

# Softwaretechnik 1 - 2. Tutorium

Tutorium 18

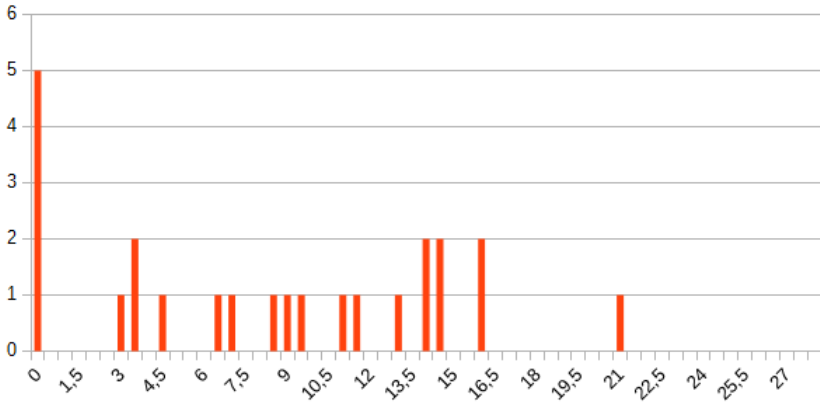
Felix Bachmann | 22.05.2018

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



## 2. Übungsblatt Statistik

$n = 24$



## 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.

- $\implies$  nur das offizielle Deckblatt verwenden! (ab nächstes Mal Abzug)

## 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.

- $\implies$  nur das offizielle Deckblatt verwenden! (ab nächstes Mal Abzug)
- häufigster Fehler: Aufgaben nicht abgegeben

## Aufgabe 1 (Lastenheft)

- Unnötiges aus Vorlage durfte man löschen (z.B. “Siehe <https://en.wikibooks.org/wiki/LaTeX/Glossary>“ oder “Szenarien“)

## Aufgabe 1 (Lastenheft)

- Unnötiges aus Vorlage durfte man löschen (z.B. “Siehe <https://en.wikibooks.org/wiki/LaTeX/Glossary>“ oder “Szenarien“)
- Anwendungsfälle beschreiben, was man in dem System tun kann  
⇒ beinhalten immer Verben! (z.B. “Hobbyfotograph“ oder “JMJRST“ ist kein Anwendungsfall)

## Aufgabe 2 (Klassendiagramm)

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen

## Aufgabe 2 (Klassendiagramm)

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen



## Aufgabe 2 (Klassendiagramm)

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen
- Ausrichtung des Bildes wurde nicht als Enum modelliert

## Aufgabe 2 (Klassendiagramm)

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen
- Ausrichtung des Bildes wurde nicht als Enum modelliert
- falsche UML-Syntax (insb. Methode, Attribute)

## Aufgabe 3 (Durchführbarkeitsanalyse)

- sehr häufig nicht abgegeben

## Aufgabe 3 (Durchführbarkeitsanalyse)

- sehr häufig nicht abgegeben
- Fragen beantworten, nicht stellen!  
z.B. “Es werden 3 Java-Entwickler benötigt.“ ergänzen durch “Da wir 5 zur Zeit untätige Java-Entwickler in der Firma haben, ist das Projekt aus personeller Sicht für die Pear Corp. durchführbar.“

## Aufgabe 4 (Geometrify programmieren)

- NullPointerExceptions wurden geworfen  
⇒ deswegen sollte man seine eigene Lösung testen

## Aufgabe 4 (Geometrify programmieren)

- NullPointerExceptions wurden geworfen  
⇒ deswegen sollte man seine eigene Lösung testen
- Java-Koordinaten != “normale” Koordinaten

## Aufgabe 4 (Geometrify programmieren)

- NullPointerExceptions wurden geworfen  
⇒ deswegen sollte man seine eigene Lösung testen
- Java-Koordinaten != “normale“ Koordinaten
- Alpha-Kanal nicht beachtet

