Klausur Softwaretechnik

25. April 2003

MUSTERLÖSUNG

Aufg.0	Aufg.1	Aufg.2	Aufg.3	Aufg.4	Aufg.5	Σ	Note
/1	/20	/8	/11	/9	/11	/60	

Die Klausur bestand aus 17 Seiten und war Ugeheftet abzugeben. Für die Vollständigkeit nicht gehefteter Klausuren wird keine Verantwortung übernommen.

Punktgrenze	**Note**
0	**5.0**
21	**4.0**
24	**3.7**
27	**3.3**
30	**3.0**
33	**2.7**
37	**2.3**
41	**2.0**
45	**1.7**
49	**1.3**
53	**1.0**

Aufgabe 0 (1 Punkt)

Schreiben Sie auf jedes Blatt Ihren Namen und Ihre Matrikelnummer in die dafür vorgesehenen Felder.

Aufgabe 1 (20 Punkte)

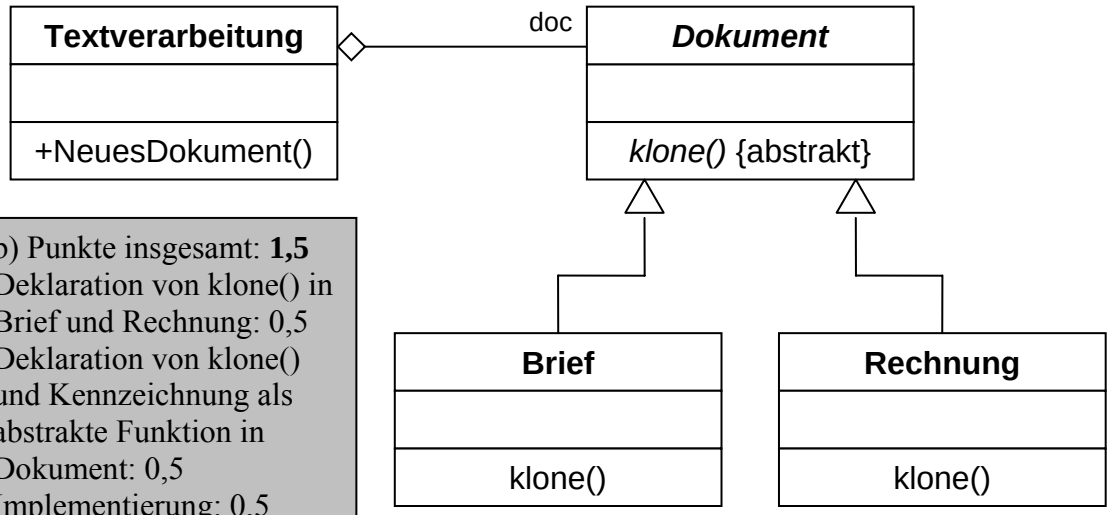
Hinweis: Falsche Kreuze geben negative Punkte, fehlende Kreuze, sowie fehlende oder falsche Freitext-Antworten bewirken nichts. Weniger als 0 Punkte können Sie mit dieser Aufgabe insgesamt nicht erreichen.

a) Nennen Sie vier auf die Planungs- und Definitionsphase folgende Phasen der Softwareentwicklung.

a) Punkte insgesamt: **2,0**

4 Phasen: 4 x 0,5 Punkte.

- Entwurfsphase
- Implementierungsphase
- Testphase
- Abnahme-/Einführungs-/Installationsphase
- Wartungs- und Pflegephase



b) Punkte insgesamt: **1,5**
Deklaration von klone() in Brief und Rechnung: 0,5
Deklaration von klone() und Kennzeichnung als abstrakte Funktion in Dokument: 0,5
Implementierung: 0,5

b) Das obige Klassendiagramm zeigt, wie das Entwurfsmuster *Prototyp* verwendet wird, um in einer Textverarbeitung Dokumente wie z.B. Briefe oder Rechnungen zu erzeugen. Ergänzen Sie in dem Diagramm die entsprechenden Klassen um die für einen Prototypen notwendige Funktion. Geben Sie nachfolgend eine Pseudocode-Implementierung dieser fehlenden Funktion in der Klasse **`Brief`** an.

return Kopie von sich selbst;

Name: _____ Matrikelnummer: _____

c) Geben Sie eine Pseudocode-Implementierung der Funktion **`NeuesDokument()`** der Klasse **`Textverarbeitung`** an.

return doc.klone(); oder p = doc.klone();

c) Punkte insgesamt: **0,5**

d) Das Entwurfsmuster *Einzelstück* (Singleton) wird verwendet, wenn es nur eine einzige Instanz von einer bestimmten Klasse geben darf.

```
class Singleton {
   private static Singleton instance = null;
   public static Singleton getInstance();
   protected Singleton();
};
```

d) Punkte insgesamt: **1,0**

Pro Zeile 0,5

Bei der Initialisierung dieser Java-Klasse wird **`instance`** auf **`null`** gesetzt.

Vervollständigen Sie die Implementierung der Funktion **`getInstance()`**.

```
public static Singleton getInstance () {

   if (instance == null)  { instance = new Singleton() }

   return                   instance ;

}
```

e) Sie haben ein Programm geschrieben, das Sie nun optimieren möchten. Nennen Sie ein Optimierungsprinzip, mit dem Sie die Anzahl der durch Schleifen verursachten Unterbrechungen in der Befehlspipeline vermindern können.

Loop unrolling, function inlining

e) Punkte insgesamt: **1,0**

Pro Zeile 0,5

Benennen Sie die Optimierungsebene, auf der Sie dieses Prinzip anwenden.

Feinoptimierung

f) Punkte insgesamt: **1,0**

Pro Zeile 0,5

f) Welche Eigenschaft muss eine Benutztrelation zwischen abstrakten Maschinen haben?

Die Relation zwischen abstrakten Maschinen ist zyklenfrei.

Man nennt solch eine Benutztrelation deshalb auch hierarchisch .

g) Betrachten Sie die folgende Entscheidungstabelle zum Planen einer Dienstreise. Beantworten Sie die folgenden Fragen.

