# KLAUSUR SOFTWARETECHNIK I

*12.10.2009*

Prof. Dr. Walter F. Tichy
Dipl.-Inform. Andreas Höfer
Dipl.-Inform. David J. Meder

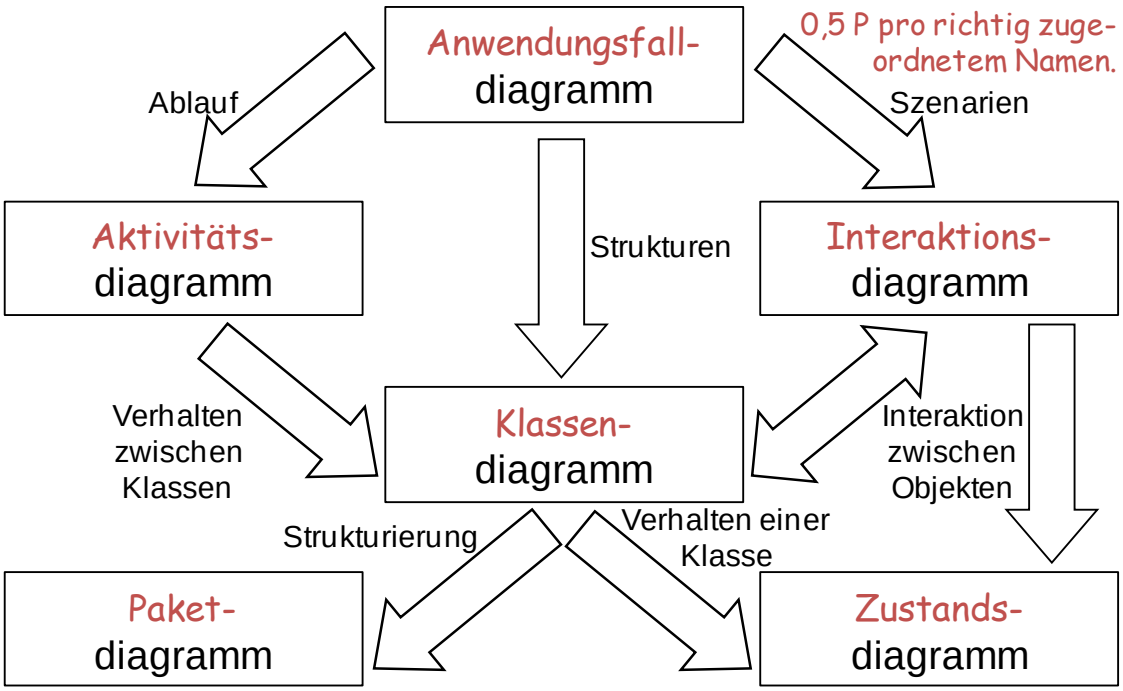
Musterlösung

Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen.
Die Bearbeitungszeit beträgt 60 Minuten.
Die Klausur ist vollständig und geheftet abzugeben.
Mit Bleistift oder roter Farbe geschriebene Angaben werden nicht bewertet.

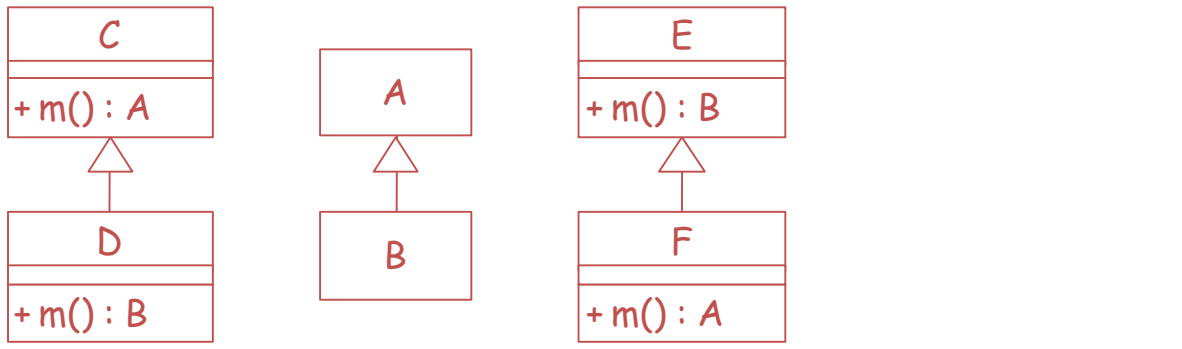
| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | ∑ |
|---|---|---|---|---|---|---|---|
| Maximum | 12 | 8 | 10 | 13 | 11 | 6 | 60 |
| Korrektor 1 | | | | | | | |
| Korrektor 2 | | | | | | | |
| Korrektor 3 | | | | | | | |

