**Pool III: Methoden für die Softwareentwicklung (UML & Patterns)**

**Frage 1: Modellierung von Softwaresystemen (statische Sicht)**

* Ausgangssituation: Firma; Sie arbeiten im Bereich der Datenmodellierung. Aufgabe: Verwaltungssystem für Gebäudereinigungsfirma konzipieren.

Fragen:

1. Geben Sie einen Überblick über alle Ihnen bekannten UML-Diagramme zur Darstellung der statischen Sicht (Strukturdiagramme) (4) (formale Notation)
2. Erstellen Sie für den am Beiblatt beschriebenen Sachverhalt ein Domain Model.
3. Klassendiagramme werden in verschiedenen Phasen der Softwareentwicklung eingesetzt. Beschreiben Sie den Einsatz am Beispiel des RUP in den Workflows „Business Modelling“ und „Analysis & Design“.

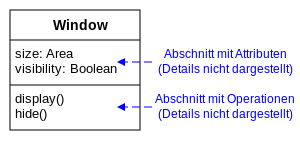
Antworten:

a)

Folgende statische Modellierungsdiagramme sind vorhanden:

* Class Diagram (Klassendiagramm)
* Object Diagram (Objektdiagramm)
* Package Diagram (Paketdiagramm)
* Component Diagram (Komponentendiagramm)
* Composite Strucure Diagram (Kompositionsdiagramm)
* Deployment Diagram (Verteilungsdiagramm)

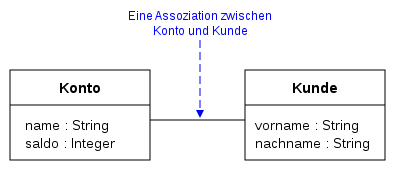
**Class Diagram**



### **Assoziation**

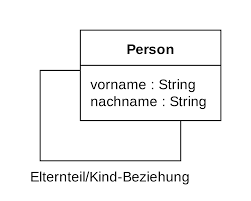
Eine Assoziation modelliert eine Verbindung zwischen zwei Klassen. Jene zwei Klassen verbindet man mittels einer Linie, wobei zu beachten ist, dass man der Assoziation (Linie) einen Namen zuweist. Der Name beschreibt die Assoziation und sollte idealerweise aus einem einzigen Verb bestehen. Außerdem sind an beiden Enden der Linie sogenannte Kardinalitäten hinzuzufügen.

#### Binäre Assoziation



Verbindung zwischen genau zwei Klassen.

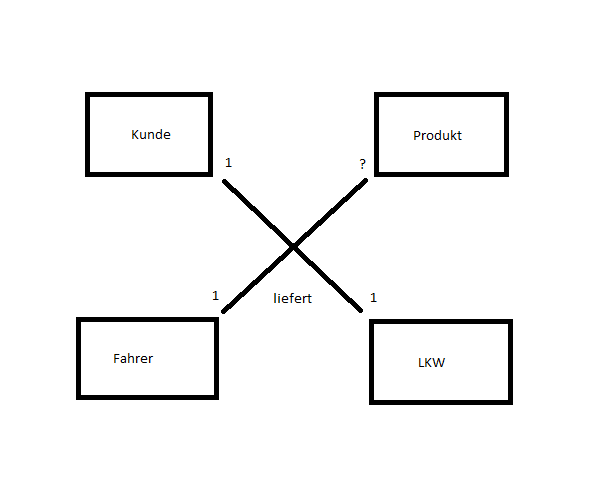
#### Unäre Assoziation



Die Assoziation ist rekursiv und anhand des Namens erkennt man, dass eine Person mit einer anderen Person verbunden sein kann, weil man ja möglicherweise die Eltern oder das Kind modellieren möchte.

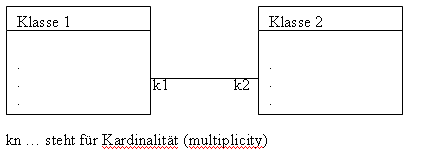
#### N-äre Assoziation

Bei dieser Assoziationsvariante muss es mehr als zwei beteiligte Klassen geben. Man beschreibt alle Klassen mit der Kardinalität 1 bis auf eine einzige, um die Kardinalität der verbliebenen Klasse herauszufinden. Dieses Szenario wiederholt man solange, bis jede Klasse einmal an der Reihe war. Anschließend fasst man die Ergebnisse in einem Klassendiagramm zusammen, d.h. man schreibt zu jeder Klasse die Kardinalität, die man vorher herausgefunden hat.



In diesem Fall lautet die dazugehörige Fragestellung: 1 Fahrer liefert mit 1 LKW an 1 Kunden wie viele Produkte?

#### Multiplicity



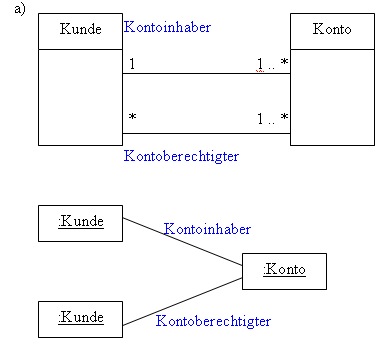
Diese steht an, wie viele Objekte die Klasse 1 von der Klasse 2 besitzen kann bzw. umgekehrt.

Beispiele:

* 1 => Objekt1 kann nur ein Objekt von Objekt 2 besitzen
* 0…2 => 0 bis 2 Objekte
* \* => 0 bis n Objekte
* 2…\* => 2 bis n Objekte
* 3 => genau 3 Objekte

#### Rolle

Optional kann man noch Rollen vergeben. Sie kommen üblicherweise zum Einsatz, wenn es mehrere Assoziationen gibt und man diese beschreibt.



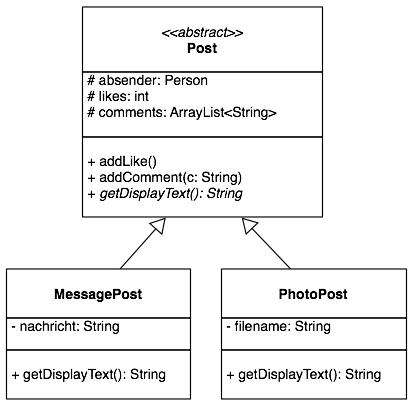
### **Geltungsbereich (Privacy)**

Der Geltungsbereich von Operationen und Attributen wird wie folgt gekennzeichnet:

* „+“ für [public](https://de.wikipedia.org/w/index.php?title=Public&action=edit&redlink=1) – unbeschränkter Zugriff
* „#“ für [protected](https://de.wikipedia.org/w/index.php?title=Protected&action=edit&redlink=1) – Zugriff nur von der Klasse sowie von Unterklassen (Klassen, die erben)
* „−“ für [private](https://de.wikipedia.org/w/index.php?title=Private_(Informationstechnik)&action=edit&redlink=1) – nur die Klasse selbst kann es sehen
* „~“ für [package](https://de.wikipedia.org/w/index.php?title=Package_(Sichtbarkeit)&action=edit&redlink=1) – innerhalb des Pakets sichtbar (nur in wenigen Programmiersprachen, etwa Java und C#, implementierbar)

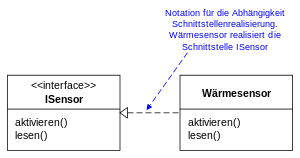
**Abstrakte Klassen**

Abstrakte Klassen besitzen normale Attribute und abstrakte Methoden, die mit dem Schlüsselwort „abstract“ gekennzeichnet sind. Allerdings müssen nicht alle Methoden in dieser Klasse abstrakt sein. Sobald mindestens eine Methode abstrakt ist, muss die ganze Klasse als solche deklariert sein. Man muss beachten, dass von abstrakten Klassen keine Instanz angelegt werden kann, weil nicht alle Methoden implementiert sind. Abstrakte Methode implementiert man üblicherweise in Subklassen. Zu jenen Klassen schreibt man oberhalb des Klassennamens das Wort „abstract“ in doppelter Spitzklammer. Alternativ kann der Klassenname auch kursiv geschrieben werden, wenn dies gut erkennbar ist.



**Interface**

Eine Schnittstelle wird gleich wie eine Klasse mit einem Rechteck dargestellt, zur Unterscheidung aber mit dem Schlüsselwort „interface“ in doppelter Spitzklammer gekennzeichnet. In dieser Klasse gibt es nur die Signaturen der Methoden. Implementiert werden diese in Subklassen, wobei dabei alle Methoden zu implementieren sind.



