

Objektorientierte Software-Entwicklung

Priv.- Doz Dr. Rolf Hennicker

04.10.2002



Kapitel 2

Objektorientierte Modellierungstechniken

Ziele

- Klassendiagramme in UML erstellen können.
- Objektdiagramme in UML erstellen können.
- Das Vererbungsprinzip verstehen, insbesondere
 - abstrakte Klassen und Operationen
 - Schnittstellen
 - dynamische Bindung
- Flache und hierarchische Zustandsdiagramme in UML erstellen können (einschl. Aktivitätsdiagramme).
- Zustände, Ereignisse und Transitionen verstehen.

Grundidee

Modellierung dient dazu, ein System zu verstehen, bevor es gebaut wird.

Wesentliches Prinzip

Abstraktion (auf wesentliche Aspekte konzentrieren, keine Details)

Es werden i.a. verschiedene Sichten auf ein System modelliert:

- *Statisches Modell*: beschreibt strukturelle und datenbezogene Eigenschaften
- *Dynamisches Modell*: beschreibt das Verhalten der Objekte, deren Zustandsänderungen und Interaktionen.

Notation

UML (Unified Modelling Language), 1997 Version 1.0 (Booch, Rumbaugh, Jacobson), aktuelle Version 1.4

2.1 Statisches Modell (Objektmodell)

2.1.1 Klassen und Objekte

Klassen

Allgemeine Form:

Klassenname
operation operation(Argumentenliste) operation(Argumentenliste): Typ
attribut attribut: Typ attribut: Typ = Defaultwert

Beispiel:

Kunde
name: String adresse: String umsatz: Real
getName(): String umsatzErhoehen(n: Real)

Kurzform:

Klassenname

Objekte

Allgemeine Form:

<u>Objektname:Klassenname</u>
attribut = Wert attribut: Typ = Wert

Beispiel:

<u>:Kunde</u>
name = "Fritz Meier" adresse = "München" umsatz = 4590,30

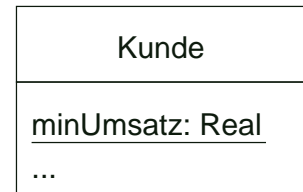
Bemerkungen

- Typen sind Standarddatentypen (Boolean, Integer, Real, String) oder Klassennamen.
- In der Analysephase sollten als Typen von Attributen nur Standardtypen verwendet werden.
- Operationen mit Ergebnistyp liefern nach Ausführung einen Wert dieses Typs. Der Rückgabewert kann auch ein Objekt sein.
- Operationen ändern i.a. den Zustand eines Objekts.
- Operationen, die den Zustand nicht ändern, nennt man "Queries".

Weiterführende Begriffe und Notationen

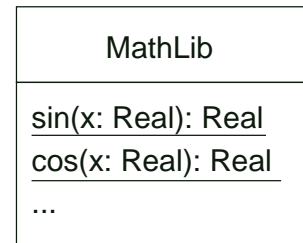
- Ein Attribut, das zu jedem Zeitpunkt bei jedem existierenden Objekt der Klasse denselben Wert hat, heißt *Klassenattribut* und wird in UML unterstrichen.

Beispiel:



- Eine Operation, die vom Zustand konkreter Objekte unabhängig ist, heißt *Klassenmethode* und wird in UML unterstrichen.

Beispiel:

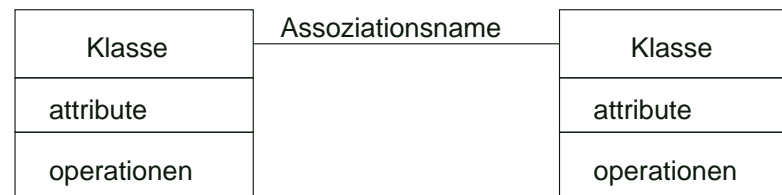


2.1.2 Assoziationen und Objektbeziehungen

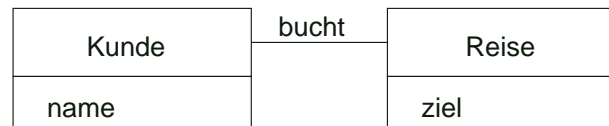
- Eine *Objektbeziehung (Link)* ist eine semantische (physikalische oder konzeptionelle) Verbindung zwischen Objekten.
- Eine *Assoziation* beschreibt eine Menge gleichartiger Beziehungen zwischen Objekten bestimmter Klassen.

Assoziationen

Allgemeine Form:

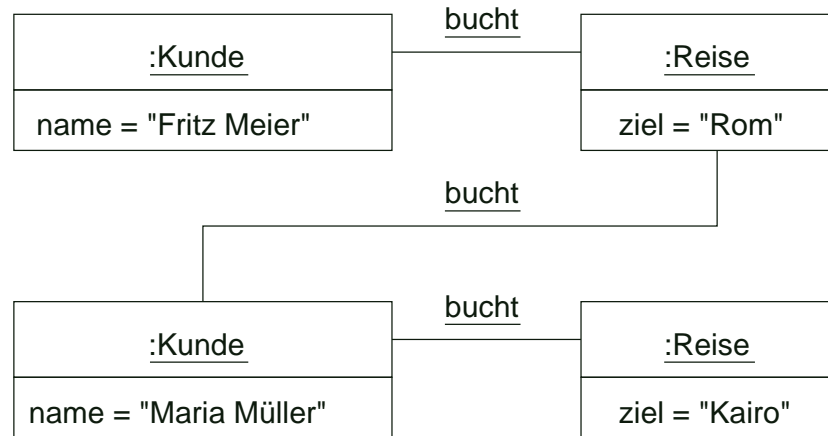


Beispiel:



Objektbeziehungen

Beispiel:



- Ein UML-Diagramm mit Klassen und Assoziationen (und Vererbung) heißt *Klassendiagramm*.
- Ein UML-Diagramm mit Objekten und Objektbeziehungen heißt *Objektdiagramm* (oder *Instanzendigramm*). Es stellt einen augenblicklichen Systemzustand ("Snapshot") dar.

Multiplizitäten

Geben an, wieviele Objekte einer Klasse mit wievielen Objekten einer (meist anderen) Klasse gemäß einer Assoziation in Beziehung stehen können.

Notation:

Bedeutung:



Jedes Objekt von A steht in Beziehung mit

genau einem Objekt von B



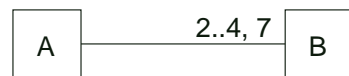
keinem oder einem Objekt von B



einem oder mehreren Objekten von B



keinem oder einem oder mehreren Objekten von B



2 bis 4 oder 7 Objekten von B

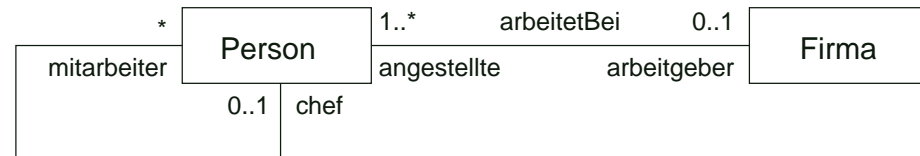
Beispiel:



Assoziationsrollen

Beschreiben, welche Funktionen die Objekte einer Klasse in einer Assoziation einnehmen.

