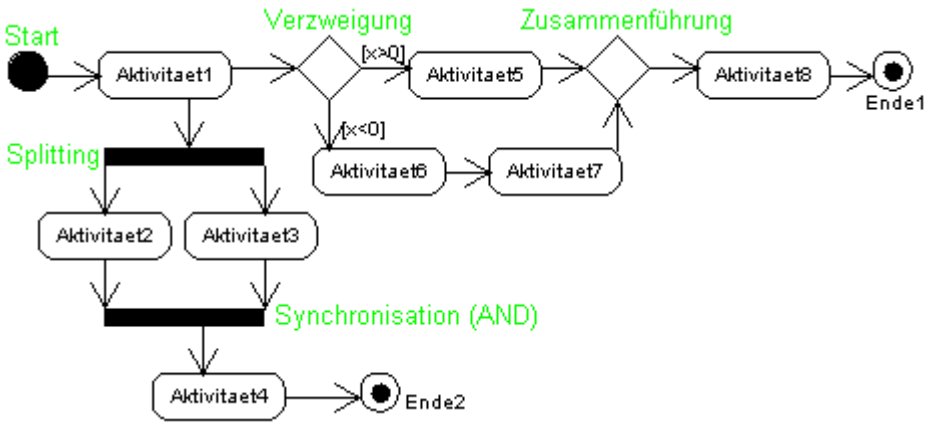
Synchronisationslinien/Pfeile werden durch Striche dargestellt und **schalten, wenn alle Eingangstransitionen vorhanden** sind.

"Linien" teilen ein Aktivitätsdiagramm so ein, dass die Bereiche, die sie abgrenzen, einzelnen Klassen zugeordnet werden können.

1.11.1. Beispiel für ein Aktivitätsdiagramm



1.11.2. Zusammenfassung: Aktivitätsdiagramm-Symbole



1.12. +Implementierungsdiagramme

Implementierungsdiagramme zeigen Aspekte der Implementierung. Diese umfassen die Codestruktur und die Struktur des Systems zur Ausführungszeit. Insbesondere für verteilte Anwendungen und Komponentensoftware ist das wichtig. Von den Implementierungsdiagrammen gibt es zwei Formen:

☑ Komponentendiagramme und
☑ Deployment-Diagramme.
Komponentendiagramme zeigen die Struktur des Codes und Deployment-Diagramme die des Laufzeitsystems.

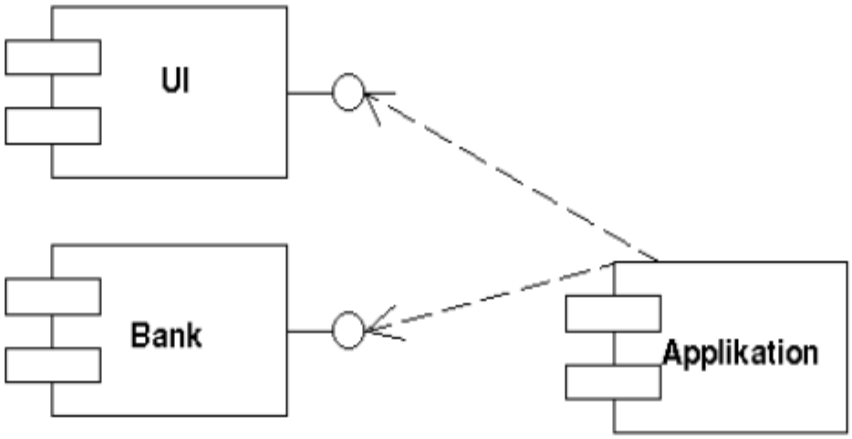
Mit zunehmender Bedeutung der verteilten Anwendungen und Komponentensoftware werden diese Diagramme an Bedeutung gewinnen.

1.12.1. +Komponentendiagramm

Das Komponentendiagramm zeigt die **Abhängigkeiten unter den Softwarekomponenten**, genauer die Abhängigkeiten zwischen Quellcode, Binärcodekomponenten und ausführbaren Programmen.

