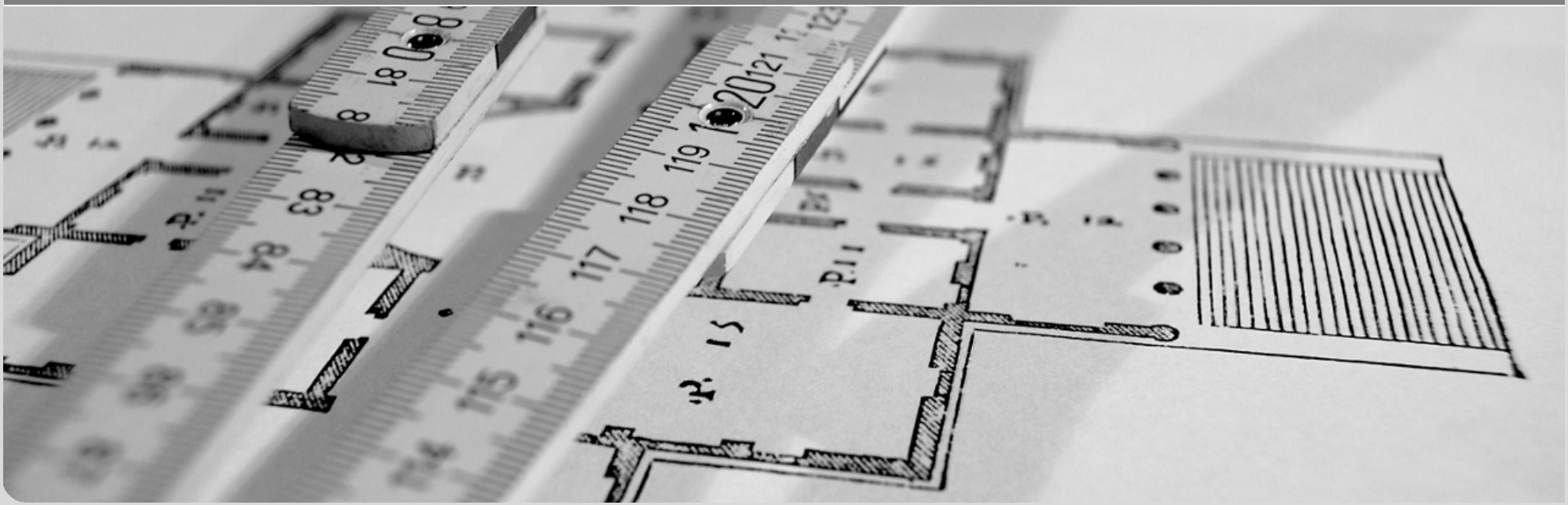
Karlsruher Institut für Technologie

Softwaretechnik 1 - 2. Tutorium

Tutorium 17

Felix Bachmann | 28.05.2019

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



Themen

Abgabemodalitäten

Sprache

- letztes Mal kam eine Frage, ob Abgaben auf Englisch auch erlaubt
- für mich ok
- laut Übungsleitern in Klausur nicht ok
  - also am besten jetzt auch schon auf Deutsch abgeben

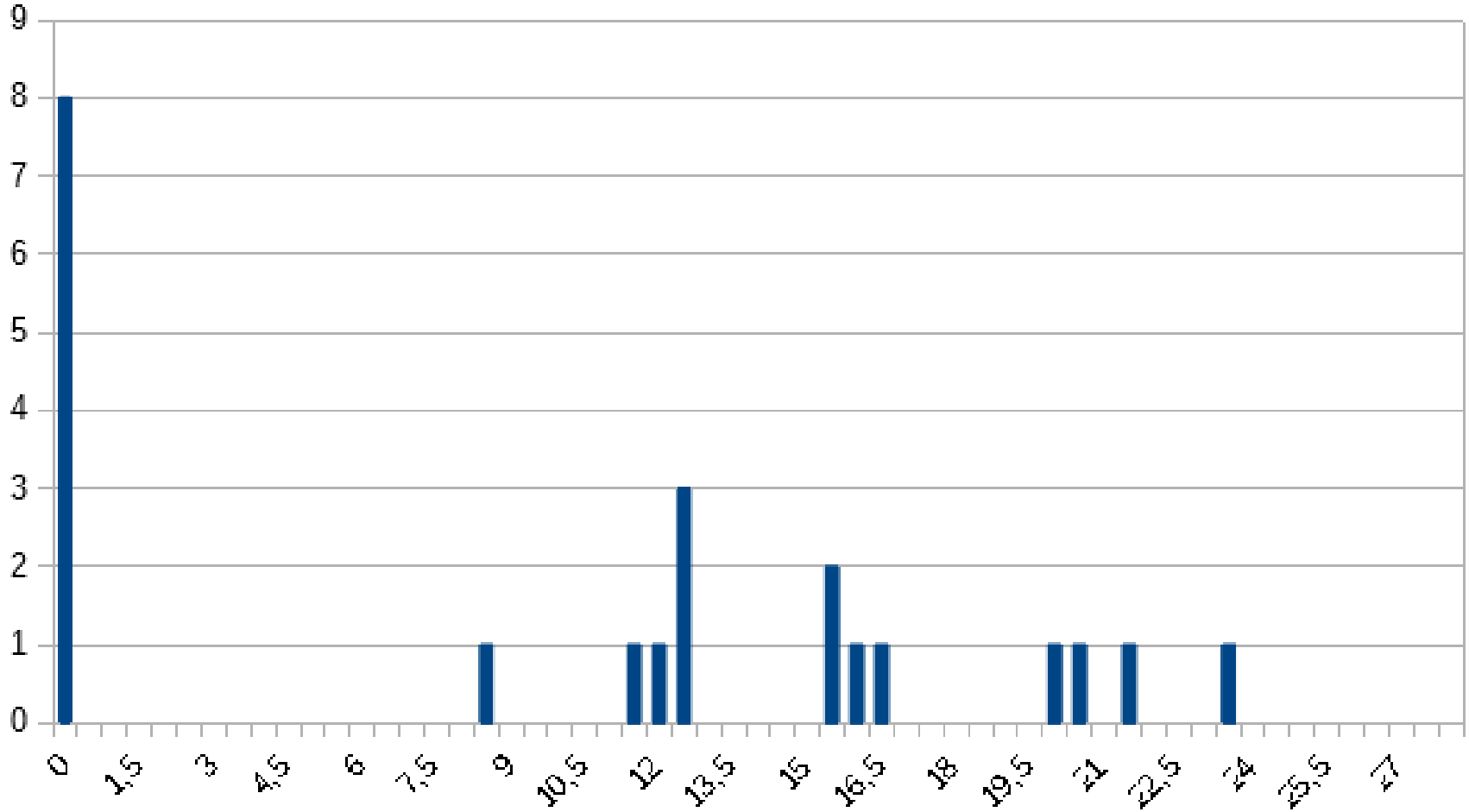
Notation Abstrakte Klassen

- an VL-Notation halten (insbesondere nicht `<<abstract>>`)
- siehe UML-Spec
  - `https://www.omg.org/spec/UML/2.5.1/PDF`, p.101

Tutorientermine

- entsprechend der Abstimmung vom vorletzten Mal ist das nächste Tut am 04.06 (= nächste Woche)
- also
  - 28.05. (heute): Tutorium
  - 04.06.: Tutorium
  - 11.06.: kein Tutorium
  - 18.06.: kein Tutorium
  - ab 25.06. wieder regulär 14-tägig
    - 25.06, 09.07, 23.07

2. Übungsblatt Statistik



Häufige Fehler

Allgemein

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und die Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

- $\Longrightarrow$ nur das offizielle Deckblatt verwenden!

Häufige Fehler

Allgemein

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und die Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

- $\Longrightarrow$ nur das offizielle Deckblatt verwenden!
- häufigster Fehler: Aufgaben nicht abgegeben
- Checkstyle, JavaDoc,. . .

Häufige Fehler

Aufgabe 1 (Lastenheft)

- „relevante Daten“ bei Produktdaten ist unpräzise

Häufige Fehler

Aufgabe 1 (Lastenheft)

- „relevante Daten“ bei Produktdaten ist unpräzise
- Verwirrung: Szenario vs. „Aufgaben-Szenario“

Häufige Fehler

Aufgabe 1 (Lastenheft)

- „relevante Daten“ bei Produktdaten ist unpräzise
- Verwirrung: Szenario vs. „Aufgaben-Szenario“
- Anwendungsfalldiagramm
  - Syntax (Pfeile, System-Kasten, Aktoren,. . .)
  - Anwendungsfälle enthalten Verben
    - „Vorschau“, „Kundenbilder“ sind **keine**
    - richtig wäre z.B. „Kundenbilder speichern“
  - direkte Verbindungen zwischen Anwendungsfällen nicht definiert
    - include vs. extend
    - auch nicht zwischen Aktoren

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- äußere Form, UML-Syntax
  - gerichtete Assoziation vs. Vererbung
  - `name:Typ` statt `Typ name`

UML Java

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- äußere Form, UML-Syntax
  - gerichtete Assoziation vs. Vererbung
  - `name:Typ` statt `Typ name`

    UML: `name:Typ` — Java: `Typ name`
- alle Klassen, die ihr verwendet, müssen auch im Diagramm sein
  - z.B. „Bild“ als Parameter verwendet, aber nirgendwo definiert

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- äußere Form, UML-Syntax
  - gerichtete Assoziation vs. Vererbung
  - `name:Typ` statt `Typ name`

    UML: `name:Typ` — Java: `Typ name`
- alle Klassen, die ihr verwendet, müssen auch im Diagramm sein
  - z.B. „Bild“ als Parameter verwendet, aber nirgendwo definiert
- `verarbeite(b:Bild):Bild` in Einzelbildop./Mehrbildop. nicht als abstrakt gekennzeichnet
  - kann man aber nicht sinnvoll implementieren

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- äußere Form, UML-Syntax
  - gerichtete Assoziation vs. Vererbung
  - `name:Typ` statt `Typ name`

    UML: `name:Typ` — Java: `Typ name`
- alle Klassen, die ihr verwendet, müssen auch im Diagramm sein
  - z.B. „Bild“ als Parameter verwendet, aber nirgendwo definiert
- `verarbeite(b:Bild):Bild` in Einzelbildop./Mehrbildop. nicht als abstrakt gekennzeichnet
  - kann man aber nicht sinnvoll implementieren
- Farbtiefe sollte Enum sein (nur bestimmte Werte erlaubt)

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- äußere Form, UML-Syntax
  - gerichtete Assoziation vs. Vererbung
  - `name:Typ` statt `Typ name`

    UML: `name:Typ` — Java: `Typ name`
- alle Klassen, die ihr verwendet, müssen auch im Diagramm sein
  - z.B. „Bild“ als Parameter verwendet, aber nirgendwo definiert
- `verarbeite(b:Bild):Bild` in Einzelbildop./Mehrbildop. nicht als abstrakt gekennzeichnet
  - kann man aber nicht sinnvoll implementieren
- Farbtiefe sollte Enum sein (nur bestimmte Werte erlaubt)
- oft wurden die konkreten Operationen als Methoden hingeschrieben
  - sollten aber eigene Klassen sein
  - Wieso ist das sinnvoll? siehe „Befehl/Command“-Entwurfsmuster
  - (nächstes Tut)

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- bei Collections immer []-Schreibweise nutzen, nicht `ArrayList<XY>`

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- bei Collections immer []-Schreibweise nutzen, nicht `ArrayList<XY>`
- uses. vs Assoziation

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- bei Collections immer []-Schreibweise nutzen, nicht `ArrayList<XY>`
- uses. vs Assoziation

Aufgabe 3 (Durchführbarkeitsanalyse)

- nur Stichpunkte...

Häufige Fehler

Aufgabe 2 (Klassendiagramm)

- bei Collections immer []-Schreibweise nutzen, nicht `ArrayList<XY>`
- uses. vs Assoziation

Aufgabe 3 (Durchführbarkeitsanalyse)

- nur Stichpunkte...
- Fragen beantworten, nicht stellen!
  z.B. "Es werden 3 Java-Entwickler benötigt." $\Longrightarrow$ "Da wir 5 zur Zeit untätige Java-Entwickler in der Firma haben, ist das Projekt aus personeller Sicht für die Pear Corp. durchführbar."

