Klausur Softwaretechnik

06. August 2003

Musterlösung

Aufg.0	Aufg.1	Aufg.2	Aufg.3	Aufg.4	Aufg.5	Σ	Note
1 /1	19 /19	8 /8	9 /9	12 /12	11 /11	60 /60	1

Aufgabe 0 (1 Punkt)

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

Aufgabe 1 (19 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) Warum ist speziell Software so schwer zu entwickeln? Nennen Sie zwei Gründe.

- immaterielles Produkt
- nicht durch physikalische Gesetze begrenzt
- leicht änderbar
- Software altert
- Software ist schwer zu vermessen

b) Nennen Sie drei Prinzipien der Software-Technik, mit denen versucht wird, die zunehmende Komplexität bei der Software-Entwicklung zu beherrschen.

- Hierarchisierung
- Modularisierung
- Strukturierung
- Abstraktion

c) Nennen Sie drei Phasen der Software-Entwicklung und jeweils ein Software-Dokument, das in diesen Phasen erstellt wird.

- Planungsphase – Lastenheft
- Entwurfsphase – Pflichtenheft
- Implementierungsphase – Quelltexte, Dokumentation

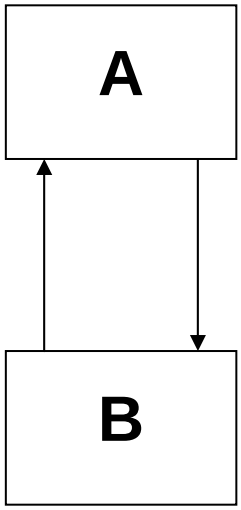
d) Nennen Sie die vier Bearbeitungszustände eines Programms nach dem V-Modell.

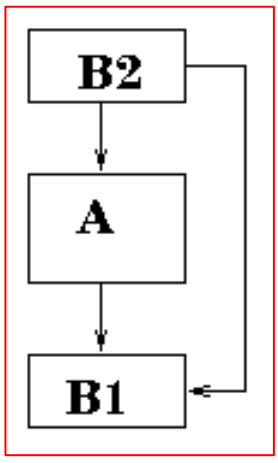
- Geplant
- In Bearbeitung
- Vorgelegt
- Akzeptiert

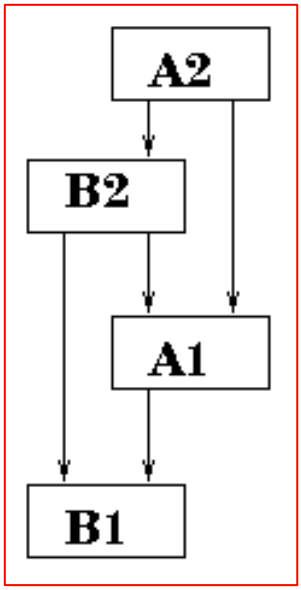
e) Bei der Modularisierung wird ein Programm nach dem Geheimnisprinzip in einzelne Module zerlegt. Nennen Sie die drei möglichen Vorgehensweisen mit denen das Programm beim Entwurf zerlegt wird.

- Schrittweise Verfeinerung (top-down)
- Schrittweiser Aufbau (bottom-up)
- Entwurf von der Mitte her (middle-out, yo-yo)

f) Lösen Sie die folgende gegenseitige Benutzung der Module A und B durch *Sandwiching* auf. Benennen Sie die entstehenden Module A1, A2,... bzw. B1, B2,... .



