4 Analyse

4.1 Grundlagen

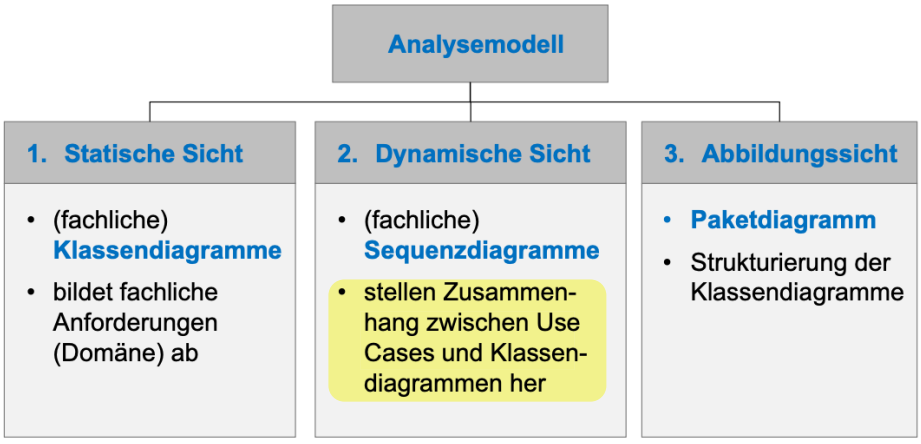
Aufgaben der Analyse

- **Ziel: Anforderungen detaillierter ausarbeiten**
  - verständlich für Fachabteilung, präzise für Entwickler*innen **(interne Sicht!)**
  - Analyse setzt auf Ergebnisse der Anforderungsanalyse auf
  - iterativ/ inkrementelles Vorgehen bei UP
    - *es werden nur die Use Cases der aktuellen Iterationsstufe analysiert (d.h. nur deren notwendige Klassen spezifiziert)*
    - *nachfolgende Iterationen führen zu Erweiterungen und ggf. Änderungen der Analyseergebnisse*
- zentrale Aspekte:
  a) **statische Modellierung**(Klassendiagramme)
     a. Modell der Gegenstände, Beziehungen und Operationen
  b) **dynamische Modellierung** (Sequenz-, Kommunikationsdiagramme)
     a. Modell der Abläufe und Wechselwirkungen

4.2 Ergebnisse

Ergebnisse der Analyse im Überblick

- Gesamtheit der Ergebnisse wird als **Analysemodell** (Spezifikation, Fachkonzept) bezeichnet

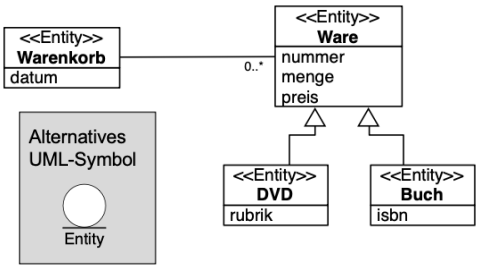


4.2.1 Statische Sicht: UML-Klassendiagramm (Class Diagram)

- Ziel: Modell der Gegenstände/Konzepte der fachlichen Domäne, deren Beziehungen und Operationen
- Modellierung der statischen Sicht durch **UML-Klassendiagramme**
  - Modelle enthalten ausschließlich **fachliche** Klassen
  - zeigen welche Klassen und Methoden es gibt aber nicht deren Zusammenspiel

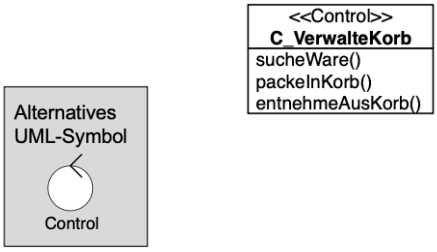
Klassentypen

1. Entity-Klassen (Entitätsklassen)
- repräsentieren **Objekte** der realen Welt, z.B. Person, Kunde, Adresse, Vertrag, Buch
- modellieren **Daten** (= Attribute und Beziehungen), die i.d.R. **persistent** abgespeichert werden
- bieten Methoden für konsistenten Zugriff auf Daten (= Attribute)



