VU Objektorientierte Modellierung **Übung 3**

Übungsgruppen: 04.05.2009 - 08.05.2009

Aufgabe 1: Verhaltensmodellierung mittels Sequenzdiagramm

Wiederholen Sie das Kapitel aus der Vorlesung, das sich mit Sequenzdiagrammen beschäftigt.

- Welche 4 Arten von Interaktionsdigrammen gibt es? Beschreiben Sie diese kurz. Wofür werden Interaktionsdiagramme eingesetzt?
- Wie ist ein Sequenzdiagramm prinzipiell aufgebaut? Welche Elemente kann es enthalten?
- Was ist eine Zustandsinvariante im Kontext des Sequenzdiagramms? Wie können Zeiteinschränkungen angegeben werden?
- Was ist ein aktives Objekt, was ist ein passives Objekt? Wie unterscheiden sich diese?

Aufgabe 2: Verhaltensmodellierung mittels Sequenzdiagramm

Wiederholen Sie das Kapitel aus der Vorlesung, das sich mit Sequenzdiagrammen beschäftigt.

- Beschreiben Sie die Unterschiede zwischen synchronen und asynchronen Nachrichten.
- Welche Arten von Verzweigungen und Schleifen können in Sequenzdiagrammen auftreten? Beschreiben Sie die entsprechenden Operatoren.
- Welche Operatoren stehen im Sequenzdiagramm zur Verfügung, um parallele Abläufe zu realisieren bzw. um Ordnungen im Ablauf festzulegen?
- Erklären Sie die kombinierten Fragmente aus der Gruppe “Filterungen und Zusicherungen”.

Aufgabe 3: Kommunikation im Sequenzdiagramm

- Synchrone/Asynchrone Kommunikation

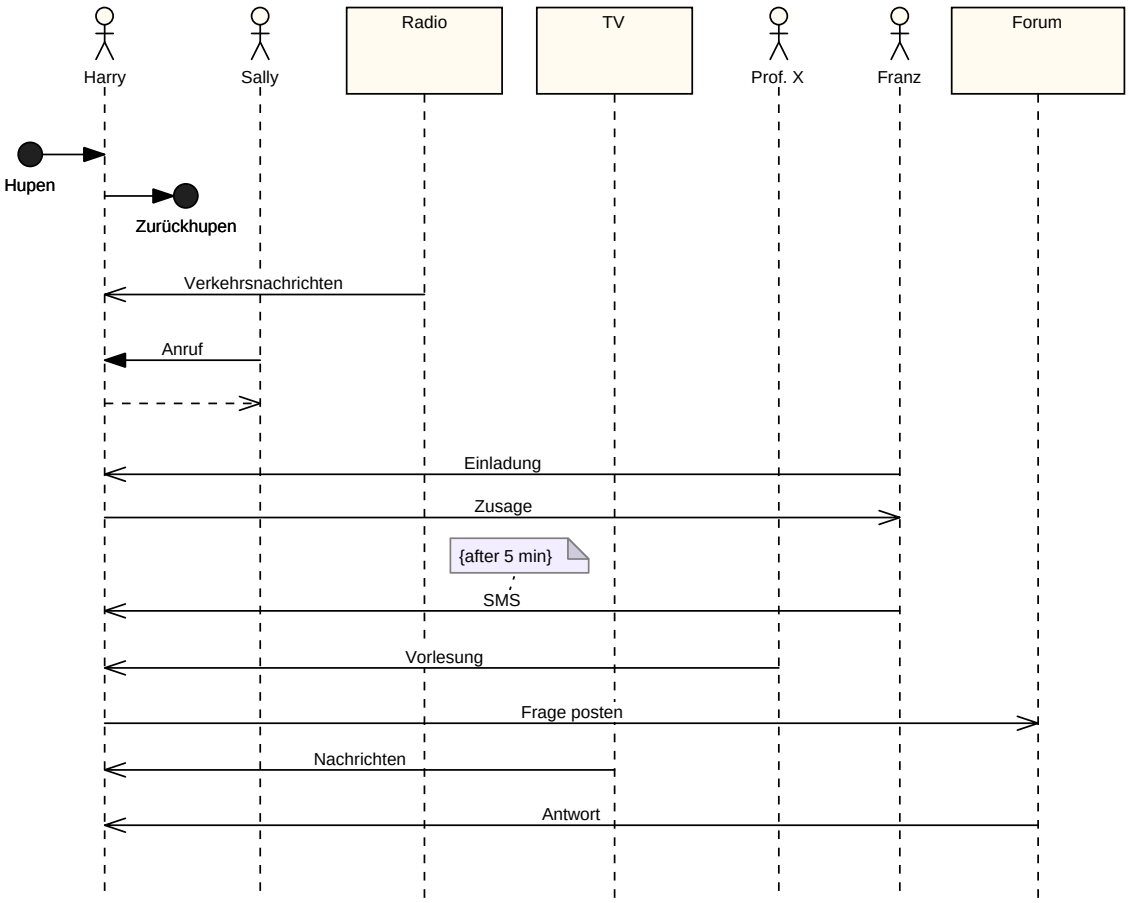
  Beschreiben Sie die im folgenden Diagramm vorkommenden Kommunikationsabläufe mittels Sequenzdiagramm.