# AUFGABE 1: AUFWÄRMEN (12 P)

a) Ergänzen Sie die Namen der UML-Diagramme in der folgenden, aus der Vorlesung bekannten, Übersicht. (3 P)




b) Erklären Sie mit Hilfe eines UML-Diagramms Ko- und Kontravarianz bei einem Ausgabeparameter. Geben Sie jeweils an, ob das Substitutionsprinzip erfüllt ist und ob es in Java erlaubt ist. (3 P)




- Kovarianz mit Diagramm 0,5 P
- Substitutionsprinzip erfüllt 0,5 P
- In Java erlaubt 0,5 P

- Kontravarianz mit Diagramm 0,5 P
- Substitutionsprinzip verletzt 0,5 P
- In Java verboten 0,5 P

c) Welche Arten von Anforderungen an Softwaresysteme gibt es? Erläutern Sie die unterschiedlichen Arten kurz. (3 P)

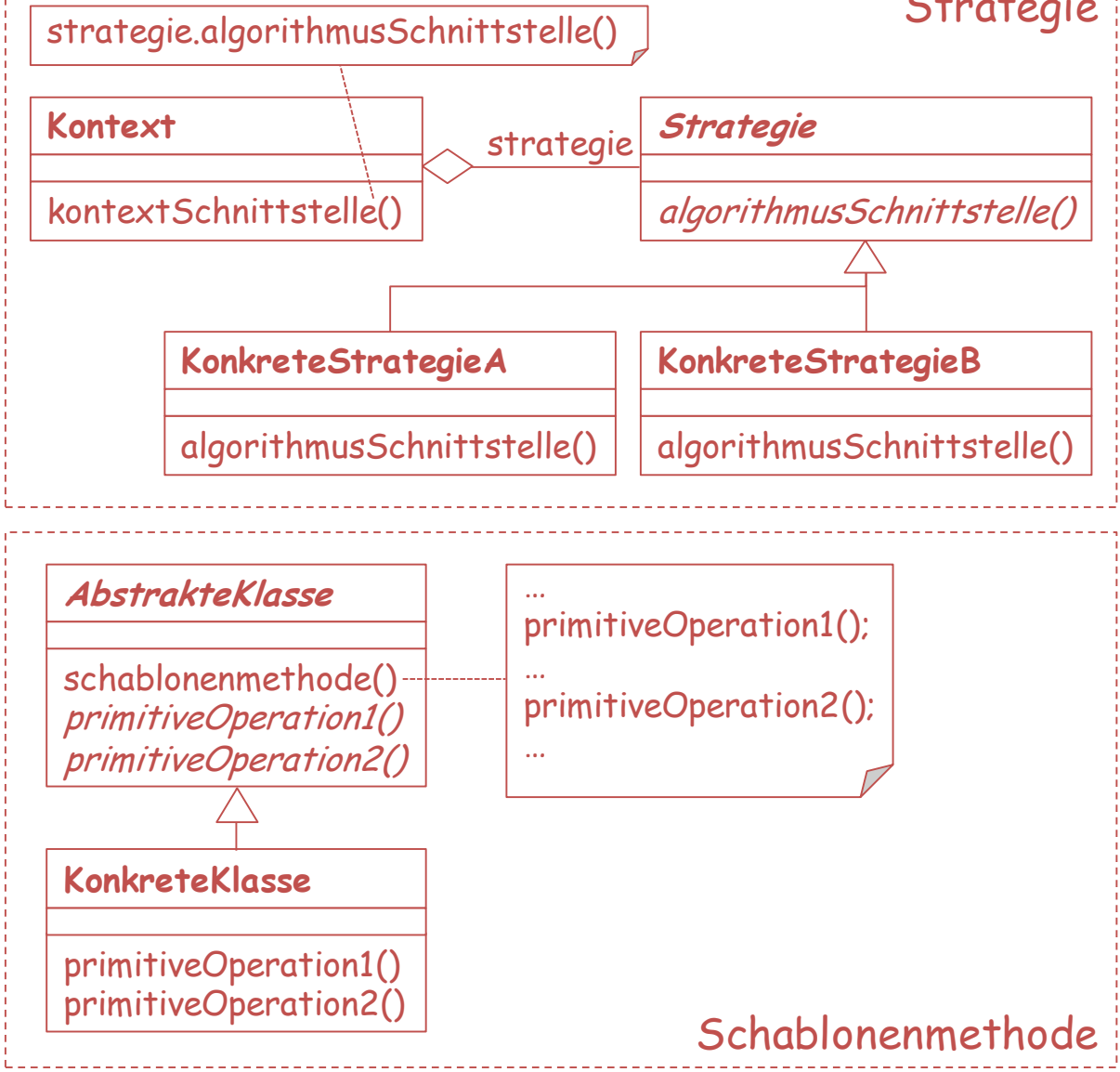
- Funktionale Anforderungen: 0,5 P
  Beschreiben die Interaktionen zwischen dem System und der Systemumgebung, unabhängig von der Implementierung. 0,5 P
- Nichtfunktionale Anforderungen: 0,5 P
  Aspekte, die nicht direkt mit dem funktionalen Verhalten des Systems in Verbindung stehen. 0,5 P
- Einschränkungen: 0,5 P
  Sind durch den Kunden oder die Umgebung vorgegeben 0,5 P

d) Erklären Sie was passiert, wenn in Java ein Faden `f1` versucht, einen bereits von einem anderen Faden `f2` besetzten Monitor zu betreten. Wieso gibt es keine Methode `wouldBlock(object)`, die überprüft, ob der Faden bei der Monitoranforderung blockiert? Welche alternative Methode bietet die Klasse `java.lang.Thread` an? Was tut diese Methode? (3 P)

