Unified Modelling Language

Software-Entwicklung

Software-Entwicklung ist Prozess von Anforderung über Modellierungen zu fertigen Programmen

Anforderungen oft informell gegeben

fertige Programme sollen am Ende Anforderungen genügen

UML ist Werkzeug zur Modellierung verschiedener Aspekte der Software, z.B.

- Welche Komponenten gibt es?
- Was sollen diese im Einzelnen leisten können?
- Wie verhalten sich diese im Einzelnen?
- Wie interagieren sie miteinander?
- ...

UML-Diagramme

UML ist grafische Modellierungssprache

Sachverhalte werden in Form von Diagrammen dargestellt

es gibt verschiedene Arten von Diagrammen mit jeweils verschiedenen Aufgaben

- Anwendungsfalldiagramme
- Klassendiagramme
- (Objektdiagramme)
- Sequenzdiagramme
- Zustandsdiagramme
- ...

Verwendung von Diagrammen

primärer Einsatz als Strukturierungs- und Modellierungsmittel auf dem Weg von informeller Beschreibung zum Programm

nicht: Beschreibung von existierender Software

Modellierung beinhaltet auch Abstraktion

z.B. unpassende Frage: *"Muss im Klassendiagramm auch der Konstruktor in den Methoden aufgelistet werden?"*

Diagramme so zu verstehen:

- Konstruktor nicht aufgelistet: Verhalten wohl Standard, Klasse muss natürlich dennoch einen haben
- Konstruktor aufgelistet: evtl. Besonderheit zu beachten, z.B. bei übergebenen Argumenten, verschiedene Konstruktoren sollen vorhanden sein, etc.

Anwendungsfalldiagramme

definieren Funktionalität (von Teilen) des Systems nach außen hin

Bsp.:

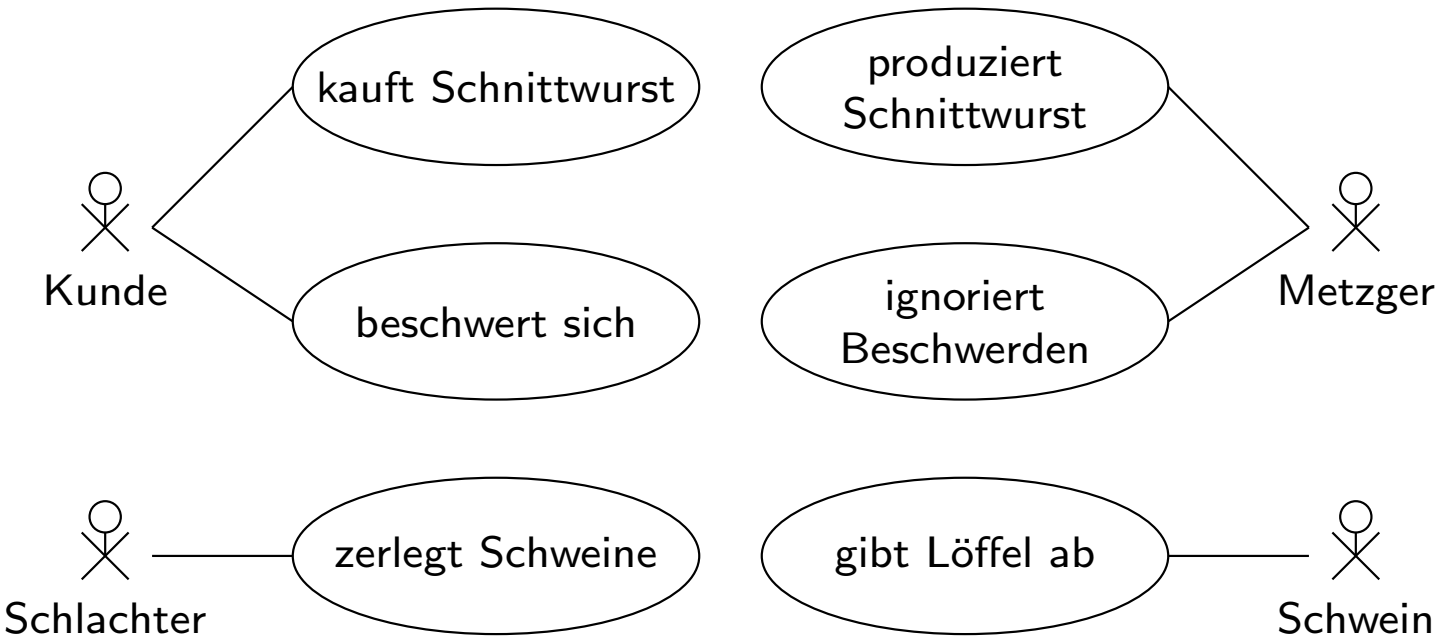
- Was macht Benutzer mit Programm?
- Was macht ein Teil des Systems mit anderem?
- ...