  Harry steht im Stau. Irgendwer hupt Harry an. Harry hupt entnervt zurück. Das Radio schaltet sich an und durch die Verkehrsnachrichten erfährt Harry, dass auf der Straße, auf der er gerade fährt, ein Stau ist. Gerade als die Verkehrsnachrichten vorbei sind, ruft ihn seine Freundin Sally an, und fragt Harry, wann er endlich kommt, da ihre Vorlesung gleich anfängt. Er antwortet, dass er im Stau steckt und zu spät kommen wird. Nachdem das Telefonat beendet ist, ruft Harry seine Mails ab. Er hat eine Nachricht von seinem Freund Franz erhalten, in der er heute Abend zu einer Party eingeladen wird. Harry sagt sofort per Mail zu. 5 Minuten später erhält er von Franz ein SMS, das die genaue Adresse enthält.

  Endlich hat sich der Stau aufgelöst und Harry kann sich doch noch den Rest der Vorlesung von Prof. X über objekt-orientierte Modellierung anhören.

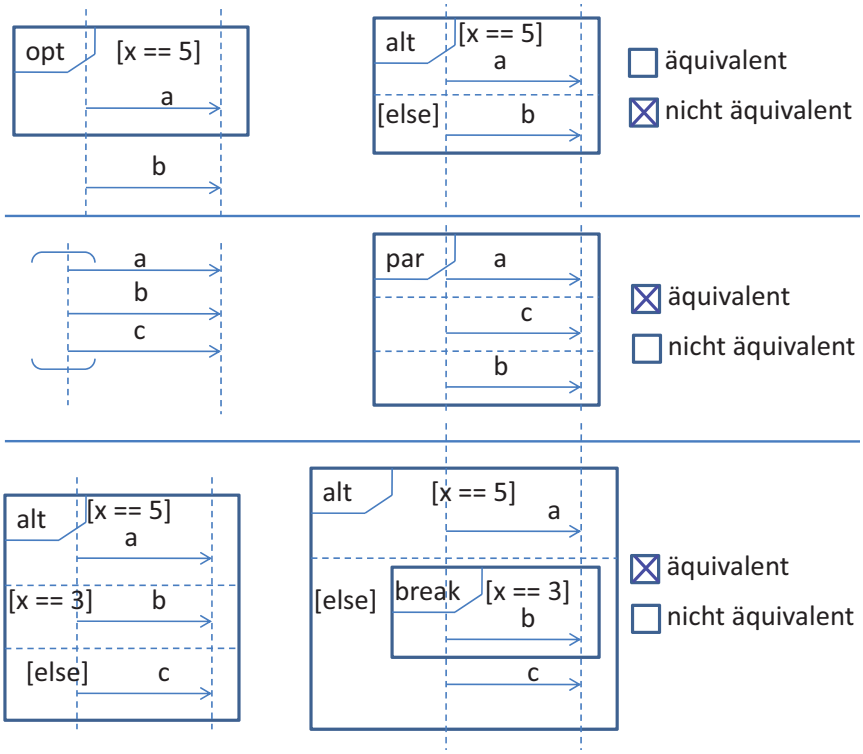
  Wieder zu Hause, fällt Harry eine sehr wichtige Frage ein, die er sofort in das LVA-Forum postet. Danach schaut er sich im Fernsehen die Nachrichten an. Inzwischen erhält Harry eine Antwort auf seine Frage.

