Technische Universität Berlin

Software and Embedded Systems Engineering Group
Prof. Dr. Sabine Glesner
www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2023/2024

Prof. Dr. Sabine Glesner
Julian Klein
Simon Schwan

Übungsaufgaben - Test 2a/2b

Dieser Aufgabenkatalog enthält Übungsaufgaben für den zweiten Test von Softwaretechnik und Programmierparadigmen. Der Katalog ist kein Probetest und entspricht nicht dem üblichen Umfang eines Tests. Die Aufgaben dienen lediglich zur Übung und können sich in Art, Umfang und Thema von den Testaufgaben unterscheiden.

1. Anforderungsspezifikation

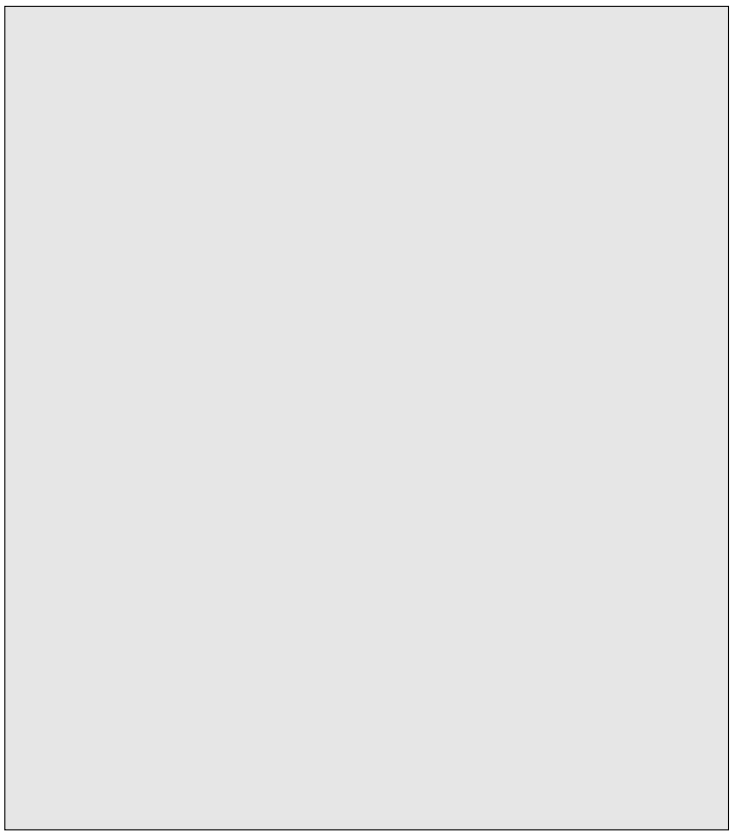
Es soll ein System entwickelt werden, mit dem Orte bewertet sowie ihre Besuche und beteiligten Personen aufgezeichnet und ausgewertet werden. Ergänzen Sie die gegebenen Diagramme zur Anforderungsspezifikation um die folgenden Anforderungen.

a) Ein Benutzer soll Bewertungen erstellen können. Für Bewertungen können Ressourcen hinzugefügt werden. Dies kann auch schon direkt bei der Erstellung einer Bewertung geschehen.

2. Anforderungsspezifikation

In einem Buchungsportal sollen Kunden Zugfahrten anzeigen und Tickets buchen können. Beides ist einzeln möglich, aber die Möglichkeit zu buchen erscheint auch bei der Anzeige von Zugfahrten. Tickets können außerdem storniert werden, dies kann aber nur durch Kundenbetreuer erledigt werden.

Bilden Sie die gegebenen Anforderungen in dem folgenden Use-Case-Diagramm ab.