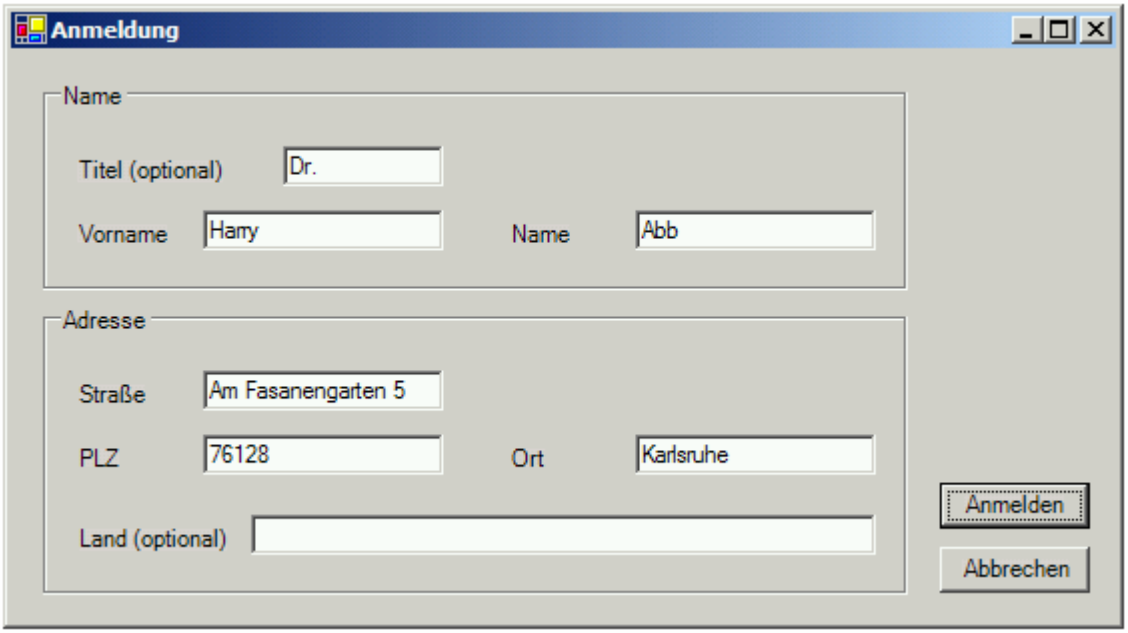




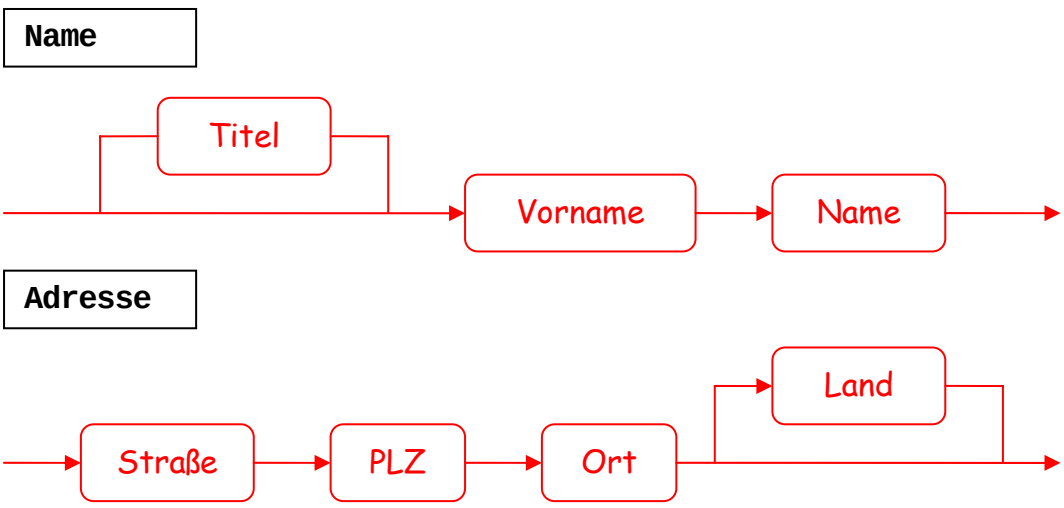
g) Welche drei Entwurfsmuster werden im Model-View-Controller, einer Klassenkombination zur Konstruktion von Benutzerschnittstellen, kombiniert?

- Beobachter
- Kompositum
- Strategie

h) Die folgende Abbildung zeigt ein Anmeldeformular einer größeren Anwendung.



Erstellen Sie aus den o.a. dargestellten Eingabelementen ein Syntaxdiagramm.



Name: _____ Matrikelnummer: _____

i) Entwurfsmuster sind für das Programmieren im Großen,
was Algorithmen für das Programmieren im Kleinen sind.

j) Sie haben in ihrem Java-Programm eine Methode geschrieben, die eine **Messagebox** mit einer Nachricht an den Benutzer anzeigt. Solch eine Meldung könnte aussehen wie folgt.

Der Aufruf der statischen Methode **show()** für das gegebene Beispiel benötigt zusätzlich zu der eigentlichen Benutzernachricht noch zwei weitere Parameter. Der Parameter **strCaption** enthält die Überschrift der **Messagebox**. Der Parameter b**uttons** gibt die Anzahl und Art der Bestätigungs-Schaltflächen an (z.B. nur ein OK-Knopf oder ein Ja- und ein Nein-Knopf).