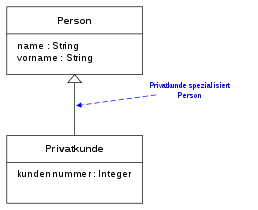
### **Generalisierung und Spezialisierung**

Eine Spezialisierung ist die Ableitung einer Subklasse von einer Superklasse, d.h. die Subklasse hat die Attribute und Methoden der Superklasse und eventuell noch weitere Elemente. Bei der Generalisierung lagert man gemeinsame Attribute und Methoden von mehreren Klassen in eine Superklasse aus und lässt die Subklassen davon erben.

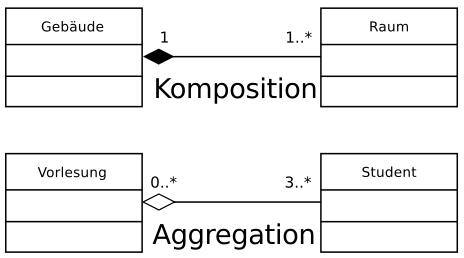
Eine Generalisierung/Spezialisierung in der UML ist eine gerichtete Beziehung zwischen einer generelleren und einer spezielleren Klasse. Attribute und Operationen der Mutterklasse sind somit auch in der Subklasse. Man sagt, dass die speziellere Klasse sie von der generelleren Klasse „erbt“ oder „ableitet“.

Eine Generalisierung wird als durchgezogene Linie zwischen den beiden beteiligten Klassen dargestellt. Auf der Seite der Mutterklasse zeichnet man eine geschlossene, nicht ausgefüllte Pfeilspitze. In gängigen objektorientierten Programmiersprachen entspricht dies dem Konzept der Vererbung.

Hilfestellung: Spezialisierte Klasse „is a“ Superklasse (z.B.: Privatkunde „is a“ Person).



**Komposition undAggregation**

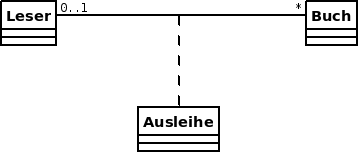
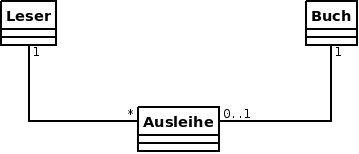


Komposition: Ein Raum kann nicht ohne Gebäude existieren.

Aggregation: Ein Student kann ohne Vorlesung existieren.

**Assoziationsklasse**

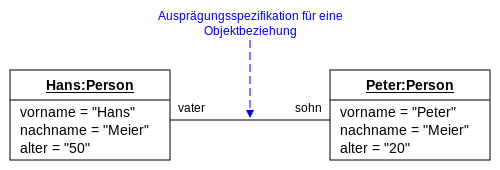
Eine Assoziationsklasse ist eine Klasse, deren Attribute und Operationen die [Assoziation](https://de.wikipedia.org/wiki/Assoziation_(UML)) zwischen anderen Klassen beschreibt.  Sie wird über eine gestrichelte Linie mit der Assoziation verbunden. Assoziationsklassen werden in Analysephasen von [Prozessmodellen](https://de.wikipedia.org/wiki/Vorgehensmodell_zur_Softwareentwicklung) verwendet und später zum Entwurf aufgelöst. Dazu werden entweder die Attribute (und Operationen) auf die Klassen der Assoziation verteilt, oder die Assoziation wird aufgeteilt, indem die Assoziationsklasse als eigenständige Klasse eingefügt wird.

Eine Assoziationsklasse Ausleihe kann hier genauere Informationen wie z. B. das Ausleihdatum speichern.

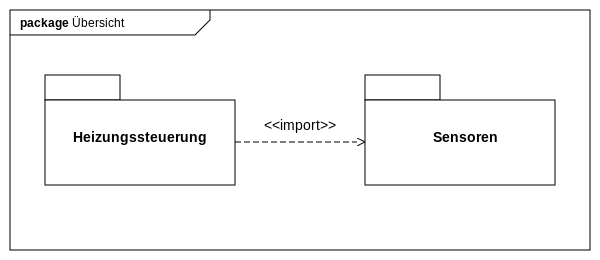
**Object Diagram**

Ein Objektdiagramm veranschaulicht den Zustand eines Systems zu einem bestimmten Zeitpunkt. Es enthält eine Menge relevanter Objekte, die Werte ihrer Attribute und die Beziehungen zwischen den Objekten.



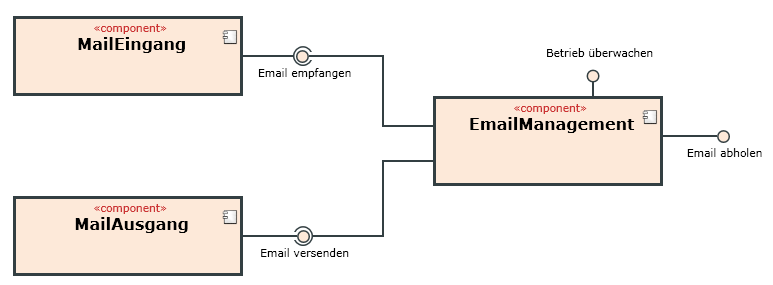
**Package Diagram**

Pakete erlauben es, verschiedene UML-Konstrukte zusammenzufassen und als eine zusammenhängende Einheit zu verstehen. Typischerweise werden unterschiedliche Klassen in einem Paket zusammengefasst. Es ist auch möglich, dass Pakete Bestandteil weiterer Pakete sind, sodass eine hierarchische Struktur zwischen Paketen gebildet wird. Das Paketdiagramm veranschaulicht die Pakete, die zur Beschreibung eines Systems benötigt werden.



**Component Diagram**

Komponentendiagramme beschreiben die physische Struktur eines Softwaresystems zur Entwicklungszeit. Die UML versteht unter Komponenten sowohl Quelltext- als auch binäre Dateien. Einfache Komponenten können zu komplexen Komponenten aggregiert werden, um Teil-Ganzes-Beziehungen abzubilden. Zwischen verschiedenen Komponenten können Abhängigkeiten spezifiziert werden, die besagen, dass eine Komponente sich auf eine andere Komponente abstützt.



**Composite Structure Diagram**

Das Kompositionsstrukturdiagramm verdeutlicht die Aufteilung einer Klasse in unterschiedliche Teile.

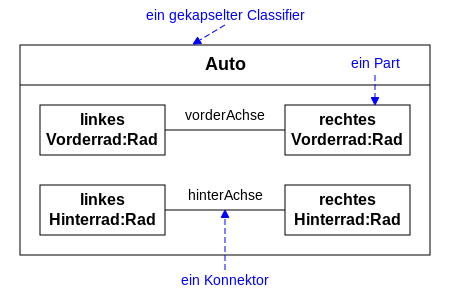
**Part**

Parts sind Bestandteile des Ganzen, die durch eine [Kompositionsbeziehung](https://de.wikipedia.org/wiki/Assoziation_(UML)#Aggregation_und_Komposition) zum Ganzen gehören bzw. vom Ganzen komponiert werden. Ein Part wird mithilfe eines Rechtecks dargestellt, der eine eigene Multiplizität besitzen kann. Sie werden vom Ganzen mit einem großen Rahmen umschlossen. Im Inneren des Rahmens zeigen die Parts ihre Beziehungen mithilfe von Konnektoren untereinander auf, nach außen hin werden über [Schnittstellen](https://de.wikipedia.org/wiki/Schnittstelle_(UML)) (Interfaces) die angebotenen und bereitgestellten [Merkmale](https://de.wikipedia.org/wiki/Merkmal_(UML)) (Features) dargestellt.

**Konnektor**

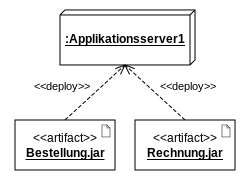
Ein Konnektor dient der Beschreibung eines Kommunikationspfades zwischen Instanzen. Instanzen die über einen Konnektor miteinander verbunden sind, haben so die Möglichkeit miteinander zu interagieren, d. h., sie „kennen“ sich zur Laufzeit. Primär werden Konnektoren eingesetzt, um die Interaktionen von Parts darzustellen.

Die Darstellung eines Konnektors erfolgt mit einer durchgezogenen einfachen Linie, identisch mit einer Assoziation. Wie auch Assoziationsenden besitzen Konnektoren ebenfalls Kardinalitäten. Diese können sich von der Kardinalitäten der Parts unterscheiden.



