VU Objektorientierte Modellierung		Übung 3: Sequenzdiagramm
188.391, SS2022	MUSTERLÖSUNG	Übungsgruppen: 28.03.2022 - 01.04.2022

Hinweise: Die meisten Beispiele bestehen aus Modellierungsbeispielen und dazu passenden Theoriefragen.

- Modellierungsbeispiele: Bilden Sie den Sachverhalt, der in der Angabe geschildert wird, möglichst genau ab. Sollte etwas in der Angabe nicht erwähnt sein, treffen Sie sinnvolle Annahmen.
- Theoriefragen: Nehmen Sie sich bei der Beantwortung die Modellierungsaufgaben der jeweiligen Aufgabe zu Hilfe.

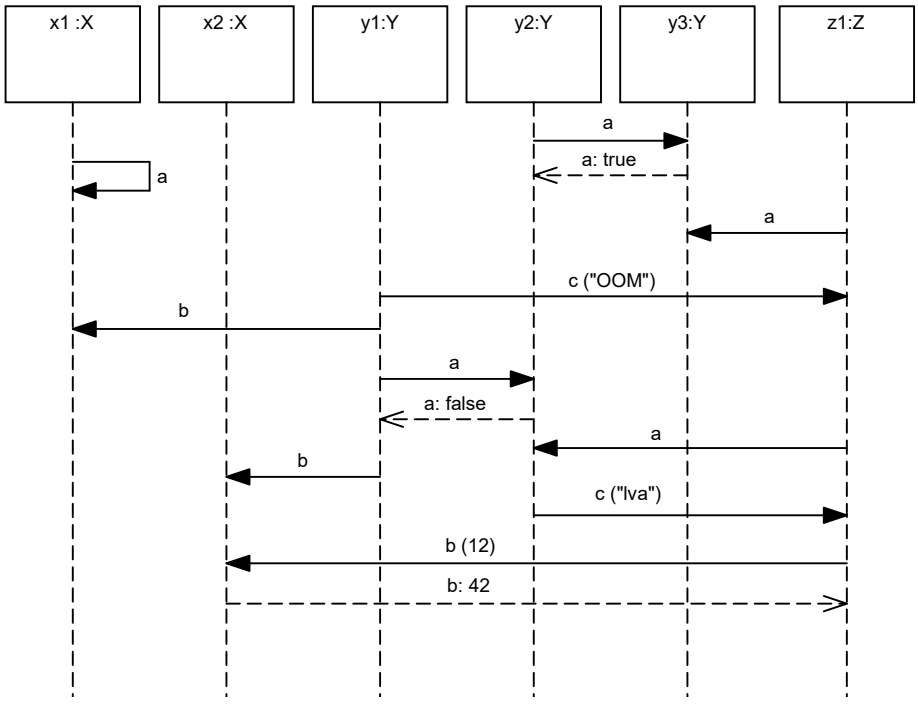
Aufgabe 1: Klassendiagramm aus Sequenzdiagramm

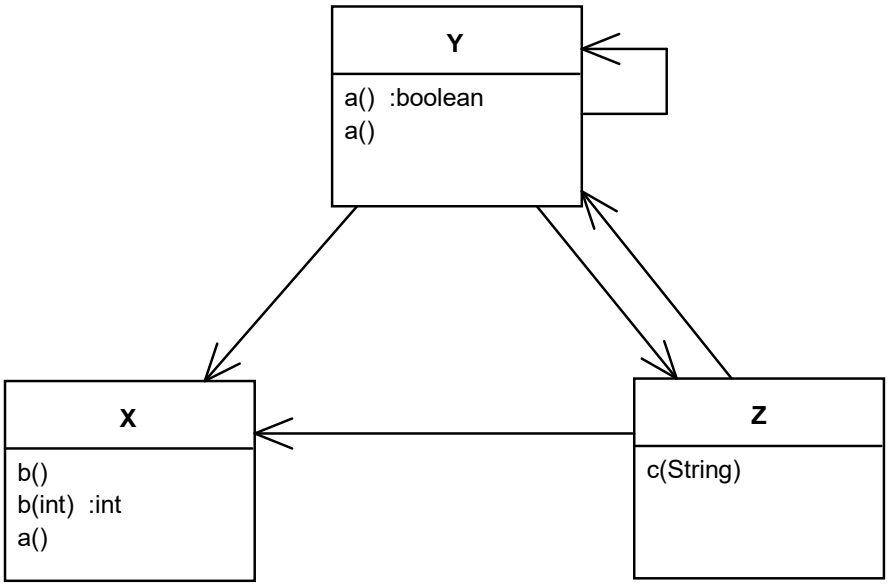
(a) Gegeben ist das nachfolgende Sequenzdiagramm. Modellieren Sie das Klassendiagramm, auf dem das gegebene Sequenzdiagramm basiert.