## Aufgabe 5 (Kommandozeilen-Tool)

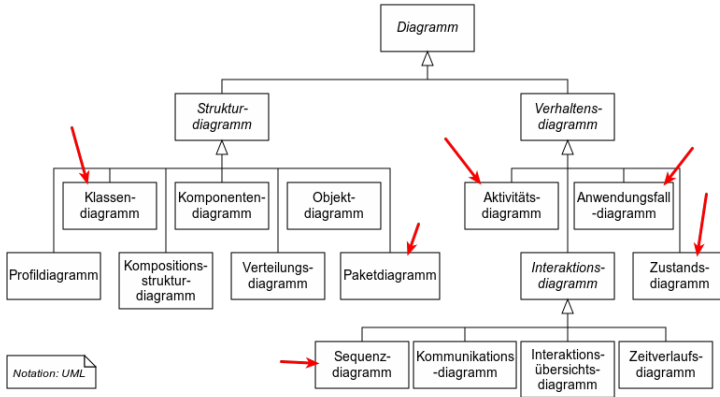
- 4 Abgaben ...

# 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  $\implies$  UML-Klassendiagramme (letztes mal)
  - **Dynamische Modelle**
    - UML-Zustandsdiagramm
    - UML-Aktivitätsdiagramm
    - UML-Sequenzdiagramm
- ⑨ Glossar

} Heute!



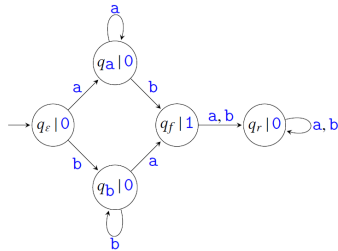


## Wozu braucht man das?

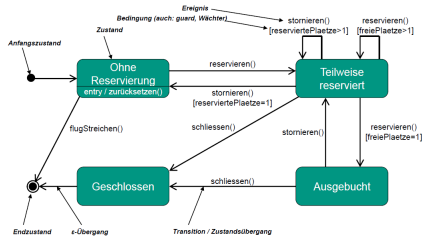
## Wozu braucht man das?

- Zustand **eines Objektes** beschreiben
- Zustandsüberföhrungsfunktion?