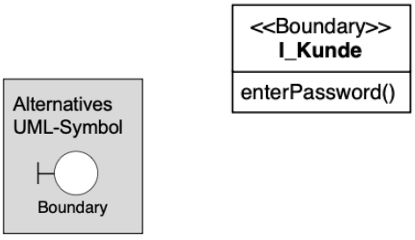
- Entities sind datentragend, EnthV§lt fachliche Logik
- Methoden bieziehen sich nur auf Daten

2. Control-Klassen (Steuerungsklassen)
- bieten nur **Dienste** (Geschäftslogik / fachliche Operationen) an, d.h. keine (Daten-)Attribute
- auch Methoden, die keiner Entity zugeordnet werden können (Entity-/Klassen-übergreifend)
- realisieren komplexe **Abläufe**, nutzen hierzu mehrere Entity- sowie andere Control-Klassen



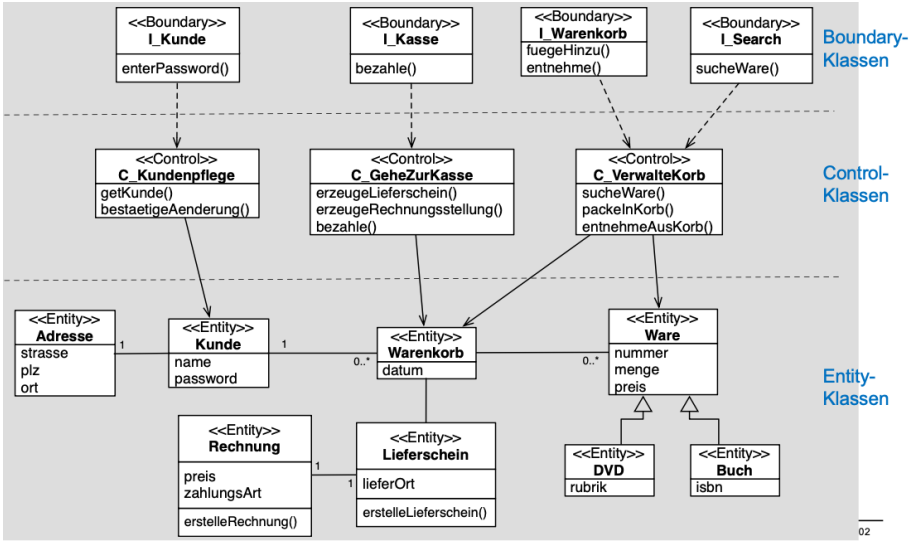
- Schnittstelle zwischen Fachlichkeit und GUI / anderes System, also die Akteure die damit interagieren
- **Ziel: Logik wird verbirgt. Damit kann man Logik ändern ohne die Schnittstelle zu ändern**

3. Boundary-Klassen (Anwenderschnittstellenklassen)
- beinhalten **Operationen** für Interaktion zwischen Akteur und System
- bilden **fachliche Schnittstelle** (stellen fachl. Operationen bereit), d.h. enthalten selbst **keine** Logik, Funktionalität, Abläufe
  - Akteur darf **nur** über Boundary-Klassen mit System kommunizieren
  - delegieren Aufrufe an Control-Klassen



- Schnittstelle & Implementierung getrennt
  (Wenig Redundanz, aber die ist gewollt. I_Klasse bezahle() ruft C_GeheZurKasse bezahle() auf)

Internet-Buchhandlung (nur Ausschnitt !) (i.w. Klassen für Kunden-bezogene Use Cases)



4.2.2 Dynamische Sicht: UML-Sequenzdiagramm

dynamische Modelle zeigen wie Klassen bzw. Objekte zusammenarbeiten, um die Abläufe in den Use Cases zu realisieren

