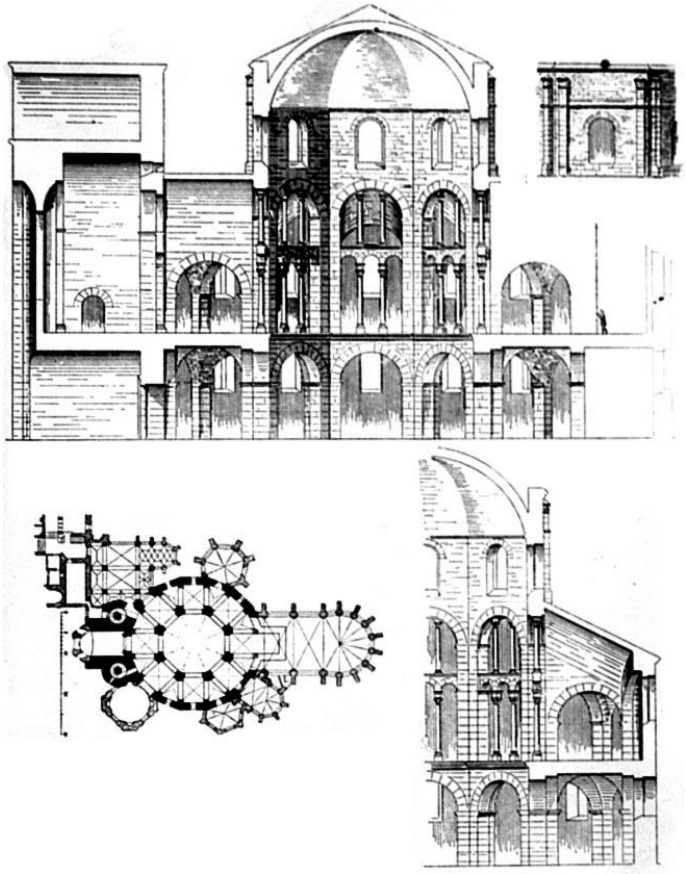
Entwurf

Prof. Sven Apel

Universität des Saarlandes

Basierend auf Material von Andreas Zeller und Norbert Siegmund



Teil I

Objektorientierter Entwurf

Was ist ein Objekt?

Ein Objekt bietet eine Sammlung von Diensten (Methoden), die auf einem gemeinsamen Zustand arbeiten.

Typischerweise entsprechen

Objekte: Gegenständen aus der Aufgabenstellung
(„ Kunde”, „Lieferant“, „Rechnung“)

Methoden: Verben aus der Aufgabenstellung
(„bestellen“, „bezahlen“, „stornieren“)

Objektorientierte Modellierung

Objektorientierte Modellierung mit UML umfasst u.a. folgende Aspekte des Entwurfs:

Objekt-Modell:

Welche Objekte benötigen wir?

Welche Merkmale besitzen diese Objekte? (Attribute, Methoden)

Wie lassen sich diese Objekte klassifizieren? (Klassenhierarchie)

Welche Assoziationen bestehen zwischen den Klassen?

Sequenzdiagramm: Wie wirken die Objekte global zusammen?

Zustandsdiagramm: In welchen Zuständen befinden sich die Objekte?