Der Faden f1 wird ununterbrechbar blockiert (0,5 P), bis der Monitor durch den anderen Faden f2 freigegeben wird (0,5 P). 1 P
Zwischen Test (wouldBlock) und Aktion danach kann sich die Situation wieder geändert haben: Monitor ist wieder freigegeben bzw. von einem anderen Faden belegt worden. 1 P
java.lang.Thread.holdsLock(Object): Prüft, ob der aufrufende Faden den angegebenen Monitor hält. 1 P

# AUFGABE 2: ENTWURFSMUSTER (8 P)

a) Zeichnen Sie die Struktur der zwei Entwurfsmuster Strategie und Schablonenmethode. Kennzeichnen Sie die Entwurfsmuster eindeutig. Unterscheiden Sie deutlich zwischen konkreten und abstrakten Klassen/Methoden. (4 P)




Strategie: 0,5 P pro korrekter Klasse; 0,5 P für die Aggregation.
Die Notiz ist optional.
Schablonenmethode: 0,5 P pro korrekter Klasse;
0,5 P für die Notiz.

b) Zu welcher/welchen Kategorie(n) gehören diese Entwurfsmuster laut Vorlesung? (1 P)

Beides sind Variantenmuster (nach Gamma: Verhaltensmuster).
0,5 P pro richtiger Zuordnung

c) Welches gemeinsame Ziel haben die beiden Entwurfsmuster? Inwiefern unterscheiden sie sich beim Erreichen dieses Ziels? (2 P)

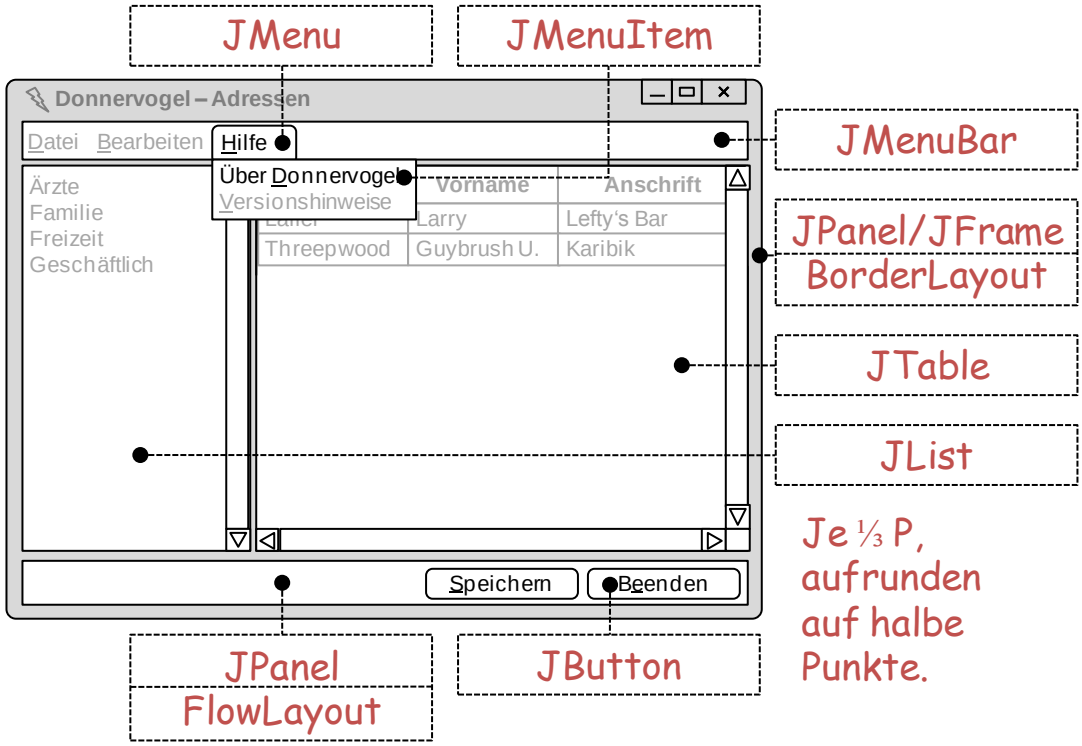
Beide Entwurfsmuster erlauben es, Varianten eines Algorithmus zu verwenden (1P). Die Schablonenmethode verwendet Vererbung, um Teile eines Algorithmus zu variieren, wohingegen die Strategie über Delegation den gesamten Algorithmus austauschbar macht (1P).