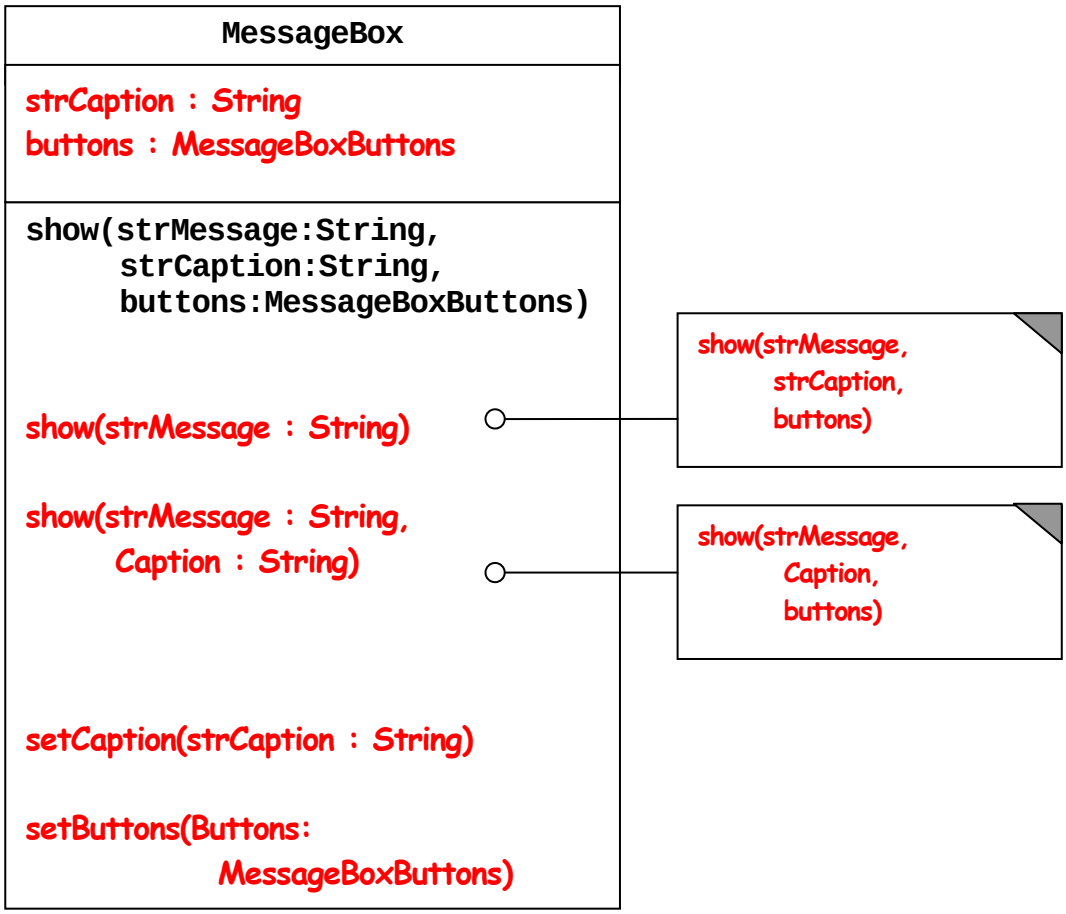
```
class MessageBox
{
    public static show(String strMessage, String strCaption,
                       MessageBoxButtons buttons)
    {
        // …
    }
}
```

```
MessageBox.show("Hello World!", "Eine neue Nachricht", MessageBoxButtons.OK);
```

- Aus Bequemlichkeitsgründen – sie möchten nicht jedes Mal die Überschrift und den Button-Text angeben müssen – verwenden Sie das Entwurfsmuster *Bequemlichkeitsmethode*. Geben Sie in dem nachfolgenden UML-Klassendiagramm die entsprechenden Funktionsdeklarationen (noch nicht die Implementierungen!) in der Klasse **MessageBox** an, so dass folgende Aufrufe möglich sind:

  i. Wenn **show()** nur mit dem Nachrichtentext als einzigem Parameter aufgerufen wird, so soll die Überschrift und der OK-Button standardmäßig so erscheinen wie in der obigen Abbildung.
  ii. Wenn **show()** nur mit dem Nachrichtentext und der Überschrift als Parameter aufgerufen wird, so soll wie in der obigen Abbildung standardmäßig ein OK-Button angezeigt werden.

- Verwenden Sie nun das Entwurfsmuster *Bequemlichkeitsklasse*. Erweitern Sie dazu die Klasse **MessageBox**, so dass Sie Einstellungen für die Überschrift und die darzustellenden Buttons jeweils als Instanz-Variablen in der Klasse **MessageBox** gespeichert werden. Definieren Sie auch Methoden zum Setzen der Variablen.



- Geben Sie in dem obigen UML-Klassendiagramm die Implementierungen Ihrer Bequemlichkeitsmethoden an.

k) Gegeben ist ein direkt abgebildeter Cache. Er ist 32 Byte groß und hat 4 Cachezeilen. Jede Cachezeile besteht aus 8 Byte. Der Datentyp **int** besteht aus 2, der Datentyp **float** aus 4 Byte. Somit passen in jede Cachezeile immer genau 4 bzw. 2 Datenelemente des Typs **int** bzw. **float**. Die folgende Zeichnung veranschaulicht noch einmal den Aufbau des Caches. Die kleinen Kästchen entsprechen einem Byte.