Häufige Fehler

Aufgabe 4 (HDR programmieren)

- Überprüfungen bei Matrix-Operationen vergessen
  - `A*B` nur möglich, wenn `A.cols() == B.rows()`
  - A^{-1} nicht-existent falls `A` nicht quadratisch
    - existiert nur, falls `det(A) != 0`

Häufige Fehler

Aufgabe 4 (HDR programmieren)

- Überprüfungen bei Matrix-Operationen vergessen
  - `A*B` nur möglich, wenn `A.cols() == B.rows()`
  - A^{-1} nicht-existent falls `A` nicht quadratisch
    - existiert nur, falls `det(A) != 0`
- die „komplizierteren“ Sachen nicht abgegeben
  - keine Angst vor ein bisschen Mathe :)

Häufige Fehler

Aufgabe 4 (HDR programmieren)

- Überprüfungen bei Matrix-Operationen vergessen
  - `A*B` nur möglich, wenn `A.cols() == B.rows()`
  - A^{-1} nicht-existent falls `A` nicht quadratisch
    - existiert nur, falls `det(A) != 0`
- die „komplizierteren“ Sachen nicht abgegeben
  - keine Angst vor ein bisschen Mathe :)
- zu wenig oder nicht sinnvoll getestet

Häufige Fehler

Aufgabe 4 (HDR programmieren)

- Überprüfungen bei Matrix-Operationen vergessen
  - `A*B` nur möglich, wenn `A.cols() == B.rows()`
  - A^{-1} nicht-existent falls `A` nicht quadratisch
    - existiert nur, falls `det(A) != 0`
- die „komplizierteren“ Sachen nicht abgegeben
  - keine Angst vor ein bisschen Mathe :)
- zu wenig oder nicht sinnvoll getestet
- Überraschung: `mtx.copy()` soll Matrix kopieren

Häufige Fehler

Aufgabe 5 (Kommandozeilen-Tool)

- 5 Abgaben . . .
- kleine Fehler in pom (dependencies kopieren)

Häufige Fehler

Aufgabe 5 (Kommandozeilen-Tool)

- 5 Abgaben . . .
- kleine Fehler in pom (dependencies kopieren)

Aufgabe 6 (Filter-Schnittstellen)

- 4 Abgaben . . .
- Filter-Liste sollte veränderbar sein
  - z.B. Pipeline-Muster

Liskov'sches Substitutionsprinzip

aka Verhaltenskonformanz, Klientencode-Wiederverwendung

Definition

In einem Programm, in dem U eine Unterklasse von O ist, kann jedes Exemplar der Klasse O durch ein Exemplar von U ersetzt werden, wobei das Programm weiterhin korrekt funktioniert.

Liskov'sches Substitutionsprinzip

aka Verhaltenskonformanz, Klientencode-Wiederverwendung

Definition

In einem Programm, in dem U eine Unterklasse von O ist, kann jedes Exemplar der Klasse O durch ein Exemplar von U ersetzt werden, wobei das Programm weiterhin korrekt funktioniert.

- aus Sicht des Verhaltens ist jedes U-Objekt auch ein O-Objekt
- Klientencode muss bei Methodenaufrufen nicht wissen, ob es sich um ein U-Objekt oder O-Objekt handelt

Liskov'sches Substitutionsprinzip

aka Verhaltenskonformanz, Klientencode-Wiederverwendung

Definition

In einem Programm, in dem U eine Unterklasse von O ist, kann jedes Exemplar der Klasse O durch ein Exemplar von U ersetzt werden, wobei das Programm weiterhin korrekt funktioniert.

- aus Sicht des Verhaltens ist jedes U-Objekt auch ein O-Objekt
- Klientencode muss bei Methodenaufrufen nicht wissen, ob es sich um ein U-Objekt oder O-Objekt handelt

Achtung

Das müssen wir als Programmierer sicherstellen. Nur weil Java uns die Möglichkeit bietet Unterklassen zu bilden, ist das Substitutionsprinzip noch lange nicht erfüllt! (dadurch ist nur die sogenannte Typkonformanz gegeben)

Liskov'sches Substitutionsprinzip

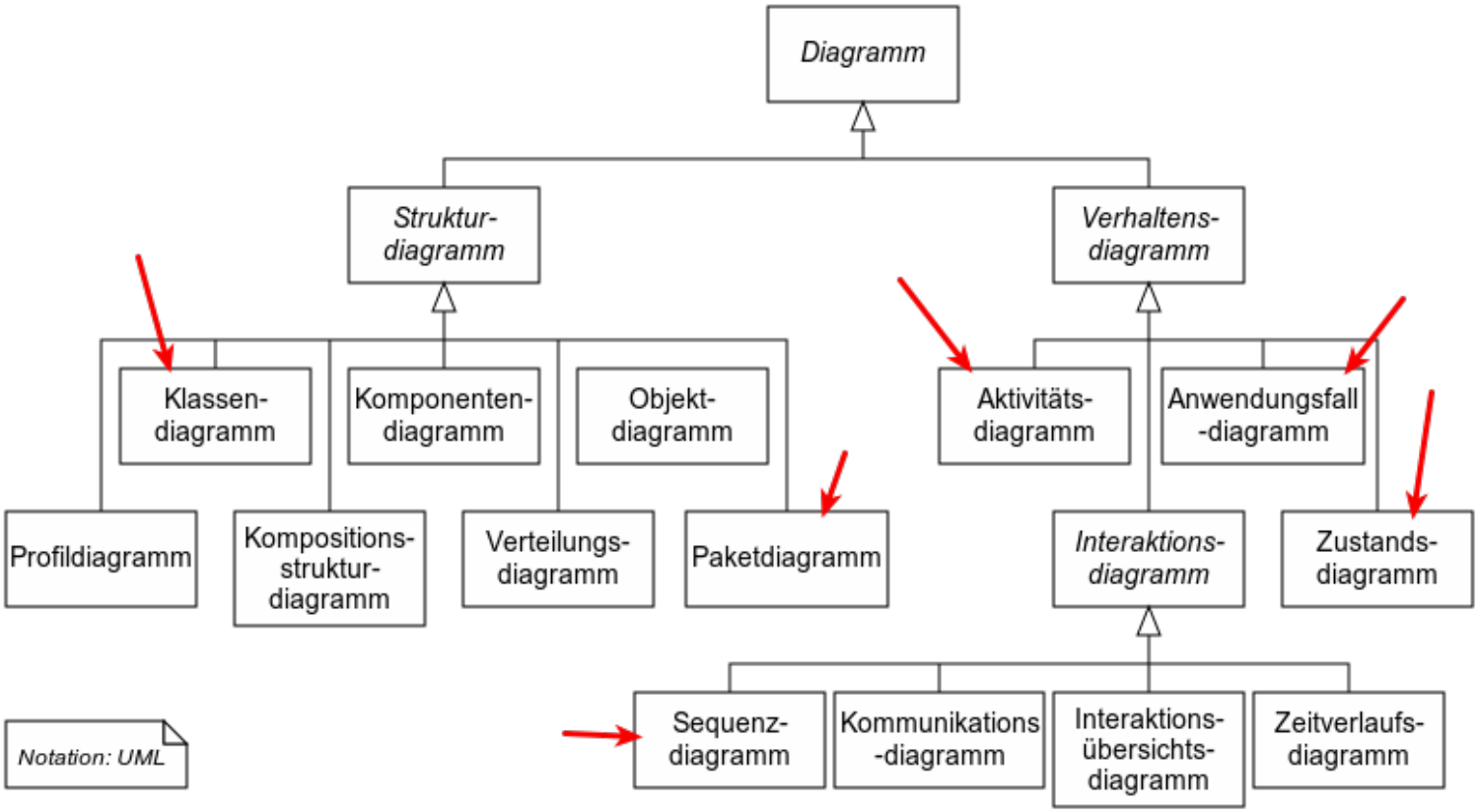
Beispiel!

Wo sind wir? Pflichtenheft!

1. Zielbestimmung
2. Produkteinsatz
3. Produktumgebung
4. Funktionale Anforderungen
5. Produktdaten
6. Nichtfunktionale Anforderungen
7. Globale Testfälle
8. Systemmodelle
   - Szenarien
   - Anwendungsfälle
   - Objektmodelle $\Longrightarrow$ UML-Klassendiagramme (letztes mal)
   - **Dynamische Modelle**
     - UML-Zustandsdiagramm
     - UML-Aktivitätsdiagramm
     - UML-Sequenzdiagramm

     } Heute!
   - Benutzerschnittstelle $\Longrightarrow$ Zeichnungen/Screenshots
9. Glossar

Begriffsklärung



Zustandsdiagramm
- Mein Entwurf bekommt Zustände

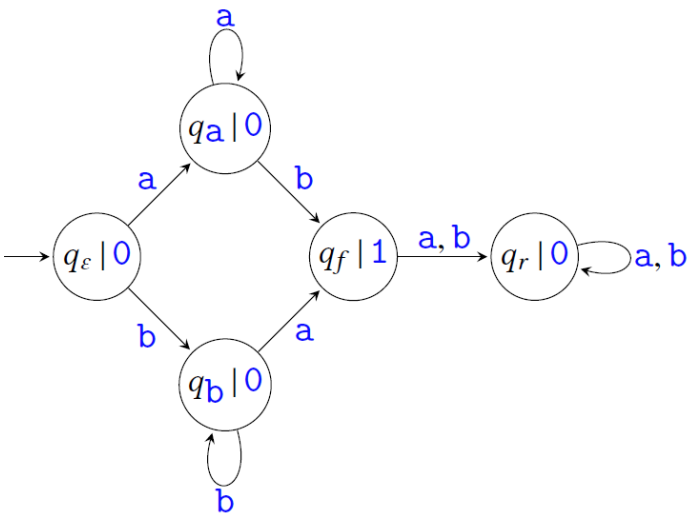
Wozu braucht man das?

Zustandsdiagramm
- Mein Entwurf bekommt Zustände

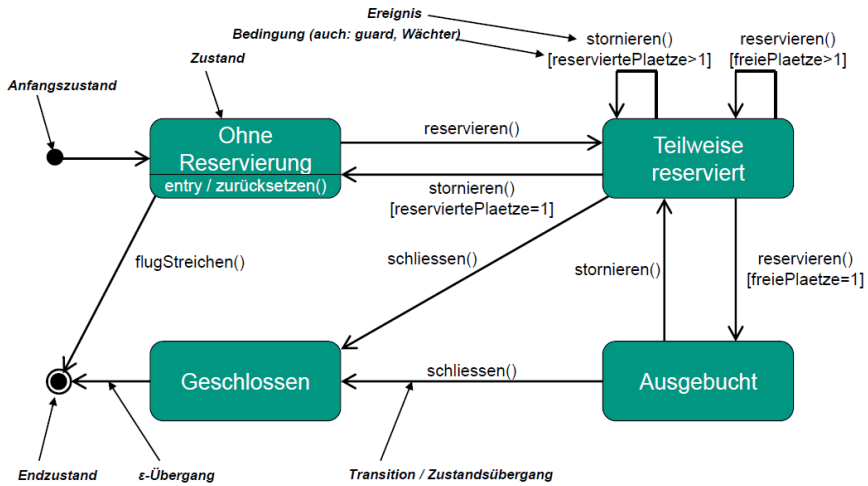
Wozu braucht man das?

- Zustand **eines Objektes** beschreiben
- Zustandsüberführungsfunktion?