Beispiel:



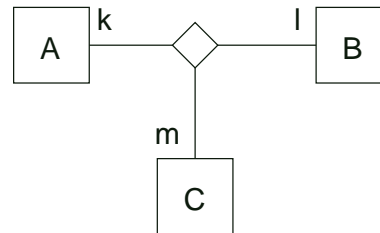
Bemerkung:

Assoziationsnamen und Rollennamen können weggelassen werden. Rollennamen sind dann implizit durch die kleingeschriebenen Klassennamen (evtl. im Plural) vorhanden.

Mehrstellige Assoziationen

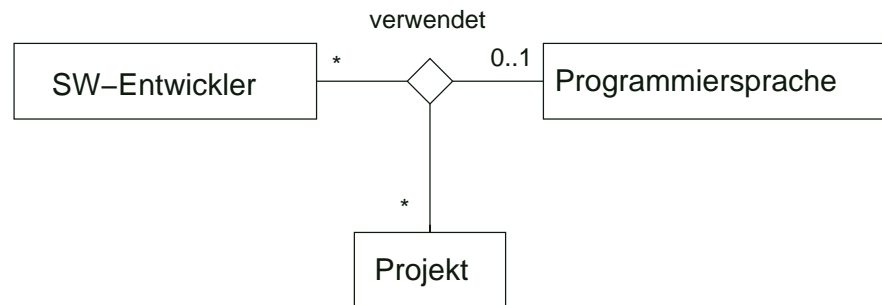
Verbinden drei oder mehr Klassen.

Darstellung:



Die Multiplizität einer Rolle in einer n-stelligen Assoziation spezifiziert, wieviele Objekte mit dieser Rolle mit fest gegebenen (fixierten) n-1 Objekten der anderen Klassen in Beziehung stehen können.

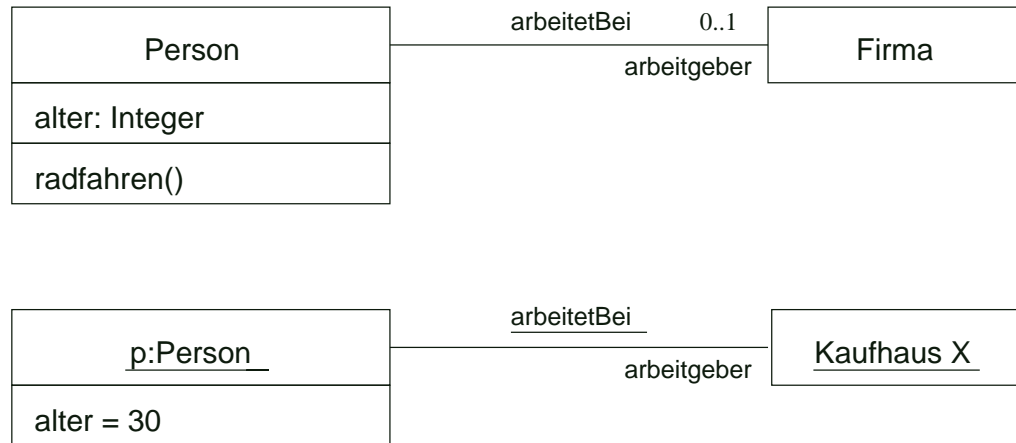
Beispiel:



Zugriff auf die Merkmale eines Objekts

Wird durch die "."-Notation ausgedrückt.

Beispiel:



```
p.alter = 30;
```

```
p.arbeitgeber = Kaufhaus X;
```

```
p.radfahren(); //Aufruf der Operation "radfahren" für das Objekt p
```

Aggregation

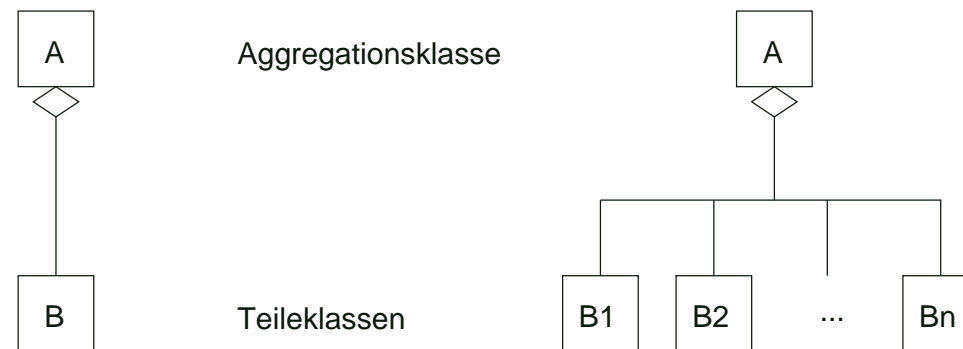
Spezielle Form der Assoziation, die eine "Gesamtheit-Teil"-Beziehung ausdrückt.

z.B.

ICE-Lok besitzt 6 Motoren (physikalisch),

Stundenplan umfasst mehrere Vorlesungen (konzeptionell)

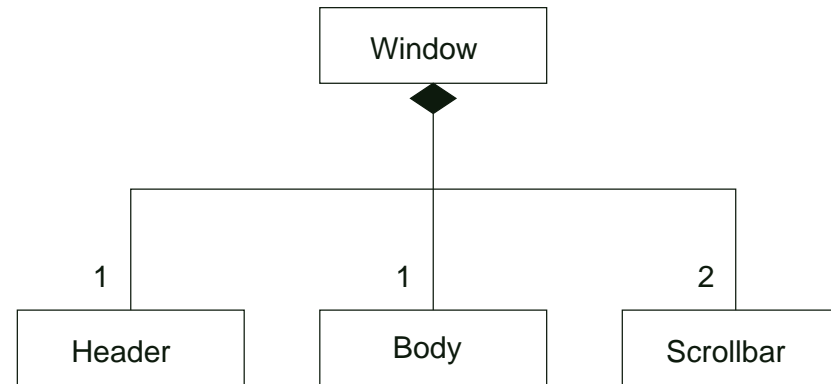
Darstellung:



Komposition

Spezielle Form der Assoziation: das Teil ist existenzabhängig vom Ganzen.