d) Nennen Sie genau zwei Gründe, die laut Vorlesung für den Einsatz von Entwurfsmustern sprechen. (1 P)

Für jede korrekte Nennung: 0,5 P, max. 1 P Muster...
verbessern die Kommunikation im Team/erfassen wesentliche Konzepte und bringen sie in eine verständliche Form/helfen Entwürfe zu verstehen/dokumentieren Entwürfe kurz und knapp/verhindern unerwünschte Architektur-Drifts/verdeutlichen Entwurfswissen/dokumentieren und fördern den Stand der Kunst/helfen weniger erfahrenen Entwerfern/vermeiden die Neuerfindung des Rades/können Code-Qualität und Code-Struktur verbessern/fördern gute Entwürfe und guten Code durch Angabe konstruktiver Bsp.

# AUFGABE 3: SWING (10 P)

Gegeben sei folgende Skizze eines Fensters einer Java-Anwendung:




a) Schreiben Sie in die gestrichelten Kästchen, um welche konkreten Swing-Benutzeroberflächenelemente es sich handelt. In die größeren, geteilten Kästchen tragen Sie bitte auch den verwendeten `LayoutManager` ein. (3 P)

b) Die Schnittstelle `ActionListener` bietet genau eine öffentliche Methode `actionPerformed(ActionEvent e)` an. Ergänzen Sie den folgenden Quelltext durch eine `ActionListener`-Implementierung, so dass beim Drücken des Knopfes `AusKnopf` das Programm beendet wird. (2 P)

```
import javax.swing.JButton;
import java.awt.event.ActionListener;

import java.awt.event.ActionEvent; 0,5 P

public class AusKnopf extends JButton /* implements ActionListener */{

  public AusKnopf() {
    super("Beenden.");
    addActionListener(
      new ActionListener() { 0,5 P
            @Override
            public void actionPerformed(ActionEvent e) { 0,5 P
                  System.exit(0);  0,5 P
            }
      }



    );
  }
/*
    @Override
    public void actionPerformed(ActionEvent e) {
      System.exit(0);
    }
*/

}
```

c) Zur `MouseListener`-Schnittstelle gibt es die entsprechende Adapter-Klasse `MouseAdapter`. Welchen Zweck hat diese Adapter-Klasse und warum gibt es zum `ActionListener` keine entsprechende Adapter-Klasse? (2 P)

Der MouseAdapter besitzt für alle (fünf) Methoden der MouseListener-Schnittstelle eine Leerimplementierung (0,5 P). Erbende Klassen müssen also nur die Methode überschreiben, die wirklich genutzt werden sollen (0,5 P). Die ActionListener-Schnittstelle definiert nur eine Methode, daher macht ein Adapter keinen Sinn (1 P).

d) Erklären Sie den Unterschied zwischen leicht- und schwergewichtigen Komponenten in grafischen Benutzeroberflächen. Welches Problem ergibt sich bei leichtgewichtigen, welches bei schwergewichtigen Komponenten? (3 P)

Leichtgewichtige Komponenten sind im Gegensatz zu schwergewichtigen nicht an plattformabhängige Komponenten gebunden; müssen aber letztendlich auch auf plattformabhängige Komponenten gezeichnet werden (1P). Problem bei leichtgewichtigen Komponenten: Aussehen der Zielplattform zu emulieren ist aufwändig (1P). Problem bei schwergewichtigen Komponenten: Auf Plattform nicht angebotene Komponenten müssen „von Hand" nachgebaut werden (1P).

## AUFGABE 4: UML-ZUSTANDSAUTOMATEN (13 P)

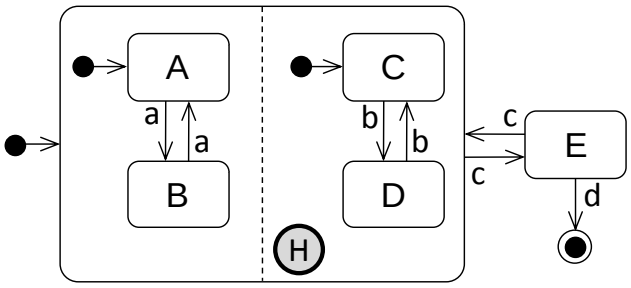
a) Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet. (1 P)