Einige dieser Komponenten existieren nur während des Übersetzungsvorgangs, einige nur während des Linkens, andere zur Ausführungszeit und wieder andere die ganze Zeit über. Im Komponentendiagramm haben die Darstellungen nur Typencharakter, im Gegensatz zu den Deployment-Diagramm, wo sie zu Instanzen werden (d. h. die Bezeichnungen werden unterstrichen).

Beispiel für ein Komponentendiagramm



Die Komponenten werden **als drei ineinander verschachtelte Rechtecke** gezeichnet;

ihre **Schnittstellen sind Striche mit Kreisen am Ende**. Dadurch können die verschiedenen Schnittstellen der Komponenten dargestellt werden.

Das Diagramm enthält ferner **Abhängigkeiten** in Form von **gestrichelten Pfeilen**.

Sie werden sich fragen, wo der Unterschied zu den Package-Diagrammen liegt. Der Unterschied besteht darin, daß die Package-Diagramme ein allgemeines Mittel für die Strukturierung darstellen, während die **Komponentendiagramme Implementierungsaspekte zeigen**.

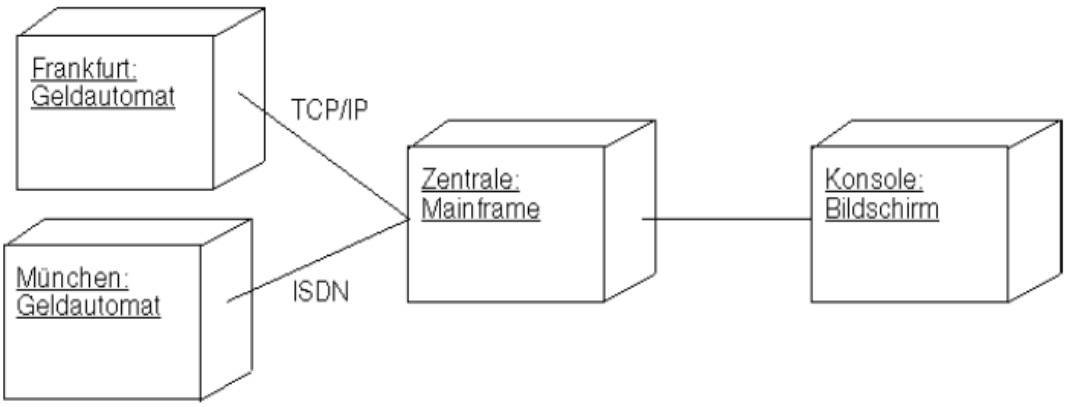
In der Praxis entspricht eine Komponente einem Package, aber nicht jedes Package einer Komponente.

Die Abhängigkeiten eines Package-Diagramms, das in ein Komponentendiagramm umgewandelt werden kann, sind genau die gleichen wie die im Komponentendiagramm.

1.12.2. +Deployment-Diagramme

Zur **Darstellung der Hardware** werden Deployment-Diagramme verwendet. Diese besitzen sogenannte Knoten, die eine Verarbeitungs- oder Hardwareeinheit darstellen. Knoten werden als Quader gezeichnet. Unter den Knoten existieren Verbindungen: Dabei handelt es sich um die physikalischen Kommunikationspfade, die als Linien eingezeichnet werden

Beispiel für ein Deployment-Diagramm



In das Deployment-Diagramm können noch die Komponenten und ihre Abhängigkeiten

eingezeichnet werden. Somit erhalten Sie einen Überblick, wo die Komponenten ausgeführt werden.

1.13. +Stereotypen

Stereotypen sind Mechanismen, die es erlauben, die **UML zu erweitern**.

Der Begriff wird nur selten ins Deutsche übersetzt, und überhaupt sind Erklärungen dafür recht selten, obwohl der Begriff "Stereotyp" häufig verwendet wird. Übersetzungen aus dem Lexikon ergeben Begriffe wie "Druckvorlage", "Muster", "Musterbeispiel", "Vorbild". Stereotypen haben in der UML die Bedeutung "so ähnlich etwa wie". Stereotypen können eigene Piktogramme (Icons) haben und werden in eckigen Klammern dargestellt, z. B. <<uses>>. Sie erlauben eine weitere Einteilung der Klassen, Abhängigkeiten, Assoziationen etc. So ist z. B. eine Einteilung der Klassen in Schnittstellen-, Kontroll- und Entitätenobjekte (irgendwelche Dinge) möglich.