Cachezeile								
0	B	B	B	B				
1					A	A		
2								
3								

Gegeben ist weiterhin der folgende Ausschnitt aus einem C-Programm:

```
10   int a[64];                        // Adresse (a[0]) = 8;
     ...
20   float b[64];                      // Adresse (b[0]) mod 32 = 16;
     ...
30   for (int i=0; i<64; i++) {
40     a[i] = b[i];
50   }
```

Name: _____ Matrikelnummer: _____

Beantworten Sie die folgenden Fragen:

i. Wie viele Bytes werden vom Hauptspeicher in den Cache übertragen, wenn der Wert a[0] in den Cache geladen wird? 8 (eine Cachezeile)

ii. An welche Stelle im Cache wird das Element a[2] geladen? Markieren Sie die entsprechenden Cachepositionen mit einem großen „A“.

iii. An welche Stelle im Cache wird das Element b[4] geladen? Markieren Sie die entsprechenden Cachepositionen mit einem großen „B“.

l) Sie möchten sich gerne das Video „Business“ von Eminem ansehen und haben dazu die folgenden Entscheidungstabellen aufgestellt.

ET 1		R1	R2	R3
B1	Persönliches Guthaben > 20 Euro.	J	N	N
B2	Video auf Kazaa verfügbar.	-	J	J
B3	Downloadgeschwindigkeit schnell genug.	-	N	J
A1	Kaufe DVD bei Amazon.com	X		
A2	Starte Download bei Kazaa			X
	Weiter bei ET		ET 2	

ET 2		R1	R2	R3	R4
B1	Video läuft auf M-TV.	N	J	N	J
B2	Video läuft auf Viva.	N	N	J	J
A1	Schaue M-TV.		X		X
A2	Schaue Viva.			X	
A3	Gehe auf das nächste Eminem-Konzert.	X			

Beantworten Sie die folgenden Fragen...

Richtig/Falsch

□/X ... Entscheidungstabelle ET1 ist formal vollständig.
□/X ... Entscheidungstabelle ET1 ist inhaltlich vollständig.
□/X ... Entscheidungstabelle ET1 ist eine erweiterte ET.
X/□ ... Entscheidungstabelle ET2 ist formal vollständig.
X/□ ... Entscheidungstabelle ET2 ist inhaltlich vollständig.
□/X ... Entscheidungstabelle ET2 ist eine erweiterte ET.

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

Richtig/Falsch

□/ X Beim selbstkontrollierten Programmieren protokolliert man die konkrete Verwendung von Entwurfsmustern um den Quelltext besser verständlich zu machen.

X /□ Die Reduktion von Speicherbedarf kann ein schnelleres Program erzeugen.

X /□ Bei der substituierten Verfeinerung werden Verfeinerungsschichten als Kommentare in den Quelltext geschrieben.

□/ X Während der Implementierungsphase wird aus dem Pflichtenheft die Systemarchitektur entwickelt.

□/ X Eine Übersetzungseinheit in einem Programm wird auch als Modul bezeichnet.

X /□ Ein Modul kann mehrere Ebenen in der Benutzthierarchie umfassen.

□/ X Wenn ein Datenflußdiagramm zu unübersichtlich wird kann man zwischen Speichern auch direkte Datenflüsse einzeichnen.

Name: _____ Matrikelnummer: _____

Aufgabe 2 (Optimierung und Entwurfsmuster) **(8 Punkte)**

In dieser Aufgabe wird die Suche in einem Binärbaum betrachtet. Die Knoten des Baumes sind Objekte, die die Schnittstelle **`Tree`** erfüllen und außerdem folgende weitere Eigenschaften haben:

1. In jedem Knoten (**`Tree`**) wird der Suchschlüssel (vom Typ **`int`**) und der zugehörige Datenwert (vom Typ **`Object`**, abzufragen mittels **`getData`**) gespeichert. Jeder Schlüsselwert tritt nur einmal auf.

2. Jeder Knoten kann einen linken und einen rechten Nachfolger haben. Es gilt, dass sich alle Datensätze mit kleineren Schlüsselwerten im linken Unterbaum befinden und alle Datensätze im rechten Unterbaum größere Schlüsselwerte haben. Hat ein Knoten keinen rechten/linken Nachfolger, so wird von den Funktionen **`getRightChild`**/**`getLeftChild`** der Wert **`null`** zurückgegeben.

3. Die Methode **`compareTo(key)`** vergleicht einen Schlüsselwert **`key`** mit dem des Objekts. Sie gibt einen negativen Wert, Null oder einen positiven Wert zurück, je nachdem ob der Schlüssel des Objekts kleiner, gleich oder größer als der übergebene Schlüssel ist.

```
interface Tree {
   public int compareTo(int key);
   public Object getData();
   public Tree getLeftChild();
   public Tree getRightChild();
}
```