Zustandsdiagramm ≈ endlicher Automat

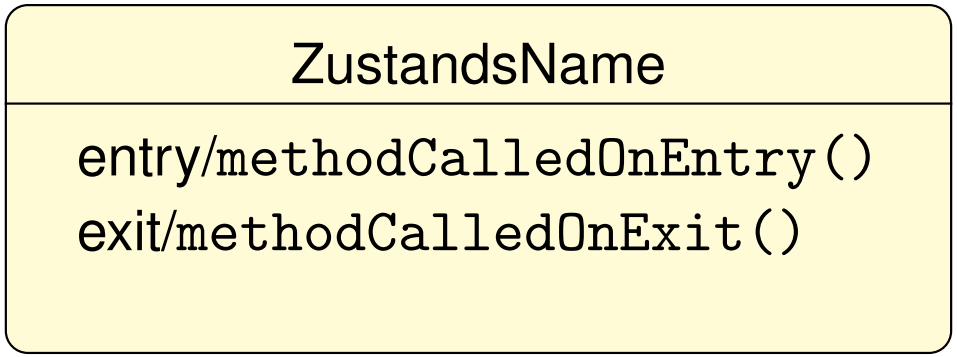


(a) GBI: DEA



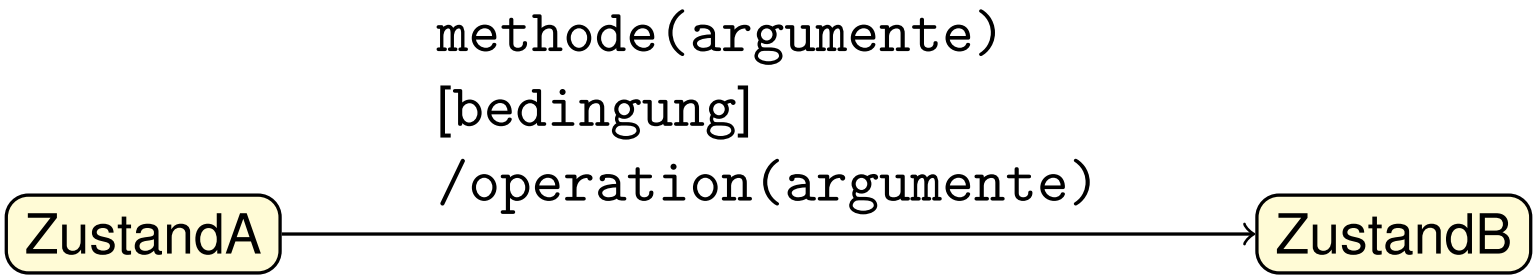
(b) SWT: Zustandsdiagramm

Zustände



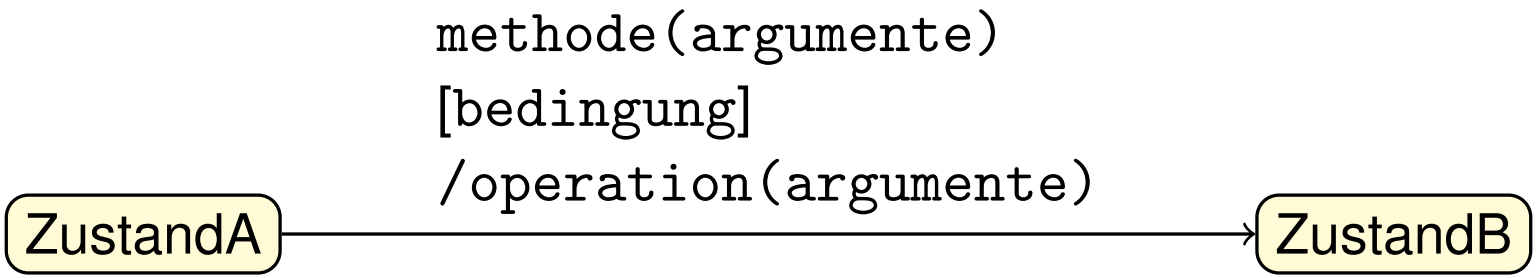
- es können Methoden angegeben werden, die ausgeführt werden bei
  - Übergang in den Zustand hinein (entry)
  - Übergäng aus dem Zustand heraus (exit)
- falls kein entry und exit, kann man Unterteilung auch weglassen

Zustandsübergänge



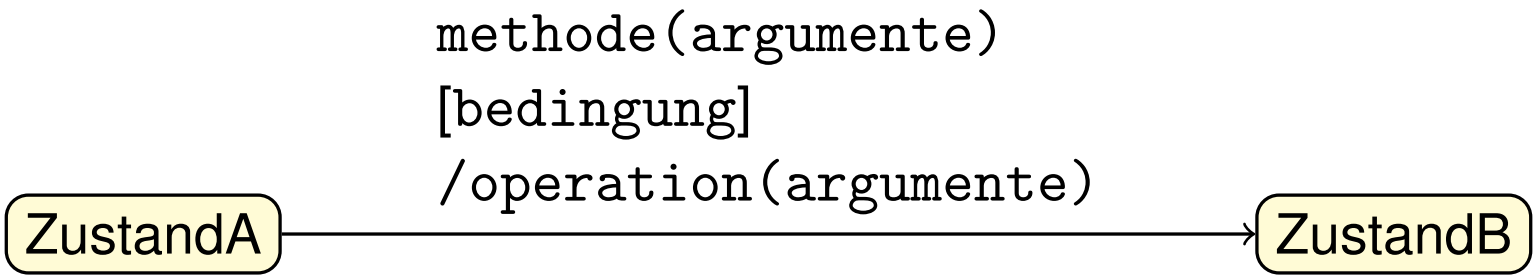
- Übergang von ZustandA nach ZustandB findet statt, wenn
  - in ZustandA `methode(argumente)` ausgeführt wird, und
  - [`bedingung`] gilt

Zustandsübergänge



- Übergang von ZustandA nach ZustandB findet statt, wenn
  - in ZustandA `methode(argumente)` ausgeführt wird, und
  - [`bedingung`] gilt
- `methode(argumente)` ist optional, ϵ-Übergänge sind möglich
- [`bedingung`] ebenfalls optional
- `/operation(argumente)` wird bei Übergang ausgeführt
  - ist auch optional

Zustandsübergänge

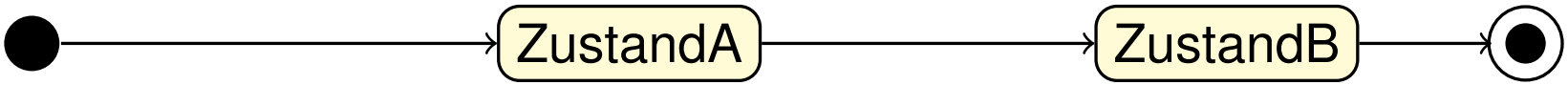


- Übergang von ZustandA nach ZustandB findet statt, wenn
  - in ZustandA `methode(argumente)` ausgeführt wird, und
  - [`bedingung`] gilt
- `methode(argumente)` ist optional, ϵ-Übergänge sind möglich
- [`bedingung`] ebenfalls optional
- `/operation(argumente)` wird bei Übergang ausgeführt
  - ist auch optional
- auch reflexive Übergänge möglich

Mögliche Übergänge

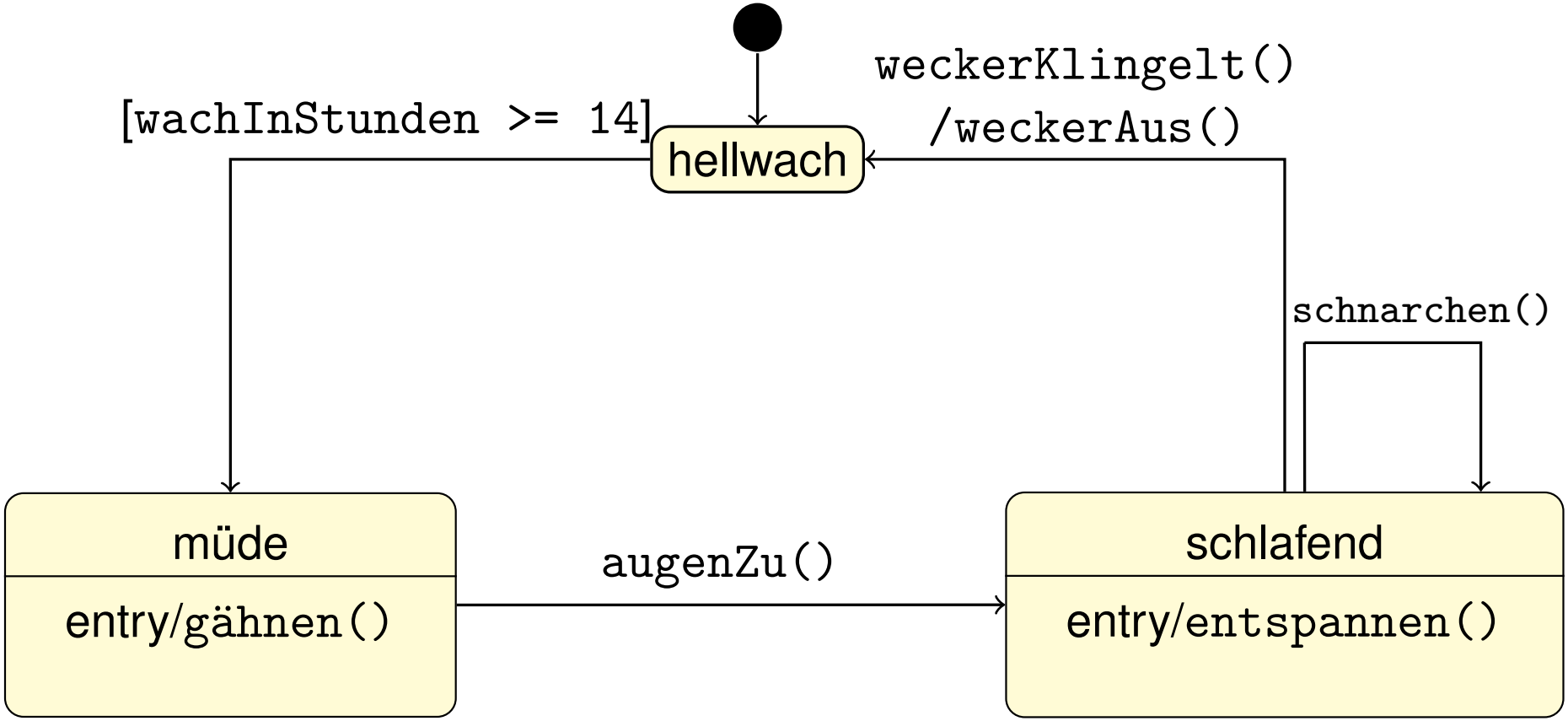
(Tafel)