1.14. Literatur

[**Boo94**] G. Booch, Object-oriented Analysis and Design with Applikations, 1994 Benjamin/Cummings Publishing Company, 1994

[**Fow97**] M. Fowler, UML Distilled, Applying the Standard Object Modeling Language, 2nd Printing, Addison-Wesley, 1997

[**Jac92**] I. Jacobson, Object-Oriented Software Engineering, A Use Case driven Approach, Addison-Wesley 1992

[**Rat97**] UML Unified Modeling Language, Version 1.0 und 1.1 (siehe: http://www.rational.com)

[**Rum91**] J. Rumbaugh, Objektorientiertes Modellieren und Entwerfen, Hanser Verlag 1993 (Titel der amerikanischen Ausgabe: Object-Oriented Modeling and Design, Prentice Hall, 1991).

[**You96**] E. Yourdon et. al, Mainstream Objects, Prentice Hall, 1996

Aus:

Dipl. Inform. Günter Wahl (E-Mail-Adresse: GWahl@t-online.de) ist Softwareingenieur bei der Firma Robert Bosch, wo er bereits zahlreiche Projekte betreut und realisiert hat. Sein Schwerpunkt liegt auf dem Gebiet der objektorientierten Methoden und Programmierung.

1.15. Weitere Beispiele

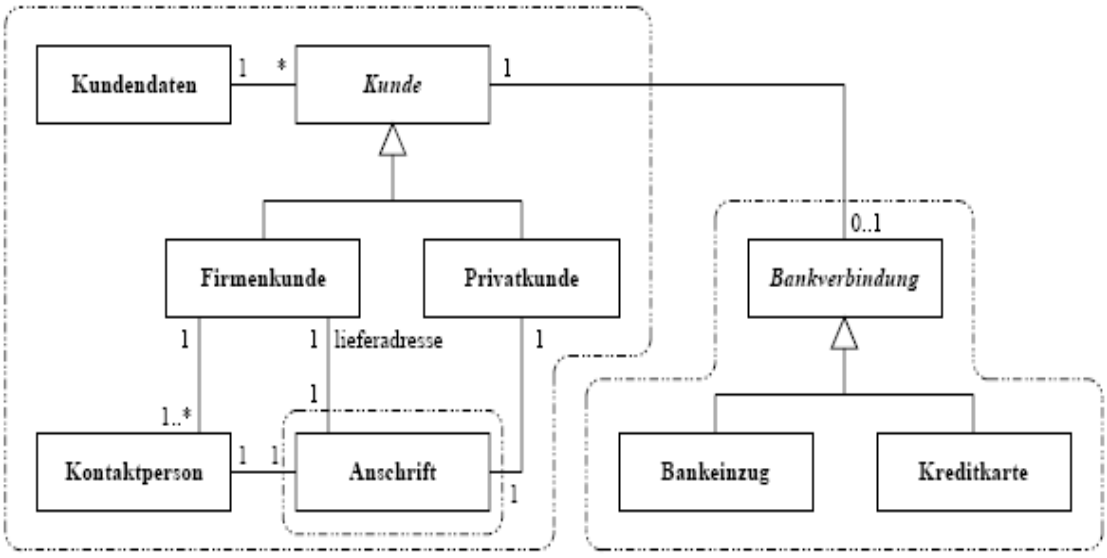


Abbildung 4.5: Erweitertes Fachklassendiagramm zu den Kunden

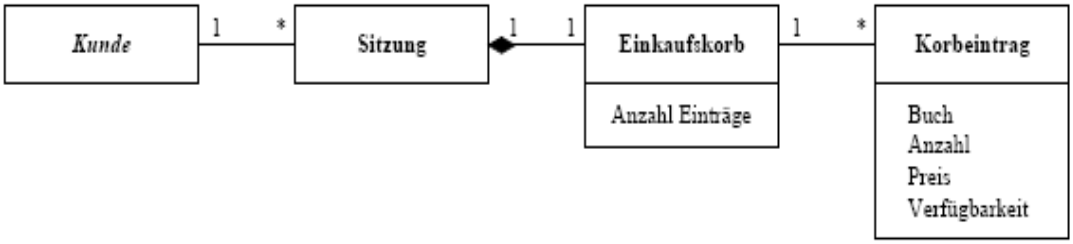


Abbildung 4.7: Verfeinerung des Fachklassendiagramms zum Einkaufskorb mit einer Kompositionsbeziehung