- **Szenario** = Abwicklung eines Anwendungsfalls (= konkrete Ereignisfolgen)
  - **Instanz** eines Use Cases (z.B. "*Fred Schneider kauft Buch xyz*")
  - beschreibt **dynamisches Verhalten** von beteiligten **Objekten**
  - besteht aus Abfolge von **Methodenaufrufen**
  - pro Use Case zunächst Standardszenarios betrachten

- Ziel: „Durchspielen" der Use Cases in Form von **Szenarios**

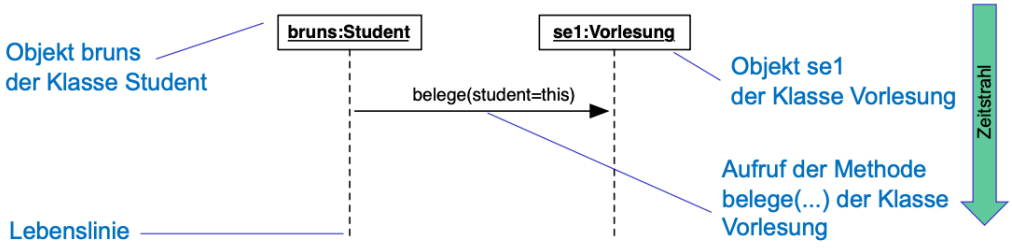
- Zusammenhang: **Szenario** und **Klassendiagramm**
  - alle benutzten Methoden muss es in Klassendiagramm geben
  - nur einander bekannte Objekte können Methoden aufrufen
  - zwischen Objekten werden alle notwendigen Daten ausgetauscht

2. Dynamische Sicht: UML-Sequenzdiagramm (sequence diagram)

- Sequenzdiagramm muss mit Klassendiagramm konsistent sein

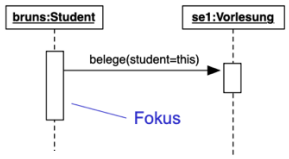
- **Aufbau dynamischer Modelle**
  - spielen genau ein Szenario (eines Use Cases) durch
  - als Abfolge von Methodenaufrufen zwischen konkreten Objekten
    - enthalten die am Use Case partizipierenden **Objekte**
  - **Akteur** (aus Use Case) kann ein Szenario initialisieren

- modelliert **Interaktionen** zwischen **Objekten**
- ein Objekt ruft eine Methode eines anderen (ihm bekannten) Objektes auf

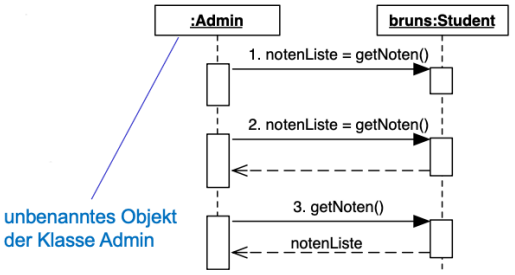


Notation

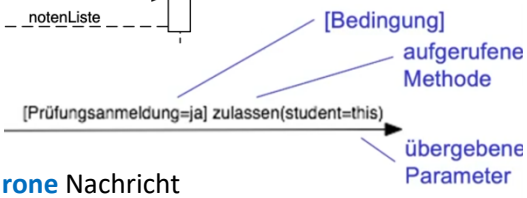
- **Fokus** (Aktionssequenz) - Box
  - zeigt an, wie lange eine Methode abgearbeitet wird
  - verdeutlicht Kaskadierung von Methoden