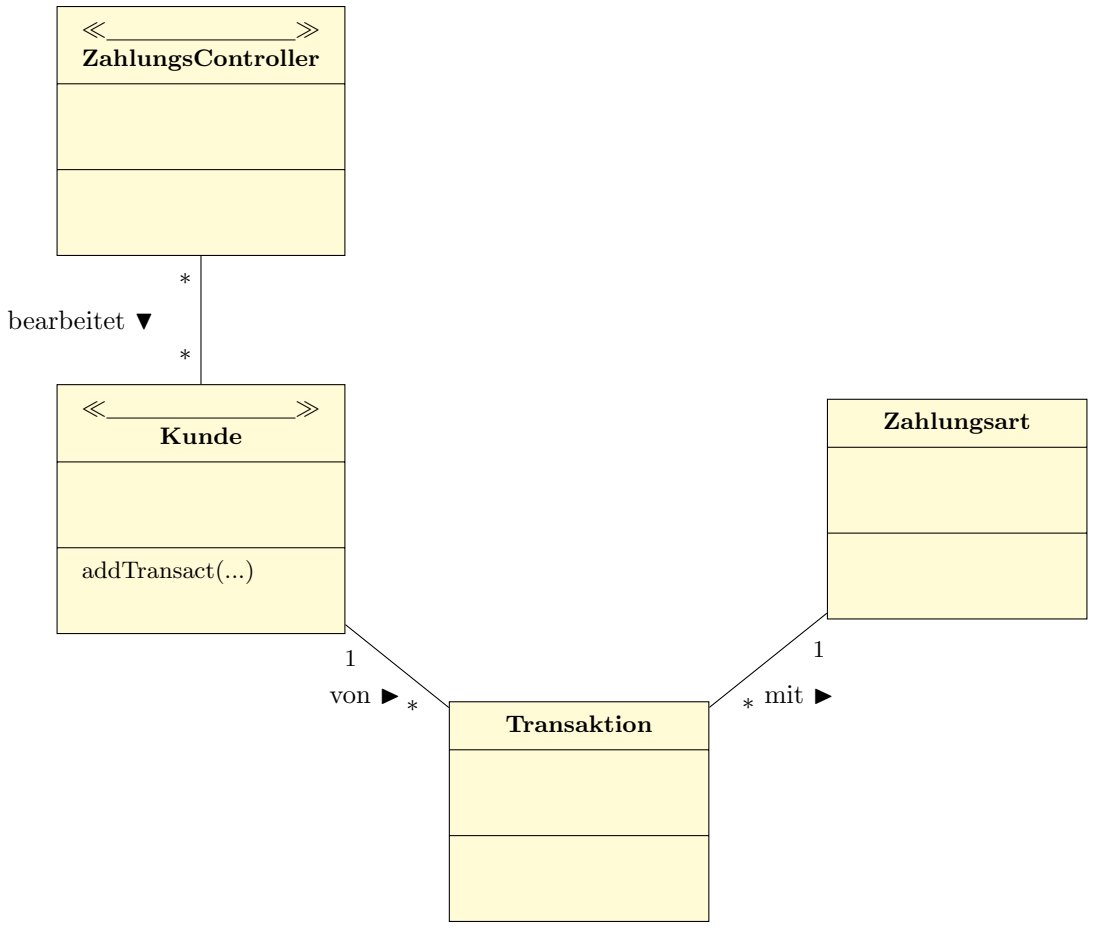
3. Code zu UML

Gegeben ist der folgende Programmausschnitt:

```
public class ZahlungsController {
    void lastschriftHinzu(Kunde k, String iban, boolean fav) {
        Zahlungsart z = new Lastschrift(iban);
        k.zahlungsarten.add(z);
        if(fav) k.favZ = z;
    }
    String bezahlen(Kunde k, int preis) {
        if(k.favZ == null) {
            return "Keine Lieblingszahlungsart.";
        } else {
            Transaktion t = new Transaktion(preis, k.favZ);
            k.addTransact(t);
            return "erfolgreich";
        }
    }
}
```

a) Erweitern Sie das Klassendiagramm um sämtliche Struktur-Informationen aus dem Programm, die die Vorlage nicht bereits enthält. Vervollständigen Sie die **Stereotypen** in den dafür vorgesehenen Lücken. Konstruktoren müssen nicht abgebildet werden, an Konstruktoren übergebene Daten müssen aber nach der Ausführung in den Objekten zur Verfügung stehen.

Attribute können ohne Sichtbarkeiten modelliert werden. Für zusätzliche Assoziationen müssen **Multiplizitäten vollständig** angegebenen werden. Rollenbezeichner sind nur anzugeben wenn sie zwingend erforderlich sind. **Operationen** dürfen ohne Parameter und Rückgabetyp angegeben werden.