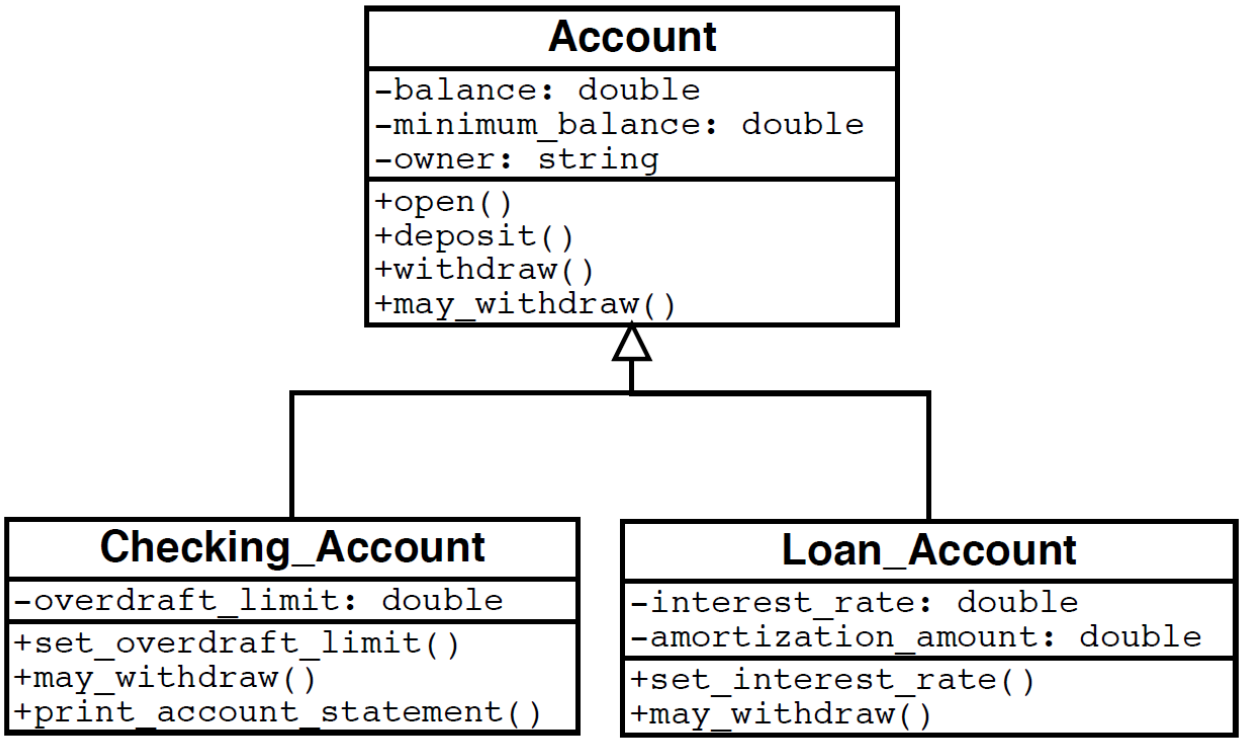
Objekt-Modell: Klassendiagramm

Darstellung der Klassen als Rechtecke, unterteilt in:

- Klassenname
- Attribute – möglichst mit Typ (meistens ein Klassenname)
- Methoden – möglichst mit Signatur

Darstellung der Vererbungshierarchie – ein Dreieck (Symbol: △) verbindet Oberklasse und Unterklassen

Beispiel für Vererbung: Bankkonten



Ererbte Methoden (wie hier: `open()`, `deposit()`) werden in der Unterklasse nicht mehr gesondert aufgeführt.

Definitionen in der Unterklasse überschreiben die Definition in der Oberklasse (hier: `may_withdraw()`)