# Zustandsdiagramm $\approx$ endlicher Automat

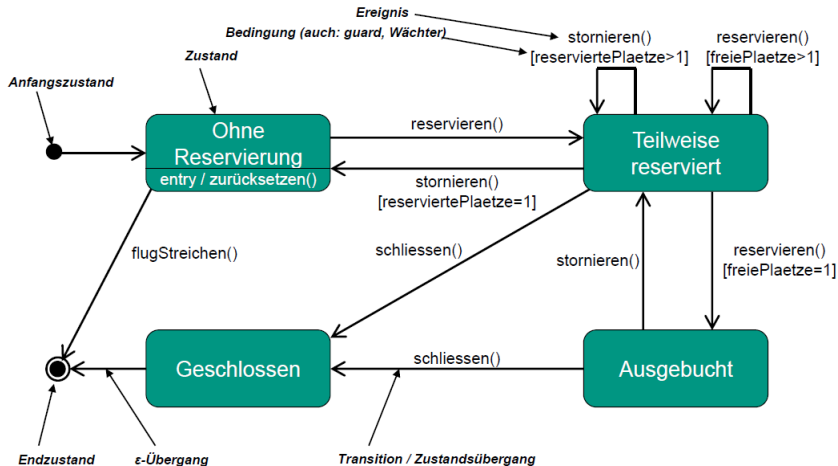


(a) GBI: DEA

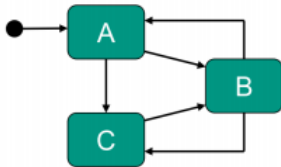


(b) SWT: Zustandsdiagramm

# Zustandsdiagramm: Syntax

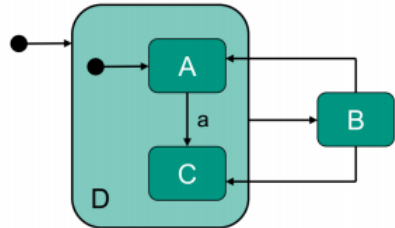


# Zustandsdiagramm: Hierarchie



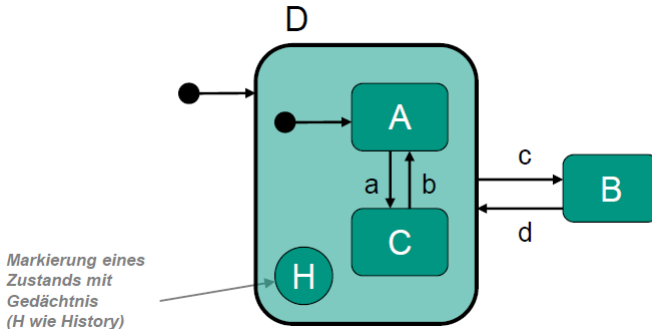
Ohne Hierarchie

$\cong$

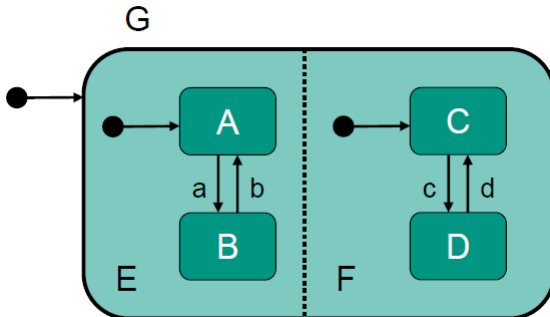


Mit Hierarchie

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

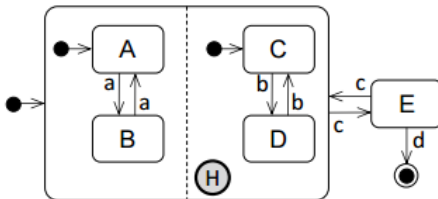


- mehrere Zustandsdiagramme in einem



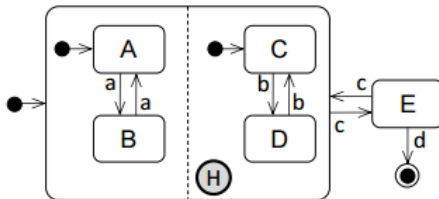


Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.



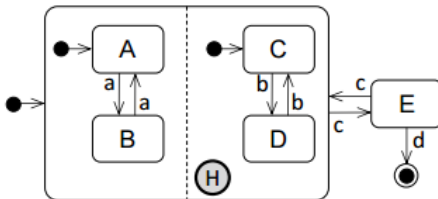
■ a, b, c, c

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



- a, b, c, c  $\implies$  AxD
- c, c, a, b, b, a, c, c, a

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



- $a, b, c, c \implies Ax D$
- $c, c, a, b, b, a, c, c, a \implies BxC$

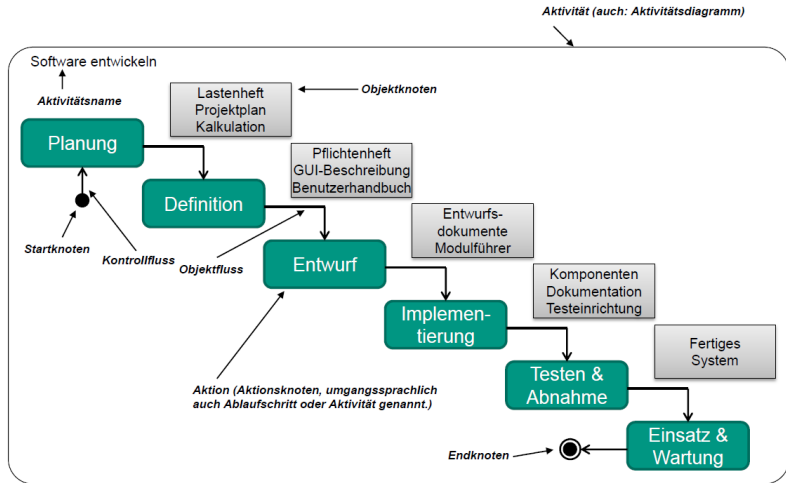
Wozu braucht man das?

## Wozu braucht man das?

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

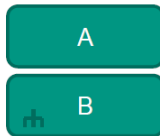
# Aktivitätsdiagramm - Beispiel

- ist ebenfalls nicht neues!