Vorsicht: hohe Variabilität bzgl. Detailliertheit

Modellierung identifiziert *Aktoren* und *Anwendungsfälle*

Anwendungsfalldiagramme – Beispiel

bayerischer Schlachthof mit integrierter Metzgerei



Klassendiagramme

Klasse = gemeinsame Ausprägung verschiedener Objekte

hier nur objekt-orientierte Software-Entwicklung in Java: Klasse = Java-Klasse

Klassendiagramme stellen Beziehungen zwischen einzelnen Klassen dar

Beziehungen spiegeln Zusammenhänge in den Anforderungen wider

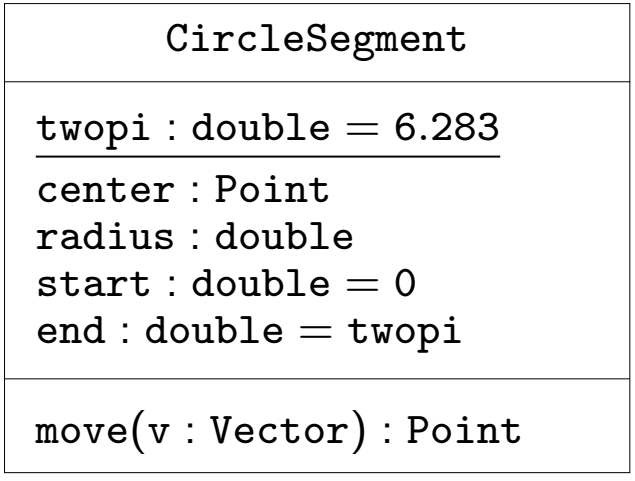
Beziehungen lassen sich oft direkt in Code umsetzen

Klassen

Klassen werden dargestellt als dreigeteiltes Rechteck

- oberer Teil enthält Klassenname
- mittlerer Teil enthält Attribute
- unterer Teil enthält Methoden

Attribute werden mit Typ spezifiert, evtl. auch mit Default-Wert



Methoden werden mit Typ spezifiziert

unterstrichene Attribute/Methoden = Klassenattribute / -methoden

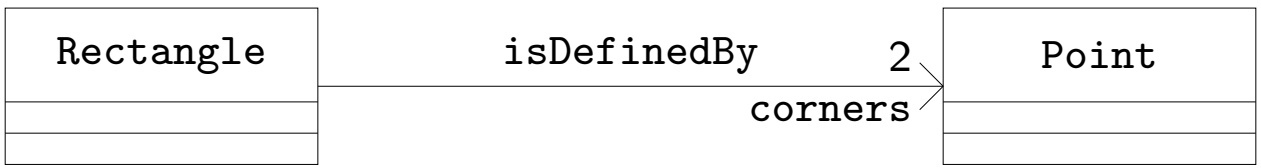
Assoziation

Objekte einer Klasse stehen in Verbindung zu Objekten der anderen Klasse

Multiplizitäten möglich, z.B. `1` oder `1..2` oder `*`

unidirektionale, bidirektionale Assoziation möglich, sogar mehrstellig

optionale Rollennamen verdeutlichen Bedeutung der Verbindung



Auswirkung auf Implementierung: `Rectangle` hat zwei Instanzvariablen vom Typ `Point`

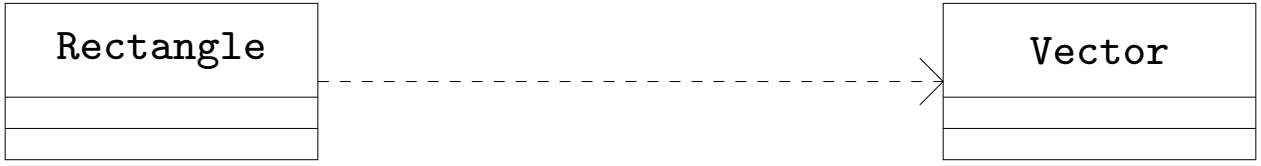
Aggregation

eine Klasse kann Teile einer Entität modellieren, die durch eine andere Klasse modelliert wird