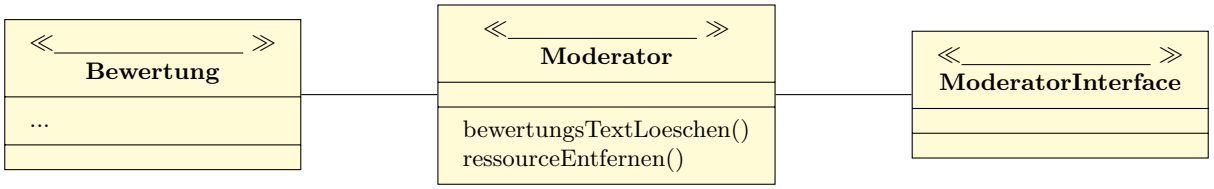
4. Klassendiagramm

Erweitern Sie die gegebenen Modell-Ausschnitte um die folgenden Teilaspekte. Beachten Sie dabei: Multiplizitäten für Assoziationen müssen vollständig angegeben werden, Rollenbezeichner hingegen nur sofern notwendig. Navigation und Sichtbarkeiten können weggelassen werden.

a) Eine Bewertung enthält eine ganzzahlige Anzahl Sterne. Eine Rezension ist eine spezielle Bewertung, die außerdem einen Text enthält und für die bis zu fünf Ressourcen hochgeladen werden können. Die Ressourcen werden im System mit der Rezension zu der sie gehören und einem Link hinterlegt.

b) Für Bewertungen können Antworten verfasst werden. Eine Antwort bezieht sich dabei entweder direkt auf eine Bewertung oder wiederum auf eine andere Antwort.

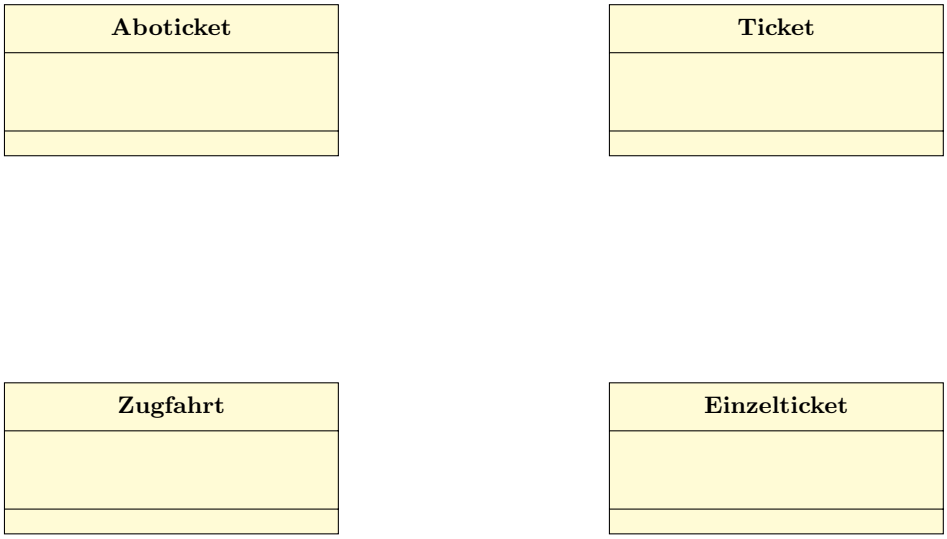
c) Für Moderatoren ist ein gesondertes Web-Interface vorgesehen, über das unangebrachte Inhalte entfernt werden können. Erweitern Sie die dafür erstellten Klassen im Modell um Stereotypen entsprechend dem Entity-Control-Boundary Schema. (1,5P)



5. Klassendiagramm

Erweiterten Sie die gegebenen Modellausschnitte um die Anforderungen der jeweiligen Texte mit den Gestaltungselementen des UML-Klassendiagramms. Attribute können ohne Sichtbarkeiten modelliert werden. Für zusätzliche Assoziationen müssen Multiplizitäten vollständig angegebenen werden. Rollenbezeichner sind nur anzugeben wenn sie zwingend erforderlich sind.

a) Abotickets und Einzeltickets sind spezielle Tickets. Jedes Ticket hat einen Preis in Cent. Ein Einzelticket gehört zu mindestens einer Zugfahrt und für jede Zugfahrt kann es beliebig viele Einzeltickets geben. Ein Aboticket hat außerdem ein Ablaufdatum.