**Deployment Diagram**

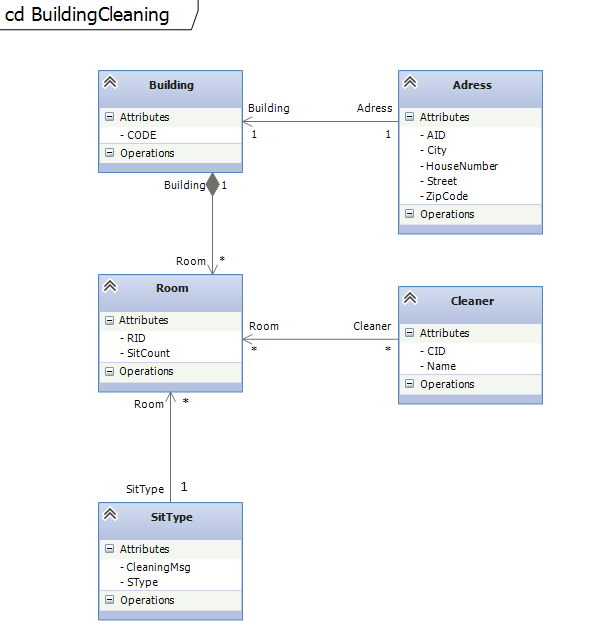
Während das Komponentendiagramm hauptsächlich die Abhängigkeiten zwischen Systemkomponenten zur Entwicklungszeit verdeutlicht, beschreiben Verteilungsdiagramme die Anordnung der Systemkomponenten auf unterschiedliche Rechenknoten zur Laufzeit. Die Darstellung umfasst dabei typischerweise Rechnerknoten, Komponenten, Artefakte, Ausprägungsspezifikationen, Verbindungen und Verteilungsbeziehungen.



b)

**Sachverhalt Gebäudereinigung:**

* Eine Reinigungsfirma hat in einer Stadt mehrere Gebäude zu reinigen.
* Jedes Gebäude hat einen Code und eine Adresse. In jedem Gebäude sind mehrere Konferenzräume zu reinigen.
* Für die Reinigung eines Raumes sind ein oder mehrere Pfleger zuständig.
* Ein Raumpfleger säubert mehrere Räume. Jeder Raum ist mit einer bestimmten Anzahl von Sitzgelegenheiten ausgestattet, die pro Raum vom gleichen Typ sind.
* Es sind mehrere Räume mit dem gleichen Sitztyp ausgestattet. Für jeden Sitztyp wird ein Pflegehinweis gegeben.
* Eine Adresse besteht aus einer Straßenbezeichnung, Hausnummer, Postleitzahl und Ort.



c)

Im Business Usecase Modelling wird das Domain Modell gezeichnet. (Struktur, Daten und ihre Beziehungen)

Im Workflow Requirements wird das Domain Model verfeinert.

In der Analyse wird das Domain Model erweitert um Boundary und Control Classes (Analyse Modell Diagramm / Entity Control Boundary-Pattern) --- warum? Fachlicge Logik – 3 Layer Konzept (Presentation L, Business Logic L, Data Layer)

**Frage 2: Dynamische Sicht**

* Ausgangssituation: Firma „OptiSoft“; Webanwendungen für Online-Geschäfte; Kunde möchte ganz speziellen Onlineshop. Alle Kundenanforderungen für die eigentliche Implementierung müssen geeignet dokumentiert werden.

Fragen:

1. Nennen Sie alle, Ihnen bekannte, UML-Diagramme zur Darstellung der dynamischen Sicht und beschreiben Sie diese kurz. (Verhaltensdiagramme) (4) (formale Notation)
2. Erstellen Sie unter Verwendung von geeigneten UML-Diagrammen ein Modell für den gewünschten Onlineshop. Dieser soll:
   * Auswählen eines Artikels,
   * Bestellung eines Artikels (inkl. Wahl der Bezahlungsmethode) und
   * Registrierung für Rechnungs- und Lieferadresse

beinhalten. Treffen Sie sinnvolle Annahmen für Ihre Anwendungsfälle.

1. Beschreiben und begründen Sie den Einsatz der UML-Diagramme zur Darstellung der dynamischen Sicht am Beispiel RUP für die ersten drei Workflows (BM, Requ. Und Analysis & Design). Überlegen Sie ebenfalls, wozu Zustandsdiagramme eingesetzt werden können.

Antworten:

a)

**Allgemein**

Dynamische UML Diagramme repräsentieren das Verhalten bzw. die Zustände der Prozesse in einem System, weshalb sie auch Verhaltensmodelle genannt werden.

**Welche UML Diagramme zur Darstellung gibt es?**

* Use Case Diagramme: Bündelt alle möglichen Szenarien die zum Erreichen eines Ziels führen können ( 2 Arten –1) Business Use Case Diagramme zur Beschreibung der Szenarien des Geschäftssystems -- Geschägtsprozesse --- 2) Use Case Diagramme zur Darstellung des gewünschten zukünftigen IT-Systems)
* Aktivitätsdiagramme: Dienen zur Beschreibung einer Ablauflogik (zeitlich) 🡪z.B. Beschreibung eines Use Cases oder Business Use Cases oder auch Use Case-übergreifend, aber nur im Business Modelling)
* Sequenzdiagramme: Dienen zur Beschreibung der Kommunikation zwischen Klassen/Objekten   
  🡪z.B. Realisierung der UseCases (fachl.Logik) od. Beschreibung der Programmlogik
* Zustandsdiagramme: Beschreiben die möglichen Zustände eines Objekts

**Use Case Diagramm**

**Business Use Case**

Wird definiert als “eine Sequenz von Transaktionen in einem System (Unternehmen)“. Die ausgeführte Aufgabe soll für den Akteur außerhalb des Geschäftssystems von messbarem Wert sein. Es handelt sich um einen Unternehmensprozess, der aus einer Anzahl von unternehmensinternen Aktivitäten besteht, die durchgeführt werden, um die Wünsche eines Akteurs (zB Kunde zu befriedigen. Anstoß durch einen Akteur außerhalb des Geschäftssystems

**Use Case in einem Informationssystem**

Wird definiert als “eine Sequenz von zusammengehörenden Transaktionen, die von einem Akteur (User der Anwendung) im Dialog mit einem System ausgeführt werden, um für den Akteur ein Ergebnis von messbarem Wert zu erstellen“.

Er spezifiziert die Interaktionen zwischen einem Akteur und dem IT-System, er definiert also eine spezielle Benutzung des Systems. (entspricht UserStory im SCRUM)

**Formale Notation**

Im Use-Case-Diagramm gibt es Akteure und das System. 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.

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

Im Use-Case-Diagramm gibt es zwei Arten von Verbindungslinien: Die durchgezogene Verbindungslinie stellt eine Assoziation zwischen dem Akteur und einem Use-Case dar. Sie bedeutet, dass der Akteur den Use-Case in irgendeiner Form anwendet (Entweder Anstoß oder Erhalten eines messbaren Ergebnisses).

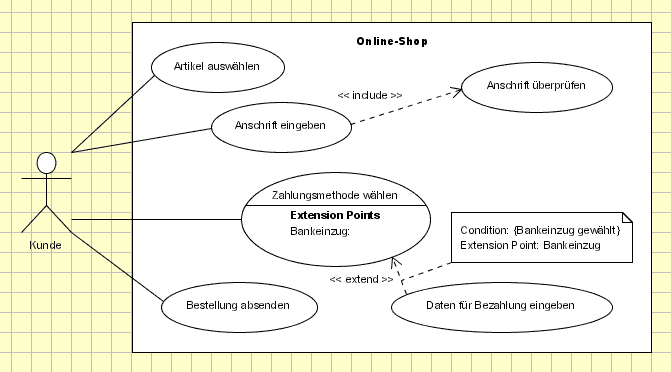
Die gestrichelte Verbindungslinie stellt eine Assoziation zwischen zwei Use-Cases dar. Da es zwei verschiedene Arten von Assoziationen zwischen Use-Cases gibt, wird neben die gestrichelte Verbindungslinie ein Schlüsselwort in Spitzklammern gesetzt.

Eine include Assoziation bedeutet, dass der Use-Case, von dem die Verbindungslinie ausgeht, den Use-Case einschließt, auf den die Verbindungslinie zeigt.

Eine extend Assoziation bedeutet, dass der Use-Case, von dem die Verbindungslinie ausgeht, möglicherweise den Use-Case erweitert, auf den die Verbindungslinie zeigt. Der entscheidende Unterschied zwischen einer include- und extend-Assoziation ist also, dass bei der include-Beziehung der zweite Use-Case immer ausgeführt wird, bei der extend-Assoziation der zweite Use-Case in Abhängigkeit von Bedingungen im ersten Use-Case ausgeführt wird.

b)

**Beispiel Use Case Diagramm:**



**Aktivitätsdiagramm**

Das Aktivitätsdiagramm beschreibt ganz allgemein Abläufe. Es stellt dar, in welcher Reihenfolge ganz bestimmte Aktionen ausgeführt werden. (= Ablauflogik)

Ein Aktivitätsdiagramm kann auf 3 Arten verwendet werden:

* im Business Use Case Modell, übergreifend zur Beschreibung der Aktivitäten-Abfolge des

gesamten Geschäftssystems.