a ,b, c, c

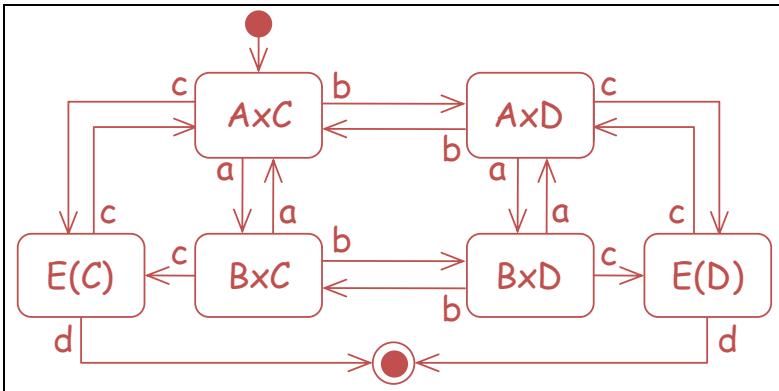
A×D
0,5P

c, c, a, b, b, a, c, c, a

B×C
0,5P




b) Wandeln Sie den UML-Zustandsautomaten aus Teil a) in einen äquivalenten neuen UML-Zustandsautomaten um, der weder nebenläufige noch hierarchische Zustände oder Zustände mit Historie enthält. Leiten Sie die Namen für die Zustände in Ihrem neuen UML-Zustandsautomaten wie folgt aus den Namen der Zustände des alten UML-Zustandsautomaten ab:
Regel 1: Die Kombination der alten Zustände 1 und 2 wird zum neuen Zustand 1×2.
Regel 2: Wurde der alte Zustand 1 vom alten Zustand 2 aus erreicht, ergibt dies den neuen Zustand 1(2). (5 P)

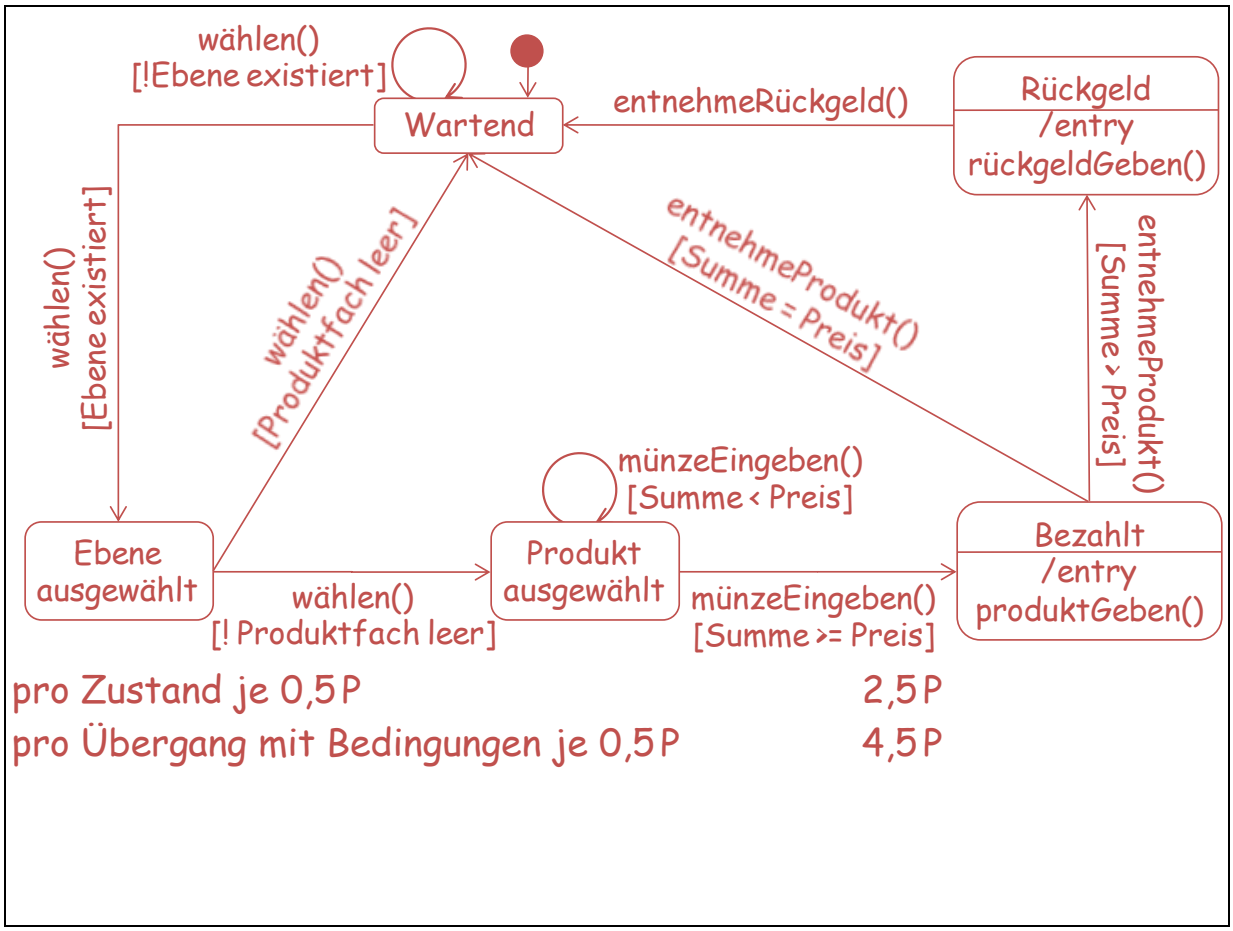



Zustand E richtig geteilt: 0,5P; Startzustand korrekt: 0,5P
Richtige Kombinationen aus den Zuständen A, B, C, D: 4 × 0,5P
Übergänge a, b, c, d korrekt: 4 × 0,5P

c) 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. (7 P)