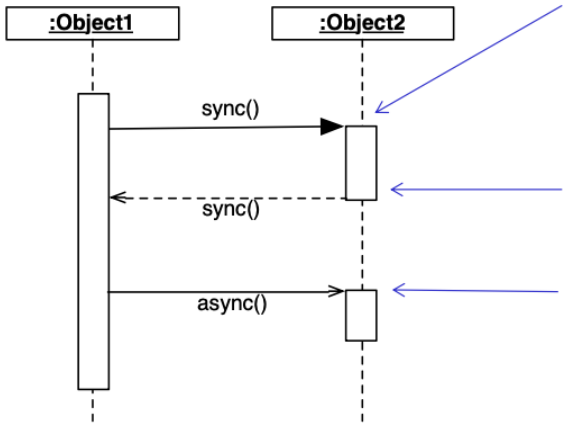
- **Rückgabewert** (reply message)
  1. In Textform an Pfeil
  2. als eigener Pfeil
  3. Mit Variable/Wert an Pfeil



- **Bedinungen (guard)** an Pfeile der Methodenaufrufe



- Interaktion zwischen Objekten durch **synchrone** oder **asynchrone** Nachricht



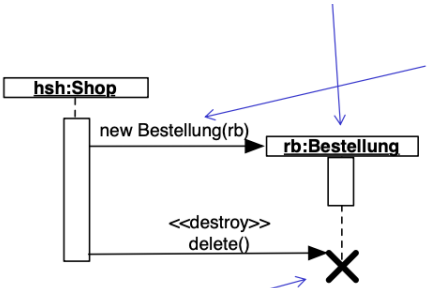
(a) synchrone Nachricht: (gefüllte Pfeilspitze)
Senderobjekt wartet, bis Empfängerobjekt die Verarbeitung komplett beendet hat

Empfänger schickt Antwortnachricht

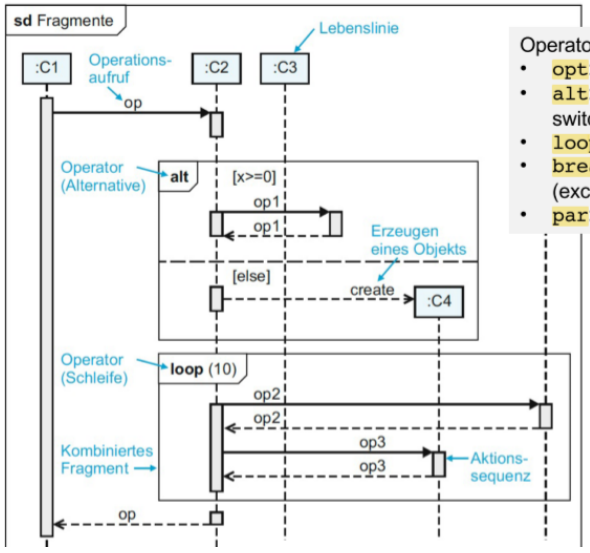
(b) asynchrone Nachricht: (offene Pfeilspitze)
Sender wartet nicht auf Verarbeitungsende durch Empfänger, sondern setzt parallel eigene Verarbeitung fort

---

- bei **Objekterzeugung** beginnt neue Lebenslinie entsprechend später
- und Zerstörung von Objekten, z.B. gekennzeichnet durch Stereotyp <<destroy>>



- Kontrollstrukturen: Alternativen und Schleifen



Operatoren:
- `opt`: optionale Interaktion (if-then)
- `alt`: alternative Abläufe (if-then-else, switch)
- `loop`: Schleife (for, while-do, do-while)
- `break`: Ausnahmebehandlung (exception)
- `par`: parallele Interaktionssequenzen