Die folgende rekursive Funktion **`search1`** wird zur Suche in den oben beschriebenen Bäumen verwendet. Sie durchsucht einen Baum nach einem Schlüssel und gibt die gefundenen Daten oder **`null`** (falls kein passender Schlüssel gefunden wurde) zurück. Da in der gegebenen Datenstruktur sehr häufig gesucht wird, muss die Suche möglichst effizient ablaufen. In dieser Aufgabe sollen nachfolgend zwei mögliche Optimierungen durchgeführt werden.

```
public Object search1(Tree tree, int searchKey) {
   if ((tree == null) || (tree.compareTo(searchKey) == 0)) {
      return (tree == null) ? null : tree.getData();
   }
   else {
      Tree next = (tree.compareTo(searchKey) < 0) ?
                  tree.getRightChild() :
                  tree.getLeftChild();

      return search1(next, searchKey);
   }
}
```

a) Zur Einführung eines *Wächterelements* wird die obige 2. Eigenschaft des Baumes leicht verändert: „Hat ein Knoten keinen rechten/linken Nachfolger, so wird von den Funktionen **getRightChild**/**getLeftChild** das Wächterelement vom Typ **Guard** zurückgegeben.“ Mit Hilfe des Wächterelements lässt sich die Suche dann wie folgt vereinfachen, so dass der Vergleich gegen **null** wegfällt:

```
public Object search2(Tree tree, int searchKey) {
   if (tree.compareTo(searchKey) == 0) {
      return tree.getData();
   }
   else {
      Tree next = (tree.compareTo(searchKey) < 0) ?
                  tree.getRightChild() :
                  tree.getLeftChild();

      return search2(next, searchKey);
   }
}
```

Vervollständigen Sie die Klasse **Guard**, so dass die Funktion **search2** wie gewünscht funktioniert und die Klasse **Guard** das Entwurfsmuster *Einzelstück* (*Singleton*) implementiert.

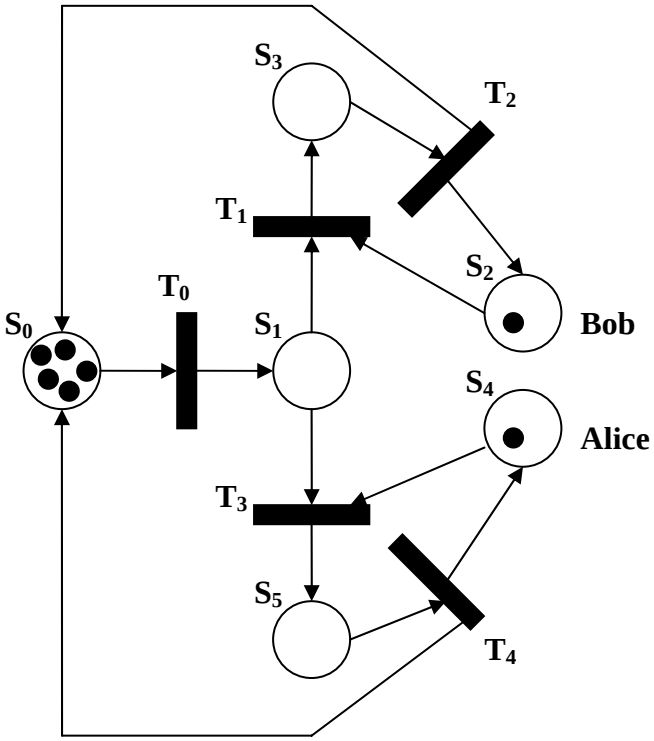
```
public class Guard implements Tree {
  private Guard instance = null;
  private Guard() {}
  public static Guard getInstance() {

     if (instance == null) { instance = new Guard() };

     return instance;
  }

  public int compareTo(int key) {

     return 0;
  }
     public Object getData() {

     return null;
  }
     public Tree getLeftChild() {

     return instance;
  }
     public Tree getRightChild() {

     return instance;
  }
}
```

b) Die Methode **search2** ist rechtsrekursiv. Wandeln Sie diese Funktion mittels des aus der Vorlesung bekannten Verfahrens in eine iterative Form **search3** um.

```
public Object search3(Tree tree, int searchKey) {
   Tree tree1 = tree;
   while (tree1.compareTo(searchKey) != 0) {
      tree1 = (tree1.compareTo(searchKey) < 0) ?
      tree.getRightChild() :
      tree.getLeftChild();
   }
   return tree1.getData();
}
```

Aufgabe 3 (Petrinetze) **(9 Punkte)**

Um die gelegentlichen Beschwerden von Konsumenten entgegenzunehmen, betreibt die Firma *MacroHard Corp.* ein Kundenzentrum mit zwei Telefon-Sachbearbeitern. Das folgende Petrinetz *N* zeigt die Situation, dass sich fünf Kunden in der Warteschlange befinden. Zwei Sachbearbeiter nehmen, wenn sie „frei“ sind, einen Anruf entgegen. Da Bob und Alice jedoch keine besonders gute Arbeit machen, reihen sich die Kunden nach Ende eines Telefongesprächs gleich wieder in die Warteschlage ein, in der Hoffnung, dass Ihnen beim nächsten Gespräch geholfen wird.



a) Stellen Sie für das Petrinetz *N* die Übergangsmatrix auf.