* in beiden Modellen pro Use Case zur Beschreibung der Aktivitäten Abfolge innerhalb eines Use Cases als Ergänzung zur Schablonen- bzw. verbalen Beschreibung.
* Zur Darstellung von Softwareabläufen (login innerhalb einer Methode)

**Formale Notation**

Ein Aktivitätsdiagramm kennt grundsätzlich zwei Arten von Bausteinen:

* Knoten sind die Stellen, an denen etwas passiert
* Kanten sind einfach nur die Verbindungslinien zwischen Knoten.

Über diese Verbindungslinien wandern sogenannte Token.

Der Token zeigt Ihnen an, wo das Aktivitätsdiagramm momentan steht.

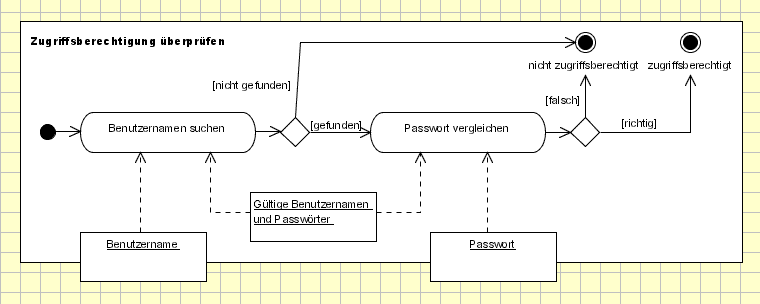
Jedes Aktivitätsdiagramm hat einen oder mehrere Start- und Endpunkte. Diese beiden Punkte sind über viele Knoten und Kanten verbunden. Wenn nun eine Aktivität ausgeführt wird, begingt ein Token im Startpunkt zu wandern. Dieses Token wandert die erste Verbindungslinie entlang zum ersten Knoten. Nachdem dort irgendetwas passiert ist, wandert das Token auf der von diesem Knoten ausgehenden Verbindungslinie zum zweiten Knoten. Diese Wanderung wird solange fortgesetzt, bis es am Endpunkt des Diagramms ankommt, womit die Aktivität endet.

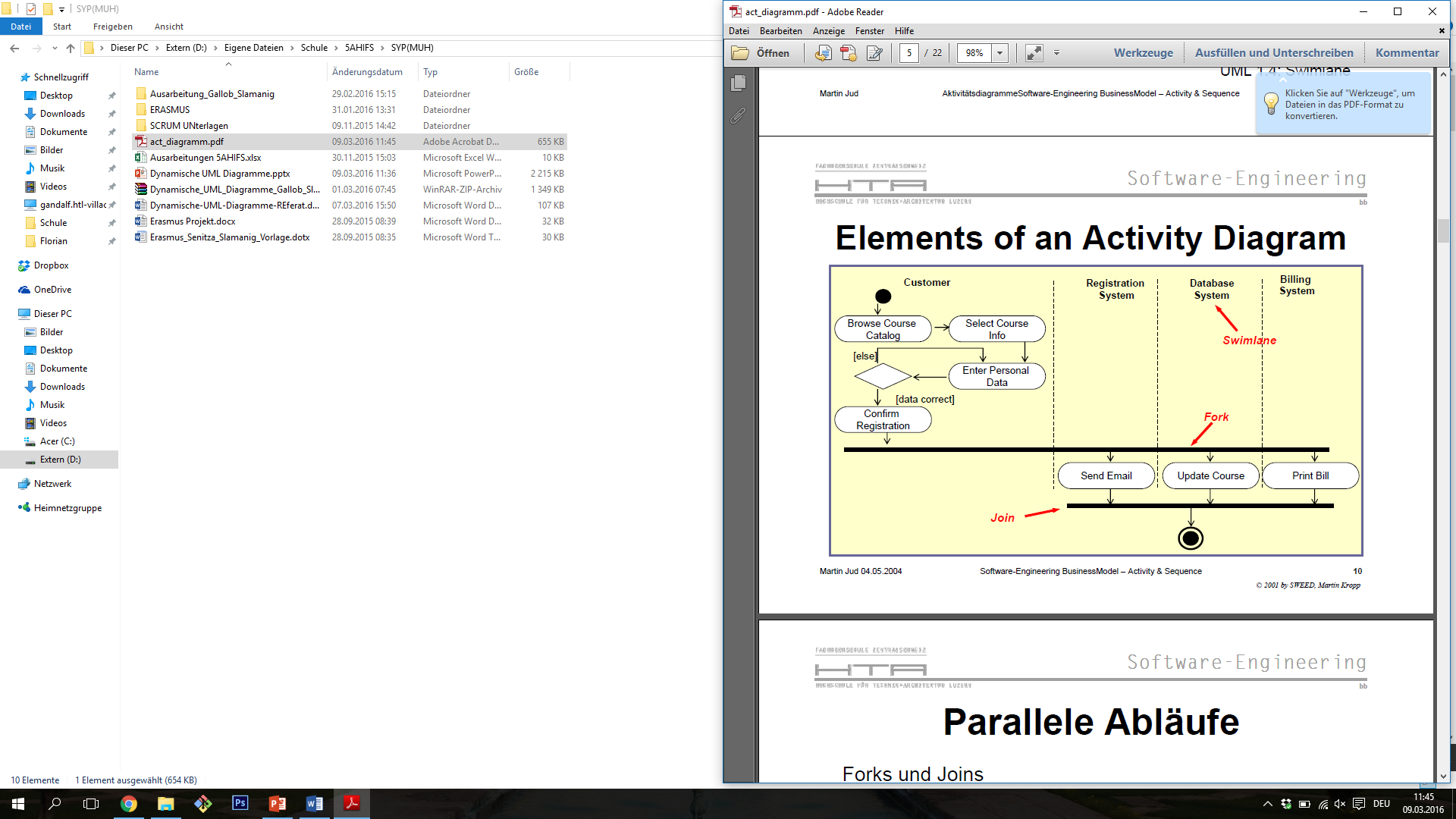
Während Kanten einfach nur durchgezogene Linien mit einer Pfeilspitze sind, die anzeigt, in welche Richtung auf dieser Linie Token wandern dürfen, gibt es unterschiedliche Arten von Knoten.

Wichtigste Knoten:

* Startpunkt – Schwarzer ausgefüllter Punkt
* Endpunkte – Schwarzer ausgefüllter Punkt mit einem umrahmenden Ring
* Aktionen – Rechtecke mit abgerundeten Ecken
* Objekte – Rechtecke ohne abgerundeten Ecken: Befindet sich ein Objektknoten zwischen zwei Aktionen, bedeutet das, dass das Token die Daten von der einen Aktion in den Speicher der anderen überträgt
* Verzweigung – Raute: Hier wird der Token nur auf einer Linie weitergeleitet, je nach der Bedingung die gestellt wurde
* Gabelung – Schwarzer Balken: Hier wird der Token kopiert und auf beiden Linien übertragen
* Vereinigung: Die parallel laufenden Abläufe werden wieder zusammengefügt
* Partitionen: Verschiedene Rollen im Aktivitätsdiagramm

**Beispiel Aktivitätsdiagramm:**





Beispiel für Fork, Join und Partitionen

# **Sequenzdiagramm**

Gezeigt wird, welche Objekte im System vorhanden sind, welche Informationen sie austauschen und in welcher Reihenfolge dies geschieht.

Durch eine Zeitachse die von oben nach unten verläuft wird die zeitliche Reihenfolge beschrieben.

Sequenzdiagramme werden häufig verwendet, um Szenarien eines Systems zu modellieren.

Ein Szenario ist die Beschreibung einer Sicht auf einen Teil eines Systems.

Ein System wird in der Regel nicht vollständig durch Sequenzdiagramme spezifiziert. Es werden nur die Szenen modelliert, die häufig vorkommen oder besonders wichtig sind.

Sequenzdiagramme können sich auf Use Cases beziehen.

# **Formale Notation**

Ein Objekt wird als Rechteck dargestellt. Von diesem geht eine senkrechte Lebenslinie aus. Auf dieser können sogenannte Aktivitäten, als schmale Rechtecke eingezeichnet werden, welche ausgeführte Methoden beschreiben.