Start- und Endzustand

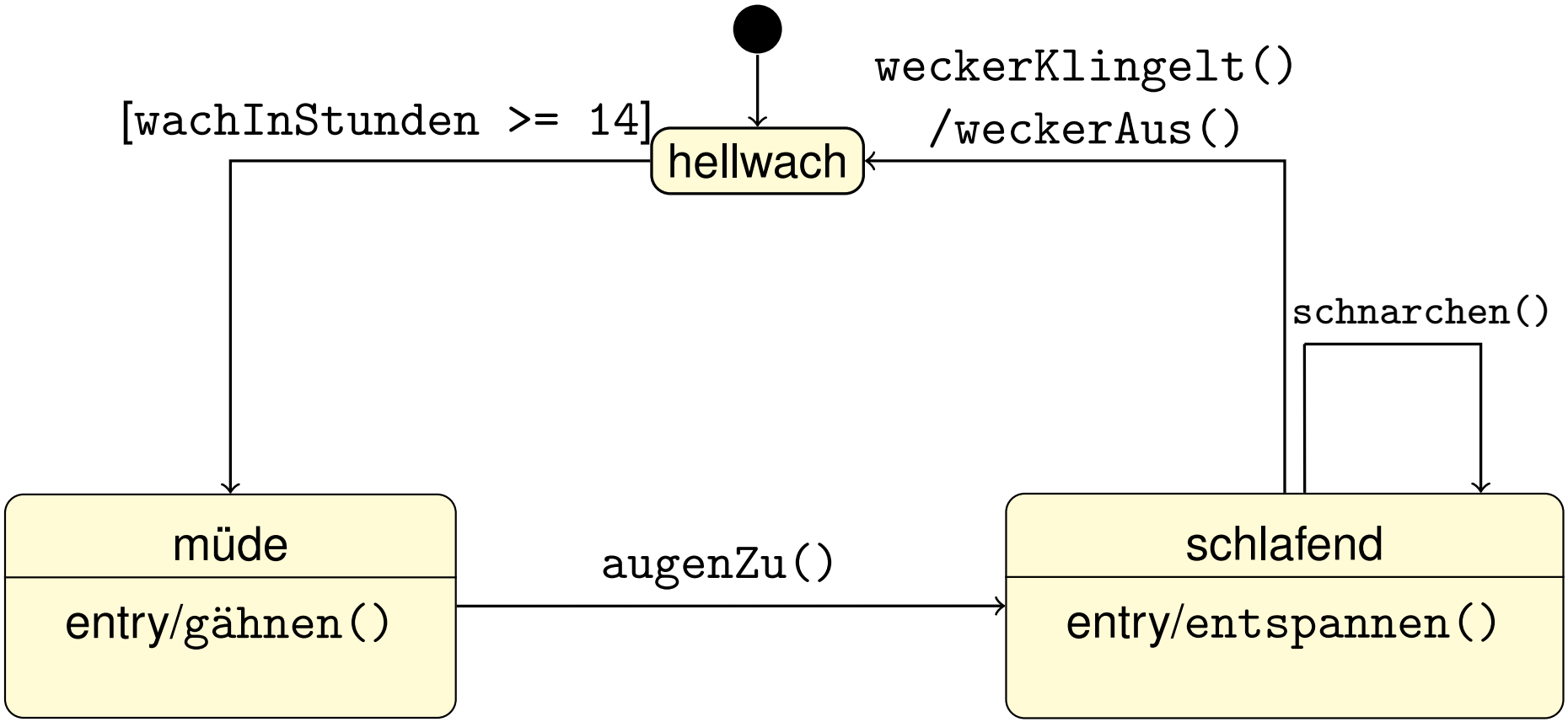


- Startzustand muss immer da sein
- Endzustand ist optional

Beispiel

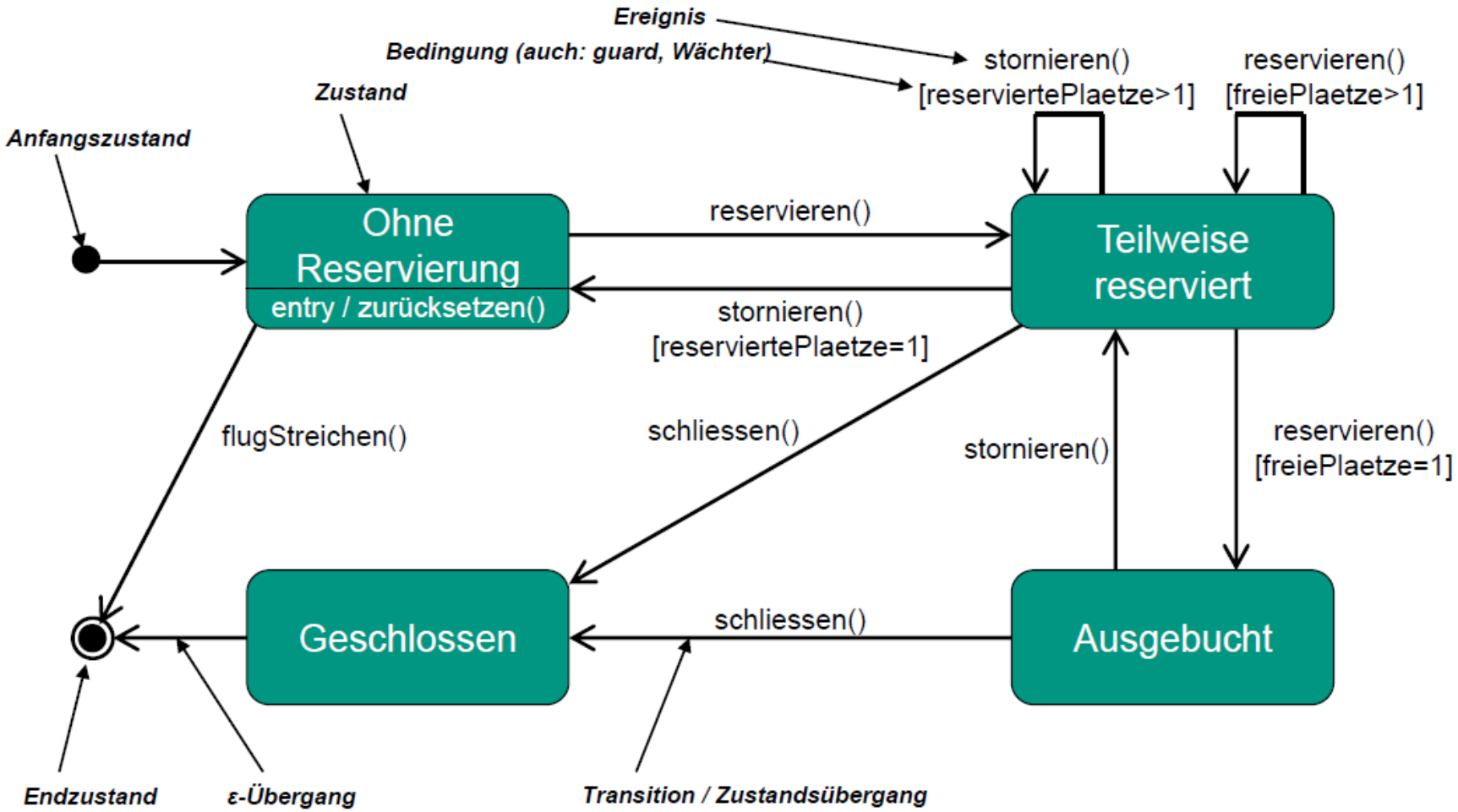

[wachInStunden >= 14]
hellwach
weckerKlingelt()
/weckerAus()
schnarchen()
müde
entry/gähnen()
augenZu()
schlafend
entry/entspannen()

Beispiel

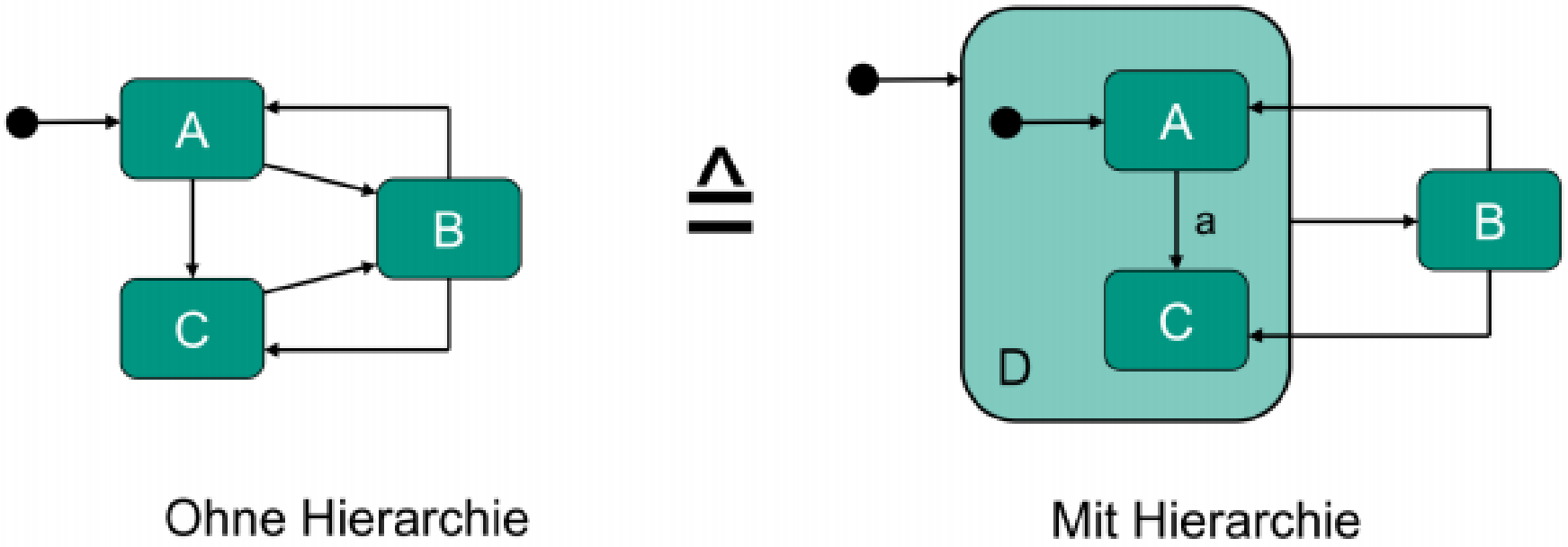


- Achtung, reflexive Übergänge führen auch entry und exit aus!
  - `entspannen()` wird ggf. mehrfach aufgerufen