  Achten Sie bei diesem Beispiel besonders darauf, ob die beschriebenen Kommunikationsabläufe synchron oder asynchron sind.


Harry
Sally
Radio
TV
Prof. X
Franz
Forum
Hupen
Zurückhupen
Verkehrsnachrichten
Anruf
Einladung
Zusage
{after 5 min}
SMS
Vorlesung
Frage posten
Nachrichten
Antwort

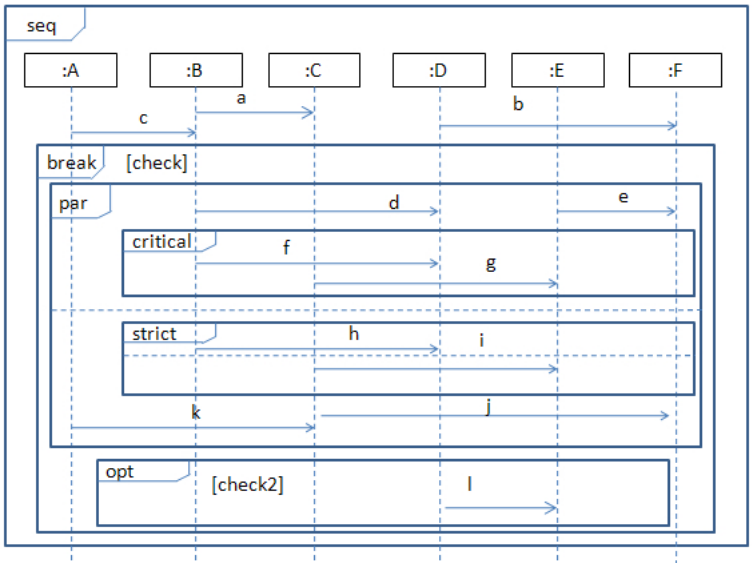
Aufgabe 4: Kombinierte Fragmente

a) **Äquivalenzen** Gegeben sind jeweils zwei Ausschnitte eines Sequenzdiagramms. Kreuzen Sie an, ob die beiden Ausschnitte jeweils „äquivalent“ oder „nicht äquivalent“ sind. Begründen Sie warum.



b) **Berechnung von Traces**

Beschreiben Sie alle möglichen Ereignisfolgen des folgenden Diagramms.



- minimaler Trace: a → b → c bzw. a → c → b bzw. b → a → c
- durch den par-Operator ergeben sich folgende Ereignisfolgen:

  1. Operand: d → e → f → g
     d → e → g → f
     e → d → f → g
     e → d → g → f
  2. Operand: h → i → j → k

- Die Traces dieser beiden Operanden können beliebig kombiniert werden, wobei zwischen f und g (egal ob f → g oder g → f) keine andere Nachricht kommen darf.
- Wenn die Variable `check` den Wert false hat, wird nur der minimale Trace ausgeführt.
- opt: Wenn die Variable `check2` wahr ist, wird l ausgeführt.
- Wenn die Traces, die sich aus dem par-Fragment ergeben, mit j → k enden, kann das l bis max. vor das j geschoben werden.
- Wenn der minimale Trace mit a → c endet, dann kann das e (d muss danach kommen) bis max. vor das a geschoben werden.

Aufgabe 5: Sequenzdiagramm aus Klassendiagramm

Gegeben sei das auf der folgenden Seite dargestellte Klassendiagramm zur Verwaltung von Produkten.