- Klassen
- Operationsdefinitionen mit Typangaben, soweit ersichtlich
- Beziehungen zwischen Klassen in Form von navigierbaren Assoziationen: Zeichnen Sie nur Navigationsrichtungen ein, die aus dem gegebenen Sequenzdiagramm ersichtlich sind.

Zusatzfragen:

(b) Wie ist ein Sequenzdiagramm prinzipiell aufgebaut? Welche Elemente kann es enthalten?




Y
a() :boolean
a()
X
b()
b(int) :int
a()
Z
c(String)

Aufgabe 2: Synchrone/Asynchrone Kommunikation

Beschreiben Sie verschiedene Arten Ihrer eigenen alltäglichen Kommunikation mit anderen Menschen oder Systemen mittels Sequenzdiagramm. Achten Sie bei diesem Beispiel besonders darauf, ob Kommunikationsabläufe synchron oder asynchron sind.

Verwenden Sie dafür mindestens 3 und maximal 5 Interaktionspartner (Menschen und Systeme) und mindestens 10 und maximal 15 Nachrichten verschiedenen Inhalts. Darin enthalten soll mindestens je ein Beispiel für folgende Nachrichtentypen bzw. Kombinierte Fragmente sein:

- Synchrone Kommunikation
- Asynchrone Kommunikation
- Verlorene Nachricht
- Gefundene Nachricht
- 2 beliebige Kombinierte Fragmente

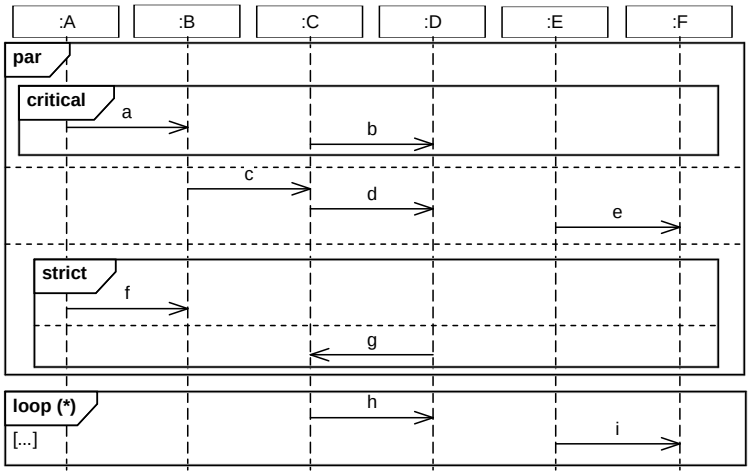
Zusatzfragen:

(a) Was sind die Unterschiede zwischen synchronen und asynchronen Nachrichten?

(b) Wie sehen die weiteren Arten von Nachrichten aus und wie funktionieren diese?

Aufgabe 3: Berechnung von Traces

Gegeben ist das folgende Sequenzdiagramm:



(a) Beschreiben Sie alle möglichen Ereignisfolgen des gegebenen Diagramms.

Die Nachrichten finden nach folgendem Schema statt:

- `par`: Die Nachrichten der drei `par`-Abschnitte können beliebig kombiniert werden, solange die folgenden Einschränkungen berücksichtigt werden:
  - Erster `par`-Abschnitt: `critical`; `a` und `b` dürfen nicht unterbrochen werden, es darf also keine andere Nachricht dazwischen stattfinden. Die Reihenfolge der zwei ist aber egal (weil sie sich keine Lebenslinie teilen).
    Daher möglich:
    (`a` → `b`)
    (`b` → `a`)
  - Zweiter `par`-Abschnitt: `c` muss vor `d` kommen (weil sie sich eine Lebenslinie teilen), für `e` gibt es keine Einschränkung.
    Daher möglich:
    `c` → `d` → `e`
    `e` → `c` → `d`
    `c` → `e` → `d`
  - Dritter `par`-Abschnitt: `strict`; die Reihenfolge der Operanden muss auf jeden Fall eingehalten werden, `f` muss also vor `g` kommen.
    Daher möglich:
    `f` → `g`
- `loop`: `h` und `i` können in beliebiger Reihenfolge stattfinden, da sie sich keine Lebenslinie teilen; Wie oft `h` und `i` stattfinden ist abhängig von der Bedingung, auch 0 Mal ist möglich. Mögliche Sequenzen zum Beispiel:
  `h` → `i`
  `i` → `h`
  `h` → `i` → `i` → `h`