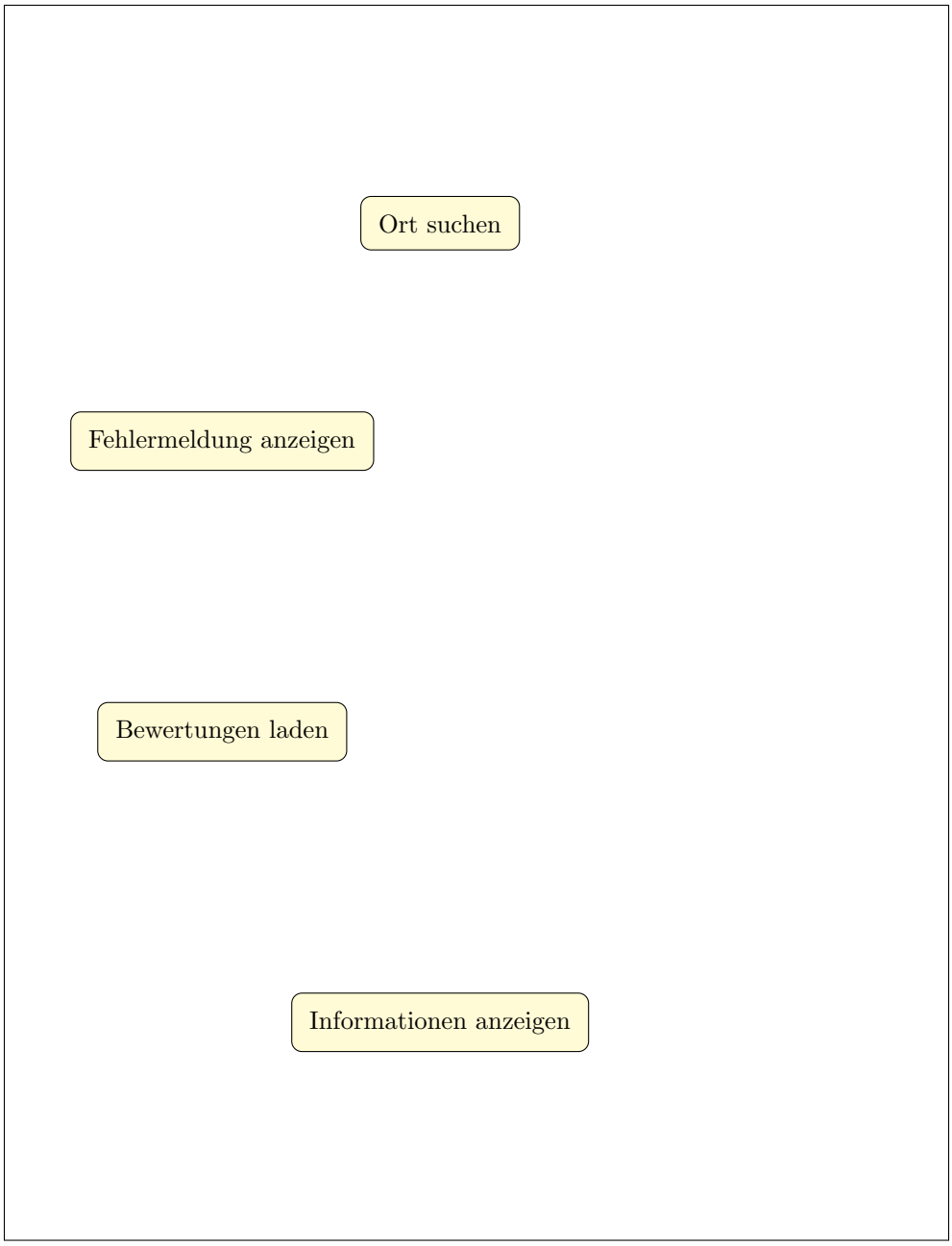


b) Kunden haben maximal ein aktuelles Aboticket, bis zu drei abgelaufene Abotickets sind für sie aber weiterhin hinterlegt. Jedes Aboticket ist maximal einem Kunden als aktuelles oder abgelaufenes Ticket zugewiesen.