ET Reiseplanung		R1	R2	R3	R4
B1	Die Entfernung ist größer als 300 km.	N	J	J	J
B2	Der Flug kostet weniger als 150 Euro.	-	N	J	N
B3	Es ist eine ICE Verbindung verfügbar.	-	J	J	N
A1	Fahre mit dem Auto	X			
A2	Fliege mit dem Flugzeug				X
A3	Fahre mit der Bahn		X	X	

Die oben angegebene Entscheidungstabelle (ET) ist eine...

Richtig/Falsch

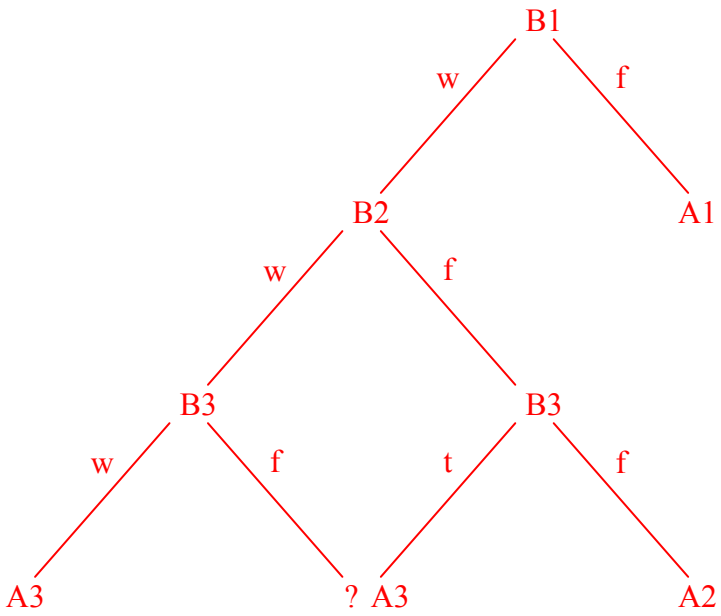
- X/□ ... Eintreffer-ET
- □/X ... Inhaltlich vollständige ET
- □/X ... Erweiterte ET

g) Punkte insgesamt: **1,5**

Pro richtigem Kreuz +0,5

Pro falschem Kreuz -0,5

h) Erstellen Sie einen Entscheidungsbaum gemäß der Reiseplanungs-Entscheidungstabelle aus g).



h) Punkte insgesamt: **2,0**

Pro falschem/fehlerhaften Pfad: -0,5

Kein „?“: -0,5

Name: _____ Matrikelnummer: _____

i) Nennen Sie 3 Entwurfsmusterkategorien und geben Sie für jede Kategorie ein Beispiel an.

i) Punkte insgesamt: **1,5**

Pro richtiger Kombination 0,5

Entkopplungs-Muster:

- Abstrakter Datentyp
  - Modul
  - Datenablage
    - Client/Server
  - Abstrakter Datentyp
  - Manger
  - Iterator
  - Datensammlung
    - Menge/Sequenz
- Schichtenarchitektur
  - Sandwich
  - Fassade
  - Vermittler
  - Brücke
  - Adapter
  - Facette
  - Stellvertreter
    - Dekorierer
    - Puffernder/Protokollierender Stellvertreter
    - Firewall
    - Synchronisierer
    - Fernzugriffsvertreter
- Fließband
- Rahmenarchitektur
- Ereignissteuerung
  - Ereignisbehandler
  - Rückruf
  - Ereignisschleife
  - Ereigniskanal
  - Propagierer
    - Genauer Propagierer
    - Fauler Propagierer
    - Anpassungsfähiger Propagierer
    - Beobachter

Variantenmuster:

- Oberklasse

    - Strategie
    - Kompositum
- Besucher
    - Einfacher Besucher
    - Äußerer Besucher
    - Aperiodischer Besucher
- Schablonenmethode
    - Farbikmethode
    - Erbauer
- Abstrakte Fabrik

Zustandshandhabungsmuster:

- Memento
- Prototyp
- Fliegengewicht
- Einzelstück

Steuerungsmuster:

- Tafel
- Befehl
- Zuständigkeitskette
- Strategie
- Steuer-Zustand
- Master-Slave
- Prozesssteuerung
    - ...

Virtuelle Maschinen:

- Interpretierer
    - Emulator
- Regelbasierter Interpretierer

Bequemlichkeitsmuster

- Bequemlichkeits-Methode
- Bequemlichkeits-Klasse
- Fassade
- Null-Objekt

Name: _____ Matrikelnummer: _____

j) Wandeln Sie den Datenwörterbuch-Eintrag A = (BC | D | DD | BCD | BCDD) in eine möglichst kompakte Form um. Verwenden Sie die in der Vorlesung eingeführte Notation für Datenwörterbücher.

A = ((BC) 0{D}2) oder
A = (BC) 0{D}2

j) Punkte insgesamt: **1,5**

Gibt's nur wenn vollständig richtig und Notation korrekt.

k) Welche der folgenden Aussagen sind richtig und welche sind falsch?

Richtig/Falsch

k) Punkte insgesamt: **2,5** Falsches Kreuz -0,5

X/□ Ein Datenflußdiagramm (DFD) enthält mindestens eine Schnittstelle.

□/ X Zwischen DFD Schnittstellen kann es Datenflüsse geben.

□/ X Zwischen DFD Speichern dürfen direkte Datenflüsse gezeichnet werden.

□/ X Zwischen DFD Schnittstellen und DFD Speichern dürfen direkte Datenflüsse gezeichnet werden.

□/ X In Datenwörterbüchern sind zirkuläre Definitionen erlaubt.