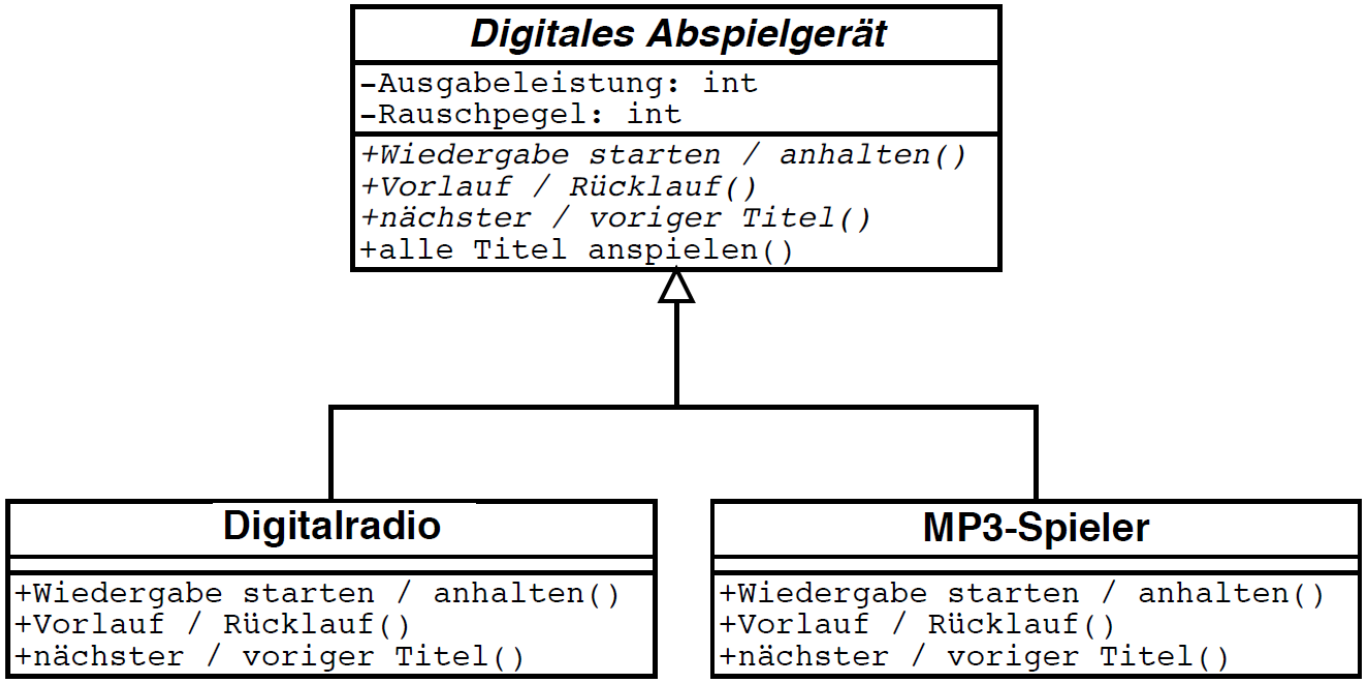
Abstrakte Klassen und Methoden

Klassen, von denen *keine konkreten* Objekte erzeugt werden können, heißen *abstrakte* Klassen. Sie verfügen meist über eine oder mehrere abstrakte Methoden, die erst in Unterklassen realisiert werden.

Eine *konkrete* Klasse ist eine Klasse, von der *konkrete* Objekte erzeugt werden können.

Beispiel für abstrakte Klasse

Ein „Digitales Abspielgerät“ ist ein abstrakter Oberbegriff für konkrete Realisierungen – z.B. ein CD- oder ein MP3-Spieler.