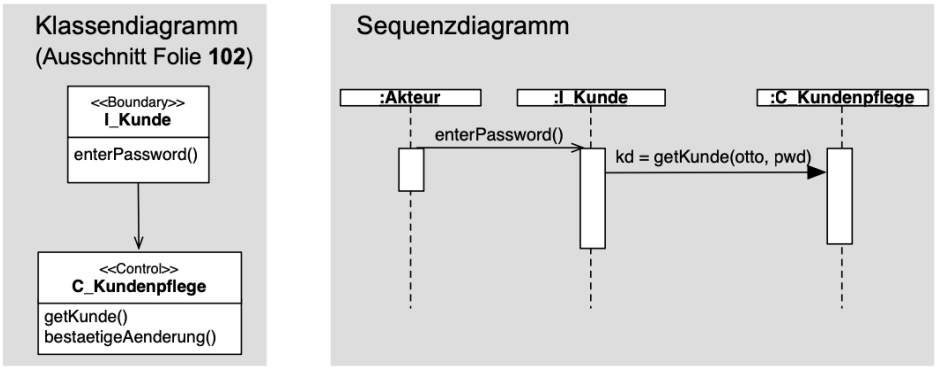
Achtung: Kontrollstrukturen erschweren die Lesbarkeit der Diagramme, besser bei fachlichen Modellen nicht verwenden

Beispiel Sequenzdiagramm 1

Akteur kommuniziert ausschließlich über Boundary-Klasse, diese nur mit Control-Klassen, diese mit anderen Control- oder Entity- Klassen

- Akteur interagiert mir GUI und ruft die Boundary-methoden auf. Es gibt kein Akteur/Kunden Objekt
- Control-Objekte werden beim Systemstart initialisiert

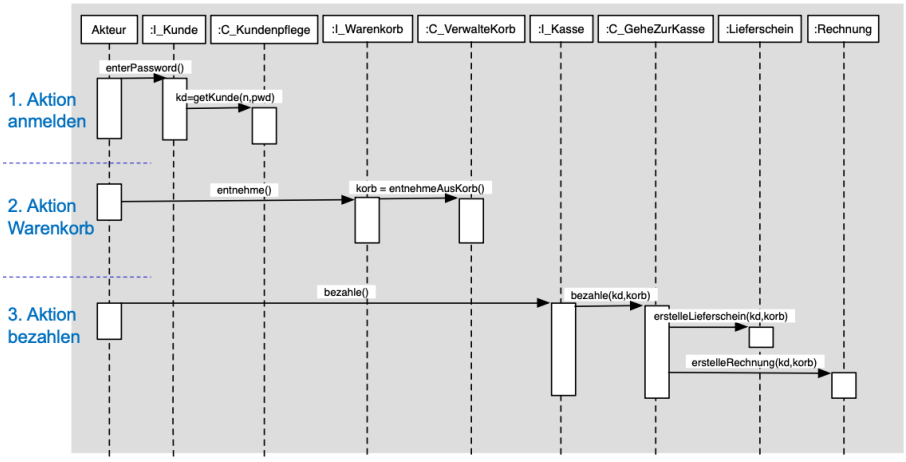
Sequenzdiagramm für Anmeldung im UC: „Benutzer meldet sich im System an“.



Beispiel Sequenzdiagramm 2

- Akteur interagiert mir GUI und ruft die boundary-methoden auf. Es gibt kein Akteur/Kunden Objekt
- Control-Objekte werden beim Systemstart initialisiert

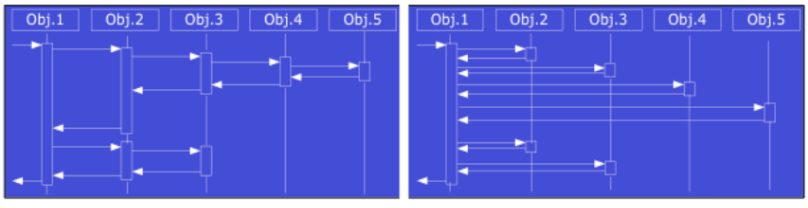
Use Case „zur Kasse gehen“ (nur Ausschnitt!) (siehe UC-Beschreibung Folie 74)



UML-Sequenzdiagramm

Vorteile:

- betonen den **zeitlichen Aspekt** des dynamischen Verhaltens
- **Reihenfolge** und **Verschachtelung** der Methoden bzw. **Zusammenspiel der Objekte sind leicht zu erkennen**
- Sequenzdiagramme zeigen Kontrollfluss innerhalb eines SW-System



- Kommunikationsstruktur gut ersichtlich: welche Objekte kennen sich
  - z.B. durch Assoziationen, Parameter, Rückgabewerte
  - welche Daten werden ausgetauscht (als Parameter oder Rückgabewerte)

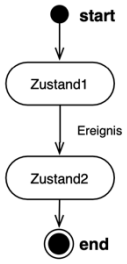
4.4 UML-Zustandsdiagramm

- Methodenaufrufe vom Zustand abhängig
- Problem: **Lebenszyklus eines Objekts** beschreiben
  - Objekte befinden sich in verschieden Zuständen mit unterschiedlichem **Verhalten**

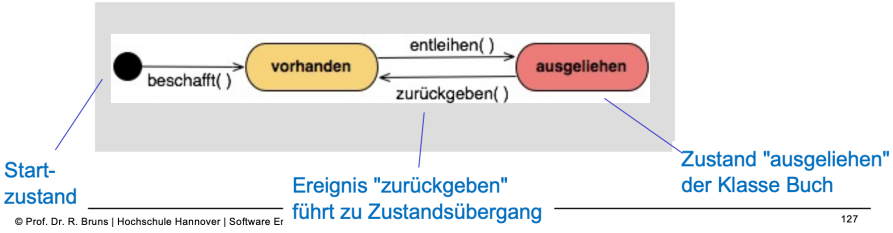
- Zustandsdiagramme sind sinnvoll
  - Wenn sich das **Verhalten** eines Objekt signifikant ändert
  - **Nur für Klassen mit komplexen** (z.B. zeitabhängigen\Verhalten, z.B. Fahrkartenautomat
  - **Theoretische Grundlage: endliche Automaten**

Elemente eines UML-Zustandsdiagramms:

- ein **Startzustand** (initial state)
- **Zustand** (repräsentiert durch Gesamtheit der Attributwerte)
- **Übergänge** (**Transitionen**) zwischen Zuständen
- **Ereignisse**, die Zustandsübergänge bewirken (z.B. Erhalt einer Nachricht, Bedingung, Zeitablauf)
- ein (oder mehrere) **Endzustand** (final state) (Mehrere Endzustände aus Übersichtlichkeitsgründen)

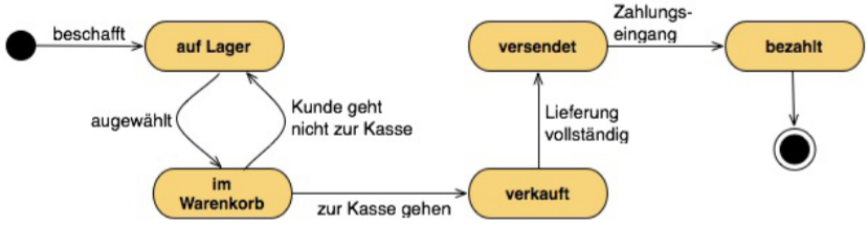


- z.B. Zustände für Klasse Buch in Bibliothek



- Zustandsdiagramme verdeutlichen **Zusammenhänge und Abhängigkeiten** von Aktionen
- Verhalten eines Objekts über **mehrere Anwendungsfälle**
- Beliebige Ereignisse möglich, müssen keine Methoden sein

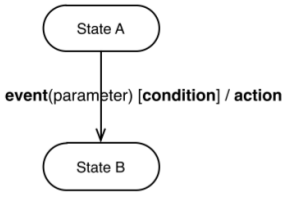
Beispiel: Zustandsmodell für Klasse Buch in Internet-Buchhandlung



Zustandsübergang mit Guard und Aktion

- Ergebnis löst Übergang zwischen zwei Zuständen aus
- Ergebnis kann mit einer Bedingung versehen werden
- Ergebnis kann beim Eintreten eine Aktion auslösen