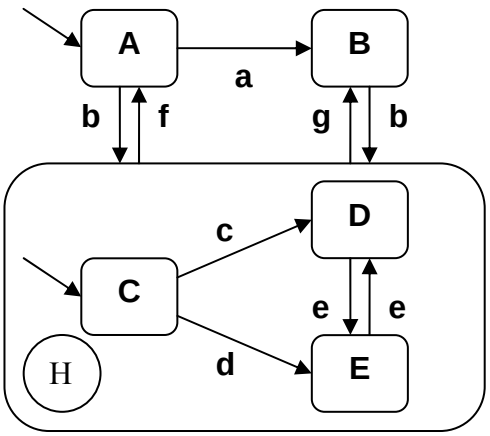
l) In welchem Zustand befindet sich der unten stehende Harelautomat nach den folgenden Eingaben? (Der Automat befindet sich zuvor jeweils im Anfangszustand.)

l) Punkte insgesamt: **1,0**

Pro Antwort 0,5

a, b, c, f, b: D

b, d, e, g, e, b: D



m) Restrekursive Funktionen können mechanisch in eine iterative Form gebracht werden. Betrachten Sie die folgende Funktion **g**:

```
int g(int x) {
  if (B(x)) {
    S(x);
    return g(E(x));
  } else {
    T(x);
    return p(x);
  }
}
```

Hinweis: **B**, **S**, **E**, **T** und **p** stehen für beliebige Operationen auf **x** ohne Verwendung der Funktion **g**.

Transformieren Sie die Funktion **g** nach dem aus der Vorlesung bekannten Schema in eine iterative Funktion **f**.

```
int f(int x) {
  int x1 = x;
  while (B(x1))
    S(x1);
    x1 = E(x1);
  }
  T(x1);
  return p(x1);
}
```

m) Punkte insgesamt: **2,0**

„E(x1)“ anstatt „x1 = E(x1)“ -0,5

Wenn „x“ anstatt „x1“: -1

n) Welche der folgenden Aussagen sind richtig und welche sind falsch?

Richtig/Falsch

n) Punkte insgesamt: **1,0** Falsches Kreuz -0,5

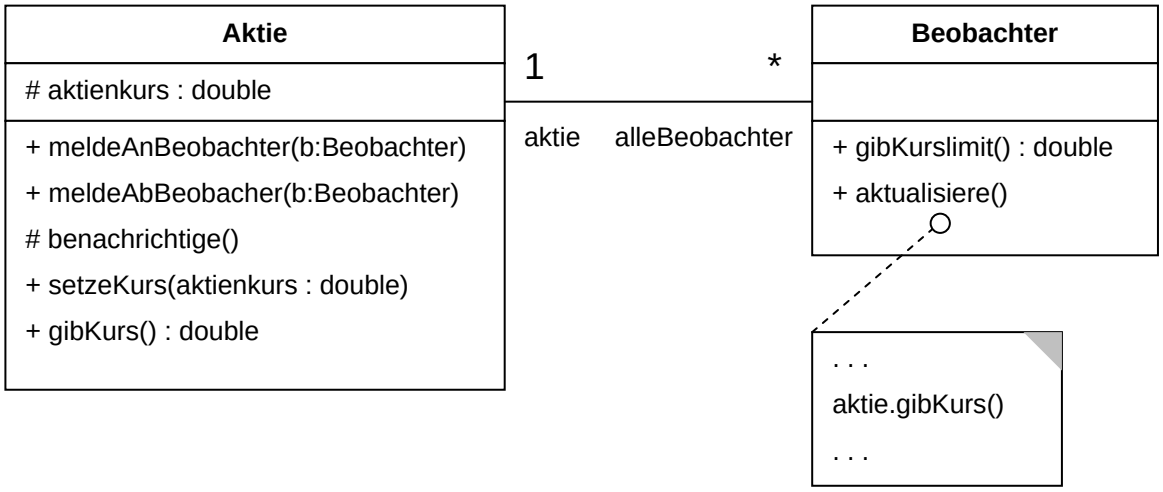
X/□ Ein Moore-Automat lässt sich in einen äquivalenten Mealy-Automaten transformieren.

□/ X Software-Entwurfsmuster sind immer objektorientiert.

Name: _____ Matrikelnummer: _____

Aufgabe 2 (Entwurfsmuster, Sequenzdiagramm) (8 Punkte)

Seitdem die Firma *ChokyCola Inc.* im Jahre 1892 gegründet wurde, hat sich der Aktienkurs sehr positiv entwickelt. In letzter Zeit ließ die Wertentwicklung aber zu wünschen übrig. Die Manager möchten deshalb unverzüglich informiert werden, wenn der Aktienkurs unter eine bestimmte Marke fällt.
Das folgende Klassendiagramm zeigt, wie das Entwurfsmuster *Beobachter* verwendet wurde, um eine Reihe von Managern über den Kurs der *ChokyCola*-Aktie im Falle des Unterschreitens bestimmter Marken zu informieren.



a) Geben Sie eine Implementierung der Funktion **`benachrichtige()`** an, so dass auf den Beobachtern nur dann die Funktion **`aktualisiere()`** aufgerufen wird, wenn der neu gesetzte Kurs der **`Aktie`** niedriger ist als das Kurslimit des Beobachters. Die Funktion **`gibKurslimit()`** der Klasse **`Beobachter`** liefert den Wert zurück, bei dem der Beobachter aktualisiert werden möchte.

Hinweis: Verwenden Sie Java-ähnlichen Pseudocode wie z.B.
```
forall Type t in myArray do ...
```

a) Punkte insgesamt: **3,0**

Auch richtig: „aktienkurs“ anstatt „this.gibKurs();“

```
benachrichtige() {
    forall (Beobachter b in alleBeobachter) {
        if (b.gibKurslimit() > this.gibKurs()) {
            b.aktualisiere();
        }
    }
}
```

Pro richtiger Zeile 1 Punkt

“>” falsch: -0,5

“in alleBeobachter” fehlt: -0,5

b) Wie kann man vermeiden, dass die Methode **`aktualisiere()`** die Methode **`gibKurs()`** aufrufen muss, um den aktuellen Kurs zu bekommen?

b) Punkte insgesamt: **1,0** Antwort kann sowohl Pseudocode als auch verbal sein.