Kursiver Klassen-/Methodenname: *abstrakte* Klasse/Methode

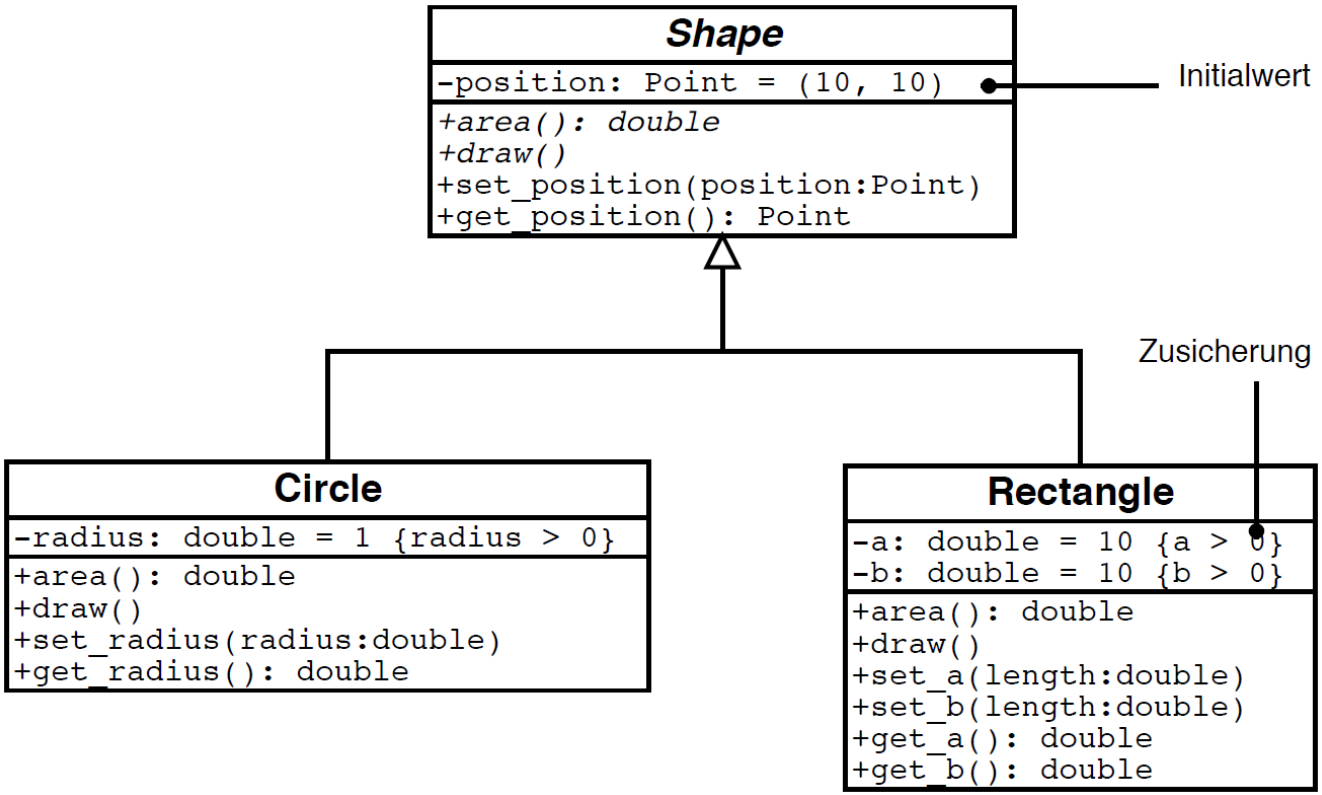
Initialwerte und Zusicherungen

Die Attribute eines Objekts können mit *Initialwerten* versehen werden. Diese gelten als *Vorgabe*, wenn bei der Konstruktion des Objekts nichts anderes angegeben wird.

Mit *Zusicherungen* werden *Bedingungen* an die Attribute spezifiziert. Hiermit lassen sich *Invarianten* ausdrücken – Objekt-Eigenschaften, die stets erfüllt sein müssen.

Beispiel für Zusicherungen

Die Zusicherungen garantieren, dass Kreise immer einen positiven Radius haben und Rechtecke positive Kantenlängen.



Objekt-Modell: Assoziationen

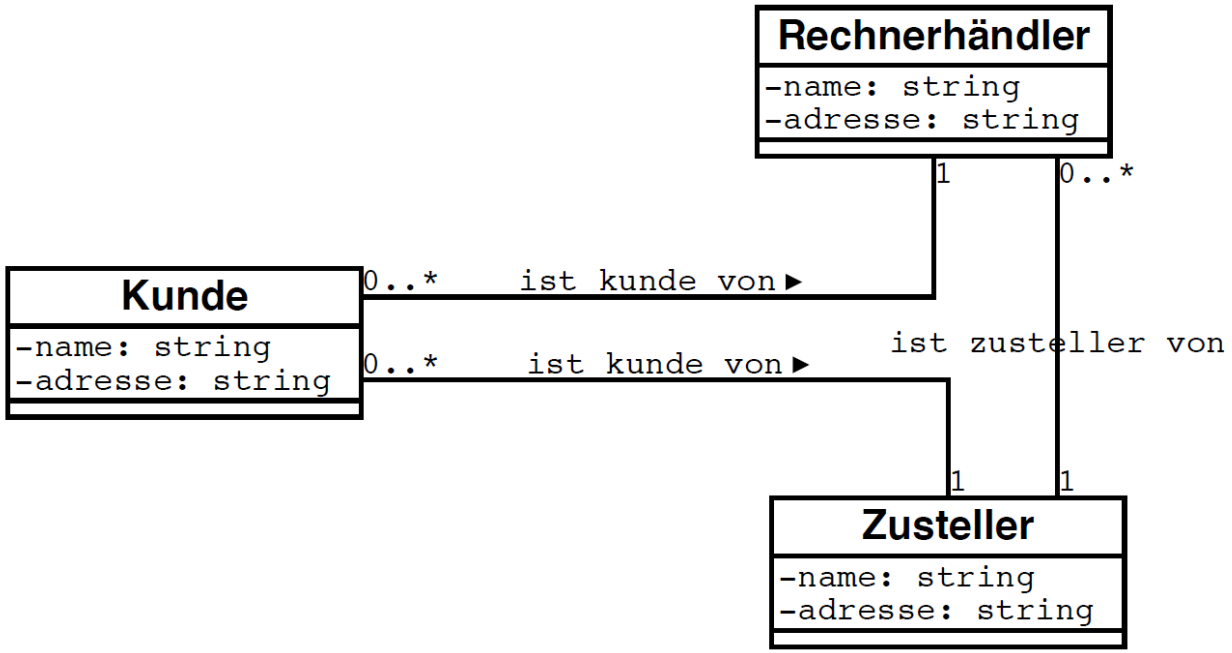
Verbindungen zwischen nicht-verwandten Klassen stellen *Assoziationen* (Relationen) zwischen diesen Klassen dar

Eine *Assoziation* beschreibt den inhaltlichen Zusammenhang zwischen Objekten (vgl. Datenmodellierung!)