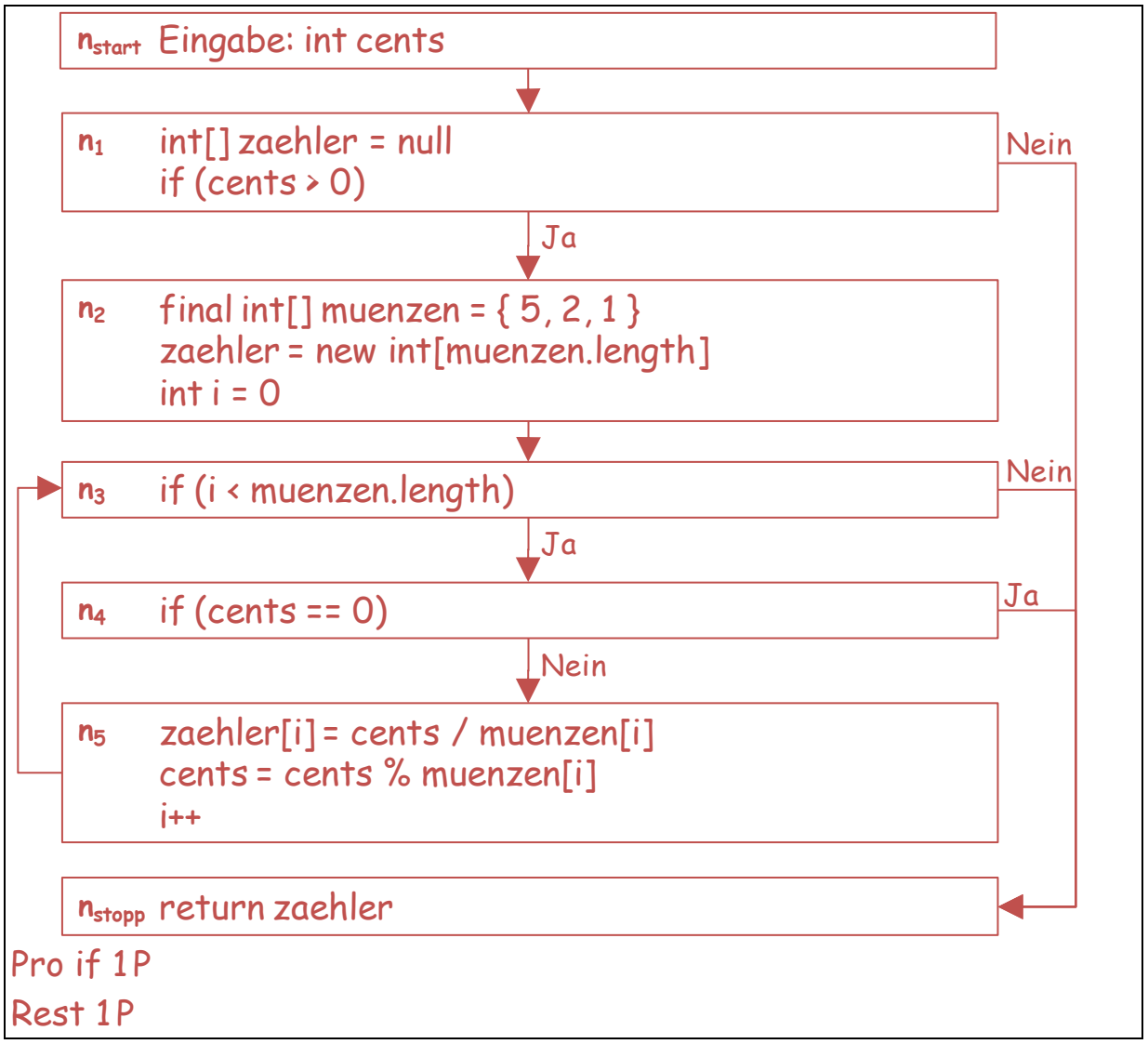


# AUFGABE 5: KONTROLLFLUSSORIENTIERTES TESTEN (11 P)

Gegeben sei folgende Java-Methode:

```
01  public static int[] wechselgeld(int cents) {
02    int[] zaehler = null;
03    if (cents > 0) {
04      final int[] muenzen = { 5, 2, 1 }; // Cent-Münzen
05      zaehler = new int[muenzen.length];
06      for (int i = 0; i < muenzen.length; i++) {
07        if (cents == 0) {
08          break;
09        }
10        zaehler[i] = cents / muenzen[i]; // Ganzzahldivision!
11        cents = cents % muenzen[i];
12      }
13    }
14    return zaehler;
15  }
```