$$C = \begin{bmatrix} -1 & 0 & 1 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

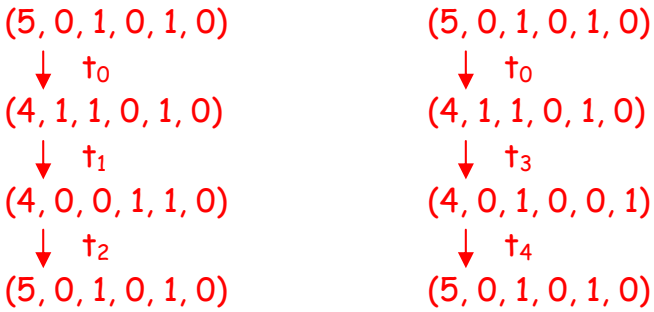
Name: _____ Matrikelnummer: _____

b) Geben Sie die elementaren Lösungen der T-Invarianten an.

$d_1 = (1, 1, 1, 0, 0)$

$d_2 = (1, 0, 0, 1, 1)$

c) Erstellen Sie für jede elementare Lösung der T-Invarianten den partiellen Erreichbarkeitsgraphen von der Anfangsmarkierung $M_N = (5, 0, 1, 0, 1, 0)$.



d) Ist das Petrinetz *N* lebendig? Begründen Sie kurz Ihre Antwort.

Ja, das Petrinetz N ist lebendig.

1. T-Invarianten überdecken alle Transitionen des Pertrinetzes N.

2. Erreichbarkeitsgraph: Jeder Präfix der Schaltfolgen bei denen die T-Invarianten ausgeführt werden besitzt eine Fortsetzung, die M_N wieder erzeugt.

e) Ist das Petrinetz *N* beschränkt? Geben Sie die Bedingung für die Beschränktheit eines S/T-Netzes im Allgemeinen und die konkrete Schranke *b* für das Netz *N* an.

Das Petrinetz N ist beschränkt. $\forall s \in S: M(s) \leq b$ $b = 5$

f) Wieviele Zustände kann das Petrinetz N höchstens annehmen?

Das Petrinetz N kann höchstens $(b+1)^{\text{Anzahl der Stellen}}$ Zustände annehmen. In diesem Fall: 20

g) Warum ist Beschränktheit wichtig bei der Implementierung von Petrinetzen?

Kein Überlauf von Stellen.

this page intentionally left blank

Name: _____ Matrikelnummer: _____

Aufgabe 4 (Entwurfsmuster, Sequenzdiagramm) **(12 Punkte)**

In dieser Aufgabe soll der Kommunikationsteil einer mobilen Anwendung, die auf einem tragbaren Computer läuft, entworfen werden. Der tragbare Computer kann je nach Verfügbarkeit eines Netzes verschiedene Protokolle zur Kommunikation nutzen. Zum Beispiel könnte ein KFZ-Mechaniker in einer Werkstatt ein drahtloses Netzwerk (WLAN) nutzen um sich Benutzerhandbücher und Reparaturanweisungen online anzusehen. Unterwegs auf einer Landstraße könnte die Verbindung über ein Mobilfunknetz (UMTS) hergestellt werden. Zu Konfigurationszwecken wie z.B. dem Installieren von Updates auf dem tragbaren Computer soll auch die die Kommunikation via Ethernet möglich sein.

Wesentliche Anforderungen an die mobile **`Anwendung`** sind also das dynamische Umschalten zwischen verschiedenen Netzwerkverbindungen. Darüberhinaus soll die **`Anwendung`** auch mit zukünftigen Protokollen umgehen können ohne neu übersetzt werden zu müssen.

Auf der nächsten Seite ist ein unvollständiges UML-Klassendiagramm gegeben. Die Klasse **`Anwendung`** repräsentiert die eigentlich Anwendungslogik. Sie verwendet die Methoden **`sende()`** und **`empfange()`** der Klasse **`Netzwerkverbindung`** zum protokollunabhängigen Senden und Empfangen von Daten. Die Methode **`ortsWechsel()`** in der Klasse **`LocationManager`** wird immer dann aufgerufen, wenn sich der Kontext der Anwendung ändert.

a) Erweitern Sie das Klassendiagramm unter Verwendung des Entwurfsmusters *Strategie* um eine abstrakte Klasse **`NetzwerkSchnittstelle`** mit den Methoden **`oeffne()`**, **`schliesse()`**, **`sende()`**, **`empfange()`**.

b) Erweitern Sie das Klassendiagramm um drei von **`NetzwerkSchnittstelle`** abgeleitete konkrete Klassen für die drei Netztypen Ethernet, WLAN und UMTS.

c) Erweitern Sie die Klasse **`NetzwerkVerbindung`** um eine Methode **`setzeNetzwerkSchnittstelle(ntzSch : NetzwerkSchnittstelle)`**, die die bestehende Verbindung schließt und eine neue Verbindung unter Verwendung der neuen **`NetzwerkSchnittstelle`** öffnet.

d) Vervollständigen Sie in dem Klassendiagramm die Implementierung der Methode **`ortsWechsel()`** in der Klasse **`LocationManager`**.