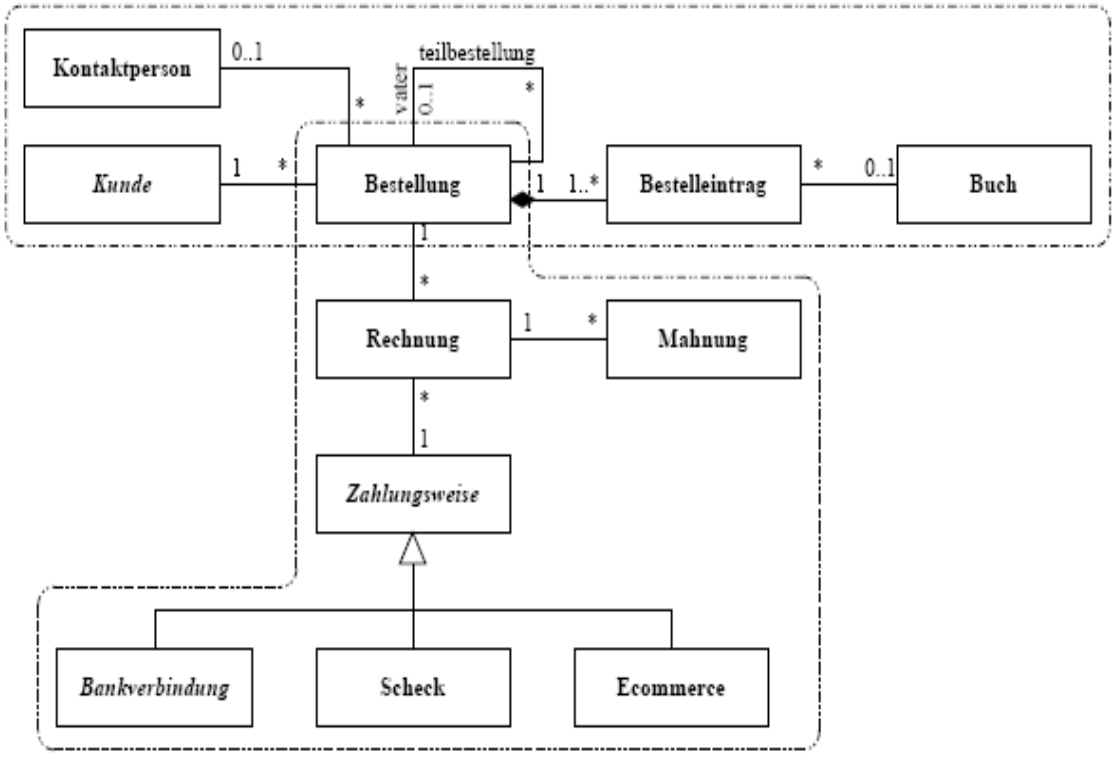


Abbildung 4.11: 4. Fachklassendiagramm zur Bestellung

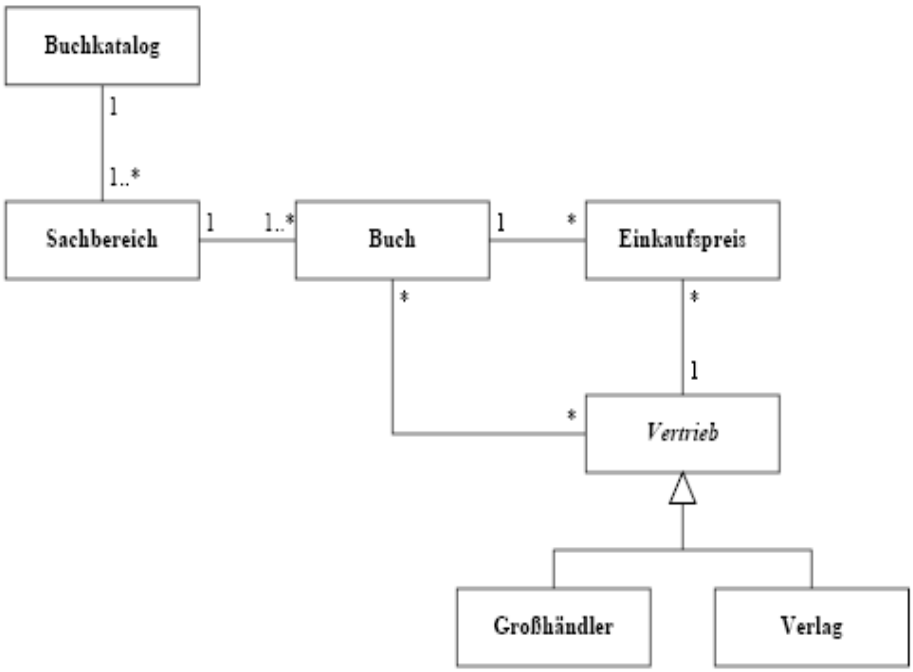


Abbildung 4.15: 