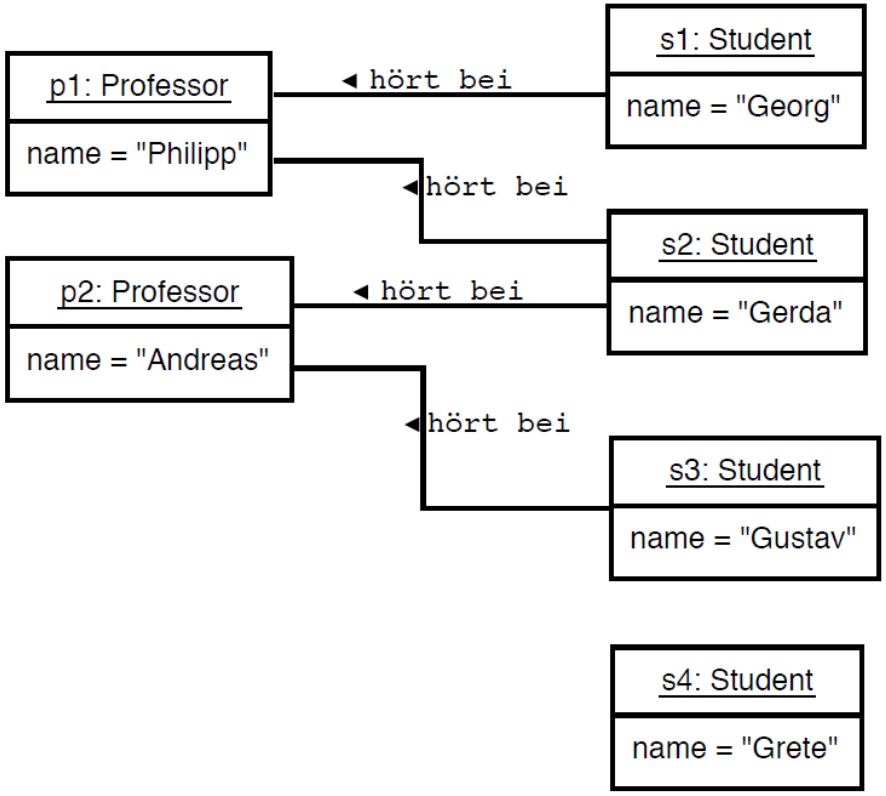
Durch *Multiplizitäten* wird die Anzahl der assoziierten Objekte eingeschränkt.

Multiplizitäten

Beispiel: Ein *Rechnerhändler* hat mehrere *Kunden*, ein *Zusteller* hat mehrere Kunden, aber ein Rechnerhändler hat nur einen Zusteller



Beispiel für Objektbeziehungen



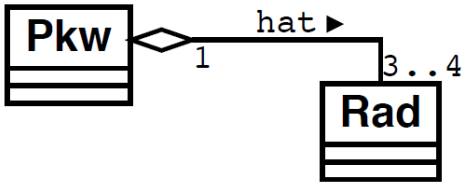
Darstellung von Exemplaren (Objekten) mit unterstrichenem Objektnamen; konkrete Werte für die Attribut

Aggregation

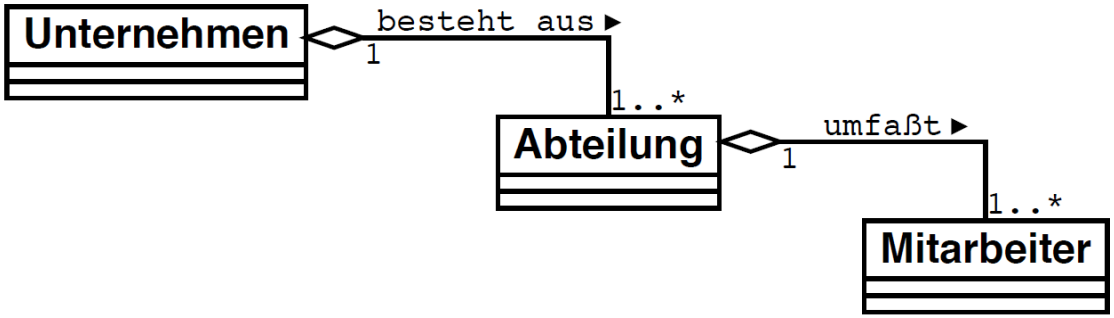
Eine häufig auftretende und deshalb mit ◊ besonders markierte Assoziation ist die *hat-Beziehung*, die die Hierarchie zwischen einem Ganzen und seiner Teile ausdrückt.