Zustandsdiagramm: Syntax



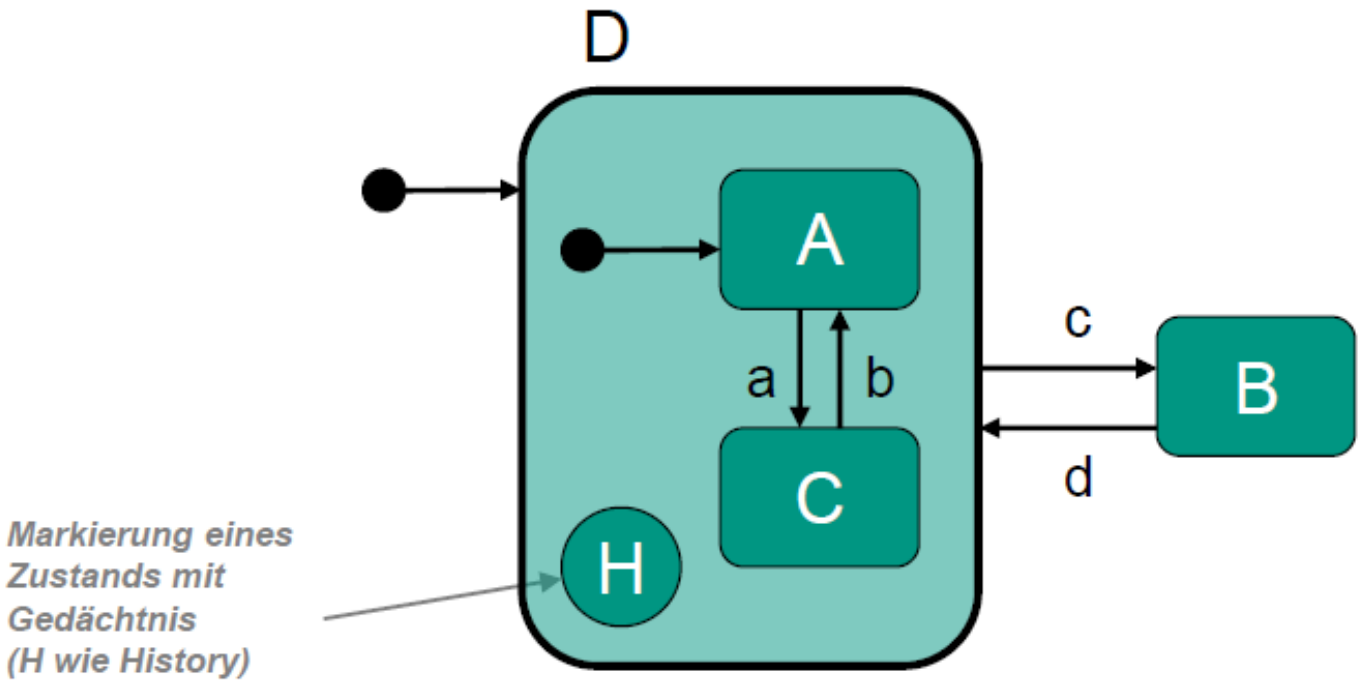
Zustandsdiagramm: Hierarchie



- nicht mächtiger, aber übersichtlicher

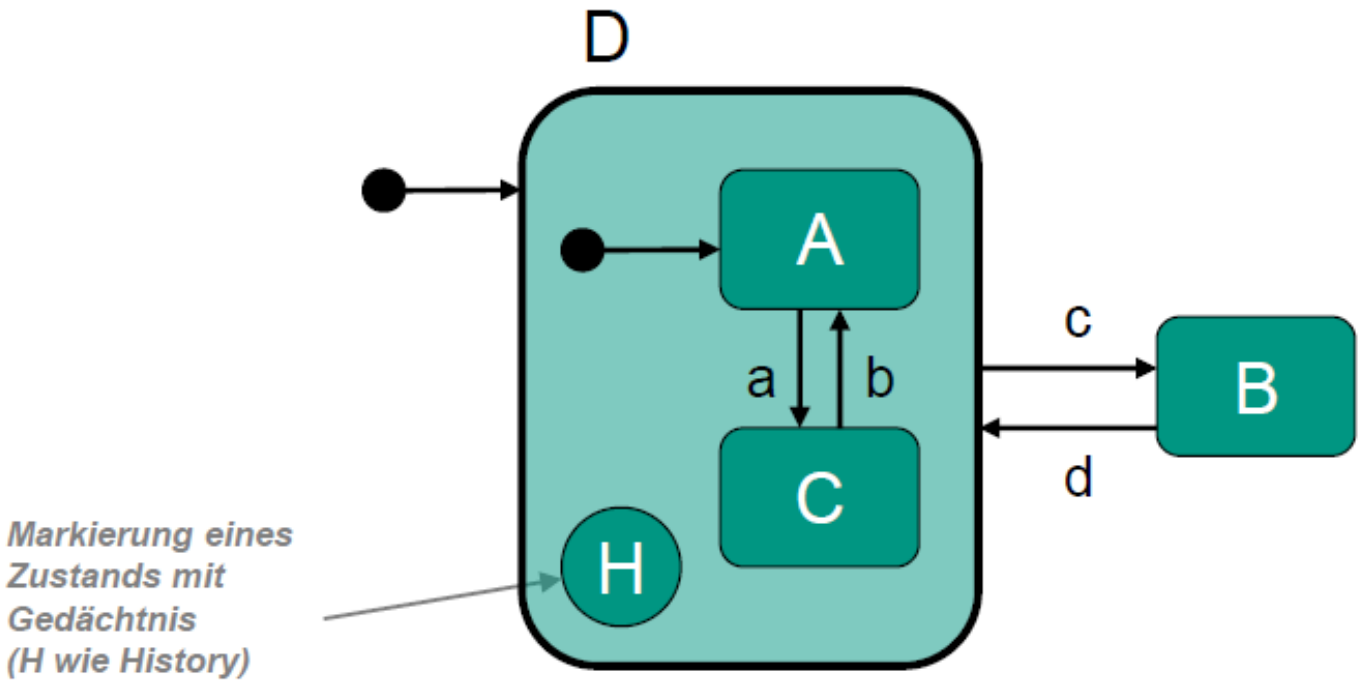
Zustandsdiagramm: Hierarchie - History

- History-Element, damit sich Hierarchie den letzten Zustand merkt



Zustandsdiagramm: Hierarchie - History

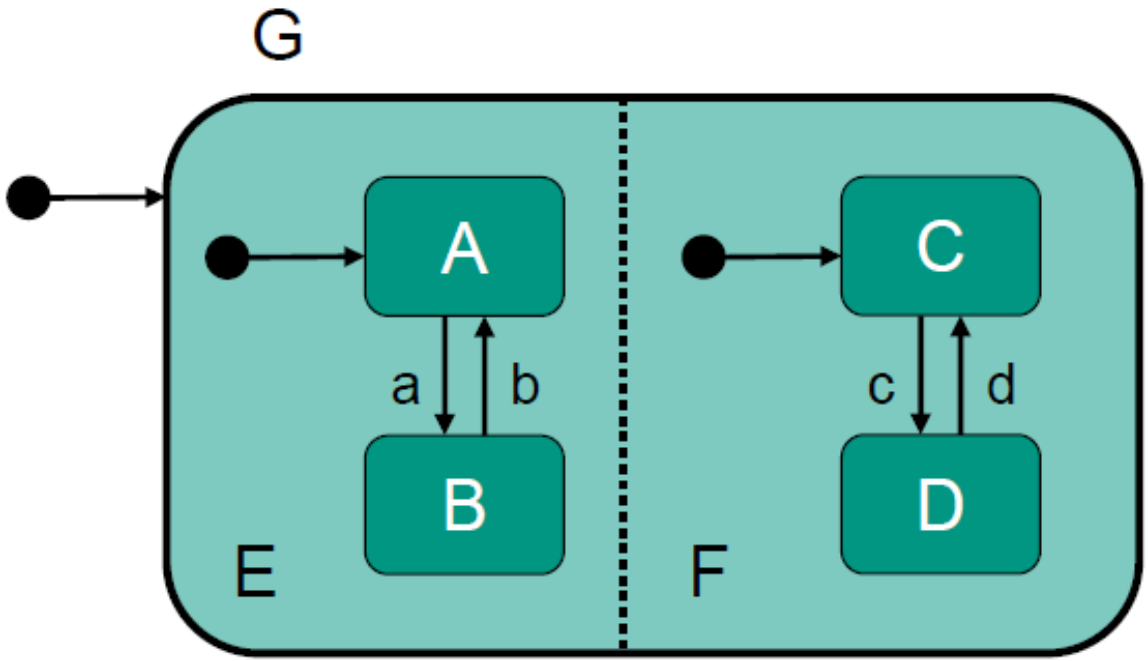
- History-Element, damit sich Hierarchie den letzten Zustand merkt



- a, c, d $\Longrightarrow$ C (ohne History wäre es A)

Zustandsdiagramm: Nebenläufigkeit

- mehrere Zustandsdiagramme in einem

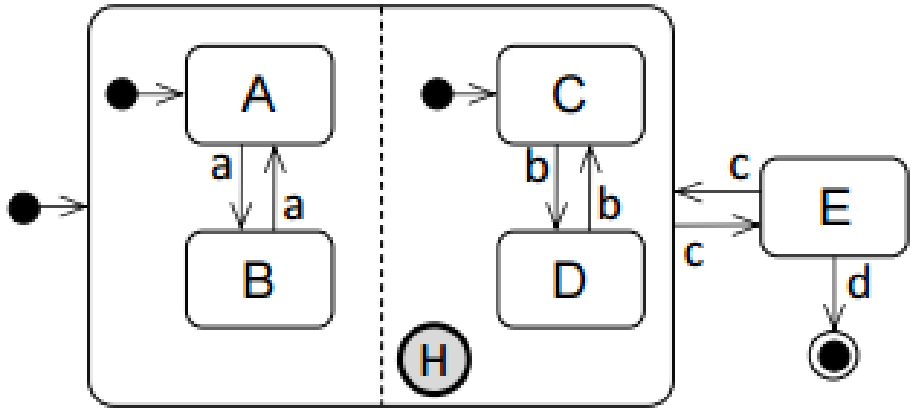


- Zustand angegeben durch Kombination aus Zuständen
  - z.B. AxC, BxC, ...

Klausuraufgabe SS09

Aufgabe

Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.

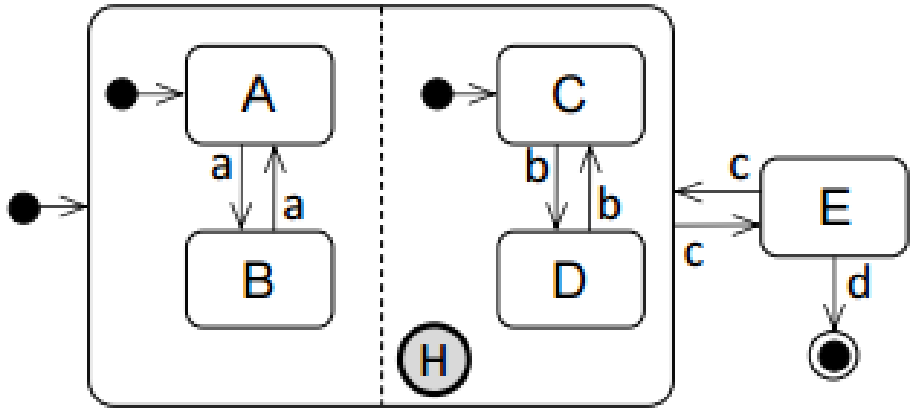


- 1. Eingabefolge: a, b, c, c

Klausuraufgabe SS09

Aufgabe

Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.

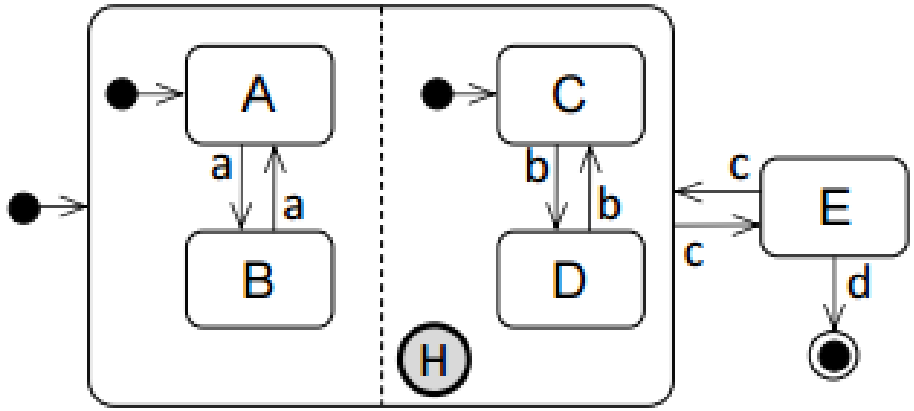


- 1. Eingabefolge: a, b, c, c $\Longrightarrow$ AxD
- 2. Eingabefolge: c, c, a, b, b, a, c, c, a

Klausuraufgabe SS09

Aufgabe

Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.



- 1. Eingabefolge: a, b, c, c $\Longrightarrow$ AxD
- 2. Eingabefolge: c, c, a, b, b, a, c, c, a $\Longrightarrow$ BxC

Klausuraufgabe SS09 (verkürzt)

Szenario

Zu Beginn wartet der Automat auf die Auswahl des Produktes durch den Kunden. Die Produktauswahl findet in zwei Schritten statt. Zunächst wählt der Kunde die Ebene, in welcher sich das gewünschte Produkt befindet. Wählt der Kunde eine Ebene aus, die nicht existiert, wartet der Automat weiter auf die Produktauswahl. Ist die Ebene gewählt, gibt der Kunde das Fach des gewünschten Produktes an. Nach der Produktauswahl wirft der Kunde so lange Münzen ein, bis der eingeworfene Betrag gleich oder größer dem Preis des ausgewählten Produktes ist. Solange der Kunde nicht ausreichend Geld in den Automaten eingeworfen hat, wartet der Automat auf den Einwurf des fehlenden Geldbetrages. Hat der Kunde ausreichend Geld eingeworfen, befördert der Automat das gewählte Produkt in den Ausgabeschacht. Danach entnimmt der Kunde das Produkt und der Automat wartet auf die nächste Produktauswahl.

Modellieren Sie das Verhalten des Automaten wie im obigen Szenario beschrieben als UML-Zustandsdiagramm. Geben Sie zu jedem Übergang das auslösende Ereignis sowie ggf. die notwendigen Bedingungen an.

Aktivitätsdiagramm - Allgemein

Wozu braucht man das?