Implementierung z.B. durch Instanzvariablen

Abhängigkeit

sollte sich `Vector` ändern, so muss evtl. auch `Rectangle` geändert werden



liegt z.B. vor, wenn Objekte vom Typ `Vector` in Methoden von `Rectangle` benutzt werden

Vererbung

eine Klasse kann spezieller als eine andere sein, (umgekehrt genereller)

in Java realisiert durch `extends`

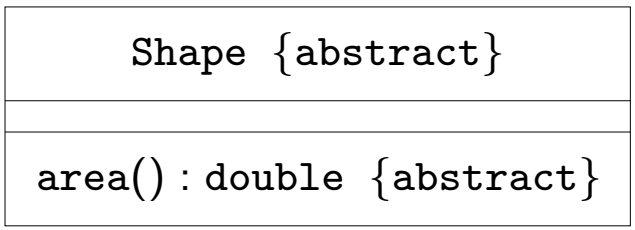
beachte Konventionen zum Vererben und Überschreiben von Instanzvariablen und Methoden

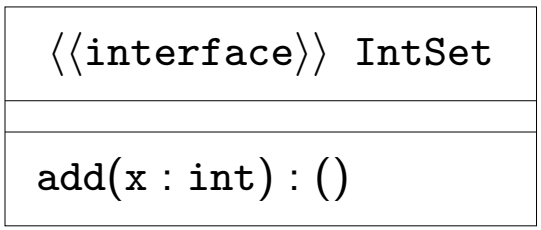
Interfaces und abstrakte Klassen

Interfaces geben Typ aber nicht Implementierung vor

abstrakte Klassen können abstrakte Methoden enthalten

abstrakte Methode: Implementierung abhängig von Untertyp





Methoden müssen in anderen Klassen implementiert werden

Zustandsdiagramme

nicht zu verwechslen mit Objektdiagrammen (= Zustand des Heaps)

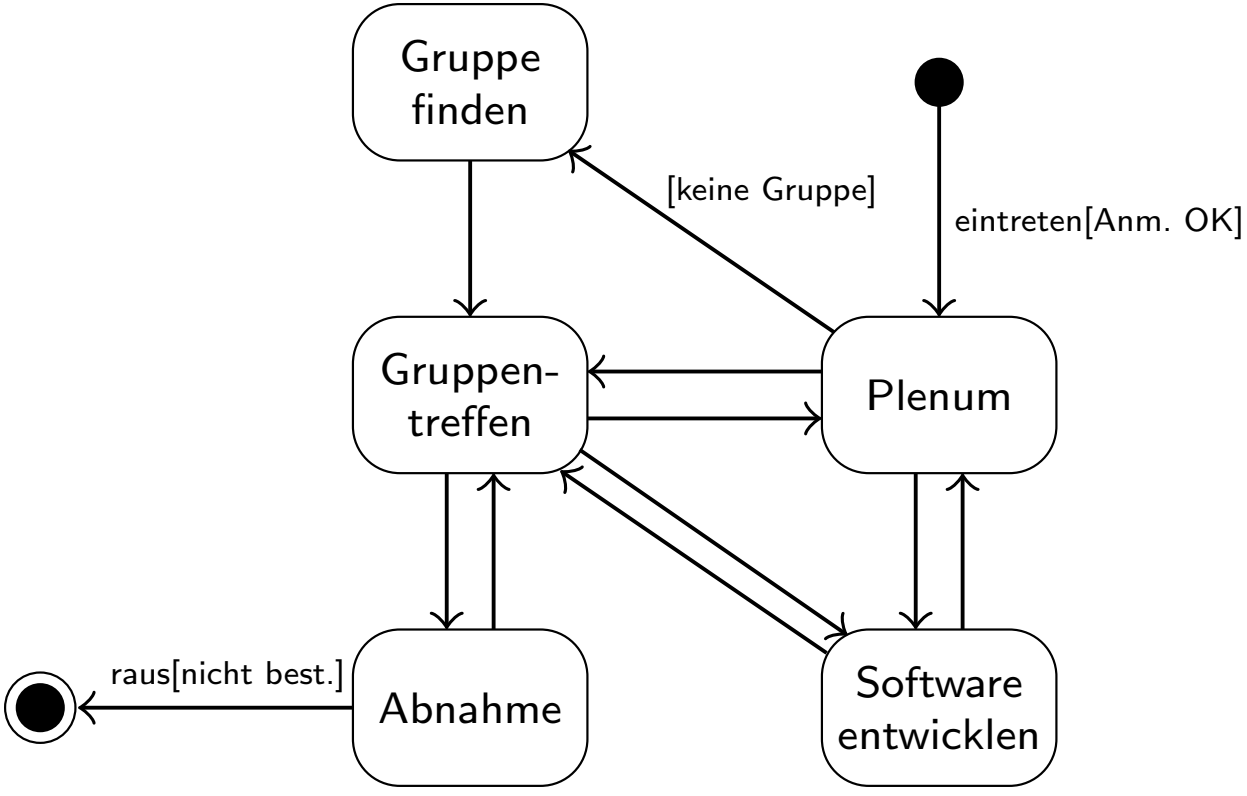
Modellierung des zeitlichen Verhaltens eines Objekts als endlicher Automat