## ■ Aktionen

- Elementare Aktion
- Verschachtelte Aktion

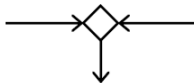
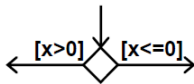


## ■ Knoten

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



- Entscheidung
  - bedingte Verzweigung
- Zusammenführung
  - „oder“-Verknüpfung
- Teilung
  - Aufteilung eines Kontrollflusses
- Synchronisation
  - „und“-Verknüpfung





## ■ Objektknoten

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

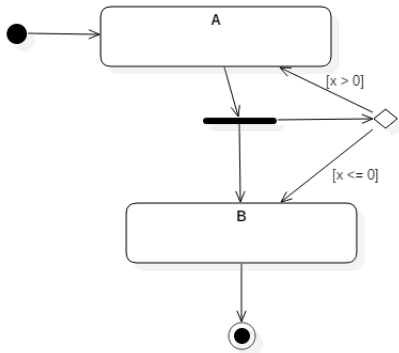


## ■ Alternative Darstellungen



- 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

Wie kommt man hier zum Endknoten?

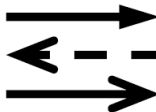


## Wozu braucht man das?

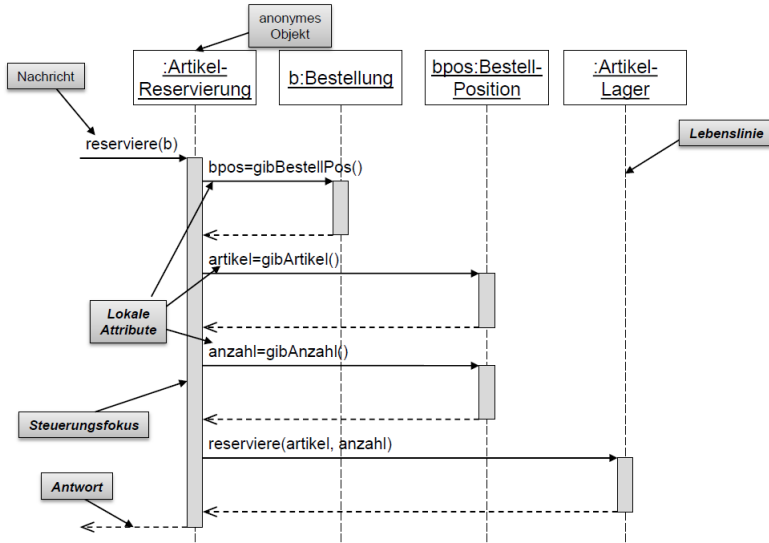
## Wozu braucht man das?

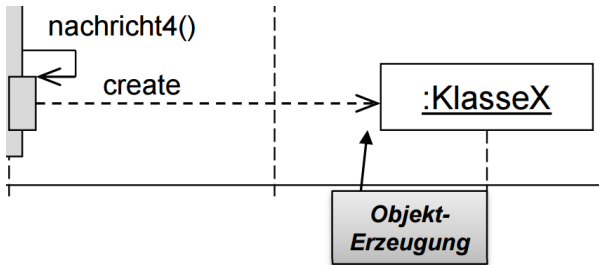
- stellt den möglichen Ablauf eines Anwendungsfalls dar
- **zeitlicher Verlauf** von Methodenaufrufen, Objekterstellung, Objektzerstörung

- Zeit verläuft von oben nach unten
- Lebenslinie
  - gestrichelte senkrechte Linie
  - eine pro Objekt
- Steuerungsfokus
  - dicker Balken über Lebenslinie
  - zeigt, dass Objekt gerade aktiv ist
- Nachrichtentypen
  - Synchrone Nachricht (blockierend)
  - Antwort (optional)
  - Asynchrone Nachricht