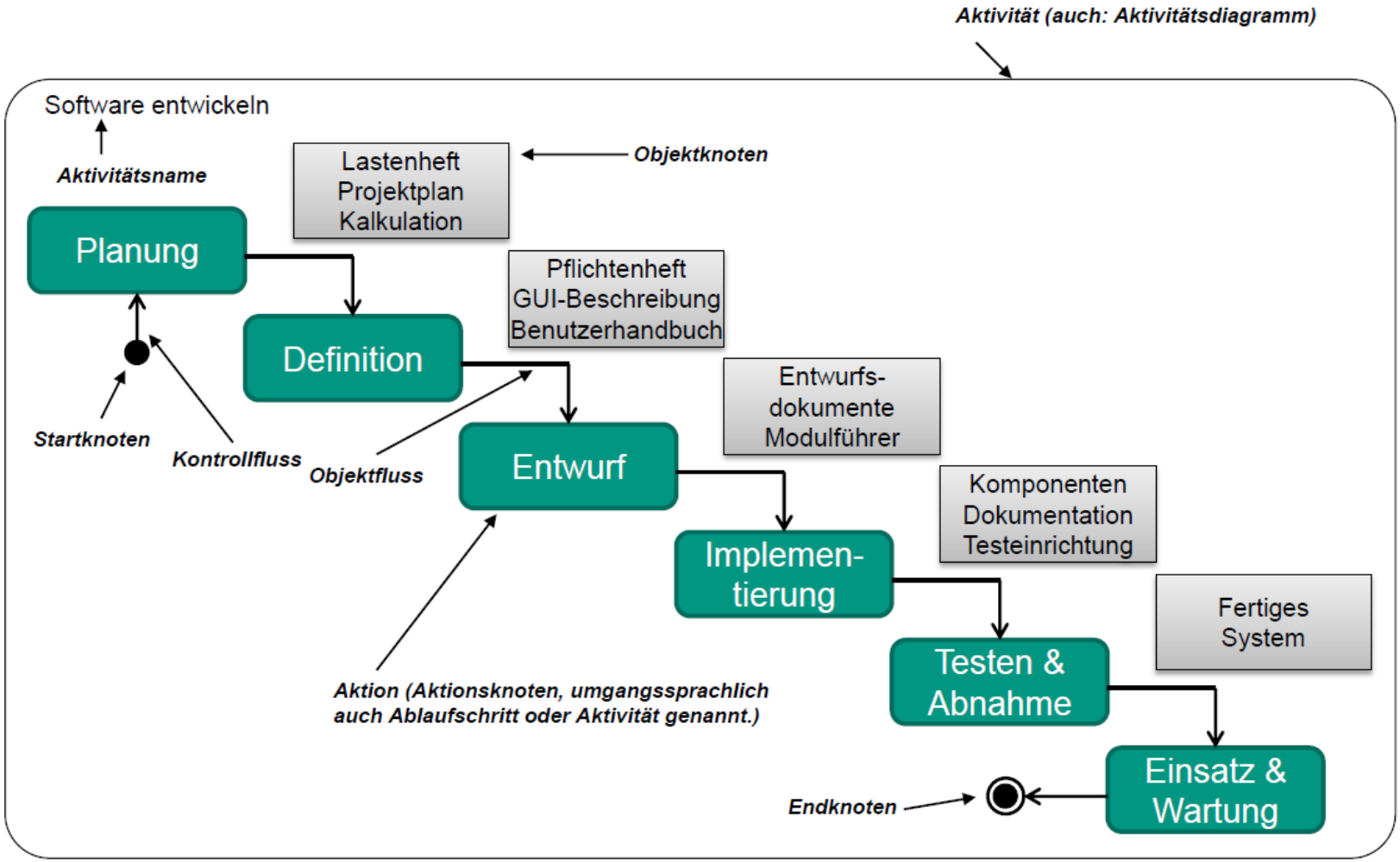
Aktivitätsdiagramm - Allgemein

Wozu braucht man das?

- Ablaufbeschreibungen (Kontrollfluss, Objektfluss)
- i.A. **mehrere verschiedene** Objekte

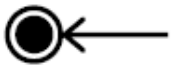
Aktivitätsdiagramm - Beispiel

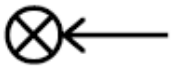
- ist ebenfalls nicht neues!



Aktivitätsdiagramm - Syntax

- Aktionen
  - Elementare Aktion
  - Verschachtelte Aktion
- Knoten
  - Startknoten
    - Startpunkt eines Ablaufs
  - Endknoten
    - Beendet alle Aktionen und Kontrollflüsse
  - Ablaufende
    - Beendet einen einzelnen Objekt- und Kontrollfluss



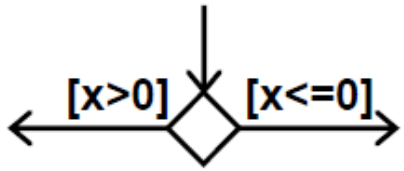


Aktivitätsdiagramm - Ablauf

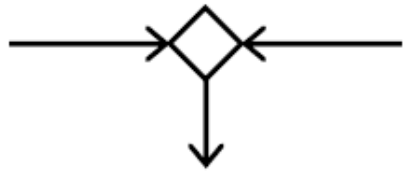
- Start am Startknoten mit einer Marke
- Aktionen werden erst ausgeführt, wenn an jedem Eingang eine Marke anliegt
- wurde eine Aktion ausgeführt, erscheinen an all ihren Ausgängen Marken

Aktivitätsdiagramm - Syntax

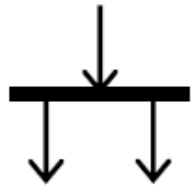
- Entscheidung
  - bedingte Verzweigung



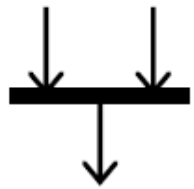
- Zusammenführung
  - „oder“-Verknüpfung



- Teilung
  - Aufteilung eines Kontrollflusses

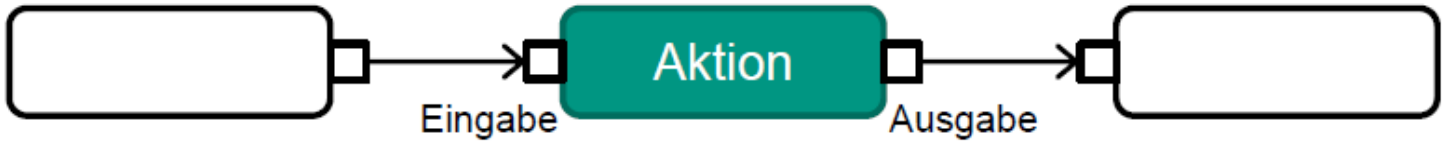


- Synchronisation
  - „und“-Verknüpfung

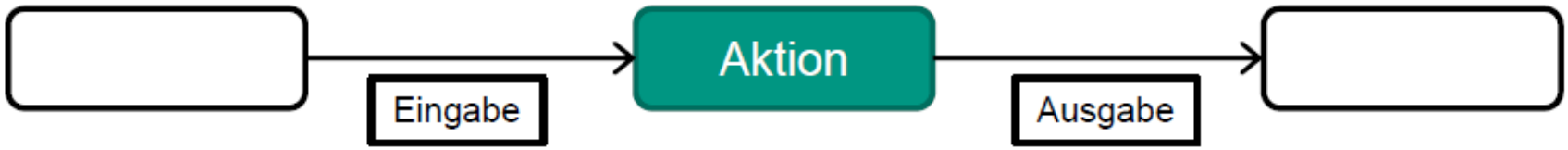


Aktivitätsdiagramm - Syntax

- Objektknoten
  - Eingabe- und Ausgabedaten einer Aktion
  - Darstellung durch Stecker (engl. *pin*)

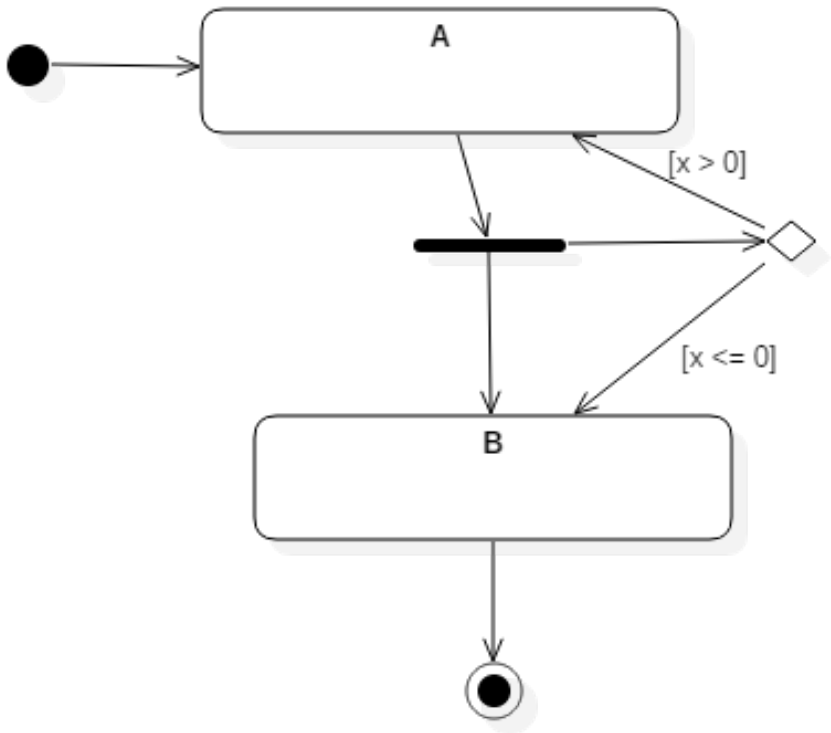


  - Alternative Darstellungen



Aktivitätsdiagramm - Beispiel

Wie kommt man hier zum Endknoten?



Noch ein Beispiel

(Tafel)

Klausuraufgabe SS 11

Aufgabenstellung

Modellieren Sie das gegebene Szenario als UML-Aktivitätsdiagramm.
Verwenden Sie bei Ihrer Modellierung korrekte UML-Notation.
Objektflüsse müssen Sie nicht modellieren.

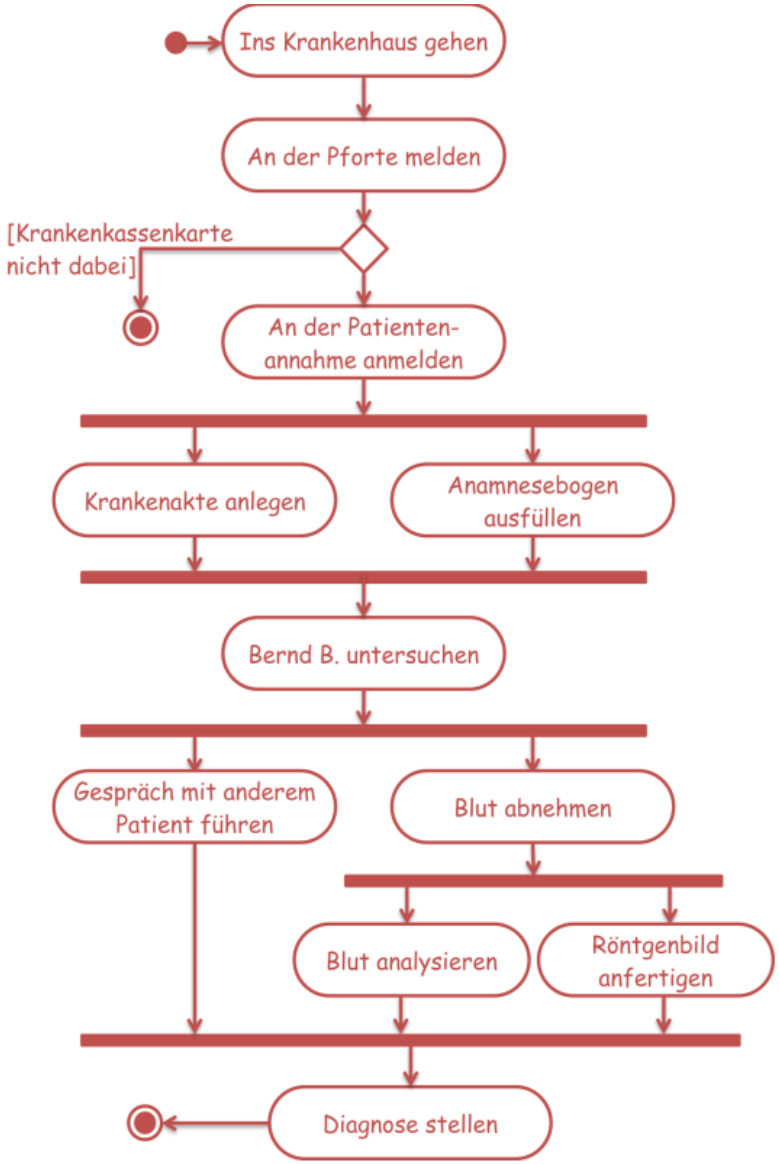
(Szenario auf nächster Folie)