Beispiel:

Ein Pkw hat 3–4 Räder



Beispiel: Ein Unternehmen hat 1..* Abteilungen mit jeweils 1..* Mitarbeitern



Aggregation

Ein Aggregat kann (meistens zu Anfang) auch ohne Teile sein: die Multiplizität 0 ist zulässig. Gewöhnlich ist es jedoch Sinn eines Aggregats, Teile zu sammeln.

Bei einem Aggregat *handelt das Ganze stellvertretend für seine* Teile, d.h. es übernimmt Operationen, die dann an die Einzelteile weiterpropagiert werden.

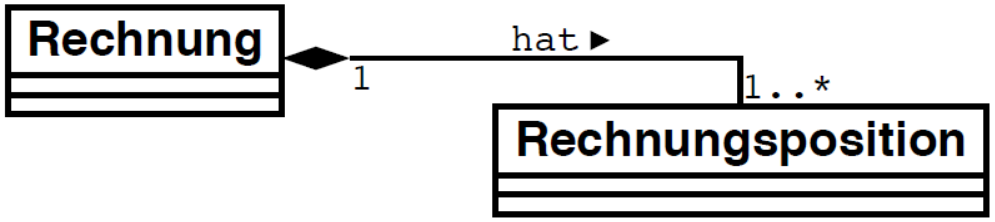
Beispiel: Methode `berechneUmsatz()` in der Klasse `Unternehmen`, die den Umsatz der Abteilungen aufsummiert.

Komposition

Ein Sonderfall der Aggregation ist die *Komposition*, markiert mit ♦.

Eine Komposition liegt dann vor, wenn das Einzelteil vom Aggregat *existenzabhängig* ist – also nicht ohne das Aggregat existieren kann.

Beispiel: Eine Rechnungsposition gehört immer zu einer Rechnung.



Sequenzdiagramme

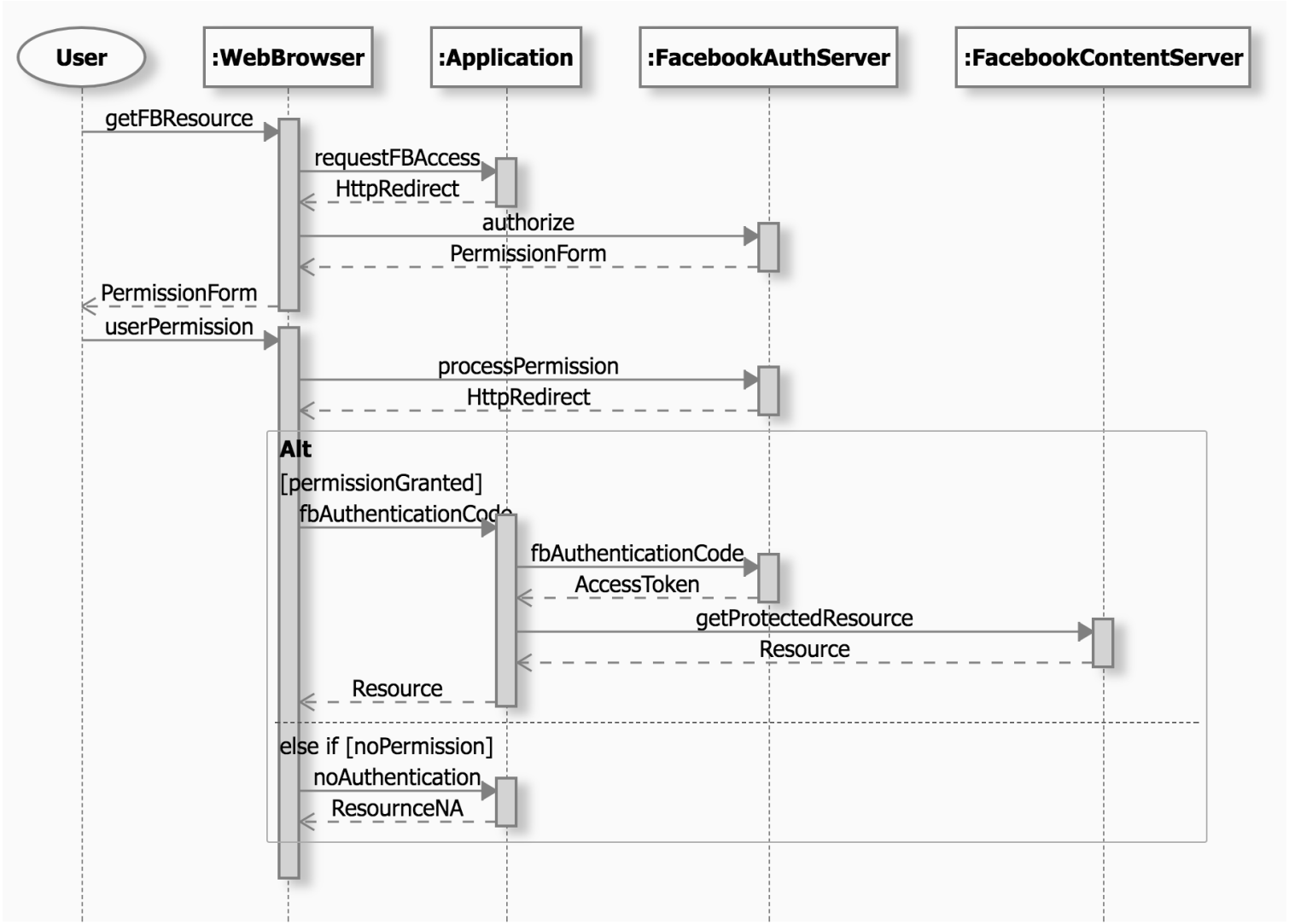
Ein *Sequenzdiagramm* gibt den Informationsfluss zwischen einzelnen Objekten wieder mit Betonung des *zeitlichen Ablaufs*.