a) Erstellen Sie den Kontrollflussgraphen der Methode `wechselgeld(…)`. Bitte schreiben Sie den Quelltext in die Kästchen, Verweise auf die Zeilennummern der Methode sind nicht ausreichend. (4 P)



b) Geben Sie eine minimale Testfallmenge an, welche die Anweisungsüberdeckung der Methode `wechselgeld(…)` erfüllt. Geben Sie die durchlaufenen Pfade an. (1 P)

{5}: $n_{start}$, $n_1$, $n_2$, $n_3$, $n_4$, $n_5$, $n_3$, $n_4$, $n_{stopp}$ 1 P

c) Ergänzen Sie die Testfallmenge aus b) so, dass Sie eine minimale Testfallmenge erhalten, welche die Zweigüberdeckung der Methode `wechselgeld(…)` erfüllt. Geben Sie für die neuen Testfälle die durchlaufenen Pfade an (2 P)

{5}: s. o.
{0}: $n_{start}$, $n_1$, $n_{stopp}$ 1 P
{1}: $n_{start}$, $n_1$, $n_2$, 3 × ($n_3$, $n_4$, $n_5$,) $n_3$, $n_{stopp}$ 1 P

d) Erfüllt die minimale Testfallmenge aus c), welche die Zweigüberdeckung erfüllt, auch die Pfadüberdeckung? Begründen Sie Ihre Antwort. (1 P)

Nein. Es fehlt z. B. die {2}. 1 P

e) Nehmen Sie an, die `if`-Bedingung aus den Zeilen 7 – 9 sei nun wie folgt in den Schleifenkopf integriert:

```
06    for (int i = 0; i < muenzen.length && cents != 0; i++) { …
```

Ändert sich nun die minimale Testfallmenge für die Zweigüberdeckung? Begründen Sie Ihre Antwort. (1 P)

Ja. Man kann entweder die {1} oder die {5} weglassen, da der Pfad $n_4$, $n_{stopp}$ entfällt. 1 P

f) Bewerten Sie folgende Aussage:

> *„Gegeben sei eine minimale Testfallmenge, welche die Pfadüberdeckung für eine Methode erfüllt. Laufen alle Tests aus dieser Menge erfolgreich, so ist die Korrektheit der Methode garantiert.“.*

Begründen Sie: Ist diese Aussage korrekt? Geben Sie ein Beispiel an, das Ihre Bewertung belegt. (2 P)

Nein. 1 P
Begründung/Gegenbeispiel: 1 P

```
public static void foo(boolean a, boolean b) {
   if (a || (b && bar())) { System.out.println("Bla"); }
}
public static boolean bar() {
   while (true) { System.out.println("Blub"); }
}
```

Minimale Testfallmenge für foo():
{a = wahr, b = falsch; a = falsch, b = falsch}.
Bei {a = falsch, b = wahr} läuft das Programm unendlich.