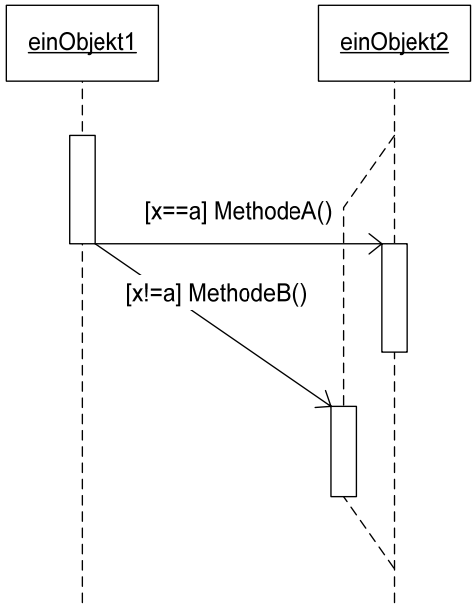
aktualisiere (aktuellerKurs : double) { ... } oder

Man übergibt den aktuellen Kurs der Methode aktualisiere als Parameter.

c) Ergänzen Sie das UML Sequenzdiagramm auf der nächsten Seite wie folgt:

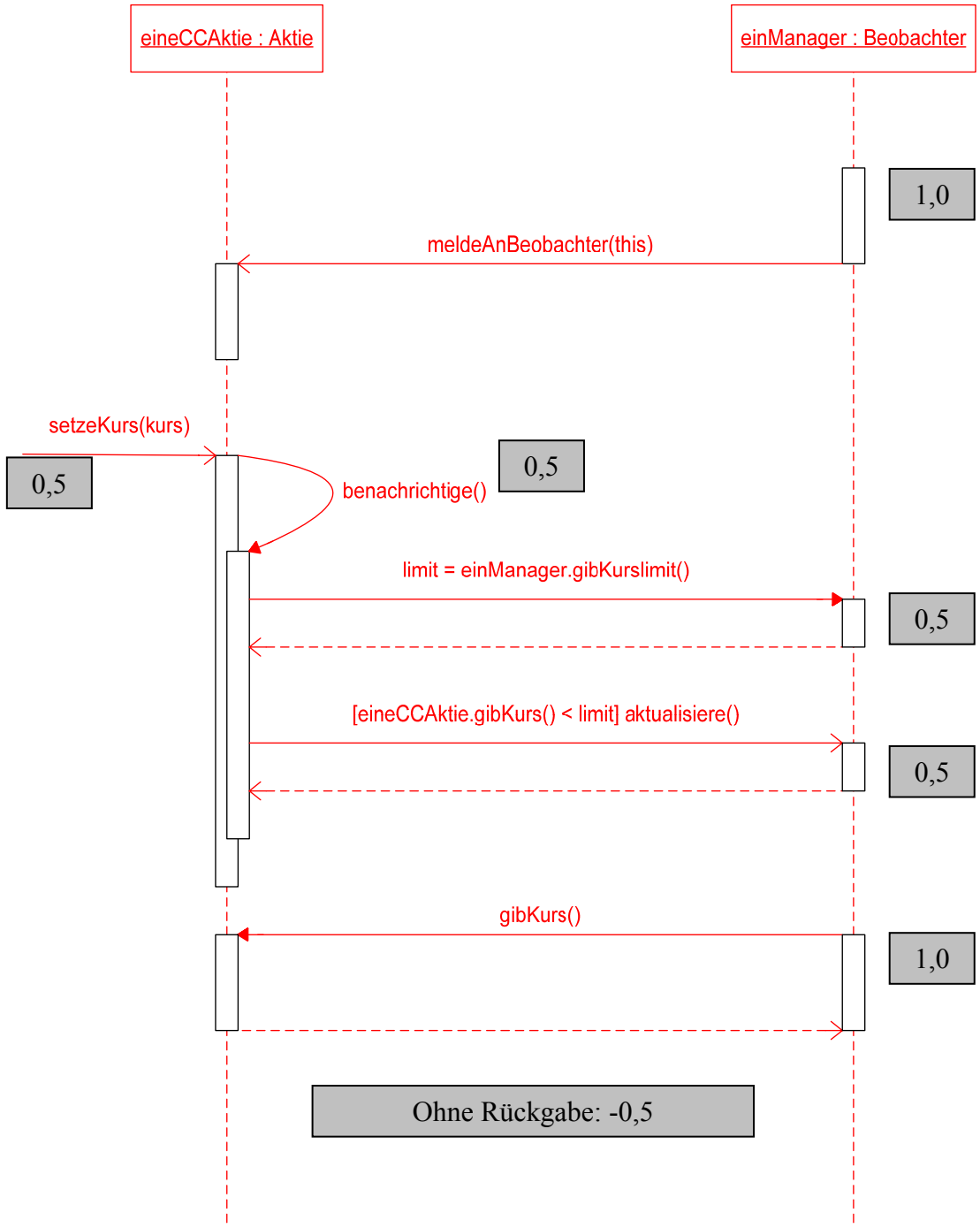
- **`einManager`** meldet sich bei einer *ChokyCola*-Aktie (**`eineCCAktie`**) an.
- Der Kurs von **`eineCCAktie`** wird gesetzt.
- Die Funktion **`aktualisiere()`** wird auf **`einManager`** nur dann aufgerufen, wenn der Aktienkurs kleiner als das Kurslimit ist.

Hinweis: Fallunterscheidungen werden in UML Sequenzdiagrammen wie in dem folgenden Beispiel durch mehrfache Lebenslinien dargestellt. Bedingungen werden in eckigen Klammern notiert.
Wenn **`x==a`** gilt, wird **`MethodeA`** aufgerufen.
Wenn **`x!=a`** gilt, wird alternativ **`MethodeB`** aufgerufen.



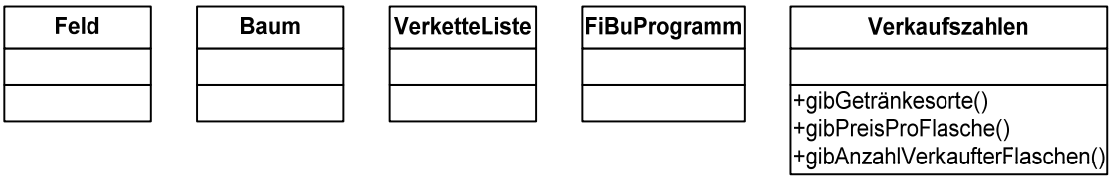
Name: _____ Matrikelnummer: _____

c) Punkte insgesamt: **4,0**
Jeweils 1 + 2 + 1 Punkt auf den oberen Teil, den Mittelteil und den unteren Teil
Für ein fehlendes Element: -0,5