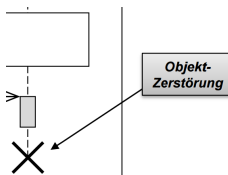


# Sequenzdiagramm - Syntax



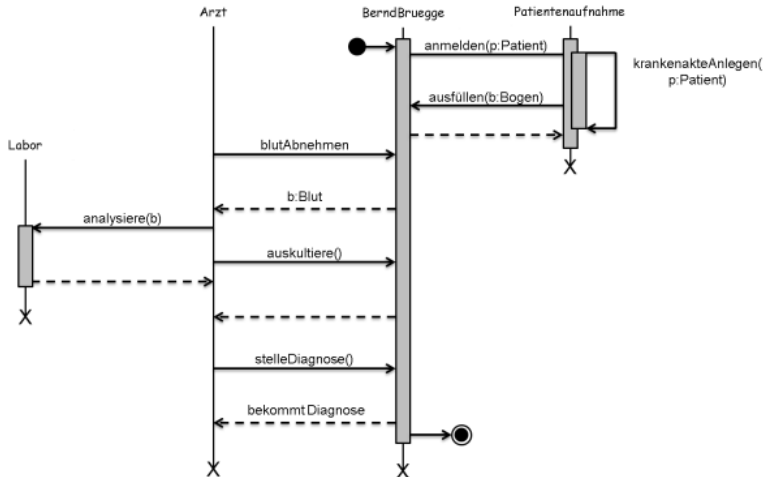


(a) Objekt-Erzeugung



(b) Objekt-Zerstörung





Hier stimmt was nicht...

# Gruppenarbeit: Jetzt seid ihr dran!

## Ablauf:

- ➊ Einteilung in Expertengruppen
- ➋ Bearbeitet eure Aufgabe in der Gruppe
  - Falls ihr Hilfe braucht, sagt Bescheid!
- ➌ Vergleicht eure Lösung mit der Musterlösung
- ➍ Durchmischen der Gruppen
- ➎ Stellt den anderen eure Aufgabe & Lösung vor
  - Erklärt insbesondere, was ihr evtl. falsch gemacht habt

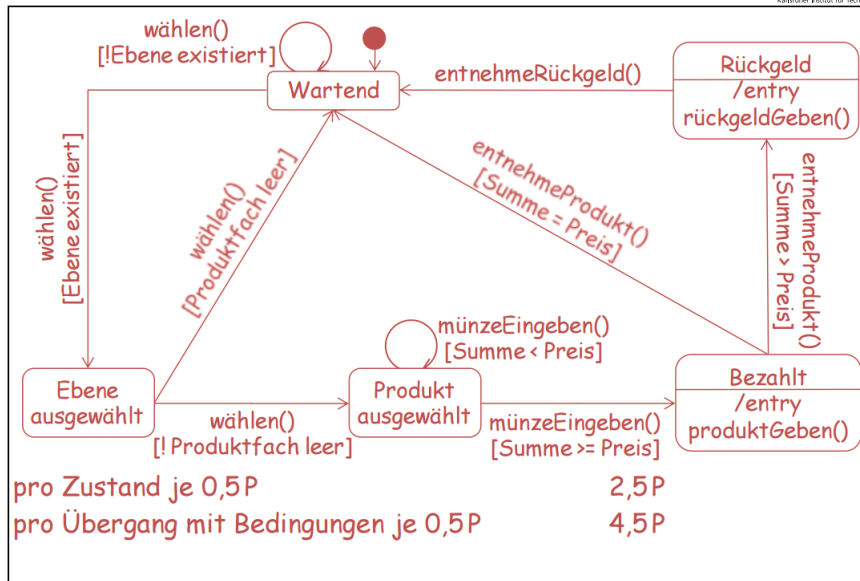
# Gruppe 1: Zustandsdiagramm (Nachklausur 09)

Gegeben ist die folgende Beschreibung eines Automaten zum Verkauf von Getränken und Süßwaren:

*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. Ist das gewählte Produktfach ausverkauft, bricht der Automat den Kaufvorgang ab und wartet erneut auf die Produktauswahl. Nach erfolgreicher 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. Hat der Kunde genau so viel Geld eingeworfen, wie das Produkt kostet, wartet der Automat auf die nächste Produktauswahl. Hat der Kunde das Produkt entnommen und mehr Geld eingeworfen, als das ausgewählte Produkt kostet, so gibt der Automat das Rückgeld in den Ausgabeschacht aus. Nachdem der Kunde das Rückgeld entnommen hat, wartet der Automat wieder 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 notwendige Bedingungen an.