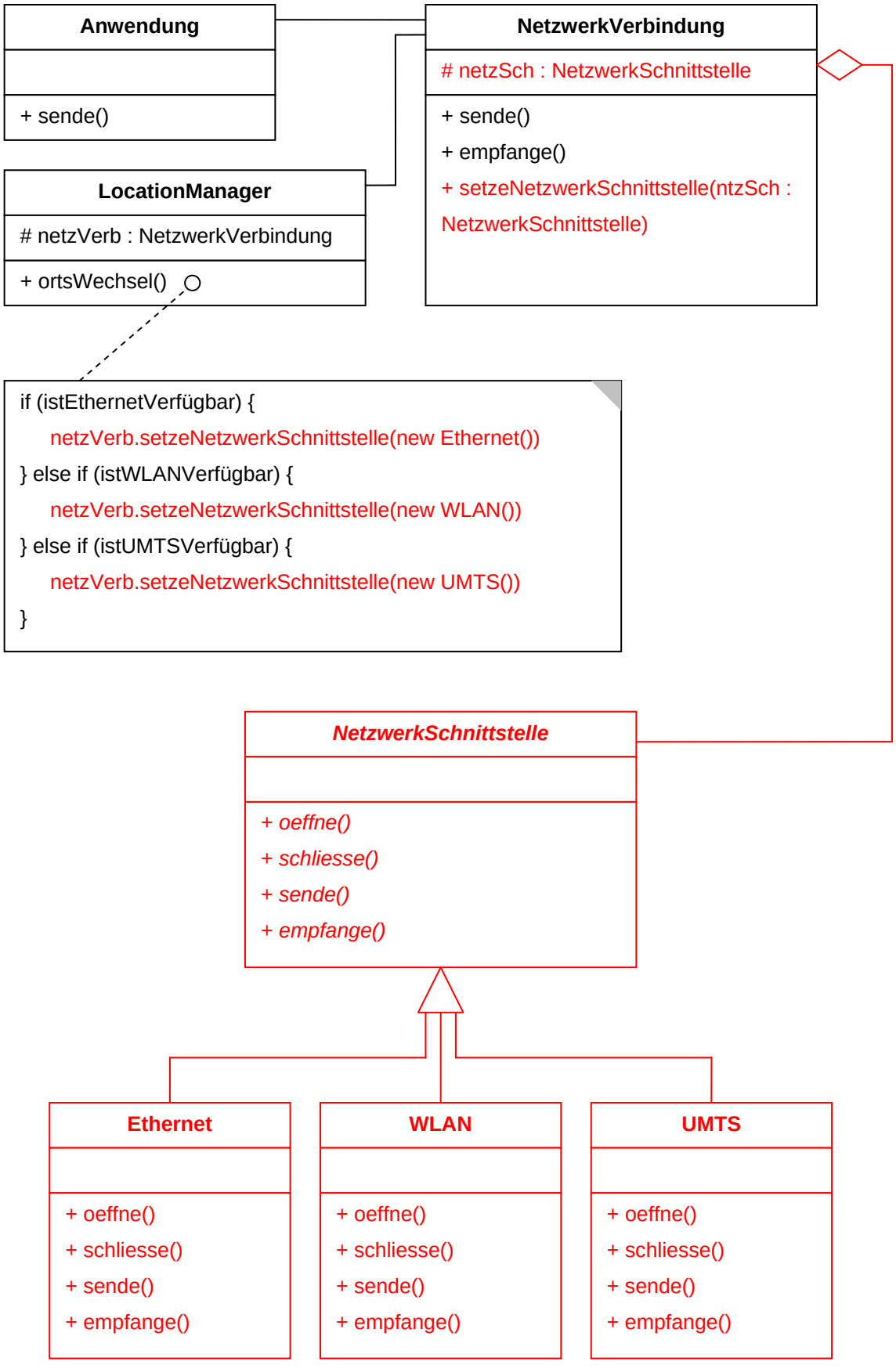
e) Eine Alternative zur Verwendung des Entwurfsmusters *Strategie* wäre, dass man protokollabhängige Verbindungsklassen einfach von der Klasse **`NetzwerkVerbindung`** ableitet.

   Was wäre der Nachteil dieses Vorgehens?

   Man hätte eine Menge von prokollabhängigen aber sonst identischen Klassen.