6. Aktivitätsdiagramm

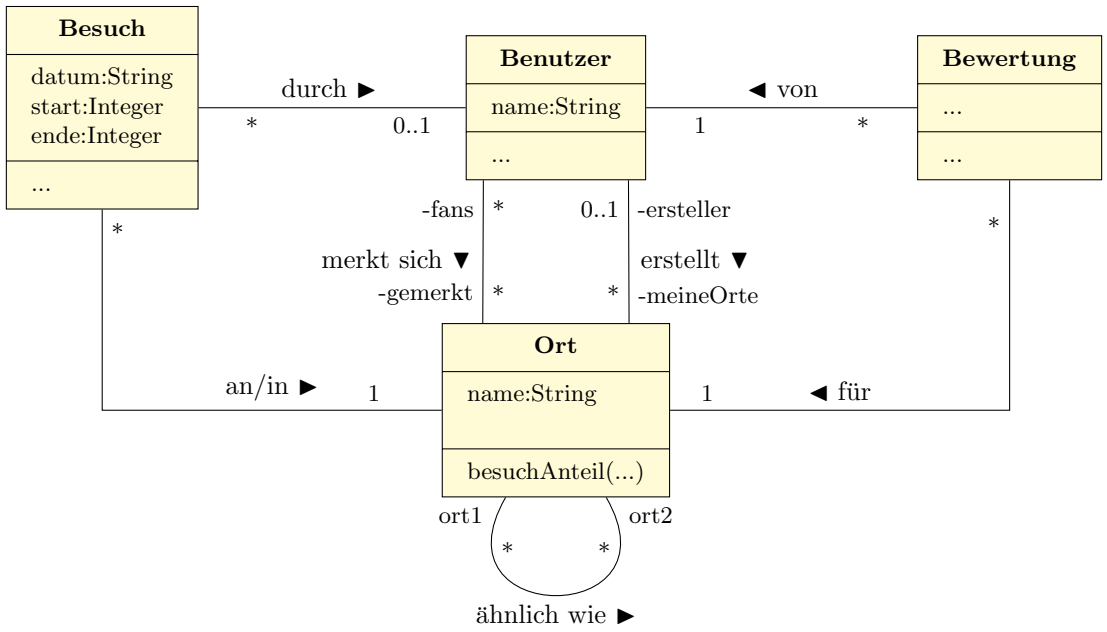
Im Webbrowser sollen die Orte des Bewertungssystems zusammen mit nützlichen Produktinformationen dargestellt werden. Vervollständigen Sie das gegebene Aktivitätsdiagrammen mit den folgenden Informationen, ohne dabei weitere einfache Aktionen hinzuzufügen.

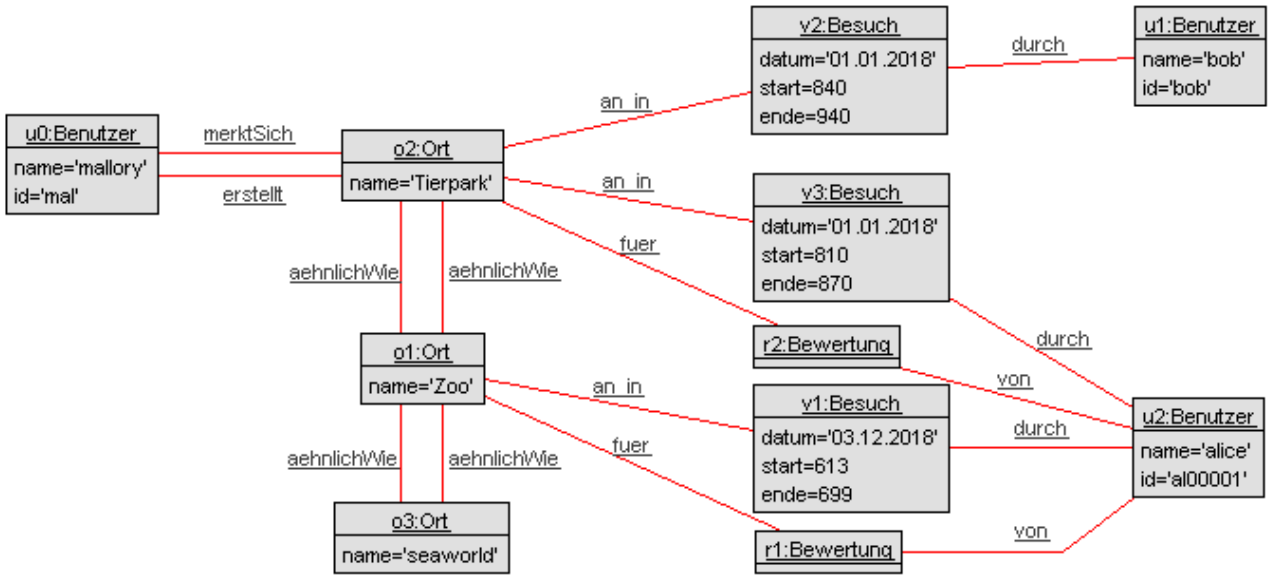
- Zuerst wird nach einem Ort gesucht, der zu dem Suchwort passt. Wenn keiner gefunden wird, endet die Anfrage mit einer Fehlermeldung.
- Ist ein Ort gewählt, werden die Bewertungen geladen. Parallel dazu werden die Produktinformationen bei einem externen Werbe-Server angefragt und die Antwort abgewartet.
- Wenn alle Informationen vorliegen werden sie gemeinsam angezeigt. Damit endet die Aktivität.