# Gruppe 1: Musterlösung



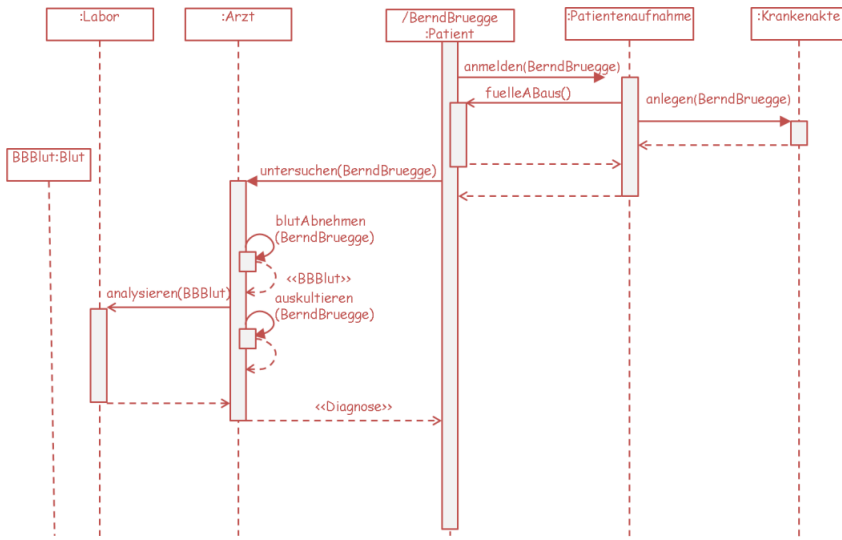
# Gruppe 2: Sequenzdiagramm (Nachklausur 11)

Gegeben sei folgendes Szenario, welches eine Untersuchung in einem Krankenhaus beschreibt:

*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 Patientenaufnahme an. Während der Sachbearbeiter an der Patientenaufnahme die Krankenakte anlegt, füllt Bernd B. den Anamnesebogen aus, den ihm der Sachbearbeiter gegeben hat. Nachdem Bernd B. den ausgefüllten Bogen dem Sachbearbeiter zurückgegeben hat, wird Bernd B. vom Arzt untersucht. Dazu nimmt er Bernd B. zunächst Blut ab. Während das Labor das Blut analysiert, führt der Arzt bei Bernd B. eine Auskultation durch. Schließlich bekommt Bernd B. vom Arzt seine Diagnose.*

Modellieren Sie das gegebene Szenario als UML-Sequenzdiagramm im Kasten auf der nächsten Seite (Querformat!). Verwenden Sie bei Ihrer Modellierung korrekte UML-Notation. Achten Sie bei Ihrer Modellierung darauf, auf welchen Objekten die Methoden sinnvollerweise aufgerufen werden müssen. Geben Sie ggf. Argumente der Methoden an.

# Gruppe 2: Musterlösung



## Gruppe 2: Musterlösung

Methodenaufruf mit Antwort:	7 × 0,5 P
Benennung der Lebenslinien:	5 × 0,5 P
Aktivitätsbalken:	5 × 0,5 P
Instanz „BerndBruegge“	1 × 0,5 P
Rest (UML-Notation, etc.)	1 × 1,0 P

Aktivitätsbalken fehlen selten:	-0,5 P
Aktivitätsbalken oft/immer:	-1,0 P
Antworten fehlen selten:	-0,5 P
Antworten oft/immer:	-1,0 P

(Alternative im Diagramm: Patientenaufnahme ruft  
Arzt.untersuchen() auf. Insbesondere aber ist die Methode untersu-  
chen() keine Methode auf Patient.)