Klausuraufgabe SS 11: Szenario

Szenario

Bernd Bruegge fühlt sich nicht wohl und möchte sich untersuchen lassen, um eine Diagnose zu erhalten. Er geht dazu ins Krankenhaus und meldet sich an der Pforte. Der Pförtner macht ihn darauf aufmerksam, dass er seine Krankenkassenkarte dabei haben muss – falls nicht, kann ihm keine Diagnose gestellt werden. Bernd B. hat seine Krankenkassenkarte dabei und lässt sich daher den Weg zur Patientenaufnahme zeigen, da er sich dort zunächst anmelden muss. Während der Sachbearbeiter an der Patientenaufnahme die Krankenakte mit den persönlichen Daten anlegt, füllt Bernd B. den Anamnesebogen aus. Danach wird Bernd B. vom Arzt untersucht. Die Untersuchung ergibt, dass der Arzt zur weiteren Diagnose ein Blutbild und ein Röntgenbild der Lunge benötigt und er in der Zwischenzeit ein Gespräch mit einem anderen Patienten führen kann. Dazu wird Bernd B. zunächst Blut abgenommen. Während das Blut im Labor analysiert wird, geht Bernd B. in die radiologische Abteilung und lässt das Röntgenbild der Lunge anfertigen. Als der Arzt von seinem Gespräch mit dem anderen Patienten zurückkommt, liegen das Röntgen- und das Blutbild vor und er stellt Bernd B. eine Diagnose.

Musterlösung



Start- und Endzustand: Je 0,25 P.
Je Zustand: 0,25 P.
Je Splitter/Joiner 0,25 P.
Geschachtelte Parallelitaet: 0,5 P
Fallunterscheidung, deren Beschriftungen und dortiger Endzustand: Je 0,25 P.

Sequenzdiagramm - Allgemein

Wozu braucht man das?

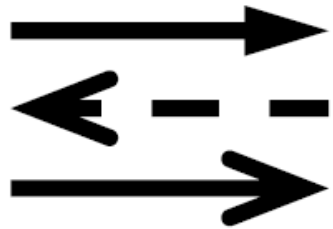
Sequenzdiagramm - Allgemein

Wozu braucht man das?

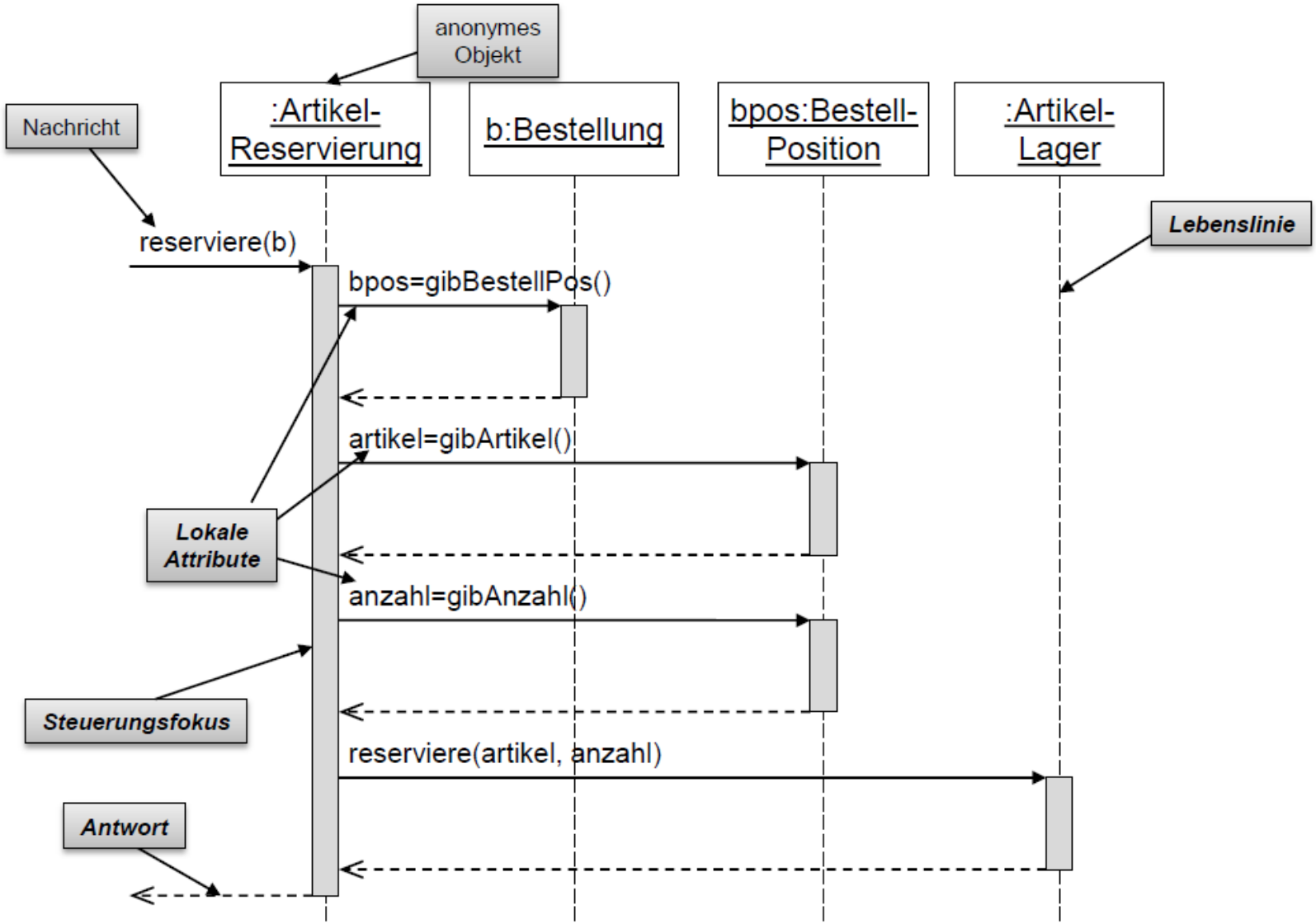
- stellt den möglichen Ablauf eines Anwendungsfalls dar
- **zeitlicher Verlauf** von Methodenaufrufen, Objekterstellung, Objektzerstörung

Sequenzdiagramm - Syntax

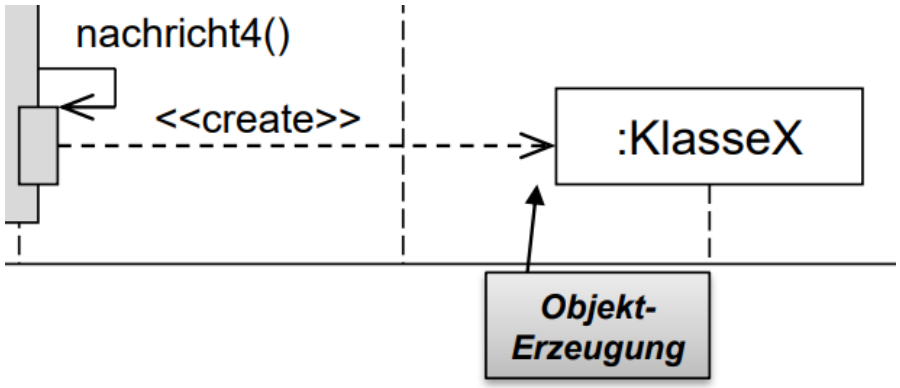
- Zeit verläuft von oben nach unten
- ein Kasten pro Objekt (`instanzName: KlassenName`)
- Lebenslinie
  - gestrichelte senkrechte Linie
  - eine pro Objekt, solange wie Objekt „lebt“
- Steuerungsfokus
  - dicker Balken über Lebenslinie
  - zeigt, dass Objekt gerade aktiv ist
- Nachrichtentypen (für Methodenaufrufe und Rückgabe)
  - Synchrone Nachricht (blockierend)
  - Antwort (optional)
  - Asynchrone Nachricht