Aufgabe 3 (Objektorientierter Entwurf, Entwurfsmuster) **(11 Punkte)**

Die Getränkefirma *ChokyCola Corp.*, ein Hersteller von Limonade, unterhält Niederlassungen in vielen Ländern. Für die Bilanzierung über Ländergrenzen hinweg verwendet die Firma eine in Java geschriebene Software (**`FiBuProgramm`**). Jede Niederlassung verwendet die unten dargestellte Klasse **`Verkaufszahlen`** mit den Methoden **`gibGetränkesorte()`**, **`gibPreisProFlasche()`** und **`gibAnzahlVerkaufterFlaschen()`** (Der Preis für eine bestimmte Getränkesorte ist über das Jahr hinweg konstant).
Da in jedem Land eine Reihe verschiedener Getränkesorten vertrieben werden (z.B. *ChokyCola*, *FancyFanta*), speichert die Niederlassung jedes Landes für jede Getränkesorte eine Instanz der Klasse **`Verkaufszahlen`** in einer Behälterklasse. Dummerweise verwendet dabei jedes Land für diese Behälterklasse eine andere Implementierung (z.B. Feld, verkettete Liste, Baum, etc.). Die folgenden Klassen sind gegeben:



Die Methode **`gibPreisProFlasche()`** liefert den Verkaufspreis einer Flasche Limonade vom Typ **`gibGetränkesorte()`**. Die Methode **`gibAnzahlVerkaufterFlaschen()`** liefert die Anzahl verkaufter Flaschen dieser Getränkesorte.

a) Am Jahresende möchte die Zentrale den Umsatz des gesamten Konzerns ausrechnen. Warum bietet es sich an, hierfür das Entwursfmuster *Iterator* zu verwenden?

a) Punkte insgesamt: **1,0**	Jeweils 0,5 pro Aspekt.

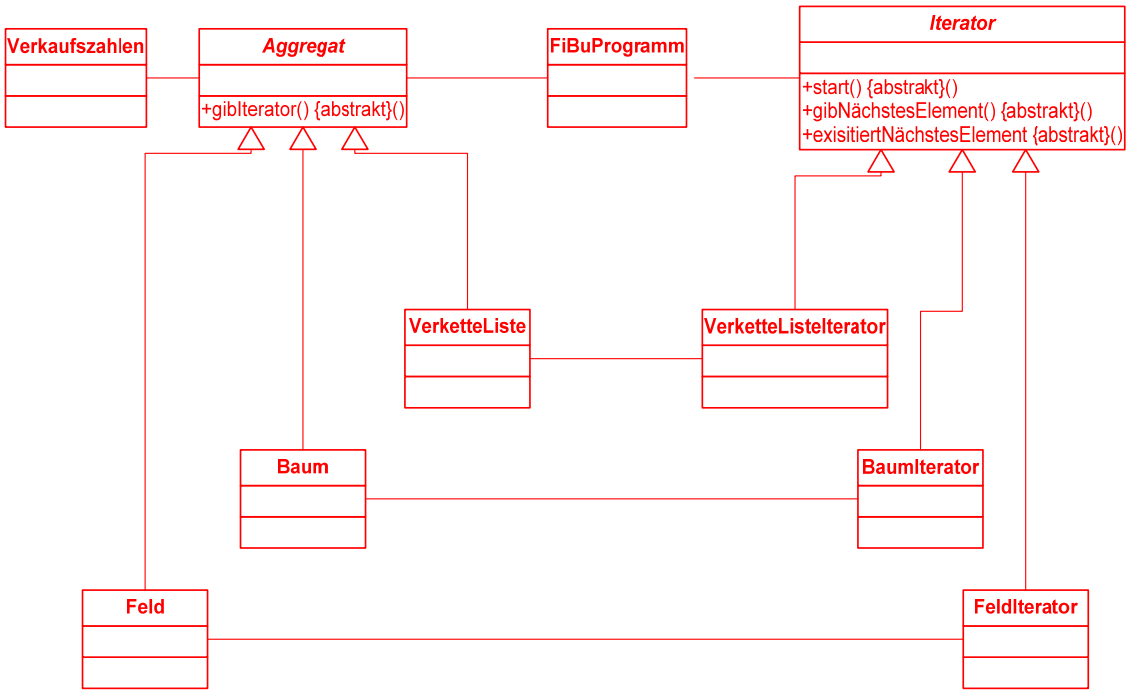
Einheitliche Schnittstelle zur Traversierung unterschiedlicher zusammengesetzter Strukturen. und Ermöglicht Zugriff auf den Inhalt eines zusammengesetzten Objekts, ohne dabei seine interne Struktur offenzulegen.

b) Wenden Sie das Entwurfsmuter *Iterator* an. Erstellen Sie auf der gegenüberliegenden Seite ein UML-Klassendiagramm, das die oben gegebenen Klassen und zusätzlich ein abstraktes **`Aggregat`**, einen abstrakten **`Iterator`** und die notwendigen konkreten Iteratoren enthält.

c) Zeichnen Sie in dem Klassendiagramm folgende *abstrakte* Funktionen in die entsprechenden abstrakten Oberklassen ein:

```
gibIterator()                 // Liefert einen Iterator.
start()                       // Setze Iterator auf Beginn des Aggregats.
gibNächstesElement()          // Liefert das nächste Element.
existiertNächstesElement()    // Liefert true wenn das letzte Element des
                                 Aggregats noch nicht erreicht wurde.
```

Name: _____ Matrikelnummer: _____



b) Punkte insgesamt: **5,0**

3x konkreter Iterator und Assoziation zu konkreter Behälterklasse: 3 x 1
Ohne Assoziationen zwischen Behälter + Iterator: 3 x 0,5

FiBuProgramm mit Assoziation zu Aggregat: 0,5

Verkaufszahlen und Assoziation zu Aggregat: 0,5

Vererbungsrelationen zwischen abstraktem Aggregat/Iterator zu konkreten Behältern/Iteratoren: 2 x 0,5

c) Punkte insgesamt: **2,0**

4 x Funktionen korrekt eingezeichnet und als abstrakt gekennzeichnet: 4 x 0,5
Kennzeichnung als abstrakt fehlt: pro Klasse -0,5

d) Erstellen Sie, basierend auf dem Klassendiagramm aus c), eine Java-Implementierung, die den in einem Land erzielten Umsatz berechnet. Der Umsatz soll berechnet werden, indem für jede Getränkesorte die Anzahl verkaufter Flaschen mit dem jeweiligen Verkaufspreis pro Flasche multipliziert wird. Vereinfachend wird angenommen, dass alle Variablen vom Typ **`float`** sind.