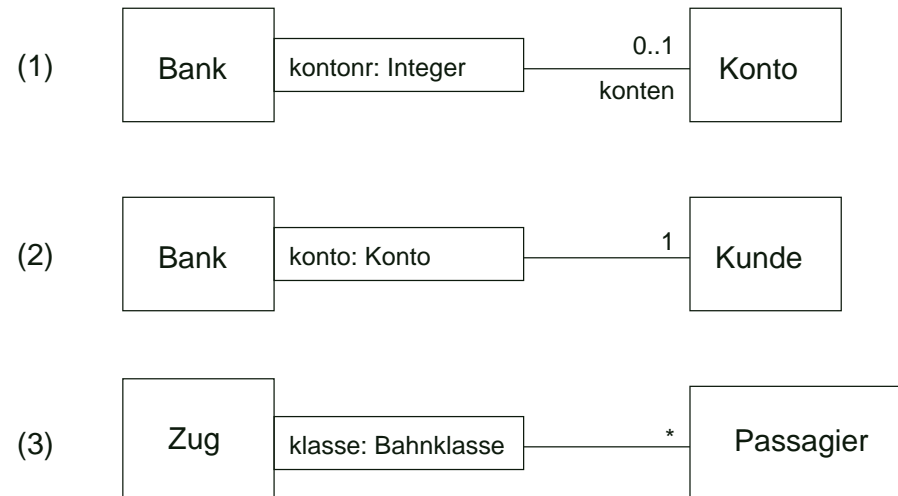
Darstellung (Beispiel):



Qualifizierte Assoziation

- Eine qualifizierte Assoziation unterteilt die Menge der Objekte auf einer Seite der Assoziation in Partitionen.
- Qualifizierer haben (wie Attribute) einen Typ, der auch eine Klasse sein kann.

Beispiele:



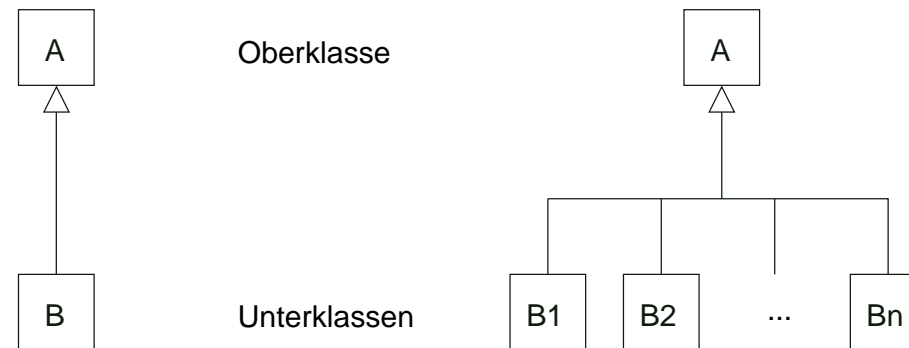
2.1.3 Vererbung

Relation zwischen einer "allgemeineren" Klasse (Ober- bzw. Superklasse) und einer "spezielleren" Klasse (Unter- bzw. Subklasse). Jedes Objekt der Subklasse ist auch ein Objekt der Oberklasse.

Beispiel:

Stoppuhr, Standuhr, Armbanduhr, Digitaluhr sind Uhren

Darstellung:

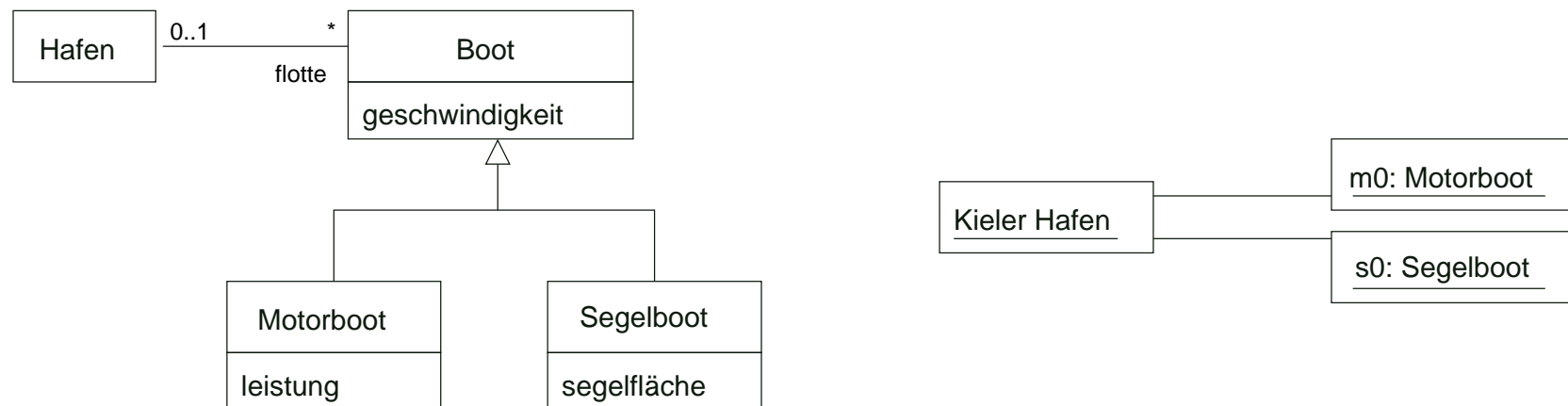


- A ist eine *Generalisierung* von B.
- B ist eine *Spezialisierung* von A.

Substitutionsprinzip

Immer wenn ein Objekt einer Oberklasse erwartet wird, kann ein Objekt einer Unterklasse von A eingesetzt werden.

Beispiel:



Jede Unterklasse besitzt (erbt) alle Attribute, Assoziationen und Operationen der Oberklasse und kann eigene hinzufügen.

Abstrakte Klasse

Klasse, von der keine Instanzen erzeugt werden können.

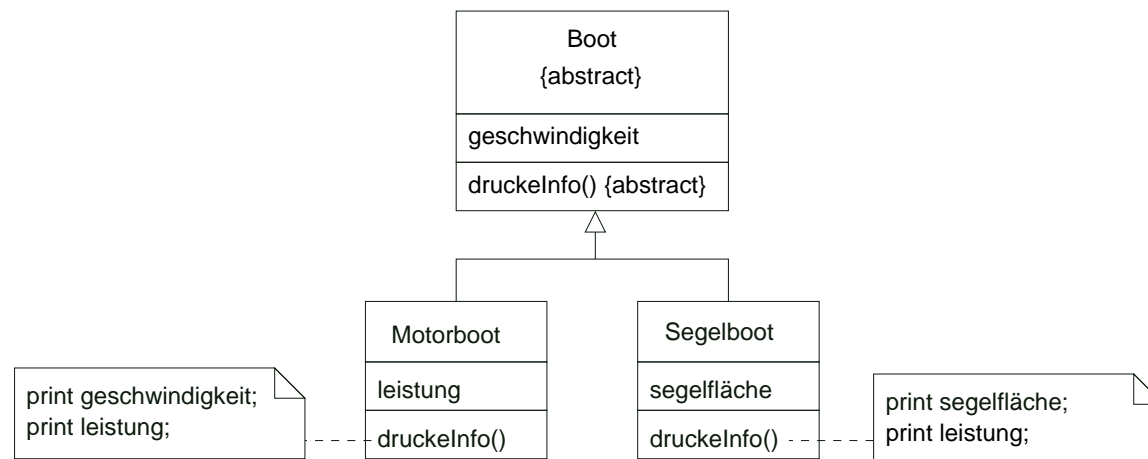
Abstrakte Operation

Operation ohne Implementierung.