endliche Anzahl von Zuständen kann z.B. durch Datenabstraktion erreicht werden (z.B. Inhalt einer `int`-Variable ignorieren)

zwei ausgezeichnete Zustände für Anfang (Konstruktion) und Ende (Destruktion)

Zustandsdiagramme – Beispiel

Teilnehmer im SEP



Zustandsübergänge z.B. durch Methodenaufrufe realisiert

Sequenzdiagramme

Beschreibung des dynamischen Verhaltens eines Systems

stellen Interaktion von Objekten in zeitlicher Reihenfolge dar

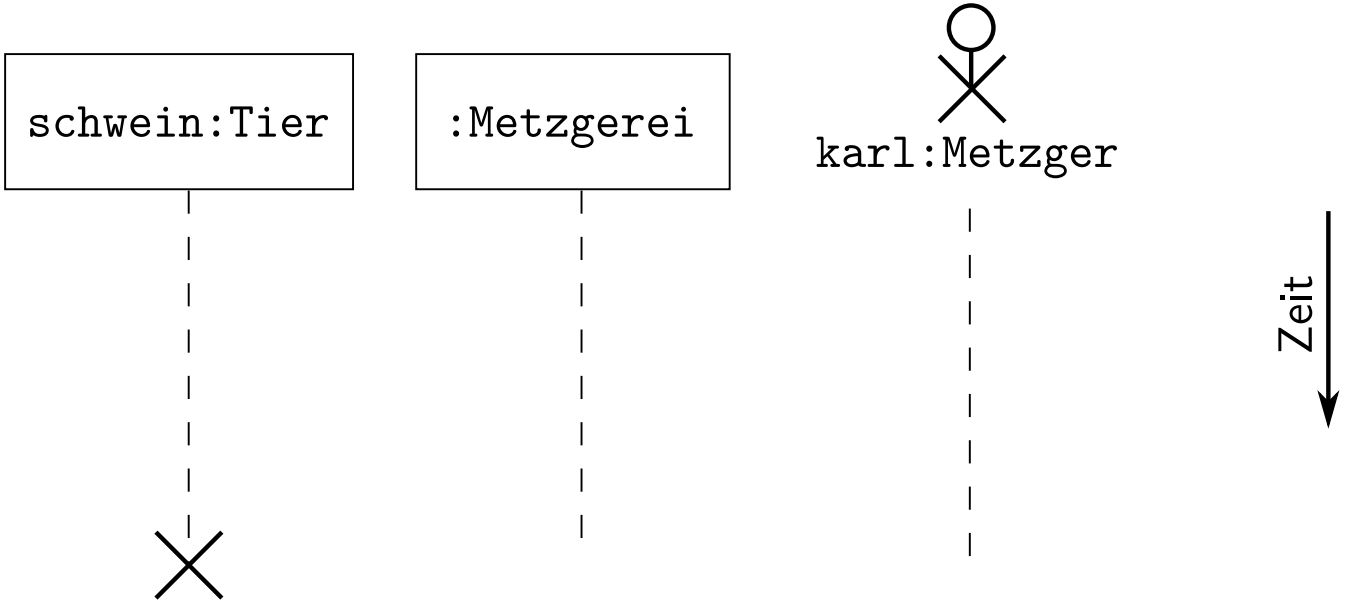
Systemverhalten wird durch Austausch von *Nachrichten* beschrieben

in Java: Nachrichtenaustausch z.B. Methodenaufruf

Zeitlinien

Sequenzdiagramme bestehen aus mehreren Zeitlinien, welche die Lebenszeit von Objekten darstellen