Ergebnis [Bedingung]/ Aktion



4.5 Aktivitätsdiagramme

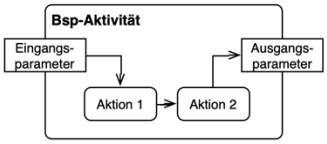
UML-Aktivitätsdiagramme

- **Problem: Beschreibung des Ablaufs komplexer Prozesse durch Aktivitäten/ Aktionen/ Schritt**
  - Zustandsdiagramme nur für **eine Klasse** (Zustandsraumexplosion)
  - Use Cases nur **exemplarische Abläufe** (typische und alternative Abläufe, aber keine Vollständigkeit)
- Ziele
  - **Verhalten von Objekten in mehreren UC-Szenarien bzw. Use Cases zeigen**
    - alternative Abläufe darstellen
    - iterative Abläufe darstellen
    - potenziell parallele Abläufe identifizieren (z.B. Threads) (Synchronisation von nebenläufigen Aktivitäten)
  - (generell) Spezifikation eines komplexen Ablaufs/ komplexer Funktionalität

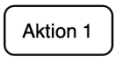
---

UML-Notation

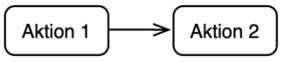
1. **Aktivität (activity)**
   - gesamtes Diagramm beschreibt eine Aktivität



2. **Aktions**knoten
   - Aktion (action): kleinste ausführbare Einheit
     - Aufgabe/Prozessschritt oder Methode
   - Aktion kann ausgeführt werden, wenn Vorgänger-Aktion beendet ist

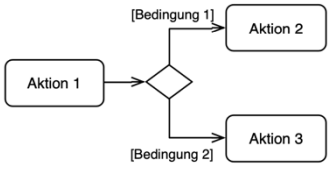


3. **Pfeil/ Kante (→):** Übergang zur folgender Aktion
   - - Aktion (action): kleinste ausführbare Einheit

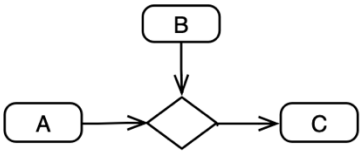


4. **Kontroll**knoten

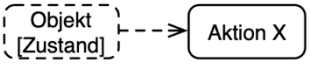
   a) Entscheidung
   - Verzweigung abhängig von Bedingung



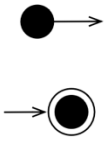
   b) Zusammenführung
   - nach Eintreffen von A oder B wird Zweig C fortgesetzt



5. **Verknüpfung Aktionen mit Objekten**sddd
   - Verknüpfung von Aktionen mit Objekten bzw. deren Zuständen

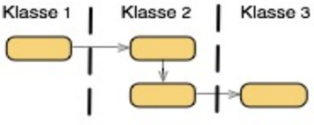


6. **Start-/Endknoten**
   - Aktivität kann einen oder mehrere Startknoten/ besitzen (Aktionen starten parallel)
   - Aktivität kann einen oder mehrere besitzen



7. **Schwimmbahnen ("swimlanes")**

   verdeutlichen die **Verantwortlichkeiten** (für Aktionen, Entscheidungen)
   - Rollen/ Organisationseinheiten (konzeptionell)
   - Klassen (spezifizierendes, konzeptionelles Modell)