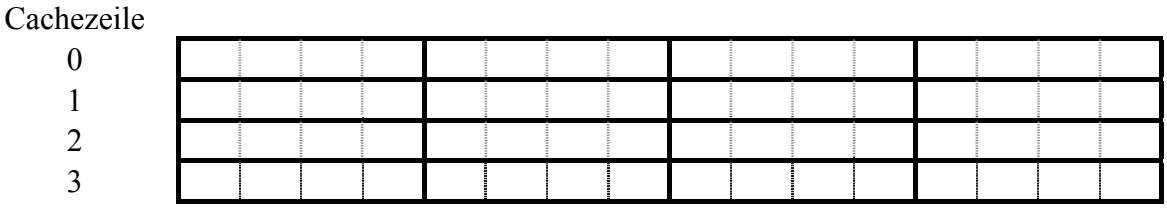
d) Punkte insgesamt: **3,0** 6 Zeilen: 6x 0,5 Punkte

```
float berechneUmsatz(Aggregat a) {

    Iterator i = a.gibIterator();

    float umsatz = 0.0;

    while(i.existiertNaechstesElement()) {

        Verkaufszahlen v = i.gibNaechstesElement();

        umsatz += (v.gibPreisProFlasche() *
                        v.gibAnzahlVerkaufterFlaschen());
      }

    return umsatz;

}
```

Name: _____ Matrikelnummer: _____

Aufgabe 4 (Caches) **(9 Punkte)**

Gegeben ist ein direkt abgebildeter Cache. Er ist 64 Byte groß und hat 4 Cachezeilen. Jede Cachezeile besteht aus 16 Byte. Der Datentyp **int** besteht aus 2, der Datentyp **float** aus 4 Byte. Somit passen in jede Cachezeile immer genau 8 bzw. 4 Datenelemente des Typs **int** bzw. **float**. Die folgende Zeichnung veranschaulicht noch einmal den Aufbau des Caches. Die kleinen Kästchen entsprechen einem Byte.



Die folgende Programmschleife in C ist ein Auszug aus einem größeren Programm, das von der Firma *ChokyCola Corp.* verwendet wird, um in 12 europäischen Ländern den Umsatz mit *ChokyCola* zu berechnen. Der in einem Land erzielte Umsatz wird sowohl in Euro als auch in Dollar abgespeichert. Ein Euro kostet 1,20 Dollar.

```
10   //*******************************
20   // Programmiersprache: ANSI C
30   //
40   int s[12];               // Stückzahlen verkaufter Flaschen
50   float p[12];             // Preise pro Flasche
60   float u[24];             // Umsätze in Euro und Dollar
70   float eurokurs = 1.20;

     ...

80   for (int i=0; i<12; i++) {
90     u[2*i] = s[i] * p[i];
100    u[2*i+1] = u[2*i] * eurokurs;
110  }
     ...
```

Die Anfangsadressen der Felder im Hauptspeicher sind wie folgt:

```
Adresse(u[0]) =    8;
Adresse(s[0]) =  144;
Adresse(p[0]) = 4096;
```

Alle skalaren Werte liegen in Registern des Prozessors. Die Elemente der Felder **u[]**, **s[]** und **p[]** werden über den Cache in den Prozessor geladen.

a) Wie lauten die folgenden Hauptspeicher-Adressen für **`i = 2`**?

Adresse(u[2*i]) = 24

Adresse(u[2*i+1]) = 28

Adresse(s[i]) = 148

a) Punkte insgesamt: **1,0**

u[2*i] + u[2*i+1]: 0,5
s[i]: 0,5

Cacheabbildung von **u[2*i]**

Cachezeile				
0	~~~~		~O~	
1	~~~~		~~~~	
2	~~~~		~~~~	
3	~~~~		~~~~	

Cacheabbildung von **u[2*i+1]**

Cachezeile				
0		~~~~		~O~
1		~~~~		~~~~
2		~~~~		~~~~
3		~~~~		~~~~

Cacheabbildung von **s[i]**

Cachezeile								
0								
1	O~	~	~	~	~	~	~	~
2	~	~	~	~				
3								

Cacheabbildung von **p[i]**

Cachezeile				
0	~O~	~~~~	~~~~	~~~~
1	~~~~	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~	~~~~
3				

b) Markieren Sie in den obigen Cacheabbildungen die Positionen der Feldelemente **u[2*i]**, **u[2*i+1]**, **s[i]** und **p[i]** bei der ersten Iteration mit einem „O“.

b) Punkte insgesamt: **2,0** 4 x 0,5

c) Markieren Sie nun den gesamten Cache-Bereich, der beim Zugriff auf die Felder **u[2*i]**, **u[2*i+1]**, **s[i]** und **p[i]** genutzt wird mit „Wellenlinien“ (~~~~).

c) Punkte insgesamt: **2,0** 4 x 0,5

d) Wieviele Cachezeilen müssen während der ersten 2 Iterationen für die jeweiligen Felder aus dem Hauptspeicher in den Cache geladen werden?

Der Übersetzer wertet Ausdrücke von rechts nach links aus. D.h. in Zeile 90 wird erst auf Feld **p**, dann auf Feld **s** und danach auf Feld **u** zugegriffen.
Der Cache ist zu Beginn der Programmausführung leer.

d) Punkte insgesamt: **4,0**

1. Iteration: 0,5 0,5 0,5

Zeile 90: Feld **p[i]**: 1 Feld **s[i]**: 1 Feld **u[2*i]**: 1

Zeile 100: Feld **u[2*i]**: 0 Feld **u[2*i+1]**: 0 beides richtig: 0,5

2. Iteration: 0,5 0,5 0,5

Zeile 90: Feld **p[i]**: 1 Feld **s[i]**: 0 Feld **u[2*i]**: 1

Zeile 100: Feld **u[2*i]**: 0 Feld **u[2*i+1]**: 0 beides richtig: 0,5

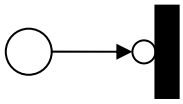
Name: _____ Matrikelnummer: _____

Aufgabe 5 (Petrinetze) (11 Punkte)

Um den Umsatz anzukurbeln, betreibt die Firma *ChokyCola Corp.* ein dichtes Netz von Getränkeautomaten. Ihre Aufgabe ist es nun, während der Entwicklung neuer Getränkeautomaten verschiedene Entwurfsaufgaben mittels Petrinetzen zu lösen.