Zeitlinien sind mit Klassennamen und evtl. Objektnamen annotiert



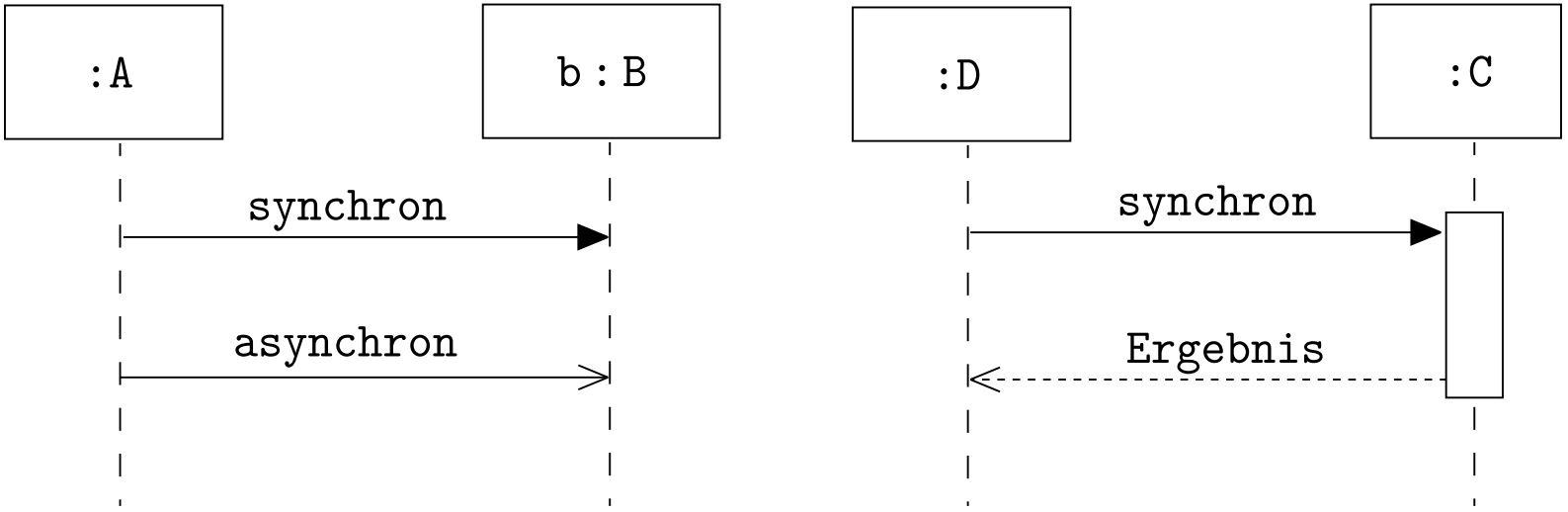
Objektdestruktion mit Kreuz gekennzeichnet