# AUFGABE 6: AKTIVITÄTSDIAGRAMM (6 P)

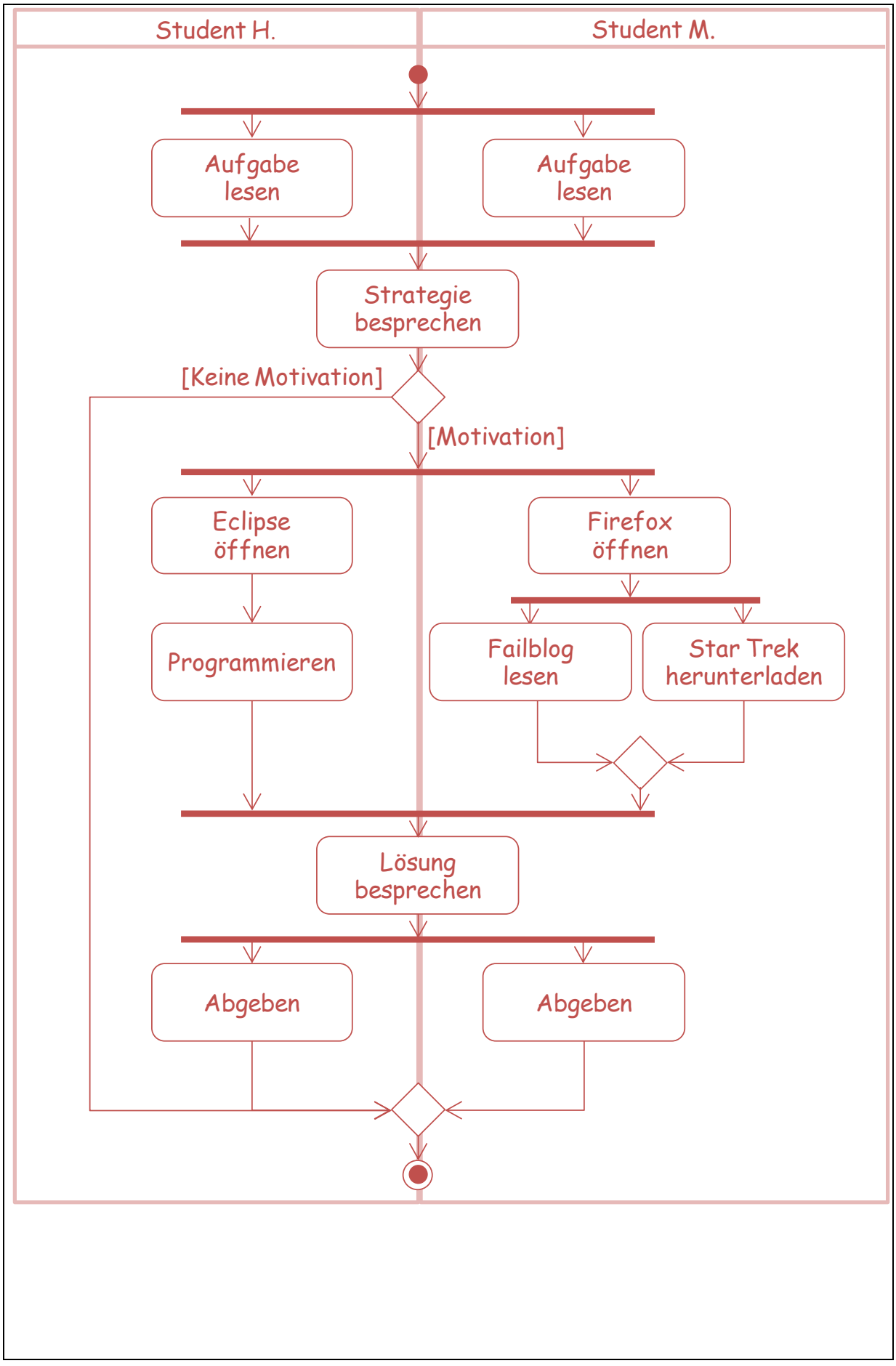
Gegeben sei folgendes Szenario, welches das Vorgehen der Studenten H. und M. bei der Lösung einer SWT-1-Programmieraufgabe beschreibt:

> *Zunächst lesen die Studenten H. und M. jeder für sich gleichzeitig die Aufgabenstellung der aktuellen Programmieraufgabe durch. Nachdem beide Studenten den Aufgabentext gelesen haben, besprechen sie gemeinsam die Strategie für das weitere Vorgehen. Sind die beiden Studenten nicht motiviert, die Aufgabe zu lösen, hören sie sofort auf. Haben sie Motivation, dann öffnet H. Eclipse und beginnt danach unmittelbar zu programmieren. Während H. Eclipse startet, öffnet M. Firefox und beginnt, den aktuellen Star Trek Film herunter zu laden und unterdessen den Failblog zu lesen. Sobald H. die Lösung fertig programmiert hat und M. den Film fertig heruntergeladen oder genug im Failblog gelesen hat, besprechen beide Studenten gemeinsam die Lösung. Danach geben H. und M. die Lösung getrennt voneinander ab, worauf die Programmieraufgabe für sie beendet ist.*

Modellieren Sie das gegebene Szenario als UML-Aktivitätsdiagramm. Kennzeichnen Sie, welche Aktivitäten von H., welche von M. und welche von beiden ausgeführt werden. Objektflüsse müssen Sie nicht modellieren. (6 P)

Je Listenpunkt 0,5 P

a. H & M.: Getrennt Aufgabe lesen
b. H. & M.: Strategie besprechen (inkl. Synchronisation)
c. H. & M.: Abzweigung „Motivation"
d. H. & M.: Abzweigung „keine Motivation"
e. H.: Eclipse starten (parallel zu g)
f. H.: Programmieren (parallel zu h, i)
g. M.: Firefox starten (parallel zu e)
h. M.: Star Trek herunterladen (parallel zu f, i)
i. M.: Failblog lesen (parallel zu f, h)
j. H. & M.: Lösung besprechen (inkl. Synchronisation)
k. H. & M.: Getrennt abgeben
l. Start-/Endzustand


Student H.
Student M.
Aufgabe lesen
Aufgabe lesen
Strategie besprechen
[Keine Motivation]
[Motivation]
Eclipse öffnen
Firefox öffnen
Programmieren
Failblog lesen
Star Trek herunterladen
Lösung besprechen
Abgeben
Abgeben