4. Fachklassendiagramm zum Buchkatalog

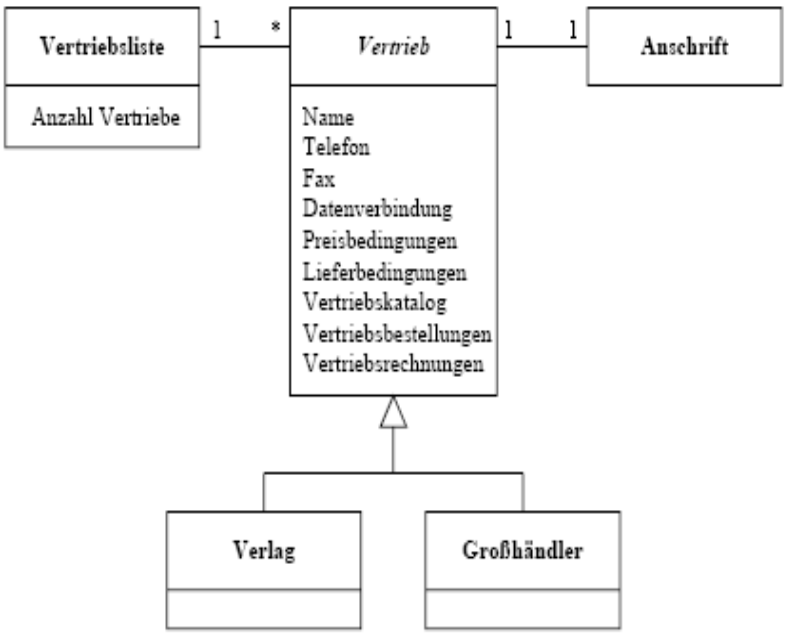


Abbildung 4.17: 2. Fachklassendiagramm zum Vertrieb

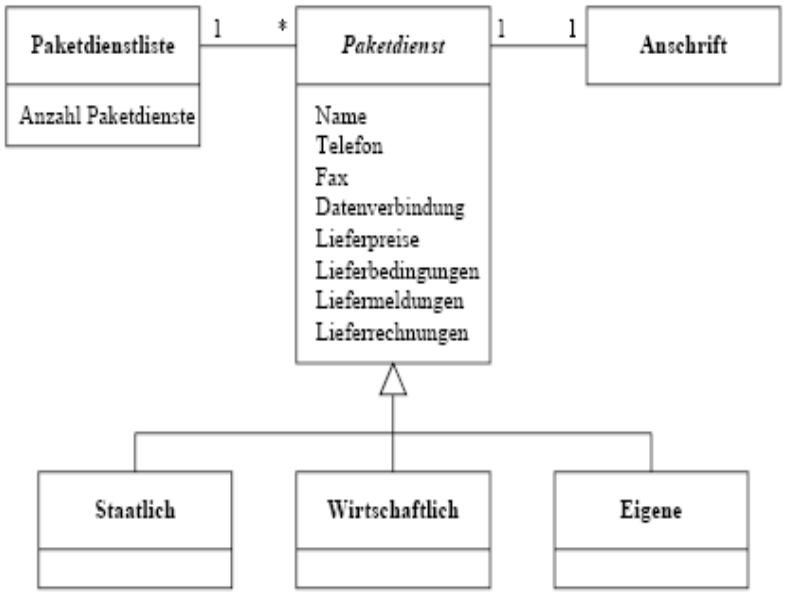