Nachrichten

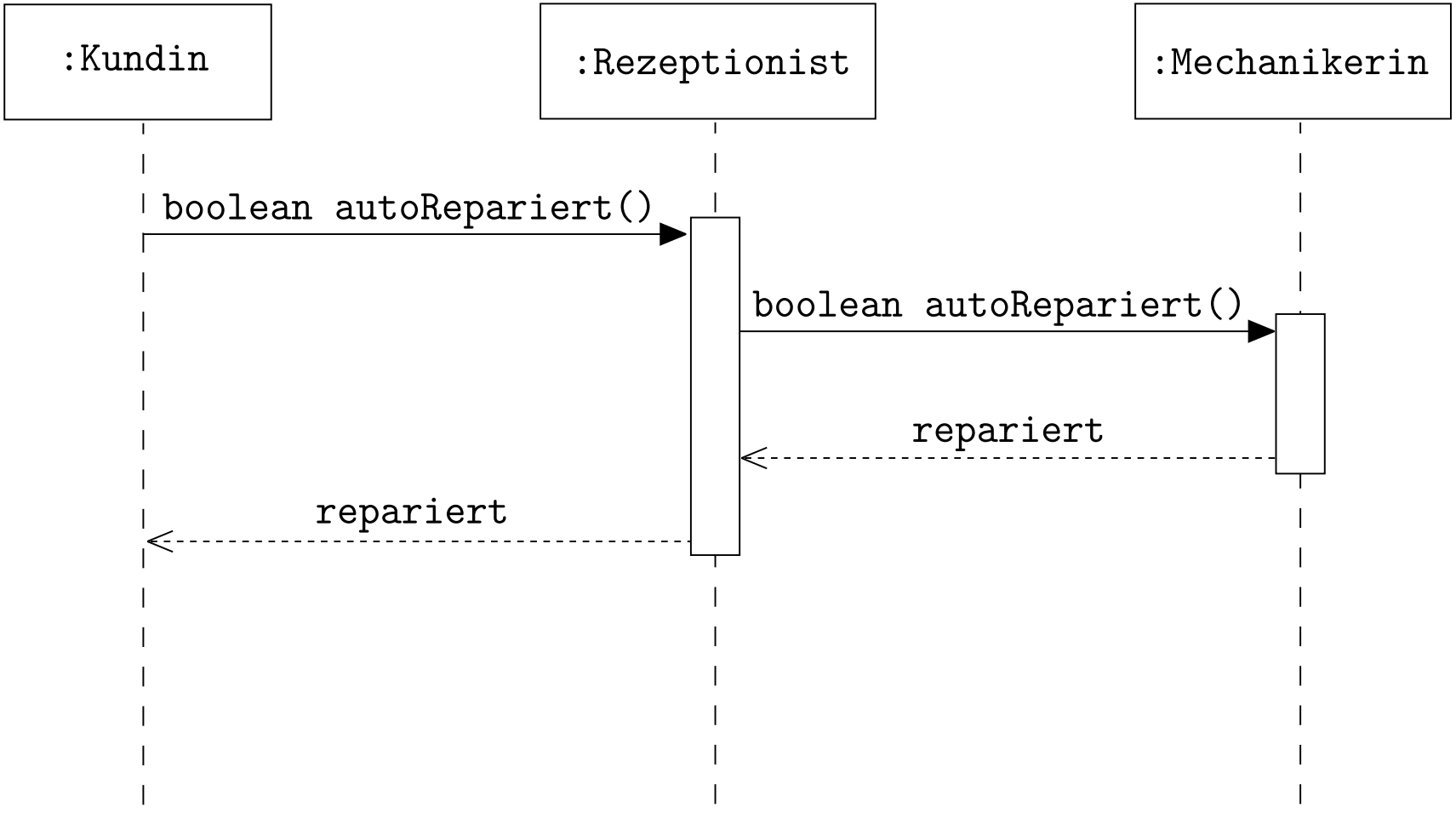
synchrone und asynchrone Nachrichten

bei synchronen Nachrichten optional Rückantwort, die mit einem Rückgabewert beschriftet sein können

Rechtecke auf den Zeitlinien symbolisieren die durch Nachrichten ausgelöste Aktivität



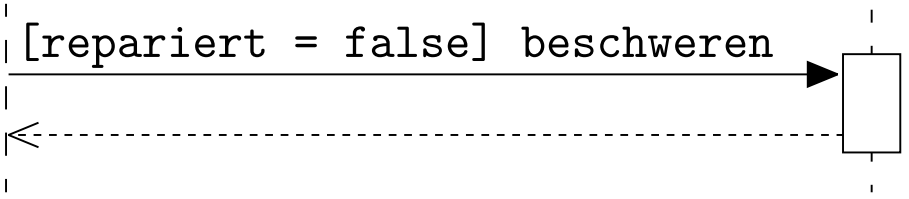
Synchrone Nachrichten – Beispiel



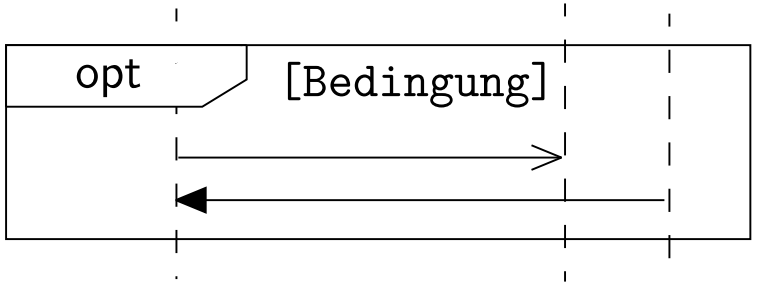
wie in Java übersetzen?

Kontrollstrukturen

Guards Nachricht wird nur geschickt, wenn eine bestimmte Bedingung erfüllt ist.