Beachte:

Eine Klasse mit mindestens einer abstrakten Operation ist abstrakt.

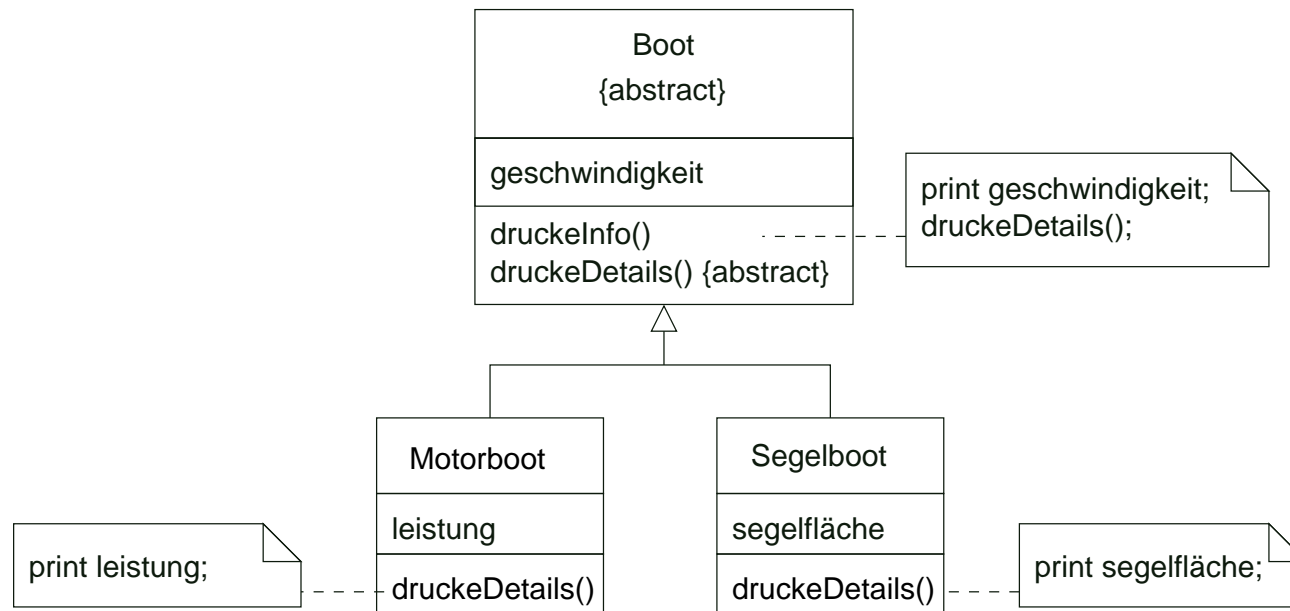
Beispiel:



Template Operation

Operation, deren Implementierung eine oder mehrere abstrakte Operationen aufruft.

Beispiel:



Überschreiben

Eine in einer Oberklasse implementierte Operation wird in einer Unterklasse neu implementiert (redefiniert).

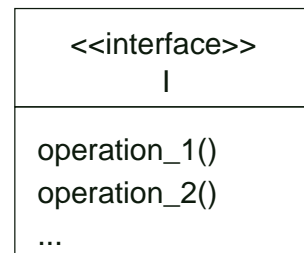
Bemerkung:

Durch Überschreiben soll die Semantik einer Operation nicht verändert werden.

Schnittstellen

Sind abstrakte Klassen, deren sämtliche Operationen abstrakt sind und die keine Attribute besitzen.

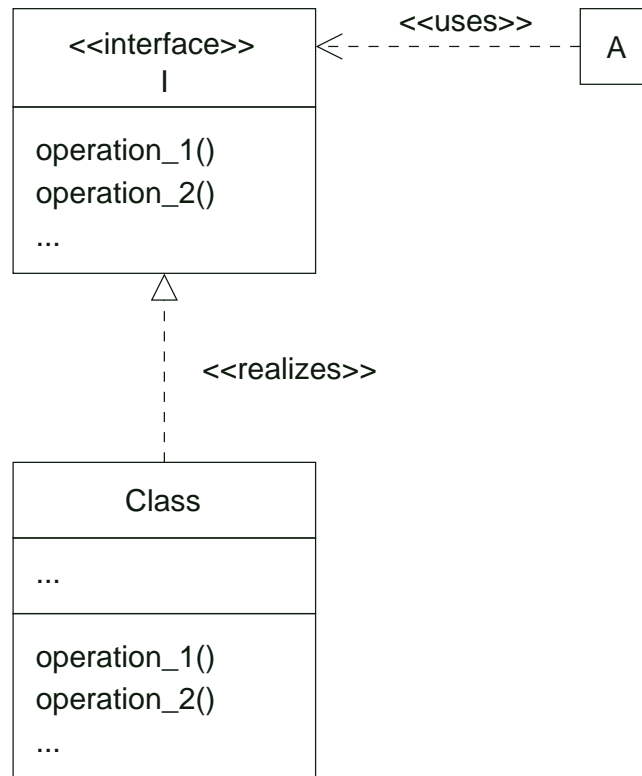
Darstellung:



Realisierung und Benutzung von Schnittstellen

Schnittstellen bieten Dienste an, die von verschiedenen Klassen realisiert (implementiert) werden können und die von anderen Klassen genutzt werden können.

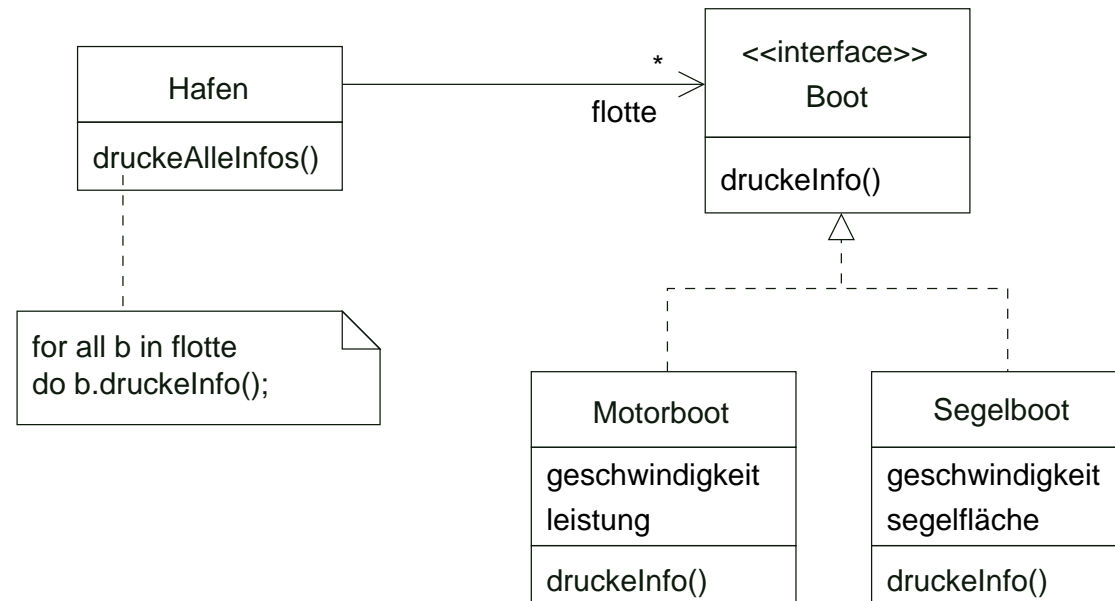
Beispiel:



(Subtyp-)Polymorphismus

Eine Operation einer Oberklasse (bzw. einer Schnittstelle) kann für alle Objekte von Unterklassen (bzw. realisierenden Klassen) aufgerufen werden.