(b) Welche der folgenden Traces sind möglich? Warum/warum nicht?

b → a → c → d → e → f → g → h → i	☒ ja	
e → a → b → c → d → f → g	☒ ja	
f → a → b → c → d → e → g	☒ ja	
h → i → h → i → h → i		☒ nein
a → b → c → d → e → f → g → h → i → h		☒ nein

a → b → c → e → d → f → g → h → i	☒ ja	
f → g → d → e → c → b → a		☒ nein
h → i → a → c → b → e → f → g → d		☒ nein
c → e → f → a → b → g → d	☒ ja	
g → a → b → e → c → d → f		☒ nein

Zusatzfragen:

(c) Erklären Sie die Funktionsweise der Operatoren seq und strict.

(d) Erklären Sie die Funktionsweise der Operatoren par und critical.

(e) Erklären Sie die Funktionsweise des loop-Operators.

Aufgabe 4: Kombinierte Fragmente

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

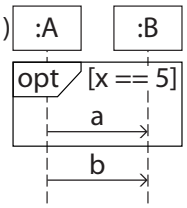
Zusatzfragen:

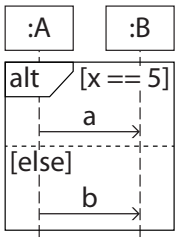
(b) Erklären Sie die Funktionsweise der Operatoren alt und opt.

(c) Erklären Sie die Funktionsweise der Coregion.

(d) Erklären Sie die Funktionsweise des break-Operators.

(a.1)

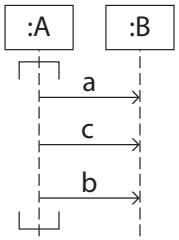


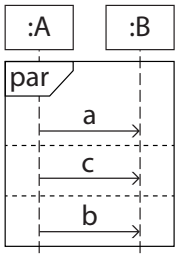


☐ äquivalent

☒ nicht äquivalent

(a.2)

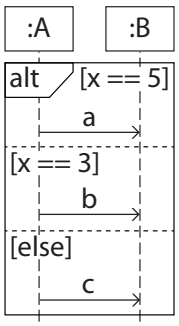


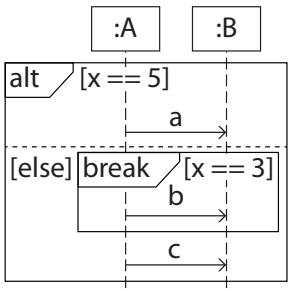


☒ äquivalent

☐ nicht äquivalent

(a.3)





☒ äquivalent

☐ nicht äquivalent

Aufgabe 5: Ausschnitte

Wie können Sie die folgenden Sachverhalte in einem Sequenzdiagramm abbilden? Modellieren Sie die geschilderten Sachverhalte.

(a) Der Client schickt Nachricht A um 8:55 zum Server und erwartet keine Antwort.

(b) Der Client schickt eine Nachricht B zum Server. 2 Sekunden nachdem er dessen Antwort erhalten hat, schickt der Client Nachricht C zum Server und erwartet keine Antwort.

(c) Der Server erstellt innerhalb von 5 Sekunden ein neues Objekt der Klasse Report, wenn dieser eine Nachricht D übermittelt hat. Um 0:05 schickt das Objekt der Klasse Report Nachricht E an den Client und erwartet keine Antwort.