7. OCL

Gegeben Sei ein Teil des Modells in Form eines Klassendiagramms und ein Beispielzustand in Form eines Objektdiagramms. Bearbeiten Sie damit die folgenden Aufgaben in OCL.





a) Werten Sie die folgenden Ausdrücke in dem gegebenen Zustand aus. Geben Sie dabei die Rückgabewerte und deren exakten Typ (inkl. Mengentypen) an. Mengen werden in geschwungenen Klammern geschrieben. Beispiel: {1, 2, 3}: Set(Integer) (5P)

```
o2.ersteller.name

--------------------------------------- : ----------------------------------

o2.besuch.benutzer

--------------------------------------- : ----------------------------------

Benutzer.allInstances()->forAll(gemerkt->includesAll(meineOrte))

--------------------------------------- : ----------------------------------

Besuch.allInstances()->select(datum='01.01.2018')->collect(b|b.ende - b.start)

--------------------------------------- : ----------------------------------

u2.besuch->including(r2)

--------------------------------------- : ----------------------------------
```

b) Formalisieren Sie die folgenden Anfragen an das System. Die Anfragen sollen dabei auch für andere Zustände, in denen die jeweils verwendeten Objekte existieren, korrekte Antworten liefern. (7P)

- Sind alle Startzeiten von Besuchen vor oder gleich dem jeweiligen Ende? (1P)

- Wie oft wurde der Ort `o2` am 01.01.2018 besucht? (1,5P)

- Welche Benutzer haben noch nie eine Bewertung hinterlassen? (1,5P)

- Waren die Benutzer `u1` und `u2` schon einmal am selben Tag am selben Ort (2P)?

- Welche Orte sind dem Ort `o2` direkt oder indirekt ähnlich? Zwei Orte sind sich dabei indirekt ähnlich wenn es eine Menge Orte gibt, über die sich Ähnlichkeitsbeziehungen zwischen den beiden Orten herstellen lässt. Es kann dabei ausgegangen werden, dass die Ähnlichkeitsbeziehung zwischen zwei Orten symmetrisch ist. (1P)

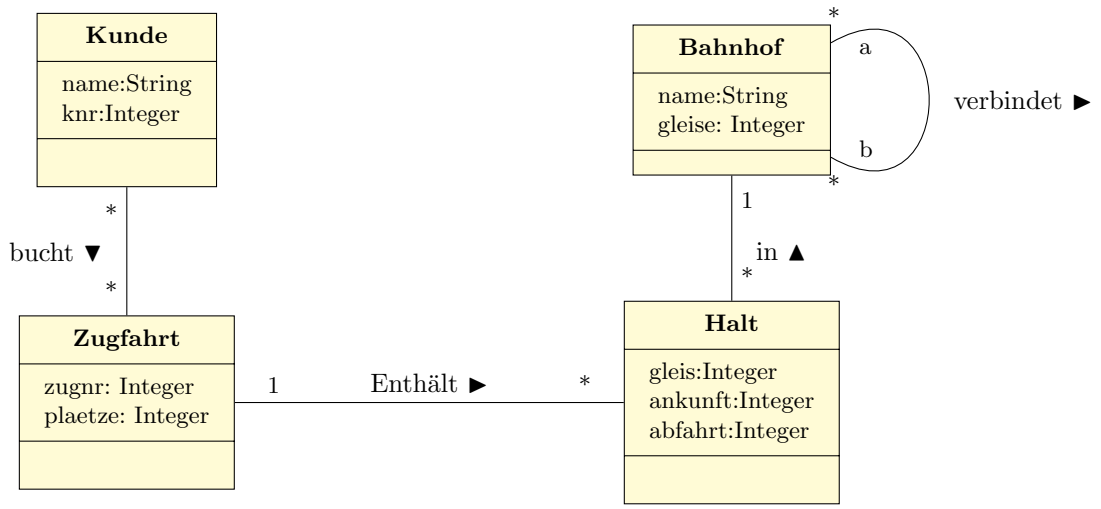
c) Erstellen Sie für die Operation `besuchAnteil` eine Spezifikation in Form eines OCL Contracts. Die Operation soll den Anteil der Besuche aller Besuche für diesen Ort in Prozent angeben, die teilweise oder ganz im übergebenen Zeitraum liegen. Vor Ausführung muss geprüft werden ob der Zeitraum gültig ist. Die übergebenen Werte sind Minuten am Tag, die Zählung startet bei 1 und ein Tag hat 1440 Minuten. (4P)

```
context Ort::besuchAnteil(startZ:Integer,endeZ:Integer):Integer
pre:


post:
```