Beispiel:



Dynamisches Binden

Beispiel:

```
Boot b; Motorboot m; Segelboot s; int x;  
... "x einlesen" ...  
if (x > 0) b = m;  
else b = s;  
(* ) b.druckeInfo();
```

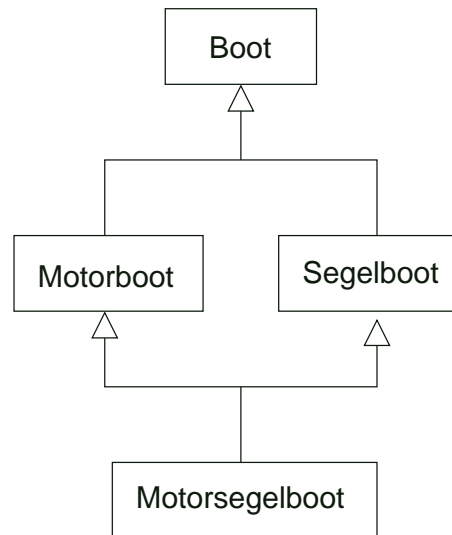
Zur Programmierzeit kann nicht festgestellt werden, welche Implementierung von "druckeInfo()" an der Stelle (*) gültig ist.

Zur Laufzeit wird festgestellt, welchen Typ das mit b bezeichnete Objekt hat. Der in der entsprechenden Klasse implementierte Code wird dann ausgeführt.

Mehrfachvererbung

Eine Subklasse hat mehr als eine (direkte) Oberklasse.

Beispiel:

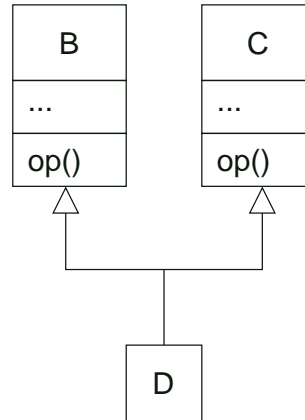


Vorteil:

Zusammenfügen von Informationen aus mehreren Quellen.

Problem:

Mögliche Konflikte



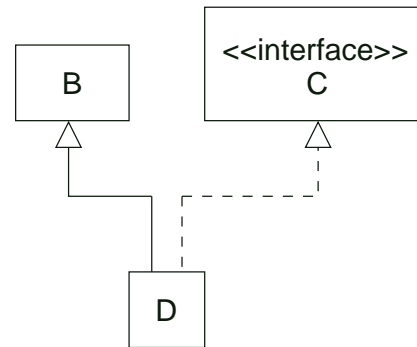
Welche Implementierung von op gilt für D-Objekte?

Konfliktauflösung:

D implementiert op neu durch Überschreiben oder op ist in B oder in C abstrakt.

Bemerkungen

- In Java ist Mehrfachvererbung nur bei Verwendung von Schnittstellen möglich.



- Mehrfachvererbung von Klassen muss beim Entwurf aufgelöst werden, wenn die Zielsprache keine Mehrfachvererbung unterstützt.

Vorteile des Vererbungsprinzips

- Konzeptionelle Vereinfachung durch Zusammenfassen gemeinsamer Merkmale verwandter Klassen in einer Oberklasse (*Generalisierung*)
- Wiederverwendung bereits vorhandener Klassen durch Subklassenbildung (*Spezialisierung*)
- Einfache Erweiterbarkeit von Vererbungshierarchien durch Hinzunahme von Subklassen (ohne Änderung der Umgebung)

Zusammenfassung von Abschnitt 2.1

- Klassendiagramme werden aus Klassen (einschl. Schnittstellen), Assoziationen und Vererbungsbeziehungen gebildet.
- Objektdiagramme werden aus Objekten und Objektbeziehungen gebildet.
- Mit Hilfe des Vererbungskonzepts kann spezialisiert und generalisiert werden.
- Es gilt das Substitutionsprinzip.
- Durch dynamische Bindung wird zur Laufzeit festgestellt, welchen Typ ein Objekt hat und der dementsprechende Code ausgeführt.

2.2 Dynamische Modellierung

Techniken

- *Interaktionsdiagramme:*
Beschreiben die Kommunikation und die Zusammenarbeit von *mehreren* Objekten.
- *Zustandsdiagramme:*
Beschreiben das Verhalten *eines* Objekts einer bestimmten Klasse während seiner Lebenszeit.
- *Aktivitätsdiagramme:*
Beschreiben den Fluss von (evtl. parallelen) Aktivitäten.

Zustände

- Die aktuellen Attributwerte und Beziehungen eines Objekts zu einem Zeitpunkt bestimmen den *aktuellen Objektzustand*.
- Objektzustände, in denen Objekte qualitativ die gleichen Reaktionen auf ankommende Ereignisse haben, sind äquivalent. Sie werden zu *einem* (abstrakten) Zustand zusammengefasst.

Beispiel: 4 Briefzustände

Standard

Kompakt

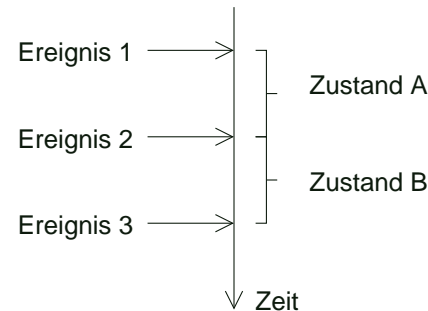
Groß

Maxi

In jedem einzelnen Zustand wird dieselbe Briefmarke aufgeklebt.

Ereignisse

Ereignis = Vorfall, der zu einem bestimmten Zeitpunkt stattfindet.



Arten von Ereignissen

- Empfang eines Signals: *signal event* (z.B. Button klicken, Telefonhörer abheben)
- Aufruf einer Operation: *call event* (z.B. `myKonto.einzahlen(1000)`)
- Eine Bedingung wird wahr: *change event* (z.B. **when** (`temperature < 0`))
- Ablauf einer Zeit: *time event* (z.B. **after**(5 sec))
- Beendigung einer Aktivität: *completion event* (z.B. WWW-Seite ist geladen)

Beachte:

Ereignisse haben keine Dauer (im Gegensatz zu Zuständen)!