Objekte werden mit senkrechten *Lebenslinien* dargestellt; die Zeit verläuft von oben nach unten.

Der *Steuerungsfokus* (breiter Balken) gibt an, welches Objekt gerade aktiv ist.

Pfeile („Nachrichten“) kennzeichnen *Informationsfluss* – z.B. Methodenaufrufe (durchgehende Pfeile) und Rückkehr (gestrichelte Pfeile).

Beispiel: Web Browser



Zustandsdiagramme

Ein Zustandsdiagramm zeigt eine Folge von *Zuständen*, die ein Objekt im Laufe seines Lebens einnehmen kann und aufgrund welcher *Ereignisse* *Zustandsänderungen* stattfinden.

Ein Zustandsdiagramm zeigt einen *endlichen Automaten*.

Zustandsübergänge

Zustandsübergänge werden wie folgt notiert:

Ereignisname [*Bedingung*] / *Aktion*

Hierbei ist

Ereignisname: der Name eines Ereignisses (typischerweise ein Methodenaufruf)

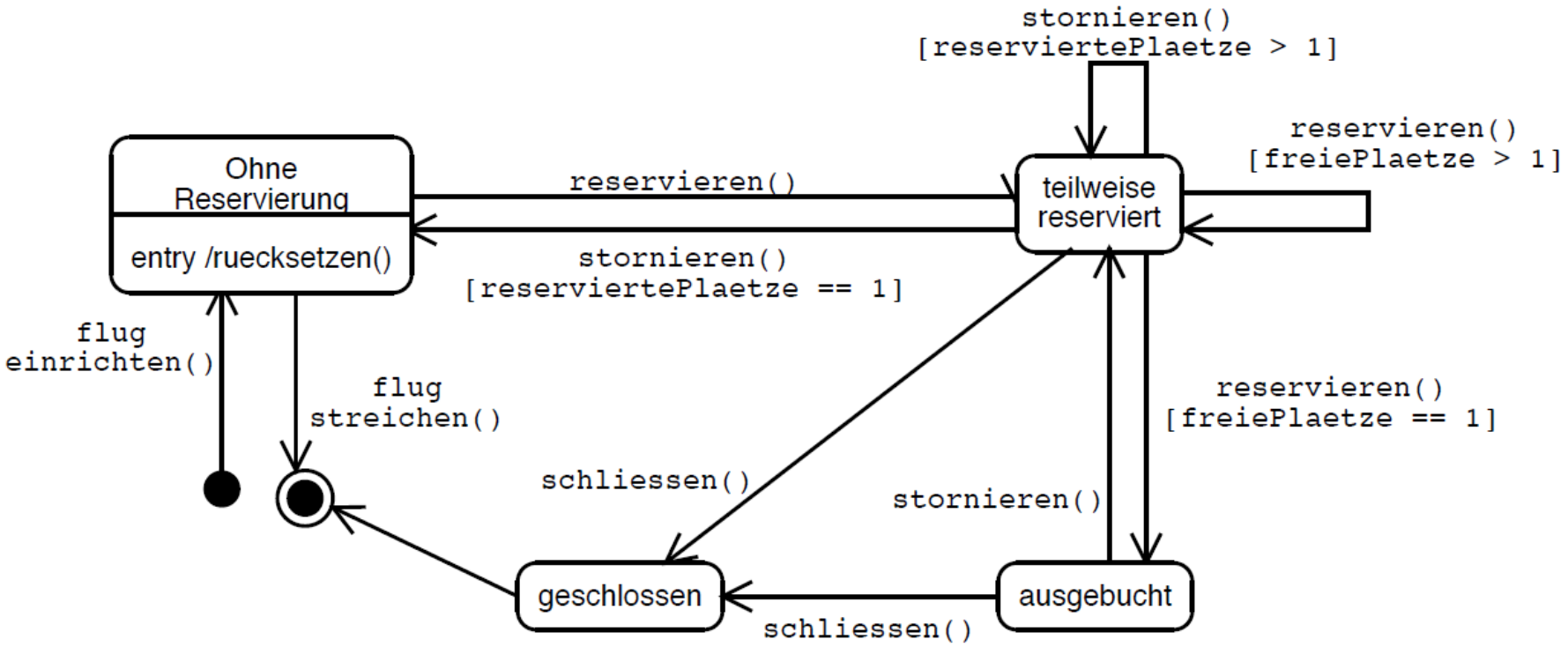
Bedingung: die Bedingung, unter der der Zustandsübergang stattfindet (optional)