Nachrichten werden zwischen Objekten mittels Pfeilen dargestellt. Der Name der Botschaft wird über den Pfeil geschrieben. Man unterscheidet zwischen folgenden Nachrichten:

* Synchrone Nachricht: Pfeil mit voller Spitze.
  + Dieser geht von der Quelle aus und endet bei dem Zielobjekt. Ist die Verarbeitung der Methode des Zielobjekts fertig, wird ein Returnwert, dargestellt durch einen gestrichelten Pfeil mit offener Spitze, an die Quelle zurückgesendet. Die Quelle ist solange im Ruhezustand und wartet auf die Antwort.
* Asynchrone Nachricht: Pfeil mit offener Spitze.
  + Hier wartet die Quelle auf keine Antwort, sondern setzt mit der Verarbeitung fort. Diese Art der Kommunikation wird bei parallelen Threads eingesetzt.

Wird ein Objekt erzeugt, führt eine Nachricht in den Kopf eines neuen Objekts. Zerstört wird es, indem eine Nachricht auf ein X zeigt.

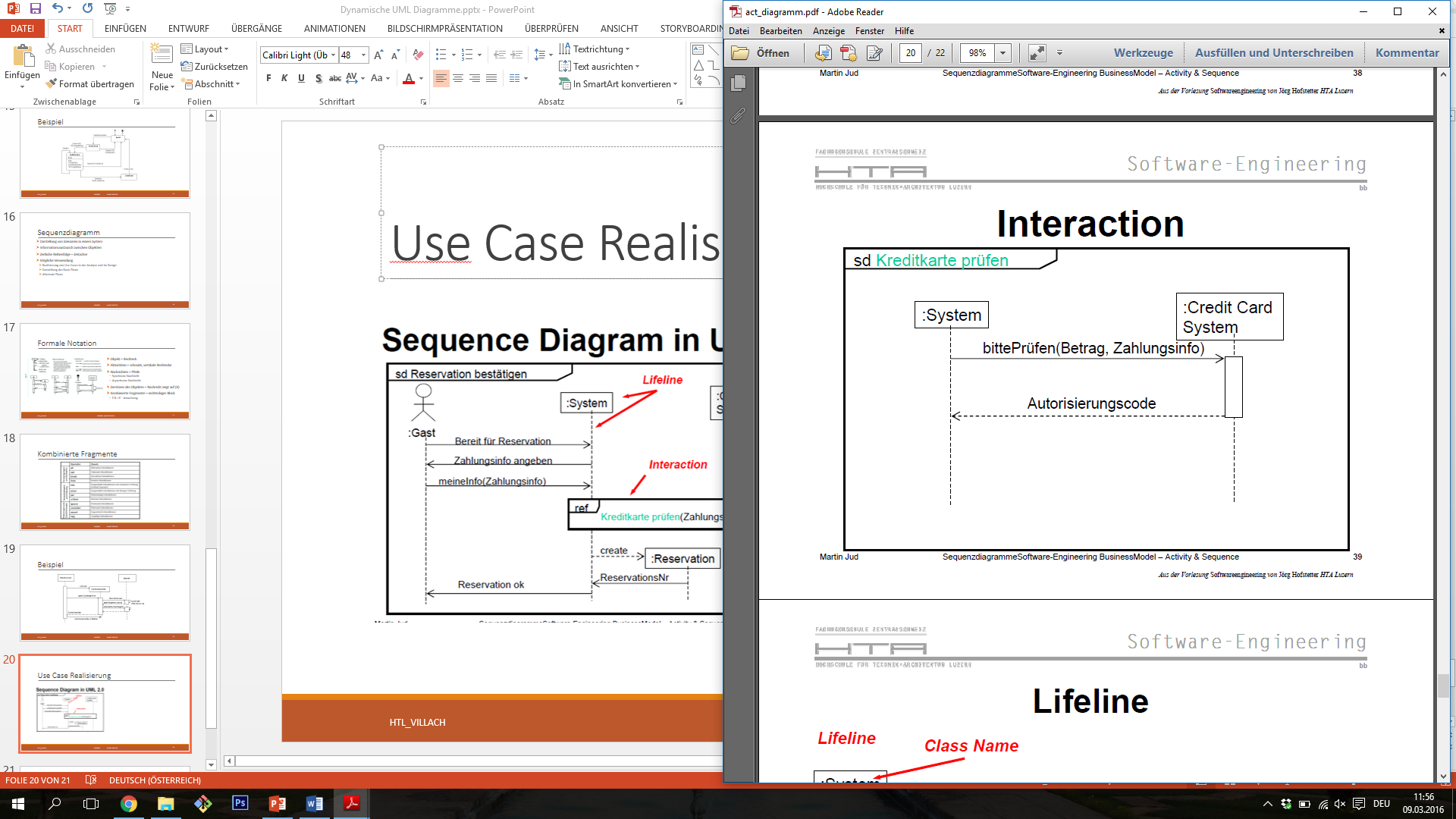
Um Teile höherer Programmiersprachen auszudrücken, gibt es sogenannte kombinierte Fragmente.

Jedes Fragment wird als rechteckiger Block dargestellt, in welchem die Bezeichnung in der linken oberen Ecke angeführt wird.

* Ein Block mit der Bezeichnung „ref“, weist auf ein weiteres Diagramm hin, welches an dieser Stelle eingebunden wird.
* Der mit „alt“ markierte Block beschreibt eine Alternative (Verzweigung). Ist die Bedingung [b=wahr] erfüllt, wird der Bereich oberhalb der gestrichelten Linie in dem alt-Block ausgeführt. Im else-Fall wird der Bereich unterhalb der gestrichelten Linie ausgeführt.
* Eine Schleife kann mit der Bezeichnung „loop“ angeführt werden. Neben der Bezeichnung steht die Abbruchbedingung der Schleife. Innerhalb des Blocks werden die durchzuführenden Aktionen dargestellt.

# **Beispiel Sequenzdiagramm**

# 



# **Zustandsdiagramm**

Zustandsdiagramme dienen dazu, das Verhalten von sämtlichen Elementen darzulegen. Zustände und Zustandsübergänge werden während ihres Lebenszyklus beschrieben. Man unterscheidet:

* behavioral state machines, die Verhalten von Instanzen einer Klasse, Systemen oder Systemteilen beschreiben
* protocol state machines, die Protokolle, die von einem Systemelement realisiert werden, modellieren.

# **Formale Notation**

Anfangs- und Endzustände geben den Beginn und das Ende eines Elements an. Der Anfangspunkt wird als ausgefüllter Punkt und der Endpunkt als ausgefüllter Punkt mit einem umliegenden Kreis dargestellt.

Eine Entscheidung beschreibt eine Alternative. Entweder wird die nach links oder die nach rechts führende Transition ausgeführt.

Eine Kreuzung verbindet mehrere Transitionen, ohne zwischengeschaltete Zustände miteinander.

Ein Terminator (X) drückt aus, dass die State Machine beendet wird; d. h. das Objekt, auf das die State Machine sich bezieht wird zerstört. Es werden keine exit-Aktivitäten ausgeführt.

Eine Gabelung teilt eine eingehende Transition in mehrere ausgehende parallele Transitionen auf. Eine Vereinigung wird verwendet, um die durch eine Gabelung aufgeteilten parallelen Zweige wieder zusammenzuführen.

Elemente können im Laufe ihrer Existenz eine endliche Anzahl Zustände einnehmen. Der Zustand wird als Rechteck mit abgerundeten Ecken dargestellt. Zustände können einen Namen erhalten. Ein Zustand kann durch eine waagrechte Linie unterteilt werden. Unterhalb der Linie kann Verhalten eines Objekts in dem Zustand angegeben werden.

Die UML definiert folgende Arten von Verhalten:

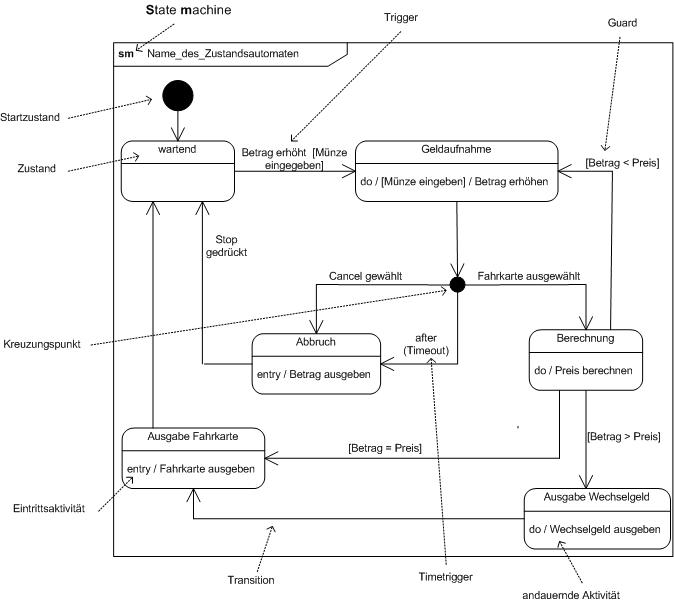
* Entry-Verhalten wird beim Eintritt in den Zustand ausgeführt
* Exit-Verhalten vor Verlassen des Zustands.
* Do-Aktivitäten werden ausgeführt, während das Objekt sich in dem Zustand befindet. Interne Transitionen werden intern ausgeführt, führen jedoch nicht zu einem Zustandsübergang. Es kann ein Auslöser (Trigger) angegeben werden, eine Bedingung (Guard), die erfüllte sein muss und eine Aktivität, die ausgeführt wird.