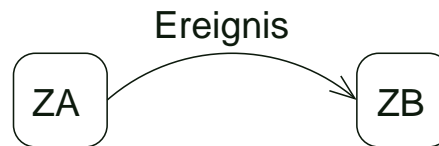
Flache Zustandsdiagramme

Gerichteter Graph mit

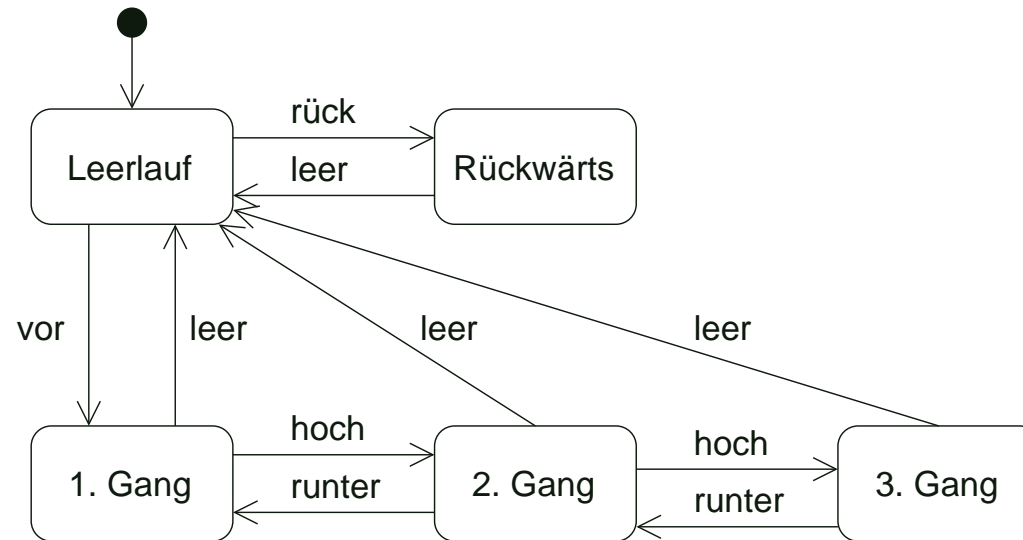
- Knoten = Zustände
- Kanten = Transitionen

Transition

Beschreibt den durch ein Ereignis verursachten Übergang von einem "alten" in einen "neuen" Zustand.



Beispiel: Automatikgetriebe



Bemerkungen

- Ein Ereignis, für das es von einem Zustand aus keine Transition gibt, wird in diesem Zustand "ignoriert" (bzw. es kann oder darf in diesem Zustand nicht auftreten, da es keine sinnvolle Verarbeitung gibt).
- Das Symbol \bullet bezeichnet den Anfangszustand ("Pseudozustand").
- Das Symbol \odot bezeichnet einen Endzustand (zumeist nach Beendigung eines Ablaufs).

Wächter (*guards*)

- Eine Bedingung (boolescher Ausdruck) kann als Wächter für eine Transition verwendet werden.
- Die Transition feuert, wenn das Ereignis eintritt *und* die Bedingung erfüllt ist.

Syntax:



Aktion

- Atomare, *nicht unterbrechbare* Tätigkeit.
- Kann als Reaktion auf ein Ereignis erfolgen.

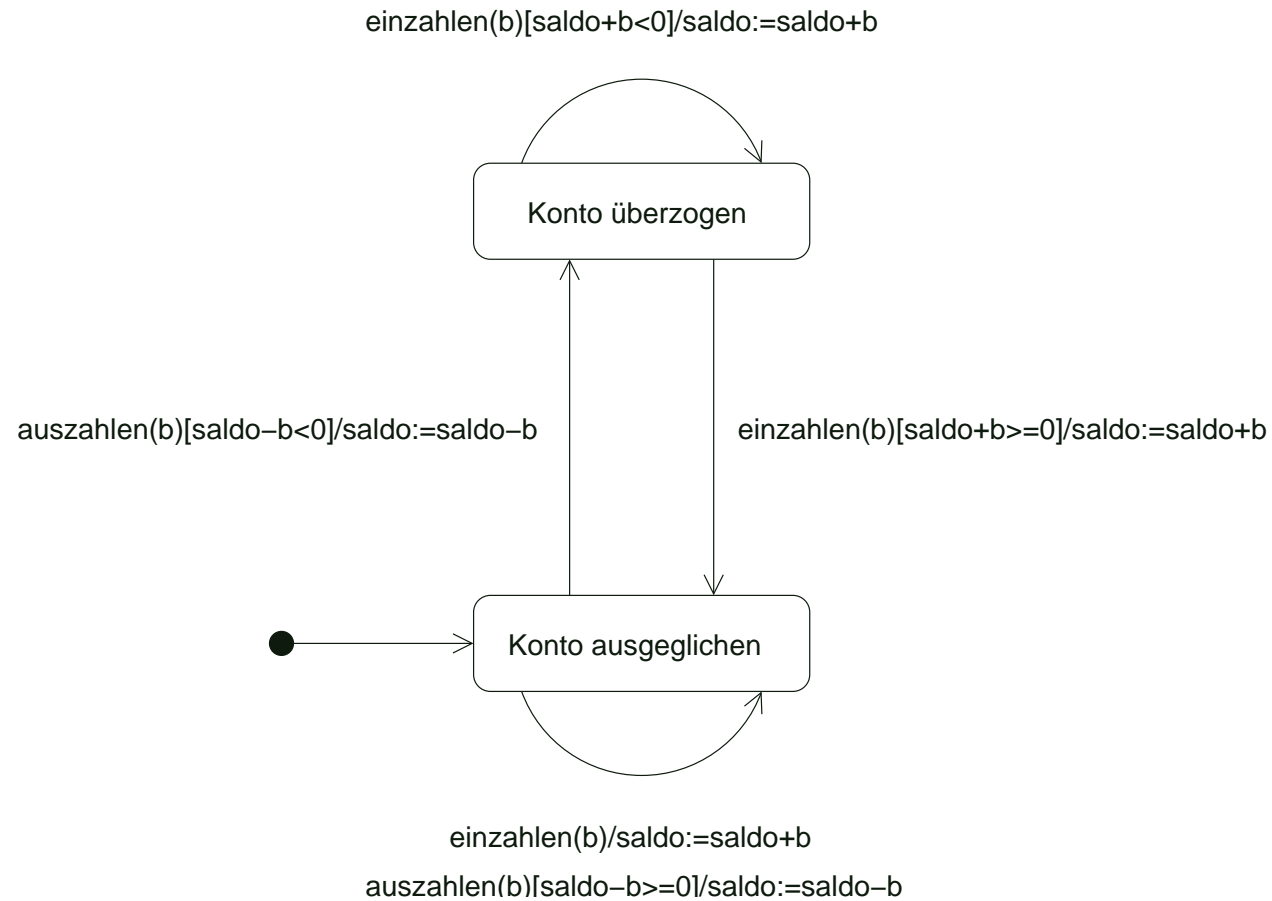
Syntax:



Beachte:

Aktionen werden meist in der Form von Statements (Anweisungen) geschrieben.