Aktion: eine Aktion, die beim Übergang ausgeführt wird (optional)

Auch Zustände können mit Aktionen versehen werden: Das Ereignis `entry` kennzeichnet das *Erreichen* eines Zustands; `exit` steht für das *Verlassen* eines Zustands.

Beispiel: Flugreservierung

Wird ein Flug eingerichtet, ist noch nichts reserviert. Die Aktion `ruecksetzen()` sorgt dafür, dass die Anzahl der freien und reservierten Plätze zurückgesetzt wird.



Finden von Klassen und Methoden

„*Wie finde ich die Objekte?*“ ist die schwierigste Frage der Analyse.

Es gibt keine eindeutige Antwort: Man kann ein Problem auf verschiedene Weisen objektorientiert modellieren.

Anwendungsfälle helfen

- Beschreiben von *typischen* Szenarien in Anwendungsfällen
- Extrahieren der *zentralen* Klassen und Dienste aus den Anwendungsfällen

Anwendungsfälle (Use Cases)

Ein *Anwendungsfall* beschreibt wie ein *Akteur* („HandelInder“) zu seinem *Ziel* kommt.

Welche Akteure gibt es?

Welche Ziele haben die Akteure?

Definitionen

Ein *Akteur* ist etwas, das sich verhalten kann

- Eine Person, ein System, eine Organisation, …

Ein *Szenario* ist eine Folge von Aktionen und Interaktionen zwischen Akteuren

Ein *Anwendungsfall* ist eine Sammlung von verwandten Szenarien

- Bestehend aus *Erfolgsszenario* und *Alternativszenarien*

Beispiel: Rechnerversand

Studentin Dagmar bestellt mit einem Brief bei der Firma Rechnerwelt einen Rechner. Dieser wird ihr nach mehreren Tagen als Paket durch die Firma Gelbe Post zugestellt.

Wer sind die Akteure?

Welche Ziele verfolgen sie?

Was kann alles fehlschlagen?

Alternative Szenarien

Sind meist spannender als Erfolgsszenarien

Beispiel: Parkhaus

Kein Zugang • Karte nicht • lesbar Stromausfall…

Beispiel: Geld abheben

Konto leer • falsche PIN • falsches Konto…

Bespiel: Flug buchen

Flug ausgebucht • Stornierung…

Beispiel: Rechnerversand

Studentin Dagmar bestellt mit einem Brief bei der Firma Rechnerwelt einen Rechner. Dieser wird ihr nach mehreren Tagen als Paket durch die Firma Gelbe Post zugestellt.

Alternative Szenarien

- Bestellung trifft nicht ein
- Rechner nicht lieferbar
- Paket nicht zustellbar
- …

Anwendungsfälle Rechnerversand