Eine Transition ist ein Zustandsübergang von einem Quellzustand zu einem Zielzustand.

Drei Zustandstypen werden unterschieden:

* Einfacher Zustand: Ein einfacher Zustand hat keine Unterzustände. Ein einfacher Zustand kann entry, exit und do-Aktivitäten enthalten und interne Transitionen. Nach Erreichen des Zustands wird sofort die entry-Aktivität ausgeführt. Vor Verlassen des Zustands wird die exit-Aktivität ausgeführt. Die do-Aktivität wird nach Beendigung der entry-Aktivität ausgeführt.
* Zustandsübergang (Transition): Eine Transition wird durch einen Auslöser (Trigger) verursacht. Für einen Zustandsübergang kann eine Bedingung angegeben werden, die erfüllt sein muss, damit die Transition ausgeführt wird. Bei einer Transition kann ein Objekt eine Aktivität ausführen, die zu dem Aufruf einer Methode führt.
* Einfacher Zustandsautomat: Zustandsautomaten beschreiben den Lebenszyklus von Objekten, können sich aber auch auf komplexere Elemente beziehen, wie z. B. Komponenten, Subsysteme oder Use Cases. Sie stellen die Zustände dar, die Objekte, Komponenten usw. im Laufe ihrer Existenz annehmen können und sie geben die Zustandsübergänge an, die möglich sind. Ein Zustandsautomat gibt auch an, in welchem Zustand ein Objekt auf welche Ereignisse reagiert.

# **Beispiel Zustandsdiagramm**



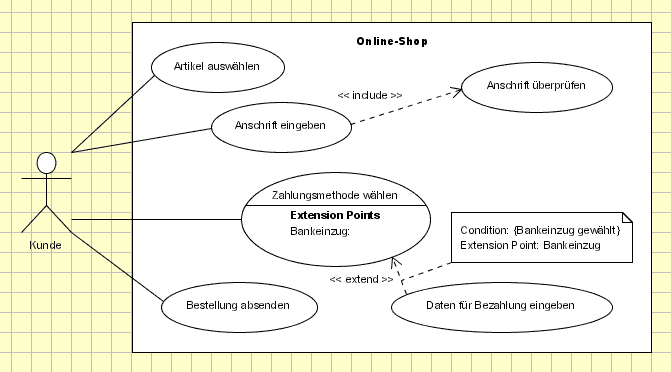
**Beispiel Fahrkartenautomat:**

* Wartender Zustand 🡪 Geldaufnahme
* Berechnung
* Fahrkartenausgabe
* Wechselgeld optional

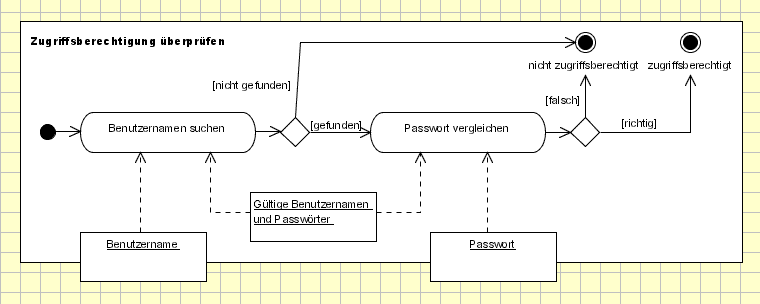
b und c)

Praktisches Beispiel: Sie sind MA in einer Software Firma und haben die Aufgabe für einen Kunden der sich einen Webshop wünscht, zu dokumentieren. Verwenden Sie dafür sinnvolle UML Diagramme: der Webshop soll das Auswählen eines Artikels, die Bestellung (inklusive Wahl der Bezahlungsmethode) und die Registrierung für Rechnungs- & Lieferadresse beinhalten (UseCase: „Suchen der Bestellung“, „Bestellung inkl. Warenkorb“, „Bezahlung“, „Registrieren“

# **Use-Case Diagramm**



# **Activity Diagramm**



c)

**Business Modelling:**

* Business Use Case Diagram
* Activity Diagram zur Beschreibung der Use Cases bei hoher Komplexität
* Ev auch use case übergreifendes Activity D zur Beschreibung des ges. GSs

**Requirements:**

* Zukünftiges IT-Sytem (IT-UC-Diagram)
* Activity Diagram zur Beschreibung der Use Cases bei hoher Komplexität

**Analyse & Design:**

* Sequence Diagram pro Use Case zur Beschreibung der fachl Logik
* (jeder Use Case wird im WF Analysis mit einer Control Klasse, den zugehörigen Boundary Klassen und den zugehörigen Entity Klassen des Domain Models dargestellt), dh. Das Sequendiagramm zeigt die INteraktionen zw diesen Klassen 🡪 Resalisierung des Use Cases)

**Frage 3: Analysemuster (Analysis Patterns)**

* Ausgangssituation: Firma „HighTronic“; Sie sind zuständig für die Datenmodellierung. Im Rahmen der Einschulung werden folgende Aufgaben gestellt:

1. Nennen und beschreiben Sie die, in der Datenmodellierung eingesetzten Analysemuster (10) und erklären Sie kurz, wofür man Sie einsetzt.
2. Erklären Sie drei Analysemuster anhand eines Beispiels genauer.
3. Das am Beiblatt befindliche Domain Model (Domain Modell v Frage 1) enthält mehrere dieser Analysemuster. Bestimmen und erörtern Sie diese.

Antworten:

a)

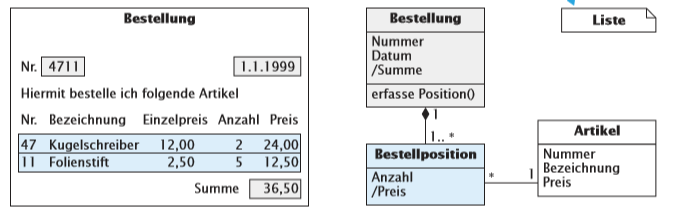
**Analyse-Pattern**

Es hat sich gezeigt, dass bei der Modellierung häufig ähnliche Probleme vorkommen. Muster beschreiben – wiederkehrende – Problemstellungen und ihre Lösungen. Im Sinne einer effizienten Softwareentwicklung ist es sinnvoll, bereits existierende Problemlösungen wiederzuverwenden. Ganz allgemein gesehen ist ein Muster (pattern) eine Idee, die sich in einem praktischen Kontext als nützlich erwiesen hat und es wahrscheinlich auch in anderen sein wird. Ein Analysemuster ist eine Gruppe von Klassen mit feststehenden Verantwortlichkeiten und Interaktionen. Es kann eine Gruppe von Klassen sein, die durch Beziehungen verknüpft sind, oder eine Gruppe von kommunizierenden Objekten.

**Muster 1: Liste**

Eine Bestellung besteht sozusagen aus einem Bestellungskopf und den einzelnen Positionen. Analog lässt sich ein Lager mit all seinen einzelnen Lagerplätzen als Liste visualisieren. Ein Lagerplatz kann nicht ohne Lager existieren. Die Attributwerte des Lagers gelten auch für jeden Lagerplatz. Diese Problemstellung kommt in vielen Anwendungsbereichen immer wieder vor und wird als Komposition modelliert. Die UML erlaubt es, auf jedem Diagramm wichtige Informationen als Notiz anzugeben.