Beispiel: Wächter und Aktionen



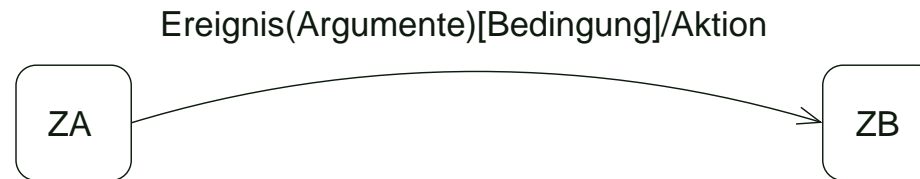
Insbesondere gibt es:

- *Sendeaktionen:*
Zielobjekt.Ereignis(Argumente)
- *zusammengesetzte Aktionen:*
Aktion1; Aktion2

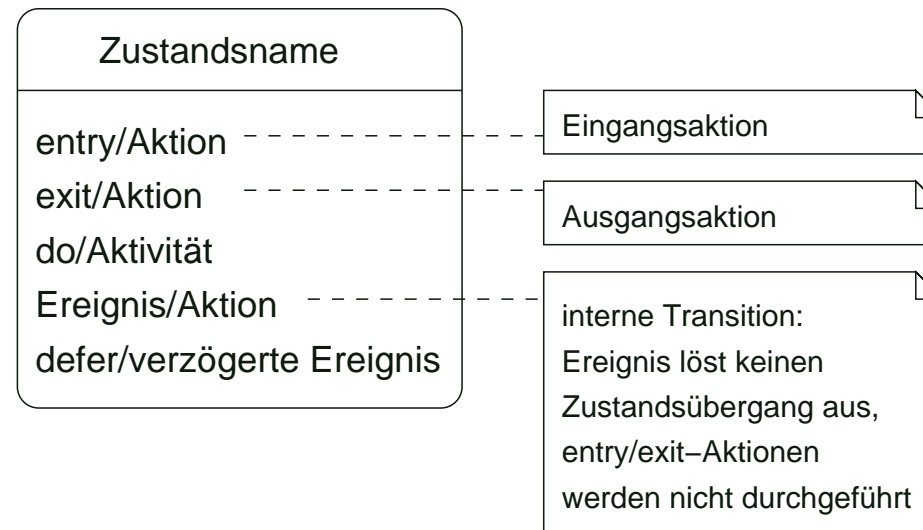
Beachte:

Eine zusammengesetzte Aktion ist weiterhin atomar.

Allgemeine Syntax von Transitionen



Allgemeine Syntax von Zuständen

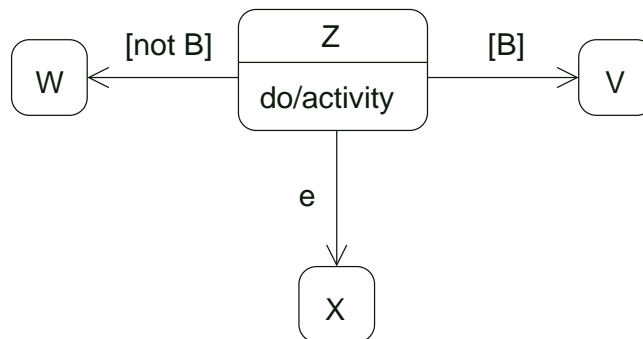


Eingangs-/Ausgangsaktionen

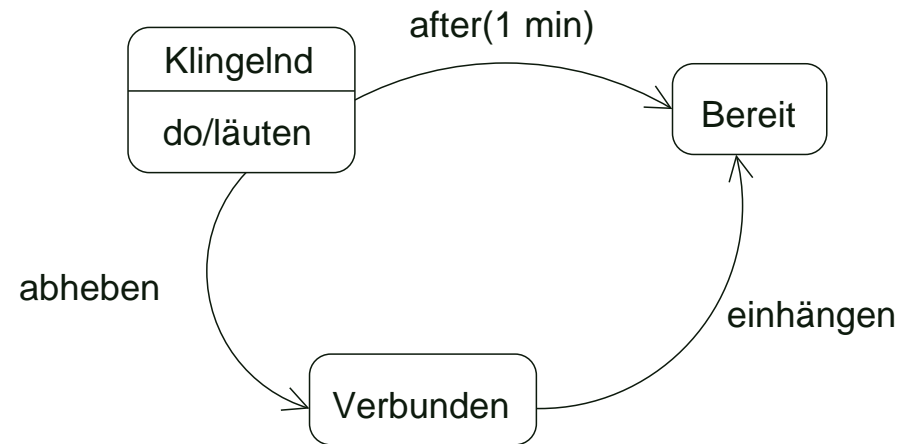
Werden jeweils beim Eintritt bzw. Verlassen eines Zustands durchgeführt.

Aktivität

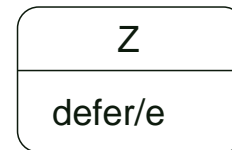
- Tätigkeit, die Zeit in Anspruch nimmt und durch ein Ereignis unterbrochen werden kann.
- Wird von einem Objekt in einem bestimmten Zustand ausgeführt.
- Wird die Aktivität von selbst beendet, dann erfolgt ein Completion Event.



Beispiel: Telefon

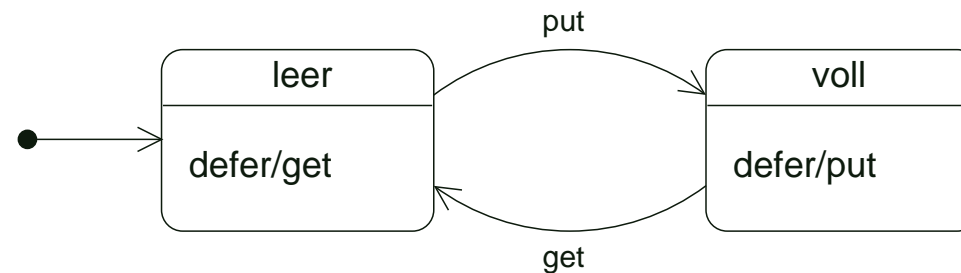


Verzögertes Ereignis



Erfolgt das Ereignis e im Zustand Z , dann wird es in eine Warteschlange des Objekts für eingehende Ereignisse ("event queue") gestellt und erst dann verarbeitet, wenn ein Zustand erreicht ist, in dem es eine Transition mit diesem Ereignis gibt.