Klasse Geschäft:

- Geschäft_ID ... die Identifikationsnummer des Geschäfts
- addProdukt ... fügt ein Produkt zu einem Geschäft hinzu
- Geschäft ... „erzeugt“ ein neues Geschäft und setzt nur die Geschäft_ID
- addKategorie ... fügt eine Kategorie zu einem Geschäft hinzu

Klasse Kategorie:

- Kategoriename ... der Name der Kategorie
- addTheProdukt ... fügt ein Produkt zu einer Kategorie hinzu
- Kategorie ... „erzeugt“ eine neue Kategorie und setzt nur den Kategorienamen
- addGeschäft ... fügt ein Geschäft zu einer Kategorie hinzu

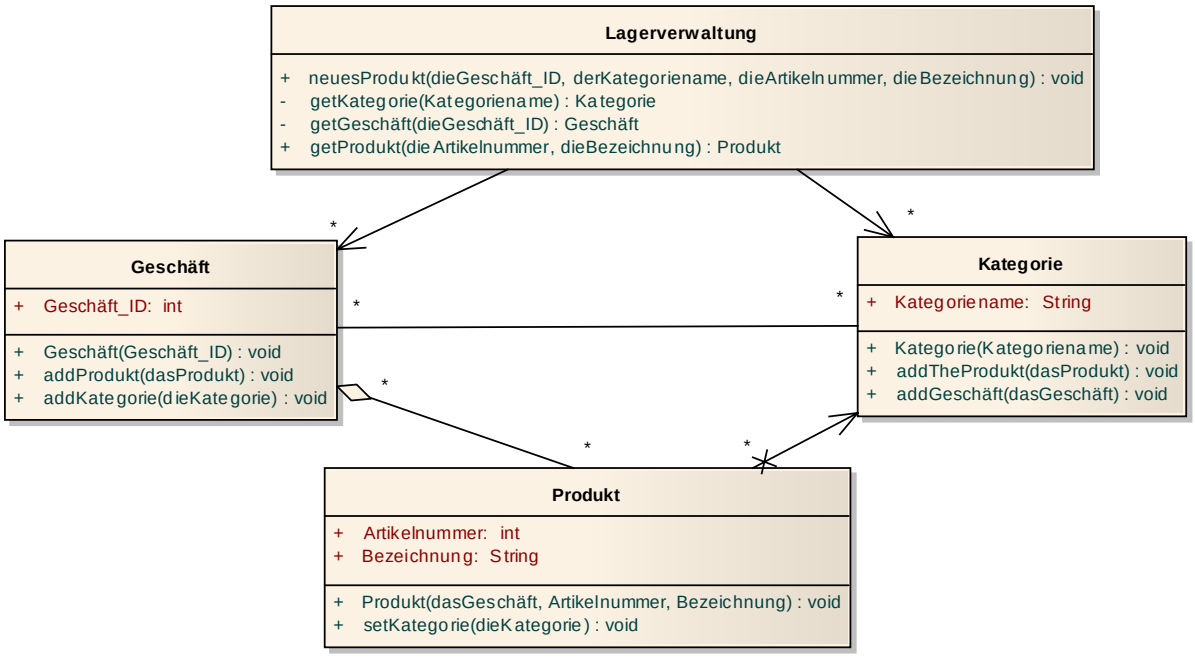
Klasse Produkt:

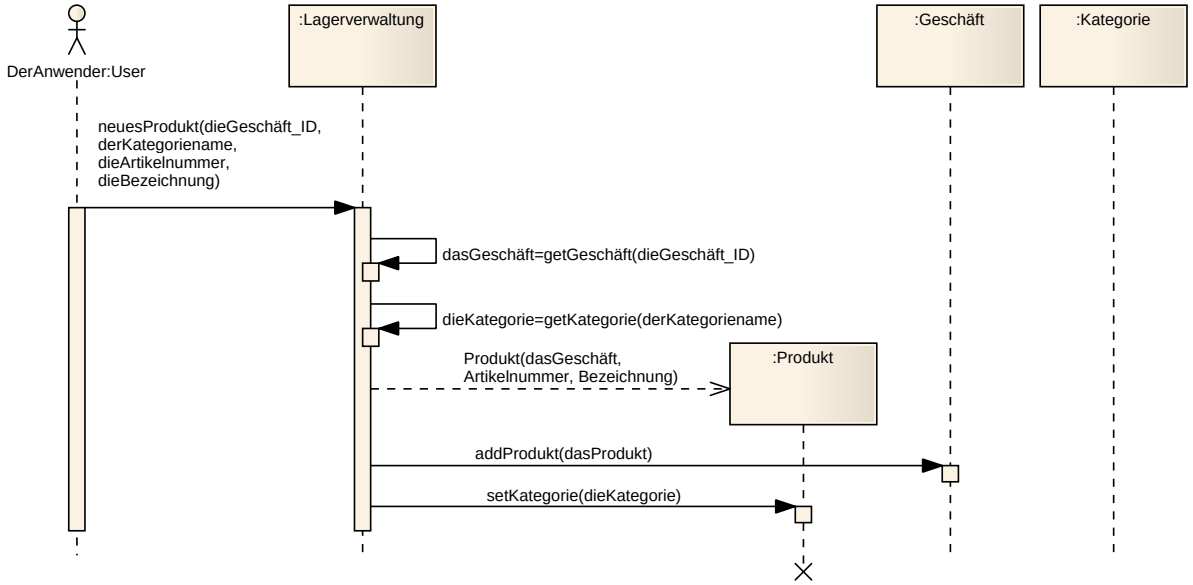
- Artikelnummer ... die Nummer des Artikels
- Bezeichnung ... der Name des Artikels
- Produkt ... „erzeugt“ ein neues Produkt und setzt Artikelnummer, Bezeichnung, aber nicht die Kategorie, zu der es gehört
- setKategorie ... fügt die Kategorie zu einem Produkt hinzu

Klasse Lagerverwaltung:

- neuesProdukt ... ein neues Produkt wird hinzugefügt
- getKategorie ... eine Kategorie wird abgefragt
- getGeschäft ... ein Geschäft wird abgefragt
- getProdukt ... ein Produkt wird abgefragt

Aufgabe: Erstellen Sie ein Sequenzdiagramm, um die Sequenz der Aufrufe von Operationen für das erfolgreiche Aufnehmen eines neuen Produktes in einem Geschäft zu zeigen (es gibt bereits Produkte der selben Kategorie; das Geschäft existiert ebenfalls schon). Wird ein Interaktionspartner erst im Laufe der Interaktion erzeugt, so geben Sie explizit den Konstruktor an, der das Objekt erzeugt.