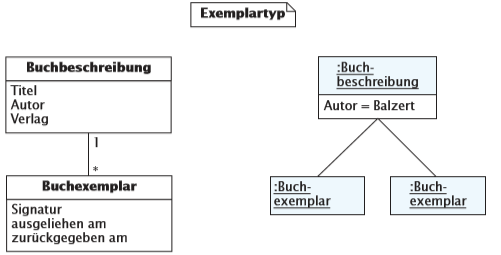
* Es liegt eine Komposition vor.
* Die Komposition, als Sonderfall der Aggregation, beschreibt ebenfalls die Beziehung zwischen einem Ganzen und seinen Teilen.
* Ein Ganzes besteht aus gleichartigen Teilen, d.h. es gibt nur eine Teil-Klasse.
* Teil-Objekte bleiben einem Aggregat-Objekt fest zugeordnet. Sie können jedoch gelöscht werden, bevor das Ganze gelöscht wird.
* Die Attributwerte des Aggregat-Objekts gelten auch für die zugehörigen Teil-Objekte.
* Das Aggregat-Objekt enthält im allgemeinen mindestens ein Teilobjekt, d.h. die Kardinalität ist meist 1..\*.



**Muster 2: Exemplartyp**

Von einen Buch sind mehrere Exemplare zu verwalten. Würde diese Problemstellung durch eine einzige Klasse Buch modelliert, dann würden mehrere Objekte bei Titel, Autor und Verlag identische Attributwerte besitzen. Eine bessere Modellierung ergibt sich, wenn die gemeinsamen Attributwerte mehrerer Buchexemplare in einem neuen Objekt Buchbeschreibung zusammengefasst werden.

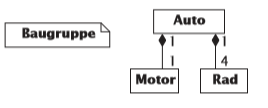
* Es liegt eine einfache Assoziation vor, denn es besteht keine whole-part-Beziehung.
* Erstellte Objektverbindungen werden nicht verändert. Sie werden nur gelöscht, wenn das betreffende Exemplar entfernt wird.
* Der Name der neuen Klasse enthält oft Begriffe wie Typ, Gruppe, Beschreibung, Spezifikation.
* Eine Beschreibung kann – zeitweise – unabhängig von konkreten Exemplaren existieren. Daher ist die Kardinalität im Allgemeinen many.
* Würde auf die neue Klasse verzichtet, so würde als Nachteil lediglich die redundante »Speicherung« von Attributwerten auftreten.



**Muster 3: Baugruppe**

Es soll ausgedrückt werden, dass jedes Auto exakt einen Motor und vier Räder haben soll. Da es sich hier um physische Objekte handelt, liegt ein physisches Enthaltensein vor, das mittels Komposition modelliert wird. Wenn ein Auto verkauft wird, dann gehören Motor und Räder dazu. Die Zuordnung der Teile zu ihrem Ganzen bleibt normalerweise über einen längeren Zeitraum bestehen. Der Motor kann jedoch durch ein neuen Motor ersetzt werden und der alte Motor in ein anderes Objekt eingebaut werden.

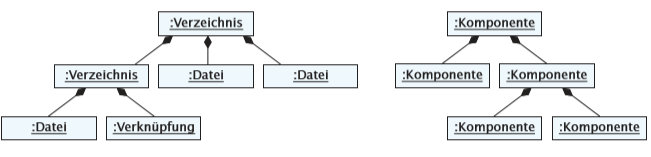
* Es handelt sich um physische Objekte.
* Es liegt eine Komposition vor.
* Objektverbindungen bestehen meist über eine längere Zeit. Ein Teil-Objekt kann jedoch von seinem Aggregat-Objekt getrennt werden und einem anderen Ganzen zugeordnet werden.
* Ein Ganzes kann aus unterschiedlichen Teilen bestehen.



**Muster 4: Stückliste**

Es soll modelliert werden, dass ein Verzeichnis Verknüpfungen, Dateien und weitere Verzeichnisse enthalten kann. Dabei sollen sowohl das Verzeichnis und alle darin enthaltenen Objekte als Einheit, als auch jedes dieser Objekte einzeln behandelt werden können. Wird beispielsweise das Verzeichnis kopiert, dann sollen alle darin enthaltenen Dateiobjekte kopiert werden. Wird das Verzeichnis gelöscht, dann werden auch alle seine Teile gelöscht. Ein Sonderfall liegt vor, wenn sich diese Enthaltensein-Beziehung auf gleichartige Objekte bezieht. Beispielsweise setzt sich jede Komponente aus mehreren Komponenten zusammen. Umgekehrt ist jede Komponente in einer oder keiner anderen Komponente enthalten. Diese Problemstellung wird durch eine Komposition modelliert, wobei die verschiedenen Teil-Objekte durch eine Vererbung dargestellt werden.

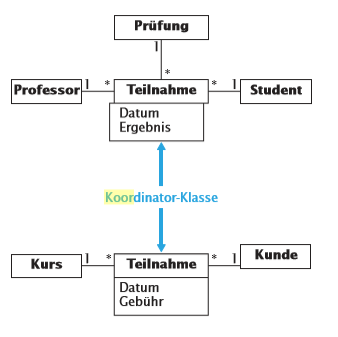
* Es liegt eine Komposition vor.
* Das Aggregat-Objekt und seine Teil-Objekte müssen sowohl als Einheit als auch einzelnen behandelt werden können.
* Teil-Objekte können anderen Aggregat-Objekten zugeordnet werden.
* Die Kardinalität bei der Aggregat-Klasse ist 0..1.
* Ein Objekt der Art A kann sich aus mehreren Objekten der Arten A, B und C zusammensetzen.
* Der Sonderfall der Stückliste ist, dass ein Stück nicht aus Objekten unterschiedlicher Art, sondern nur aus einer einzigen Art besteht.



**Muster 5: Koordinator**

Eine ternäre Assoziation verbindet Objekte der Klassen Professor, Prüfung und Student und »merkt« sich Informationen über eine abgelegte Prüfung in der assoziativen Klasse Teilnahme. Diese ternäre Assoziation kann wie abgebildet in binäre Assoziationen und eine Koordinator-Klasse aufgelöst werden. Für eine Koordinator-Klasse ist typisch, dass sie oft selbst nur wenige Attribute und Operationen besitzt, sondern sich vor allem merkt »wer wen kennt«. Als Sonderfall dieser Problemstellung kann eine binäre Assoziation mit einer assoziativen Klasse betrachtet werden.

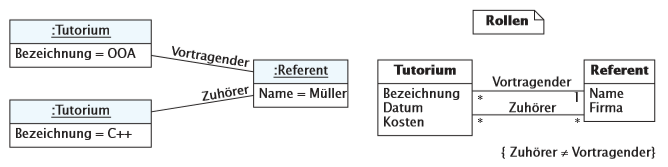
* Es liegen einfache Assoziationen vor.
* Die Koordinator-Klasse ersetzt eine n-äre (n >= 2) Assoziation mit assoziativer Klasse.
* Die Koordinator-Klasse besitzt kaum Attribute/Operationen, sondern mehrere Assoziationen zu anderen Klassen, im Allgemeinen zu genau einem Objekt jeder Klasse.



**Muster 6: Rollen**

Zu einem Tutorium sind Vortragende und Zuhörer zu verwalten. Dabei kann ein Referent sowohl Vortragender als auch Zuhörer von Tutorien sein. Mit anderen Worten: der Referent spielt – zur gleichen Zeit – in Bezug auf die Klasse Tutorium mehrere Rollen. Diese Problemstellung kommt relativ häufig vor.

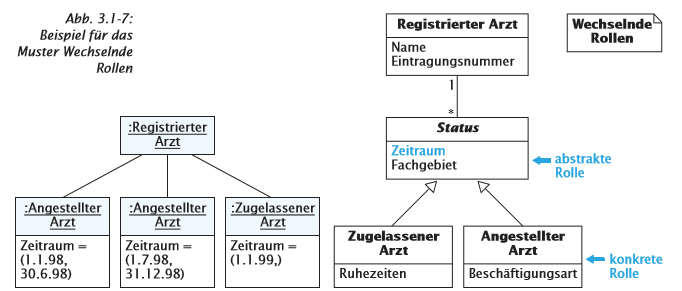
* Zwischen zwei Klassen existieren zwei oder mehrere einfache Assoziationen.
* Ein Objekt kann – zu einem Zeitpunkt – in Bezug auf die Objekte der anderen Klasse verschiedene Rollen spielen.
* Objekte, die verschiedene Rollen spielen können, besitzen unabhängig von der jeweiligen Rolle die gleichen Eigenschaften und ggf. gleiche Operationen.



**Muster 7: Wechselnde Rollen**

Wenn ein kassenärztlich registrierter Arzt im ersten und zweiten Halbjahr 1998 in unterschiedlichen Praxen jeweils eine Tätigkeit als angestellter Arzt ausübt, bevor er am 1.1.1999 seine Zulassung erhält. Für angestellte Ärzte sind teilweise andere Informationen zu speichern als für die Zugelassenen. Im Gegensatz zum Rollen-Muster spielt der registrierte Arzt während eines Zeitraum verschiedene Rollen. Da es hier darum geht, Informationen über einen Zeitraum festzuhalten, werden neue ärztliche Tätigkeiten und deren Objektverbindungen zu Registrierter Arzt immer nur hinzugefügt.