Abbildung 4.18: Fachklassendiagramm zu den Paketdiensten

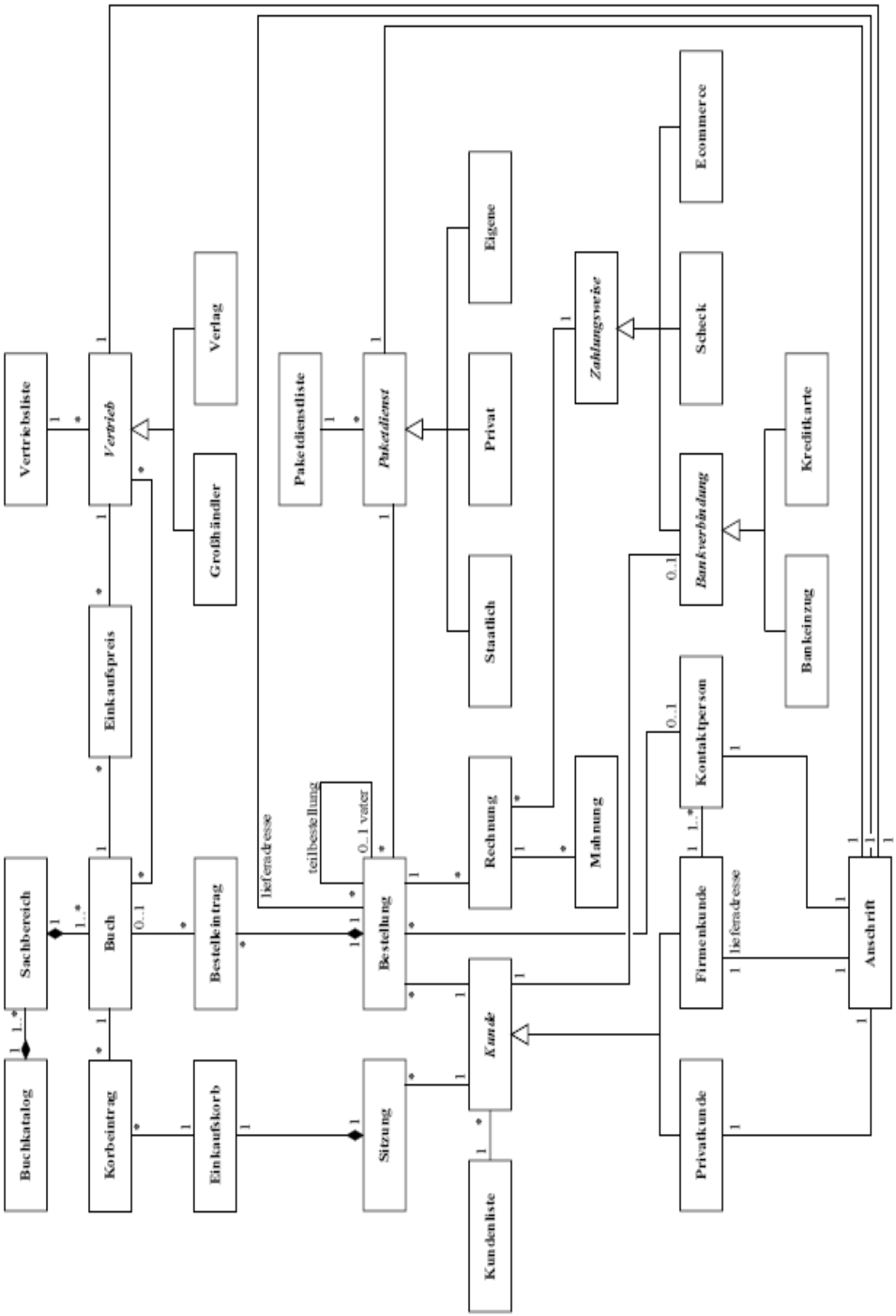


Abbildung 4.24: Gesamt–Fachklassendiagramm zum Internet–Bookshop