Die Abbildung auf der nächsten Seite zeigt ein Petrinetz, das einen Getränkeautomaten modelliert. Vereinfachend wird angenommen, dass der Getränkeautomat nur 1-Euro-Münzen akzeptiert. *ChokyCola* kostet 2 Euro, *FancyFanta* kostet 3 Euro.

Hinweise: Verwenden Sie auch so genannte Verbotskanten (Inhibitionskanten).
Bei diesen Verbotskanten ist eine Transition zulässig, wenn die üblichen Zulässigkeitsbedingungen erfüllt sind und wenn sich in der Stelle, die durch die Verbotskante mit der Transition verbunden ist, *keine* Marke befindet.

Eine Verbotskante wird wie folgt gezeichnet: 

a) Zeichnen Sie eine Anfangsmarkierung ein, so dass der Getränkeautomat 75 Dosen *ChokyCola* und 50 Dosen *FancyFanta* enthält.

a) Punkte insgesamt: **1,0** Pro Stelle 0,5.

b) Zeichnen Sie Verzögerungen ein, so dass die Transitionen T_{CC} und T_{FF} erst nach 3 Zeiteinheiten schalten und die verkaufte Dose erst mit einer Verzögerung von 2 Zeiteinheiten freigegeben wird.

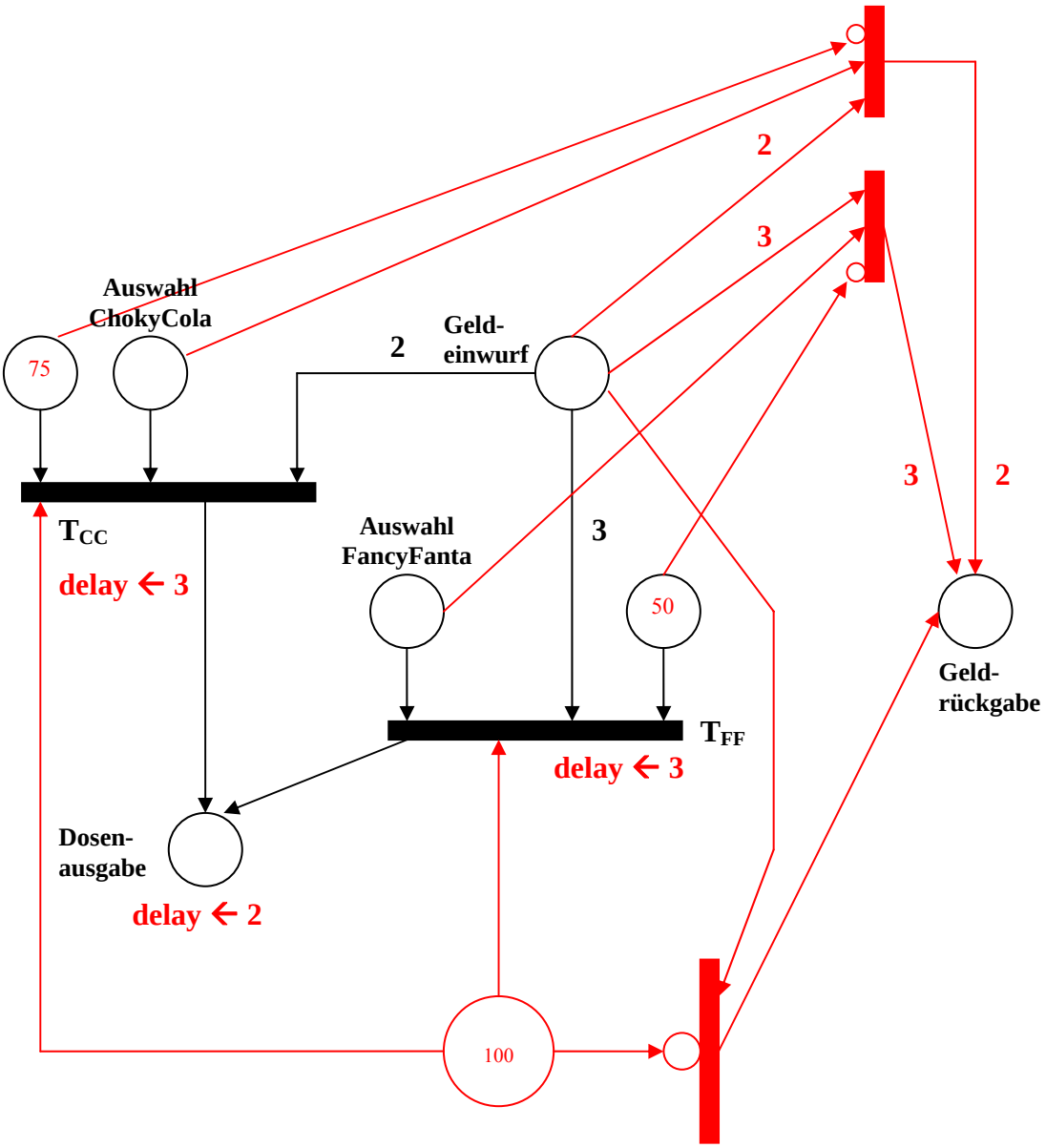
b) Punkte insgesamt: **1,0** T_{CC} und T_{FF} richtig: 0,5 Dosenausgabe: 0,5 Bei der Notation genügt es nicht, nur eine Zahl hinzuschreiben.

c) Erweitern Sie das Petrinetz wie folgt: Nachdem der Automat 100 Dosen verkauft hat, soll er aus wartungstechnischen Gründen keine Dosen mehr ausgeben, sondern eingeworfenes Geld unmittelbar zurückgeben.

c) Punkte insgesamt: **2,0** Wenn die Dosenausgabe nicht blockiert, das Netz aber sonst korrekt ist: 1,0

d) Erweitern Sie das Petrinetz wie folgt:
Wenn keine *ChokyCola* mehr da ist, der Kunde trotzdem 2 Euro einwirft und die Auswahltaste für *ChokyCola* drückt, so sollen ihm die eingeworfenen 2 Euro in die Geldrückgabe zurückgegeben werden.
Wenn keine *FancyFanta* mehr da ist, der Kunde trotzdem 3 Euro einwirft und die Auswahltaste für *FancyFanta* drückt, so sollen ihm die eingeworfenen 3 Euro in die Geldrückgabe zurückgegeben werden.