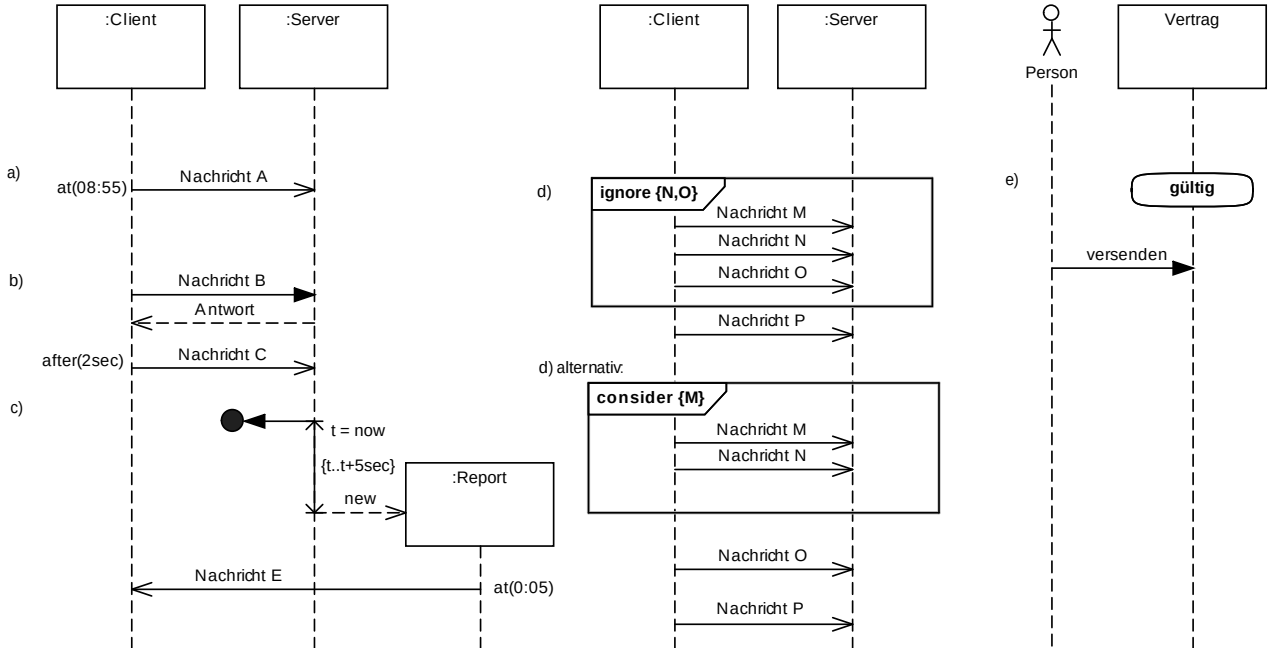
(d) Der Client schickt die asynchronen Nachrichten M, N, O und P zum Server. Nachricht N und O sind für die Programmiererin nicht relevant, die Nachricht M hingegen ist sehr wichtig. Über P haben Sie keine nähere Information.

(e) Eine Person darf einen Vertrag nur dann versenden, wenn er gültig ist. Bilden Sie diesen Sachverhalt mit Hilfe des Konzepts „Zustandsinvariante“ ab.

Zusatzfragen:

(f) Welche Konzepte werden benötigt, um die jeweilige Aufgabenstellung zu lösen?

(g) Was versteht man im Kontext des Sequenzdiagramms unter Zustandsinvarianten?