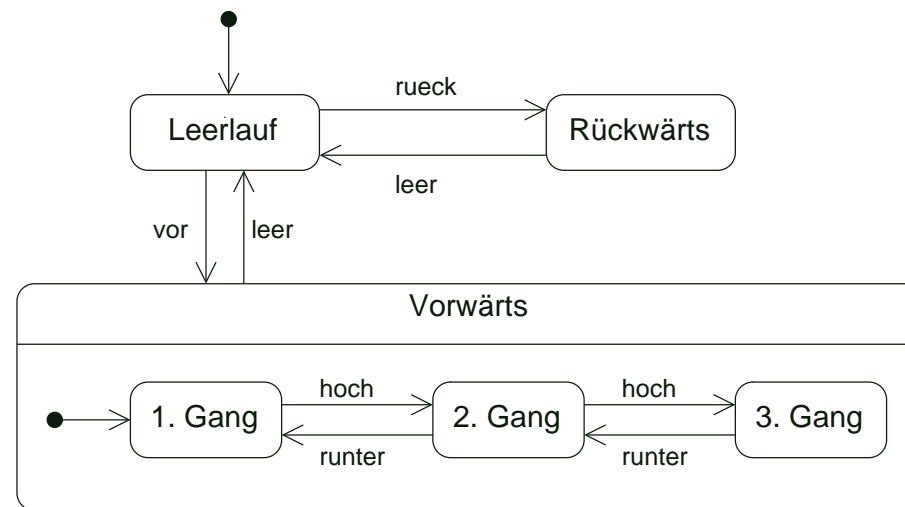
Beispiel: Einelementiger Puffer



Hierarchische Zustandsdiagramme

Ein Zustand kann in Unterzustände verfeinert werden.

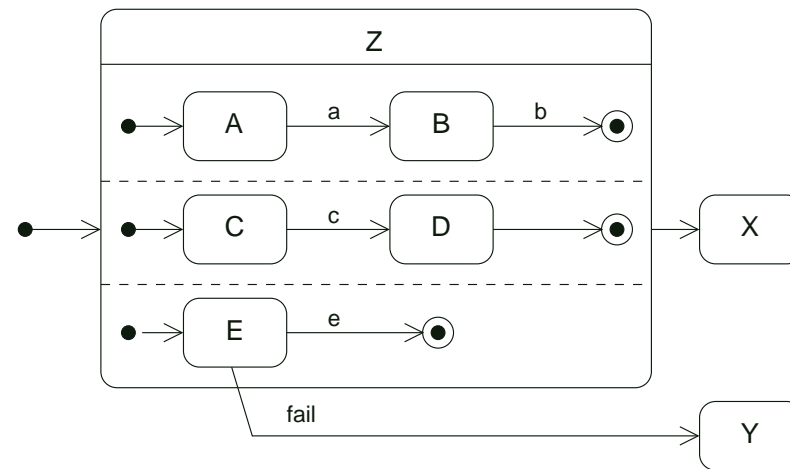
1. *Sequentielle Unterzustände*



- Transition in einen Oberzustand (hier: Vorwärts) bedeutet Transition in den Anfangszustand des geschachtelten Diagramms (hier: 1. Gang).
- Transition aus einem Oberzustand bedeutet Transition ausgehend von jedem Unterzustand.

Vorteil: Übersichtlichere Darstellung, Abstraktion

2. Parallele Unterzustände



- Ein Objekt befindet sich gleichzeitig in mehreren Unterzuständen. Beim Eintritt in den Oberzustand befindet sich das Objekt gleichzeitig in den Anfangszuständen der einzelnen (parallelen) Regionen.
- Der Oberzustand wird verlassen, wenn in jeder Region ein Endzustand erreicht ist oder wenn eine Transition direkt nach außen führt.

Aktivitätsdiagramme

Spezielle Zustandsdiagramme, in denen (fast) nur

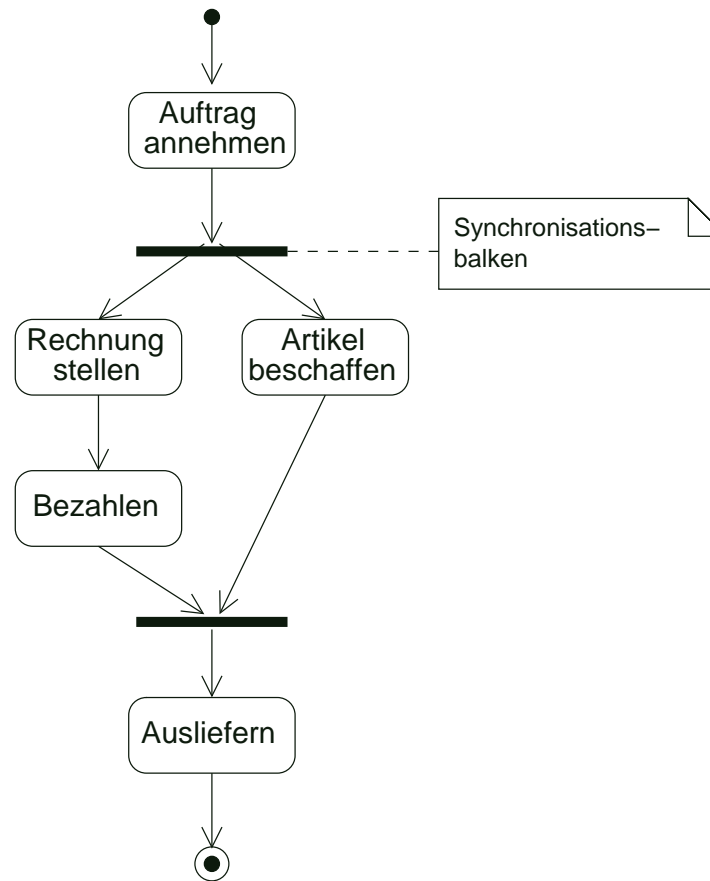
- Zustände, die Aktivitäten repräsentieren und
 - Completion Events, die den Abschluss einer Aktivität anzeigen (evtl. mit Bedingung)
- vorkommen.

Können zur Beschreibung der Abläufe von

- Geschäftsprozessen in Unternehmen
- Anwendungsfällen (Use Cases) oder von
- Operationen und Threads

verwendet werden.

Beispiel: Geschäftsprozess "Auftragsbearbeitung"



Zusammenfassung von Abschnitt 2.2

- Zustandsdiagramme beschreiben das Verhalten eines Objekts einer bestimmten Klasse während seiner Lebenszeit.
- Eine Transition beschreibt den durch ein Ereignis verursachten Zustandsübergang.
- Ereignisse sind im Gegensatz zu Zuständen zeitlos.
- Auf ein Ereignis kann eine (als zeitlos idealisierte) Aktion folgen.
- Aktivitäten dauern eine Zeit und werden von einem Objekt in einem bestimmten Zustand ausgeführt ("action state").
- Zustandsdiagramme können hierarchisch strukturiert werden.
Wir unterscheiden
 - sequentielle Unterzustände
 - parallele Unterzustände
- Aktivitätsdiagramme beschreiben Abläufe. Die Zustände repräsentieren Aktivitäten.