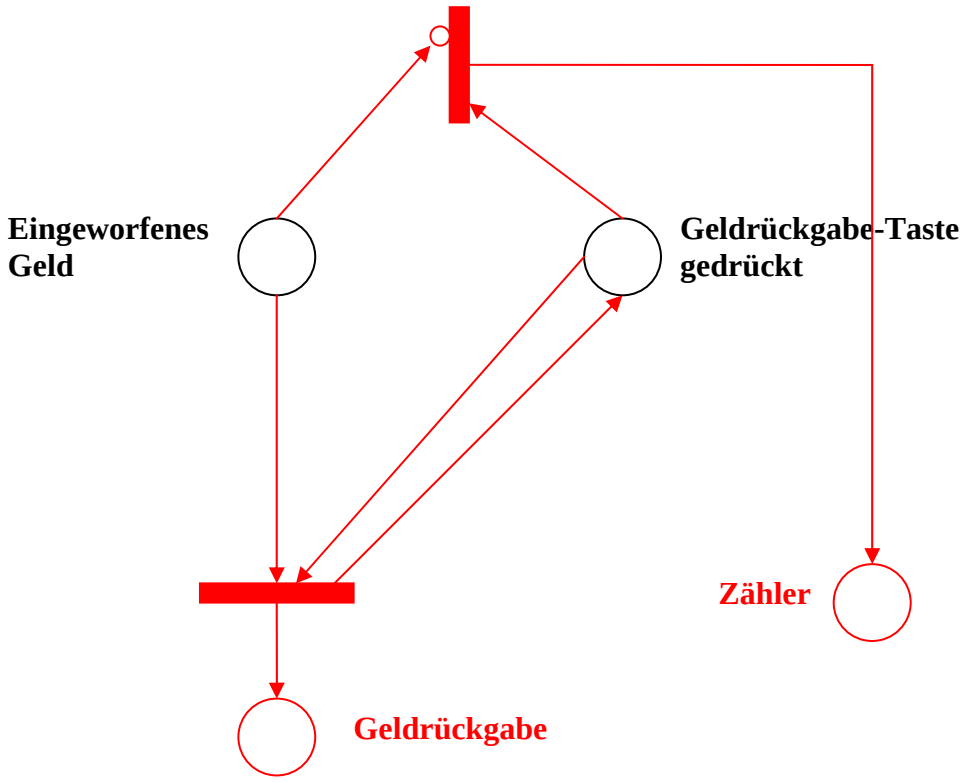
d) Punkte insgesamt: **3,0** 2 x 1,5


Auswahl
ChokyCola
75
Geld-
einwurf
2
2
3
3
2
3
2
T_{CC}
delay ← 3
Auswahl
FancyFanta
50
Geld-
rückgabe
T_{FF}
delay ← 3
Dosen-
ausgabe
delay ← 2
100

e) Das nachfolgend dargestellte unvollständige Petrinetz modelliert den Geldeinwurf und die Geldrückgabe-Taste eines weiteren Getränkeautomaten. Sie sollen folgenden Geldrückgabe-Mechanismus modellieren: Der Kunde kann zu einem beliebigen Zeitpunkt die Geldrückgabe-Taste am Getränkeautomaten drücken und bekommt daraufhin sein bisher eingeworfenes Geld zurück. Darüber hinaus soll in einer Stelle gezählt werden, wie oft die Rückgabe-Taste gedrückt wurde.

Hinweis: Da nicht bekannt ist, wieviel Geld der Kunde eingeworfen hat, müssen Sie einen „Test gegen Null“ durchführen. So etwas geht am besten mit einer „Verbotskante“.

e) Punkte insgesamt: **2,0** Geldrückgabe: 1,5 Zähler: 0,5



f) Eine Transition $t \in T_N$ in einem S/T-Netz heißt zulässig bei Markierung M wenn...

$\forall s \in S_N: M(s) \geq W_N((s,t))$ und $\forall s \in S_N: M(s) + W_N((t,s)) - W_N((s,t)) \leq K(s)$

mit S_N: Stellen des Netzes
T_N: Transitionen des Netzes

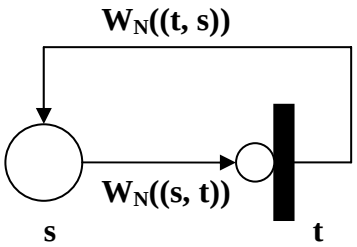
$M(s)$: Anzahl der Marken in der Stelle s
$K(s)$: Kapazität der Stelle s
$W_N((s, t))$: Gewichtung der Kante von der Stelle s zur Transition t
$W_N((t, s))$: Gewichtung der Kante von der Transition t zur Stelle s

Wie lassen sich diese beiden Schaltbedingungen für das abgebildete Petrinetz vereinfachen, wenn die Transition *t* eine Verbotskante wie oben beschrieben ist und die Stelle *s* eine Kapazität von 1 hat?

f) Punkte insgesamt: **2,0**		
1. M(s) = 0:	0,5	
2. $W_N((t, s)) <= 1$	1,0	„=1“ auch richtig
Die Transition schaltet einmal:	0,5	(kein Abzug für falsches Kreuz)

1. M(s) = 0

2. $W_N((t, s)) <= 1$



Wie oft schaltet die Transition *t* bei $M_N = (0)$ und $W_N((t,s)) = 1$?

X einmal □ zweimal □ unendlich oft □ Welche Transition?