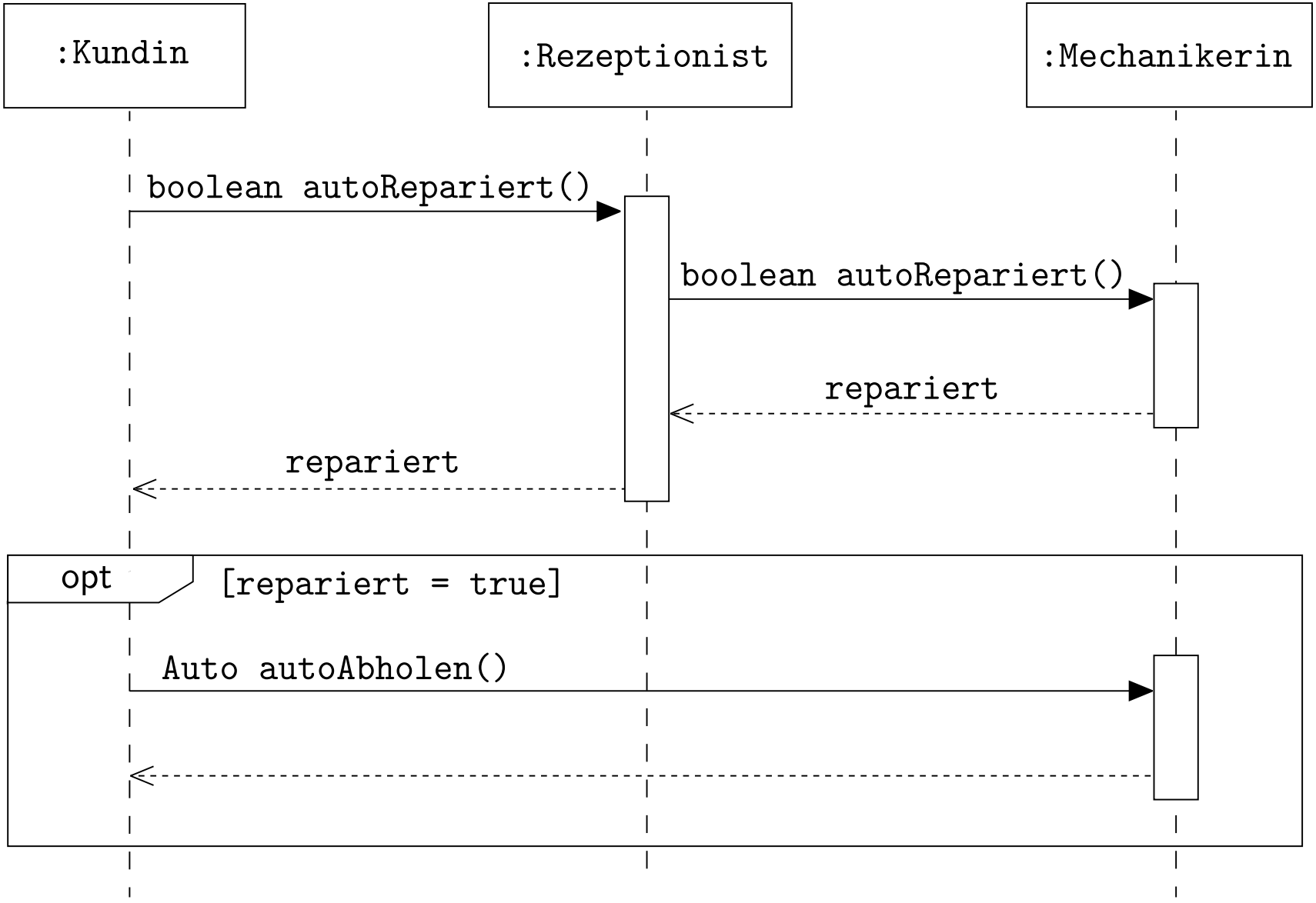
Optionen eine ganze Reihe von Nachrichten wird nur ausgeführt wenn eine Bedingung wahr ist