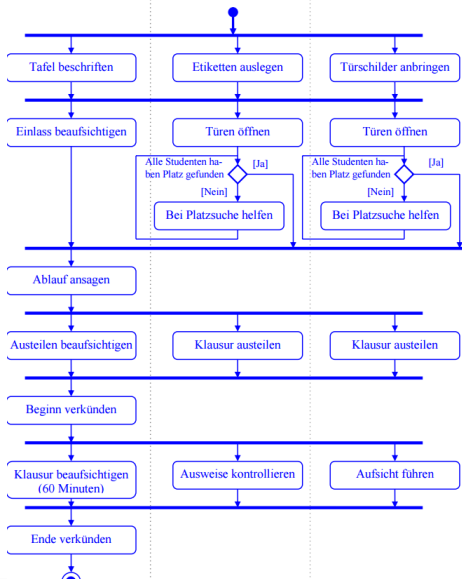
# Gruppe 3: Aktivitätsdiagramm (Hauptklausur 06)

Entwerfen Sie ein Aktivitätsdiagramm, das die Durchführung einer Klausur beschreibt. Halten Sie sich dabei so eng wie möglich an die nachfolgende Beschreibung und achten Sie darauf, parallele Aktivitäten korrekt zu synchronisieren. Beginnen Sie mit der Modellierung der Aktivitäten nach Betreten des Hörsaals. Hinweis: Aktivitäten der Studenten sind nicht zu modellieren.

*Nach Betreten des Hörsaals beginnt A1 die Tafel zu beschriften, während einer seiner Helfer die Etiketten auslegt und der andere die mitgebrachten Türschilder („Bitte nicht stören! Klausur!“) an den Türen anbringt. Auf den Etiketten stehen der Vor- und Nachname jedes Studenten, seine Matrikelnummer und eine fortlaufende Nummer, die später die Verwaltung der Klausur vereinfacht. Sind diese Aufgaben erledigt, kann der Einlass beginnen. Dazu öffnen die Helfer die Türen des Hörsaals und A1 beaufsichtigt den Einlass. Solange noch nicht alle Studenten ihren Platz gefunden haben, helfen A2 und A3 bei der Platzsuche. Der Hauptverantwortliche A1 wartet ab, bis alle ihre Plätze gefunden haben. Dann erklärt er den Ablauf, beaufsichtigt das Austeilen der Klausurblätter und verkündet den Beginn der Klausur. Anschließend beaufsichtigt er 60 Minuten lang die Klausur und sagt das Ende der Klausur an. Nachdem A1 den Ablauf der Klausur angesagt hat, sind A2 und A3 für das Austeilen der Klausur, die anschließende Aufsicht zuständig. Während der Bearbeitungszeit geht zusätzlich einer von beiden herum und kontrolliert die Studentenausweise, während der andere Aufsicht führt. Während der Klausur verlässt keine Aufsicht den Hörsaal.*



# Gruppe 3: Musterlösung



## Aufgabe 1-3: Plug-In programmieren

## Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

## Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

## Aufgabe 4: Aktivitätsdiagramm

## Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

## Aufgabe 4: Aktivitätsdiagramm

- separate Diagramme  $\implies$  verschachtelte Aktionen

## Aufgabe 5: Sequenzdiagramm

## Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

## Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

## Aufgabe 6: Substitutionsprinzip



## Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

## Aufgabe 6: Substitutionsprinzip

- Folien “Folgerung aus dem Substitutionsprinzip” anschauen (Ko-/Kontravarianz)
- mal als Java-Programm hinschreiben und versuchen zu kompilieren

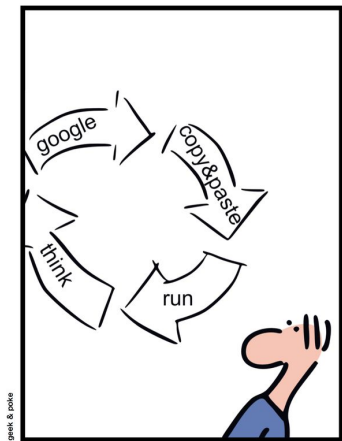
# Falls euch mal langweilig ist :D

`www.tichy.click`

## Abgabe

- Deadline am 13.6 um 12:00
- Aufgabe 4+5+6 handschriftlich (auf saubere Syntax achten!)
- an das Deckblatt denken!!

## SIMPLY EXPLAINED



DEVELOPMENT CYCLE