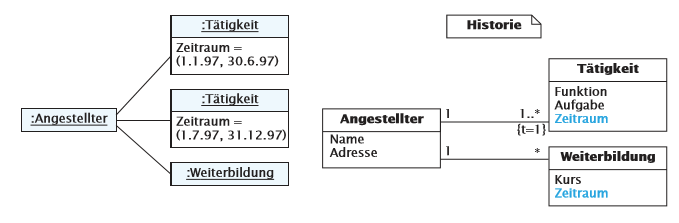
* Ein Objekt der realen Welt kann zu verschiedenen Zeiten verschiedene Rollen spielen. In jeder Rolle kann es unterschiedliche Eigenschaften (Attribute, Assoziationen) und Operationen besitzen.
* Die unterschiedlichen Rollen werden mittels Vererbung modelliert.
* Objektverbindungen zwischen dem Objekt und seinen Rollen werden nur erweitert, d.h. weder gelöscht noch zu anderen Objekten aufgebaut.



**Muster 8: Historie**

Für einen Angestellten sollen alle Tätigkeiten, die er während der Zugehörigkeit zu einer Firma ausübt, festgehalten werden. Dabei darf zu jedem Zeitpunkt nur eine aktuelle Tätigkeit gültig sein. Außerdem sind alle Weiterbildungskurse, die er im Laufe seiner Firmenzugehörigkeit besucht, aufzuzeichnen. Für jede Tätigkeit und jede Weiterbildung wird der Zeitraum eingetragen. Die Restriktion {t=1} spezifiziert, dass ein Angestellter zu einem Zeitpunkt genau eine Tätigkeit ausübt. Wenn alle Tätigkeiten und Weiterbildungen gespeichert sein sollen, dann bedeutet dies, dass die aufgebauten Verbindungen zu Tätigkeit und Weiterbildung bestehen bleiben bis der Angestellte die Firma verlässt und seine Daten gelöscht werden.

* Es liegt eine einfache Assoziation vor.
* Für ein Objekt sind mehrere Vorgänge bzw. Fakten zu dokumentieren.
* Für jeden Vorgang bzw. jedes Faktum ist der Zeitraum festzuhalten.
* Aufgebaute Objektverbindungen zu den Vorgängen bzw. Fakten werden nur erweitert.
* Die zeitliche Restriktion {{t=k} (k = gültige Kardinalität) sagt aus, was zu einem Zeitpunkt gelten muss.



**Muster 9: Gruppe**

Es bildet sich eine Gruppe, wenn mehrere Angestellte zu einer Abteilung gehören. Da die Abteilung auch – kurzfristig – ohne zugehörige Angestellte existieren soll, wird die manyKardinalität gewählt. Sollte modelliert werden, dass beim Eintragen der Abteilung mindestens ein Angestellter zugeordnet wird, dann wäre die Kardinalität 1..\* zu wählen. Wenn ein Angestellter aus einer Abteilung ausscheidet, dann wird die entsprechende Objektverbindung getrennt.

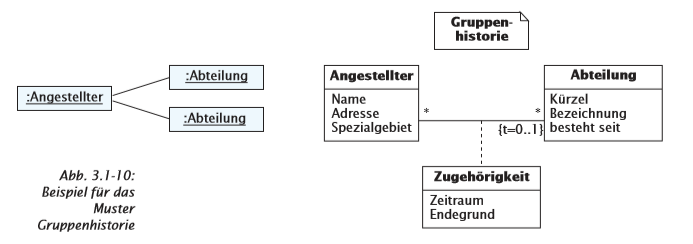
* Es liegt eine einfache Assoziation vor.
* Mehrere Einzel-Objekte gehören – zu einem Zeitpunkt – zum selben Gruppen-Objekt.
* Es ist jeweils zu prüfen, ob die Gruppe – zeitweise – ohne Einzelobjekte existieren kann oder ob sie immer eine Mindestanzahl von Einzel-Objekten enthalten muss.
* Objektverbindungen können auf- und abgebaut werden.



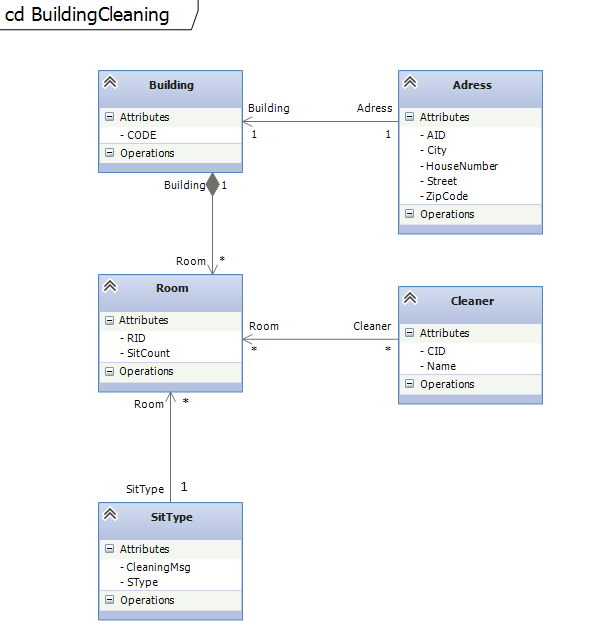
**Muster 10: Gruppenhistorie**

Soll die Zugehörigkeit zu einer Gruppe nicht nur zu einem Zeitpunkt, sondern über einen Zeitraum festgehalten werden, dann ist eine Problemstellung wie folgt zu modellieren: Für jeden Angestellten wird festgehalten, über welchen Zeitraum er zu einer Abteilung gehört hat. Die Restriktion {t=0..1} sagt aus, dass er zu einem Zeitpunkt in maximal einer Abteilung tätig sein kann. Wenn ein Angestellter eine Abteilung verlässt, dann wird dies durch die Attributwerte im entsprechenden Objekt von Zugehörigkeit beschrieben.

* Ein Einzel-Objekt gehört – im Laufe der Zeit – zu unterschiedlichen Gruppen-Objekten.
* Die Historie wird mittels einer assoziativen Klasse modelliert. Dadurch ist die Zuordnung zwischen Einzel-Objekten und Gruppen deutlich sichtbar.
* Die zeitliche Restriktion {t=k} (k = gültige Kardinalität) sagt aus, was zu einem Zeitpunkt gelten muss.
* Da Informationen über einen Zeitraum festzuhalten sind, bleiben erstellte Objektverbindungen bestehen und es werden nur Verbindungen hinzugefügt.



Angabe für c)



Lösung: selbst ausarbeiten!

Analysemuster: Liste zw Building und Room

Assoziationsklasse zwischen Room und Cleaner (zumindest ein Attribut liegt auf Assoziation (zB das Datum/die Zeit)

**Frage 4: Design Patterns ????**

Die Firma BestWebApplications entwickelt seit Jahren WebAnwendungen für verschiedenste Kunden. Dem Firmenbesitzer, ein sogenannter Autodidakt der Programmierung ist bewusst, dass in seinem Unternehmen das KnowHow bezüglich neuem Programmdesign und moderner SW-Architektur derzeit etwas nachhinkt und hier ein Innovationssprung unumgänglich ist, will man im harten Geschäft der WebEntwicklung weiterhin eine Rolle spielen.

Sie haben die neu ausgeschriebene Stelle als Programmierer angenommen und Ihre Aufgabe ist nun, die SW-Entwickler des Unternehmens mit den SW-Entwicklungsmethoden vertraut zu machen, die dem heutigen Stand der Technik entsprechen.

**Fragestellung:**

## In der Programmierung bedient man sich Muster. Damit soll erreicht werden, dass gleichartige bzw. ähnliche Probleme immer gleich gelöst werden. Damit erreicht man eine Standardisierung, Allgemeingültigkeit und eine Referenzlösung. Definieren und klassifizieren Sie solche Muster und beschreiben Sie deren Anwendung.

* 1. Erklären Sie einige wichtige Entwurfsmuster (zumindest drei).

c) Die Verwendung von Design Patterns bringt viele Vorteile, allerdings spricht man auch von

Nachteilen. Diskutieren sie diese Vor- und Nachteile.

Siehe Referat:

plus

## Nachteile

Das Kennen vieler Design Patterns kann dazu verleiten

* Pattern als Wunderwaffe für ein gutes Design anzusehen
* viele bekannte Pattern zu verwenden und dabei einfachere/elegantere Lösungen zu übersehen
* den Code unnötig aufzublähen
* ein zu allgemeines (generisches) Problem zu lösen