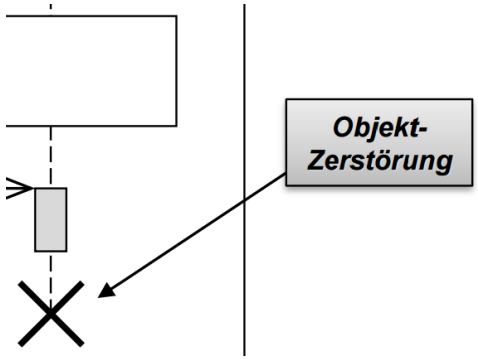
Sequenzdiagramm - Syntax



Sequenzdiagramm - Syntax

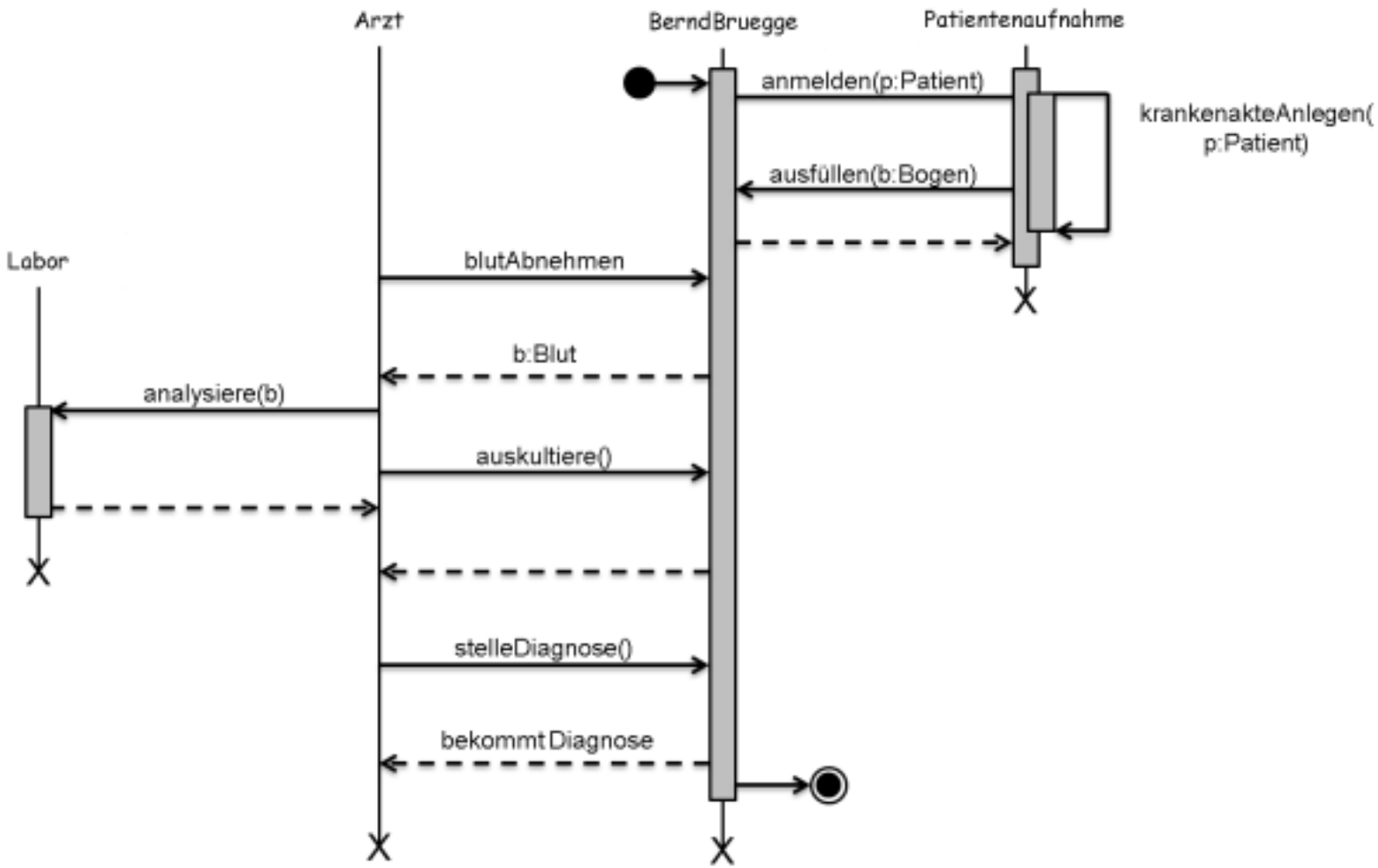


(a) Objekt-Erzeugung



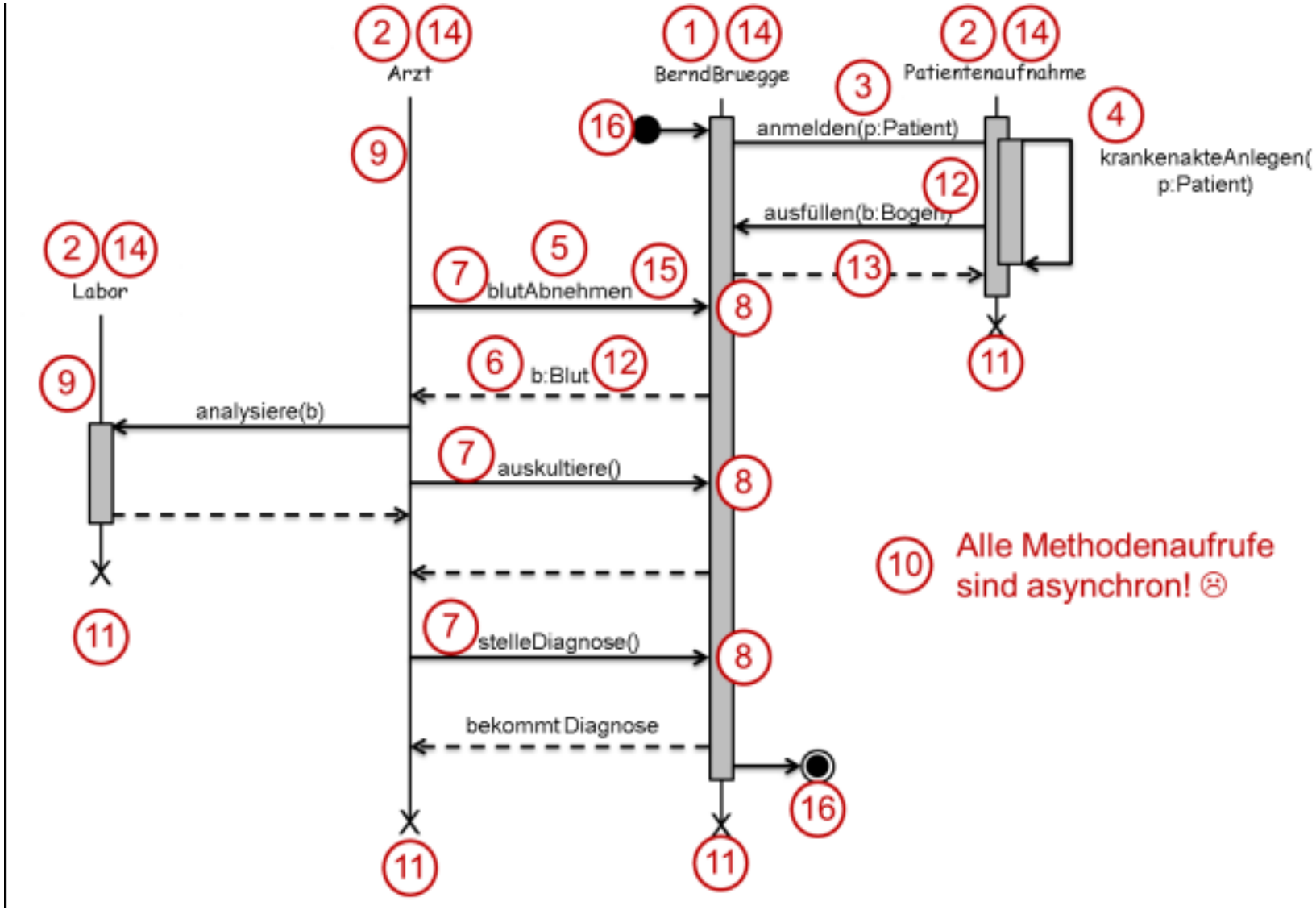
(b) Objekt-Zerstörung

Klausuraufgabe SS14



Hier stimmt was nicht. . . , aber was?

Musterlösung 1



Musterlösung 2

1. Was ist „BerndBruegge" für eine Klasse? →Patient?
2. Anonyme Instanzen mit Doppelpunkt kennzeichnen: Labor, Arzt, Patientenaufnahme.
3. Pfeilspitze & Rückgabe fehlen bei anmelden(p:Patient)! Klasse Patient und Instanz p (Argument von anmelden(...)) wird nirgendwo sonst verwendet / existiert nicht.
4. krankenakteAnlegen(p:Patient) hat keine Rückgabe bzw. erzeugt nichts (Konstruktoraufruf für Krankenakte?).
5. Für korrekte Syntax fehlen hinter „blutAbnehmen" Klammern „()". (Notiz: UML sieht Klammern nicht zwingend vor; wir akzeptieren diesen Fehler trotzdem.)
6. Bei blutAbnehmen(): <<Klammerung>> bei der Rückgabe von b:Blut vergessen.
7. Methoden werden auf der falschen Instanz aufgerufen – Welche Klasse hätte die Methoden auskultiere(), ~~blutAbnehmen()~~ und stelleDiagnose() im Klassendiagramm?
8. Der Fokusbalken fehlt bei auskultiere(), blutAbnehmen(), stelleDiagnose().
9. Notation: Lebenslinien werden nicht durchgezogen – das gilt für alle vier Instanzen ...

Musterlösung 3

10. Alle Methodenaufrufe sind asynchron dank offener Pfeilspitze – das sollte nicht überall so sein! (Nur) ok bei: krankenakteAnlegen(), ausfüllen(b:Bogen), analysiere(b). Gibt nur einen Punkt, wenn eine konkrete (d. h. szenariobezogene) Begründung gegeben wird.
11. Die Objektzerstörung ist semantisch zweifelhaft: Sollen die Instanzen tatsächlich entfernt werden? Nein!
12. Interne Konsistenz: Instanzen von Bogen und Blut haben denselben Namen „b".
13. Rückkehr von ausfülle(Bogen): Rückgabewert fehlt
14. ~~Notation: Kasten um Instanzen fehlt überall!~~
15. Klammern bei blutAbnehmen() vergessen.
16. ~~Startzustände (und Endzustände) gibt es im Sequenzdiagramm nicht.~~
17. Fokusbalken bei ausfüllen(Bogen) fehlt
18. (Mehrere) Fokusbalken bei Arzt fehlen
19. Instanznamen werden unterstrichen
20. Die Lebenslinie von Labor soll ganz oben beginnen auf einer Höhe mit den anderen Instanzen – das Labor wird nicht im Verlauf des Diagramms erzeugt!
21. Notation: Rückgabewerte in «» einklammern – hier wohl «d:Diagnose».

Aufgabe: Code → Sequenzdiagramm

```
public class Student {
    public void schlechtesGewissen(Übungsblatt b) {
        Aufgabe a = b.gibAufgabe();
        Loesung l = löseAufgabe(a);
        b.setzeLösung(a, l);
        LEZ.abgeben(b);
    }

    private Lösung löseAufgabe(Aufgabe a) {
        // SWT1-Magie erstellt Lösungs-Objekt
    }
}
```

Aufgabe

Aufruf von `schlechtesGewissen` als Sequenzdiagramm modellieren

Wozu nochmal was?

Aktivitätsdiagramm

- Kontrollfluss, Objektfluss

Wozu nochmal was?

Aktivitätsdiagramm

- Kontrollfluss, Objektfluss

Zustandsdiagramm

- Zustände eines Objektes

Wozu nochmal was?

Aktivitätsdiagramm

- Kontrollfluss, Objektfluss

Zustandsdiagramm

- Zustände eines Objektes

Sequenzdiagramm

- zeitlicher Verlauf
- Aufrufe zwischen verschiedenen Objekten/Klassen

Tipps - 3. Übungsblatt

Aufgabe 1-3: Plug-In programmieren

Tipps - 3. Übungsblatt

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle . . .

Tipps - 3. Übungsblatt

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle . . .
- Fügt Abhängigkeiten in die jeweilige Untermodul-pom ein

Tipps - 3. Übungsblatt

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Fügt Abhängigkeiten in die jeweilige Untermodul-pom ein
- Java Swing benutzen
  - zum Teil müsst ihr in existierendem JMJRST-Code rumfummeln
  - nicht an dem Stil von JMJRST orientieren!
  - relevante Klassen u.A. `JMenu`, `JMenuItem`, `JFrame`, `JPanel`, ...

Tipps - 3. Übungsblatt

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Fügt Abhängigkeiten in die jeweilige Untermodul-pom ein
- Java Swing benutzen
  - zum Teil müsst ihr in existierendem JMJRST-Code rumfummeln
  - nicht an dem Stil von JMJRST orientieren!
  - relevante Klassen u.A. `JMenu`, `JMenuItem`, `JFrame`, `JPanel`, ...

Aufgabe 4: Aktivitätsdiagramm

- Partitionen siehe VL
- Kombination als verschachtelte Aktion für Übersichtlichkeit
  - Notation!

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

- History-Vererbung? siehe flache vs. tiefe History!

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

- History-Vererbung? siehe flache vs. tiefe History!

Aufgabe 6: Sequenzdiagramm

- zum Text den Quellcode anschauen, dann werden Bezugsobjekte klarer

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

- History-Vererbung? siehe flache vs. tiefe History!

Aufgabe 6: Sequenzdiagramm

- zum Text den Quellcode anschauen, dann werden Bezugsobjekte klarer
- für Verzweigungssyntax siehe VL

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

- History-Vererbung? siehe flache vs. tiefe History!

Aufgabe 6: Sequenzdiagramm

- zum Text den Quellcode anschauen, dann werden Bezugsobjekte klarer
- für Verzweigungssyntax siehe VL
- statische Methode: bei den „Objekt-Kästen“ folgendes weglassen
  - Unterstreichung + Name des Objekts + Doppelpunkt
  - also statt `lez:LEZ` einfach `LEZ`
  - Methodenaufruf auf dem Pfeil kann man zusätzlich auch noch unterstreichen, muss man aber nicht

Tipps - 3. Übungsblatt

Aufgabe 5: Zustandsdiagramm

- History-Vererbung? siehe flache vs. tiefe History!

Aufgabe 6: Sequenzdiagramm

- zum Text den Quellcode anschauen, dann werden Bezugsobjekte klarer
- für Verzweigungssyntax siehe VL
- statische Methode: bei den „Objekt-Kästen“ folgendes weglassen
  - Unterstreichung + Name des Objekts + Doppelpunkt
  - also statt `lez:LEZ` einfach `LEZ`
  - Methodenaufruf auf dem Pfeil kann man zusätzlich auch noch unterstreichen, muss man aber nicht
- Lebenslinie + Kasten erst hinmalen wenn Objekt schon existiert

Tipps - 3. Übungsblatt

Aufgabe 7: Substitutionsprinzip

- Definition anschauen und „scharf hinsehen“, was schief gehen kann

Die nächste Frist kommt bestimmt

Abgabe

- Deadline am 05.06. um 12:00
- Aufgabe 4+5+6+7 handschriftlich
  - auf korrekte Syntax achten
  - Lineal/Geodreieck dürfen benutzt werden :-)
- an das Deckblatt denken

Bis dann! (dann := 04.06.19)