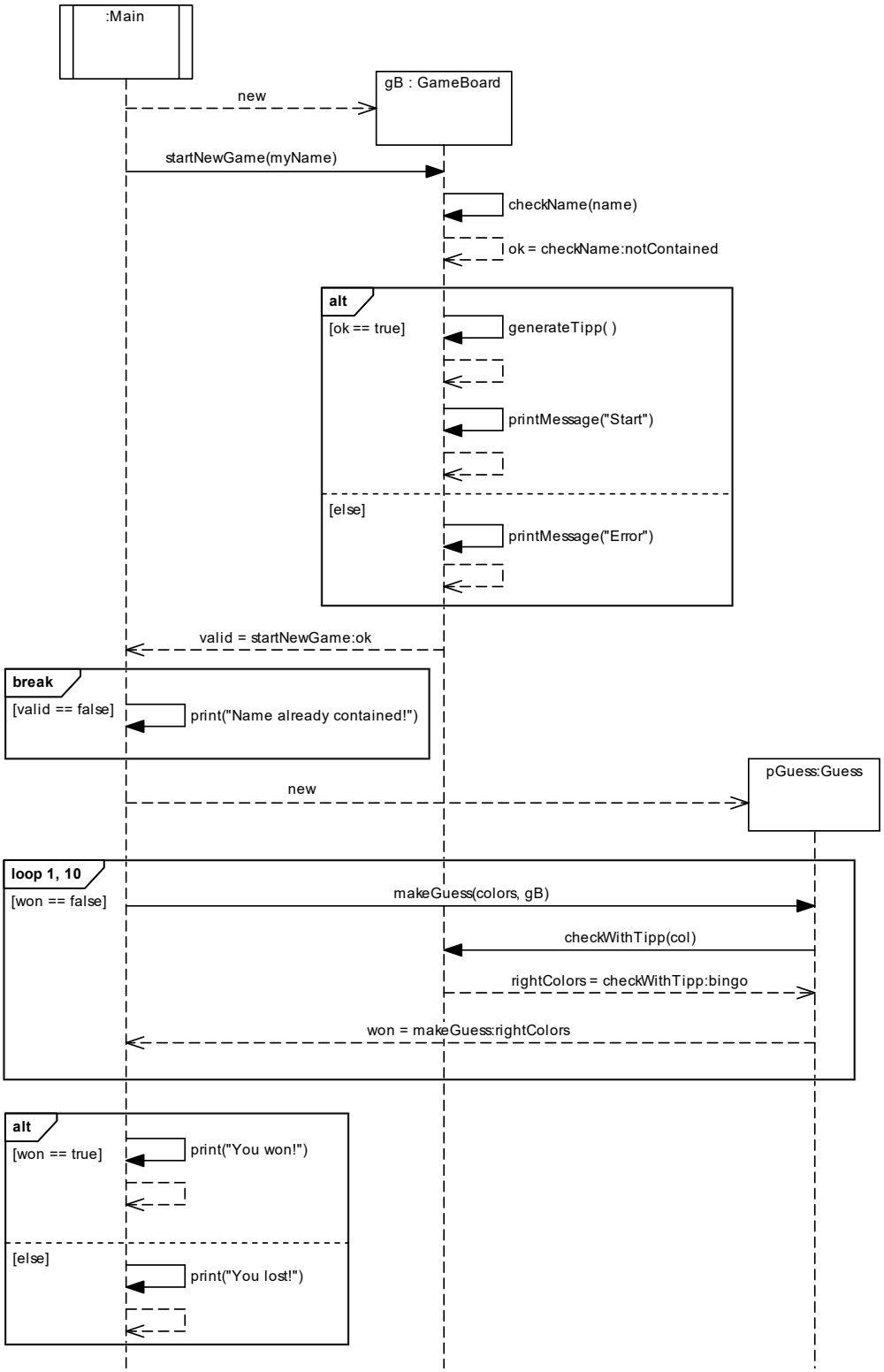


Aufgabe 6: Darstellung von Programmabläufen mittels Sequenzdiagramm

Stellen Sie die Abläufe von folgendem Programm mittels Sequenzdiagramm dar. Modellieren Sie auch allfällige Antwortnachrichten.
Sie können davon ausgehen, dass alle nicht explizit deklarierten Variablen bereits deklariert und initialisiert sind. „...“ markiert vernachlässigte Codeteile, die nicht modelliert werden müssen.

```
1  class Main {
2
3    public void main(String []) {
4      String myName;
5      Color[] colors = new Color[];
6      ...
7      GameBoard gB = new GameBoard();
8      boolean valid = gB.startNewGame(myName);
9
10     if (!valid) {
11       print("Name already contained!");
12       exit;
13     }
14
15     Guess pGuess = new Guess();
16     boolean won = false;
17
18     for (int i = 1; i <= 10 && !won; i++) {
19       ...
20       won = pGuess.makeGuess(colors, gB);
21     }
22
23     if (won) {
24       print("You won!");
25     } else {
26       print("You lost!");
27     }
28     ...
29   }
30
31   public void print(String m)
32   {...}
33 }
34
35 class GameBoard {
36   ...
37   public boolean startNewGame(String name) {
38     ...
39     boolean ok = this.checkName(name);
40     if (ok) {
41       generateTipp();
42       printMessage("Start");
43     } else {
44       printMessage("Error");
45     }
46     ...
47     return ok;
48   }
49
50
```

```
51    public boolean checkName(String name) {
52      int notContained;
53      ...
54      return notContained;
55    }
56
57    public void generateTipp() {
58      ...
59    }
60
61    public boolean checkWithTipp(Color[] colors) {
62      boolean bingo;
63      ...
64      return bingo;
65    }
66
67    public void printMessage(String m)
68    {...}
69  }
70
71  class Guess {
72
73    public boolean makeGuess(Color[] col, GameBoard gameB) {
74      ...
75      boolean rightColors = gameB.checkWithTipp(col);
76      ...
77      return rightColors;
78    }
79  }
```


:Main
gB : GameBoard
new
startNewGame(myName)
checkName(name)
ok = checkName:notContained
alt
[ok == true]
generateTipp( )
printMessage("Start")
[else]
printMessage("Error")
valid = startNewGame:ok
break
[valid == false]
print("Name already contained!")
pGuess:Guess
new
loop 1, 10
[won == false]
makeGuess(colors, gB)
checkWithTipp(col)
rightColors = checkWithTipp:bingo
won = makeGuess:rightColors
alt
[won == true]
print("You won!")
[else]
print("You lost!")