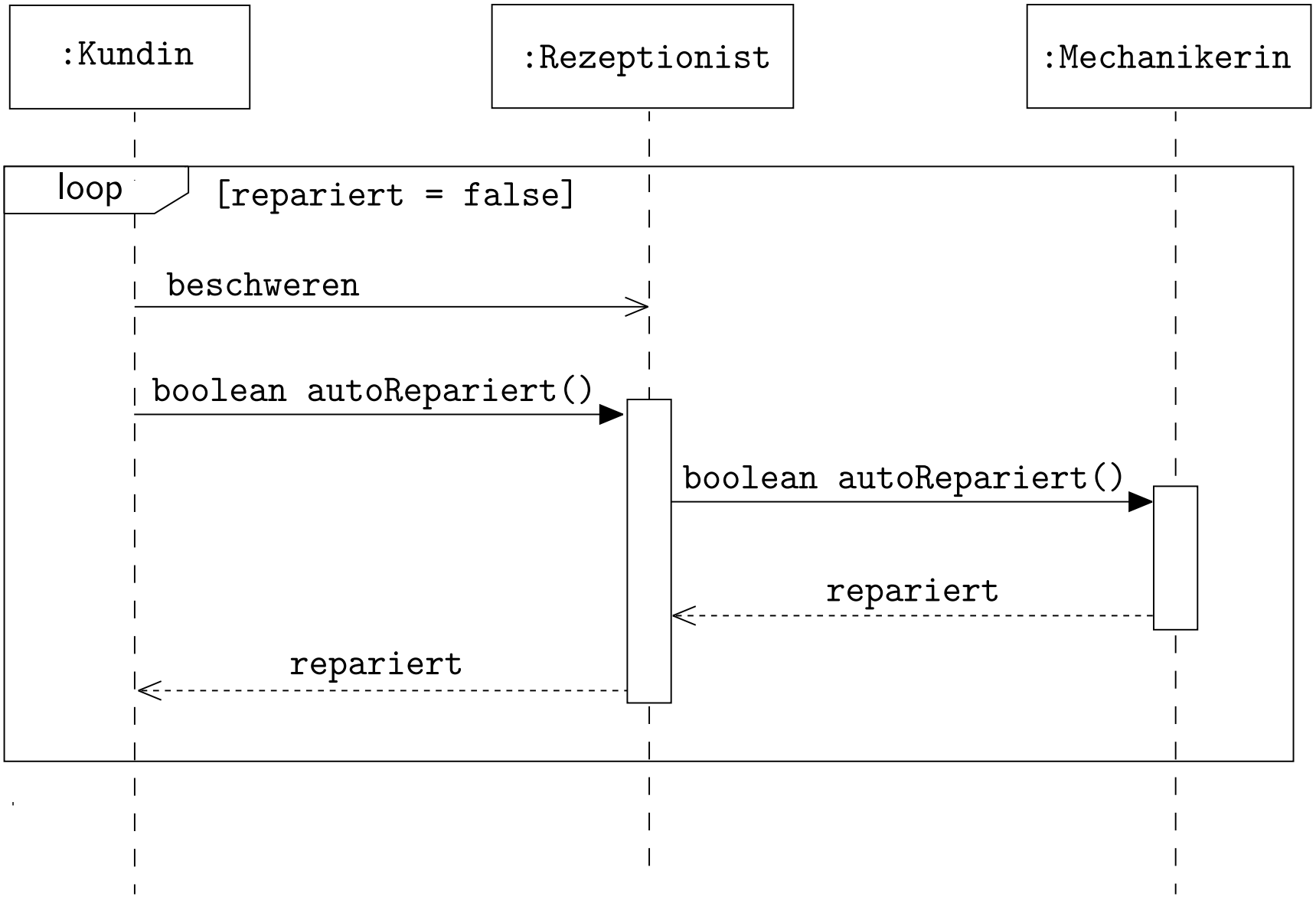
Schleifen wiederhole Nachrichten solange Bedingung wahr ist: „loop“ statt „opt“

viele weitere ...

Optionen – Beispiel



Schleifen – Beispiel



UML-Diagramme im Praktikum

- Teile der Spezifikation in UML gegeben
- Entwicklungsprozess in den Gruppen soll ebenfalls mithilfe von UML erledigt werden
  - Entwurf von detaillierter Architektur mithilfe von Klassendiagrammen, evtl. noch Anwendungsfalldiagrammen
  - Entwurf von Klassen mithilfe von Zustandsdiagrammen
  - Entwurf von Abläufen (z.B. im Server) mithife von Sequenzdiagrammen
  - (Entwurf und Debugging mit Objektdiagrammen)

Links

- Folien der Vorlesung “Objektorientierte Software-Entwicklung”

  `http://www.pst.ifi.lmu.de/Lehre/wise-07-08/oose/material`

- `http://www.uml.org/` – siehe Tutorials

- Bücher

- Tools zur Erstellung von Diagrammen, siehe z.B. “UML-Werkzeug” auf Wikipedia