8. OCL

Gegeben ist der nachfolgende Modellausschnitt in Form eines Klassendiagramms. Lösen sie die Aufgaben unter Berücksichtigung der Eigenschaften dieses Modells mithilfe von OCL.



a) Vervollständigen Sie die folgenden Invarianten, so dass sie die gegebene Anforderung spezifizieren.

- Jeder Bahnhof hat mindestens ein Gleis.

```
context ________________________ inv:
```

- Alle Verbindungen zwischen Bahnhöfen sind symmetrisch.

```
context ________________________ inv:
```

- Für jeden Halt gilt, dass die Abfahrt nach der Ankunft liegt und die anderen Halte der Zugfahrt entweder echt davor oder danach geplant sind.

```
context ________________________ inv:
```

b) Kreuzen Sie für jede der gegebenen Invarianten in OCL an, ob sie die textuelle Anforderung umsetzen (**Ja**) oder nicht (**Nein**). Es darf davon ausgegangen werden, dass die Invarianten aus a) gelten. Kreuzen Sie für jeden Lösungsvorschlag *Ja* oder *Nein* an.

Jede Zugfahrt hat mindestens zwei Halte.	**Ja**	**Nein**
`context Kunde inv: self.zugfahrt->forAll(x | x.halt->size() >= 2)`	☐	☐
`context Zugfahrt inv: self.halt->size() >= 2`	☐	☐
`context Halt inv: Zugfahrt.allInstances()->reject(halt->size() < 2) = Zugfahrt.allInstances()`	☐	☐
`context Halt inv: self.zugfahrt.halt->count(self) = 2`	☐	☐

Das Gleis eines Halts muss im Bahnhof existieren. (Annahme: Gleise sind ab 1 durchgehend nummeriert)	**Ja**	**Nein**
`context Bahnhof inv: self.halt.gleis->includes(self.gleise)`	☐	☐
`context Halt inv: Set{1}->closure(x | if x < bahnhof.gleise then x+1 else x endif)->includes(gleis)`	☐	☐
`context Halt inv: Bahnhof.allInstances()->forAll(gleise >= self.gleis)`	☐	☐
`context Bahnhof inv: halt->forAll(gleis > 0 and gleis <= self.gleise)`	☐	☐

Alle Halte einer Zugfahrt müssen (direkt oder indirekt) verbunden sein.	**Ja**	**Nein**
`context Zugfahrt inv: self.halt.bahnhof.a->includesAll(halt.bahnhof)`	☐	☐
`context Zugfahrt inv: self.halt.bahnhof->closure(x | x.a)-> includesAll(self.halt.bahnhof)`	☐	☐
`context Halt inv: Set{self.bahnhof}->closure(x | x.a)-> includesAll(self.zugfahrt.halt.bahnhof)`	☐	☐
`context Bahnhof inv: a->includesAll(halt.zugfahrt.halt.bahnhof)`	☐	☐

c) Spezifizieren Sie die Operation `zugfahrtBuchen` der Klasse `Kunde`, in der einem Kunden eine neue Zugfahrt zugewiesen wird und umgekehrt, sofern vorher genug Plätze für die Zugfahrt frei sind.

```
context Kunde::zugfahrtBuchen(z : Zugfahrt)
pre:




post:
```

d) Spezifizieren Sie die Operation `erreichbar`, mit der eine Menge von Bahnhöfen ermittelt wird, die über die von einem Bahnhof aus verlaufenden Zugfahrten innerhalb eines gegebenen Zeitfensters ohne Umsteigen erreicht werden können. Die Operation wird nur für gültige Zeiträume durchgeführt (Zeitangaben werden als Minuten am Tag übergeben).

```
context Bahnhof::erreichbar(minAbfahrt:Integer,maxAnkunft:Integer) : Set(Bahnhof)
pre:




post:
```

9. Hoare-Kalkül

a) Wenden Sie das Ersetzungsaxiom an.