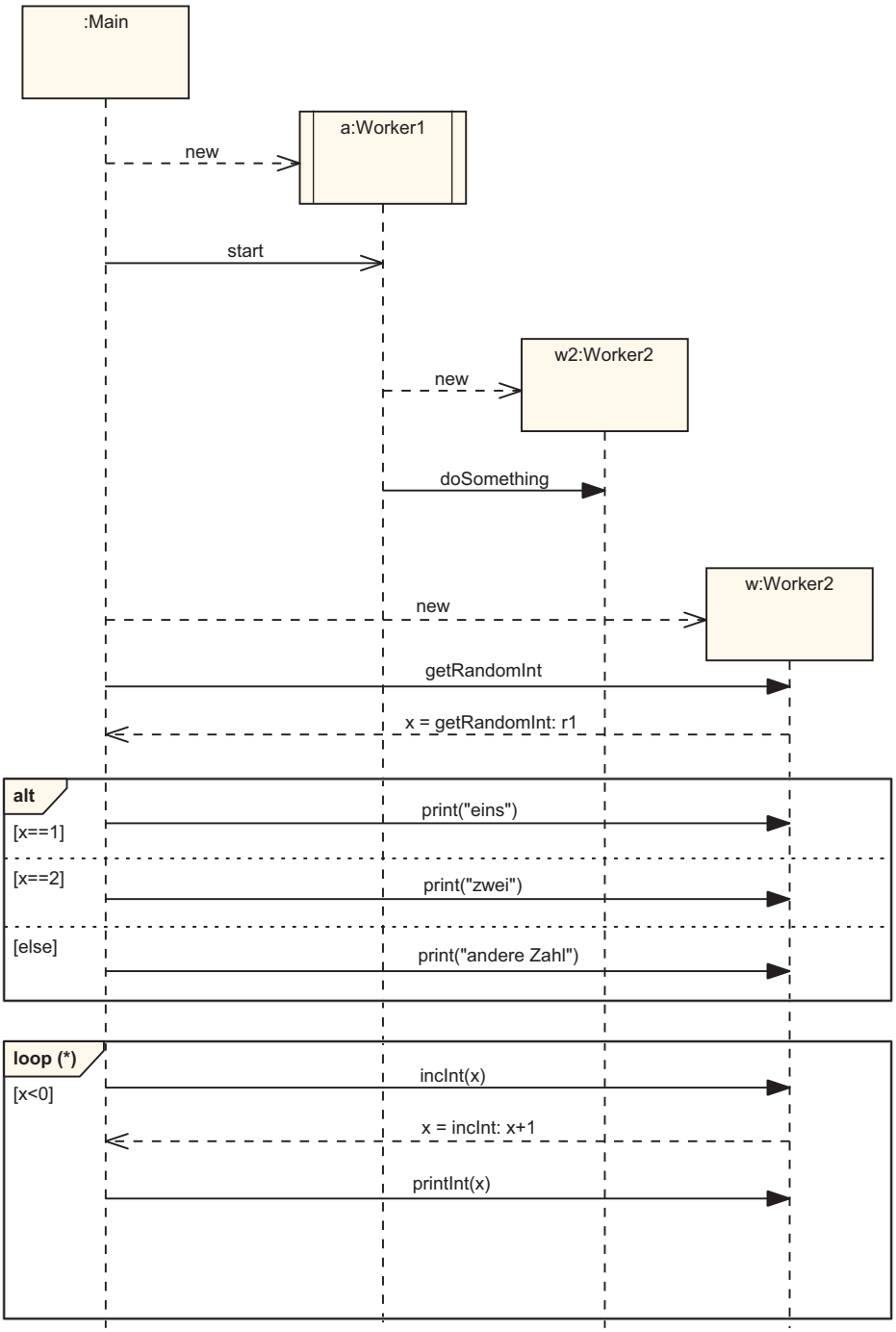


Aufgabe 6: Darstellung von Programmabläufen mittels Sequenzdiagramm

Stellen Sie die Abläufe von folgendem Programm mittels Sequenzdiagramm dar. Modellieren Sie auch allfällige Antwortnachrichten.
Anmerkungen: Die Deklaration der Variablen muss nicht angegeben werden. Beachten Sie, dass Klassen, welche die Klasse `Thread` erweitern, einen eigenen Kontrollfluss haben. Mit dem Aufruf der Methode `start` wird der Code aus `run` parallel zum auf `start` nachfolgenden Code ausgeführt. Die `break`-Anweisungen braucht nicht im Sequenzdiagramm dargestellt werden.

```
class Main {
  ...
  int x;

  Worker1 a = new Worker1();
  a.start();

  Worker2 w = new Worker2();

  x = w.getRandomInt();

  switch(x) {
    case 1: w.print("eins");
            break;
    case 2: w.print("zwei");
            break;
    default: w.print("andere Zahl");
  }


  while (x < 0); {
    x = w.incInt(x);
    w.printInt(x);
   }

}
```

```
class Worker1 extends Thread {

Worker2 w2;
// ...

  void run() {
    w2 = new Worker2();
    w2.doSomething();
  }

}


class Worker2 {
// ...

  void doSomething() { ... }

  void print(String s) { ... }

  void printInt(int i) { ... }

  int getRandomInt() { ... }

  int incInt(int x) { ... }


}
```


:Main
a:Worker1
new
start
w2:Worker2
new
doSomething
w:Worker2
new
getRandomInt
x = getRandomInt: r1
alt
[x==1]
print("eins")
[x==2]
print("zwei")
[else]
print("andere Zahl")
loop (*)
[x<0]
incInt(x)
x = incInt: x+1
printInt(x)