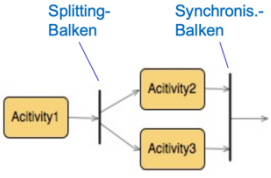


8. **Nebenläufigkeit**

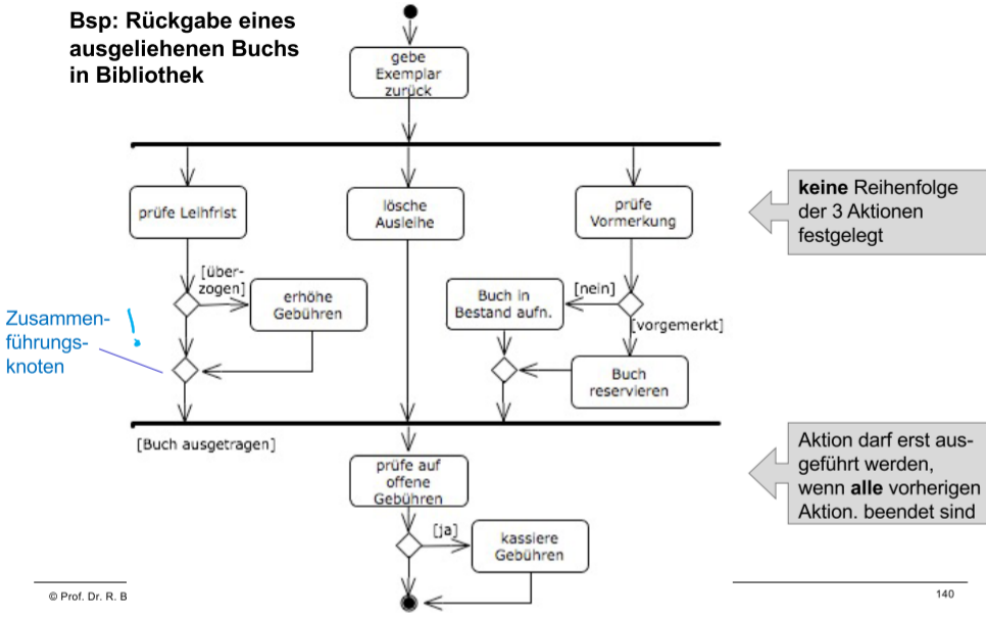
   a) Splitting (fork node, Aufspalten):
   - Kontrollfluss auf mehrere parallel Ströme aufgeteilt

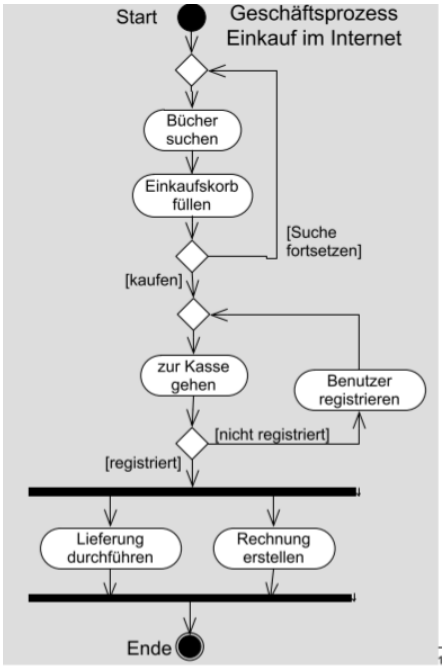
   b) Synchronisation (join node):
   - Zusammenführen mehrerer Aktionen

(alle beteiligten Aktionen müssen beendet sein, bevor nachfolgende Aktion starten kann)



Beispiele





Vor- und Nachteile

- Stärken
  - ermöglichen Modellierung komplexer Abläufe
  - bei der Darstellung nebenläufiger Prozesse
    - fachliche Analyse: Geschäftsprozesse (siehe Kap 3.2)
    - technisch: Nebenläufigkeit durch Threads realisiert
  - zeigen alternative Abläufe und Schleifen (übersichtlicher als in Sequenzdiagrammen)

- Schwächen
  - Beziehung der Aktivitäten zu Objekten schlecht sichtbar
    - Auswege: swimlanes oder Objektnamen in Aktionen aufnehmen
  - wird schnell unübersichtlich
  - für Verhalten eines einzelnen Objektes nicht geeignet
    ⇒ Zustandsdiagramm