{_____}

`a = 3 * c;`

$\{a * b = 3 * c + a\}$

b) Ist folgende Umformung im Hoare Beweis möglich? Kreuzen Sie an. (1P)

	Ja	**Nein**

`a := ...`
$\{a \geq 0 \wedge a = b + 1\}$
$\{a > 0 \wedge a - 1 = b\}$
`b := ...`
	☐	☐

c) Gegeben ist das nachfolgende Programm `mulmax(n, max)`.

$\{max \geq 0\}$

```
mulmax(n, max):
    r := 0;
    i := 0;
    while r + n <= max do
        r := r + n;
        i := i + 1
    od
```

$\{r = n * i \wedge r > max - n \wedge r \leq max\}$

Wertetabelle zum Zeitpunkt der Ausführung des Schleifenkopfs zu `mulmax(3, 11)`:

n	**max**	**i**	**r**
3	11	0	0
3	11	1	3
3	11	2	$6 = 3 + 3$
3	11	3	$9 = 3 + 3 + 3$

Geben Sie die Invariante I zu `mulmax` an, mit der sich partielle Korrektheit zeigen lässt. Die Invariante muss alle Variablen (`n`, `max`, `r` und `i`) enthalten.

$I = \{$_____$\}$

d) Verwenden Sie die Invariante aus Aufgabe 6.c), um die While-Regel anzuwenden. Fassen Sie dabei nicht zusammen.

{_____}

`while r + n <= max do`

{_____}

`r := r + n;`

`i := i + 1`

{_____}

`od`

{_____}

e) Geben Sie eine Terminierungsfunktion t für das unten stehende Programm `samesame(n)` an und vervollständigen Sie den Beweis der totalen Korrektheit.

$\{n \geq 0\}$
```
samesame(n):
     i := 0;
     while i < n do
          i := i + 1
     od
```
$\{i = n\}$

Terminierungsfunktion :

$t =$ `__________`

Erster Teil des Beweises (1P):

Zweiter Teil des Beweises (1,5P):

$\{n \geq 0\}$

`i := 0;`

$\{i \leq n\}$

`while i < n do`

$\{i \leq n \wedge i < n \wedge$ `__________` $\}$

$\{i + 1 \leq n \wedge$ `__________` $\}$

`i := i + 1`

$\{i \leq n \wedge$ `__________` $\}$

`od`

$\{i \leq n \wedge i \geq n\}$

$\{i = n\}$

10. Hoare

9 Gegeben sei die folgende Funktion $x = topfac(k, n) = \frac{n!}{(k-1)!}$:

```
topfac(int k, int n):
     x := k;
     i := k;
     while i < n do
          i := i + 1;
          x := x * i
     od
```

Werte zum Zeitpunkt der Schleifenbedingung jeder Iteration am Beispiel $x = topfac(3, 6)$:

k	n	i	x
3	6	3	3
3	6	4	$12 = 3 \cdot 4$
3	6	5	$60 = 3 \cdot 4 \cdot 5$
3	6	6	$360 = 3 \cdot 4 \cdot 5 \cdot 6$

a) Vervollständigen Sie die Invariante, passend zu dem angegebenen Code. (2 P)

$I :=$ {_____}

b) Vervollständigen Sie den Beweis der partiellen Korrektheit des Codes. Jede Lücke muss ausgefüllt werden und sich sinnvoll von seinen Vorgängern und Nachfolgern unterscheiden. (5 P)

```
topfac(int k, int n):
```
$\{k \leq n\}$

$\{k = k \wedge k \leq n\}$

```
x := k;
```
$\{x = k \wedge k \leq n\}$

{_____}

```
i := k;
```

{_____}

```
while i < n do
```

{_____}

{_____}

{_____}

```
i := i + 1;
```

{_____}

```
x := x * i
```

{_____}

```
od
```

{----------}
{----------}
$\{x = \prod_{a=k}^{n} a\}$

c) Welche Terminierungsfunktionen (auch Schleifenvarianten) t führen zu einem zulässigen bzw. unzulässigen Beweis der totalen Korrektheit der Funktion $x = topfac(k, n)$? Kreuze an! (2 P.)

	t	**Zulässig**	**Nicht zulässig**
1.	i	☐	☐
2.	$n - i$	☐	☐
3.	$-i$	☐	☐
4.	$-n + 1$	☐	☐