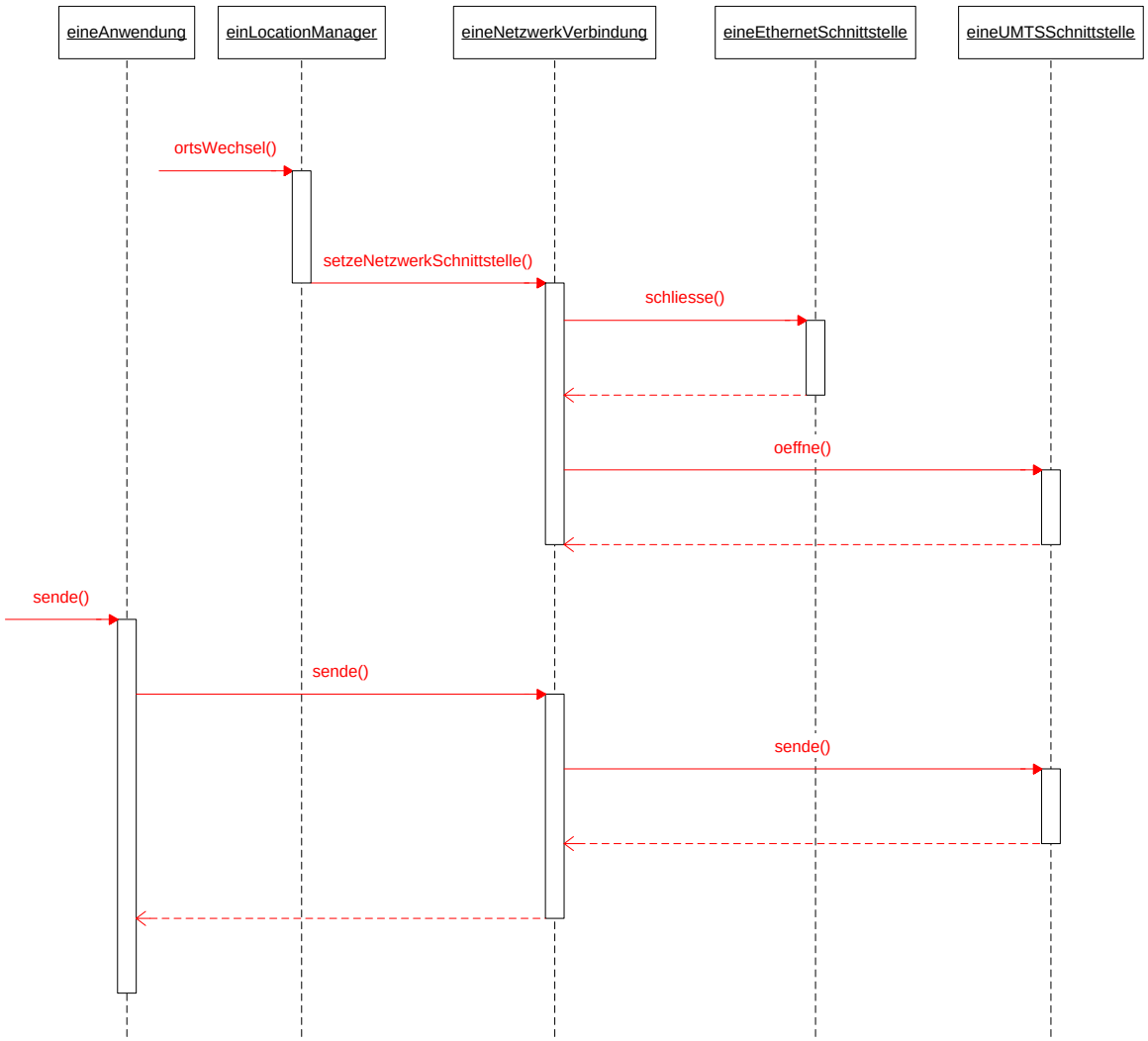
   Warum kommt dieses Vorgehen hier nicht in Frage?

   Die Anwendung soll für neue Protokolle nicht neu übersetzt werden müssen.


Anwendung
+ sende()
NetzwerkVerbindung
netzSch : NetzwerkSchnittstelle
+ sende()
+ empfange()
+ setzeNetzwerkSchnittstelle(ntzSch : NetzwerkSchnittstelle)
LocationManager
netzVerb : NetzwerkVerbindung
+ ortsWechsel()
if (istEthernetVerfügbar) {
netzVerb.setzeNetzwerkSchnittstelle(new Ethernet())
} else if (istWLANVerfügbar) {
netzVerb.setzeNetzwerkSchnittstelle(new WLAN())
} else if (istUMTSVerfügbar) {
netzVerb.setzeNetzwerkSchnittstelle(new UMTS())
}
NetzwerkSchnittstelle
+ oeffne()
+ schliesse()
+ sende()
+ empfange()
Ethernet
+ oeffne()
+ schliesse()
+ sende()
+ empfange()
WLAN
+ oeffne()
+ schliesse()
+ sende()
+ empfange()
UMTS
+ oeffne()
+ schliesse()
+ sende()
+ empfange()

f) Vervollständigen Sie das unten stehende UML-Sequenzdiagramm wie folgt:

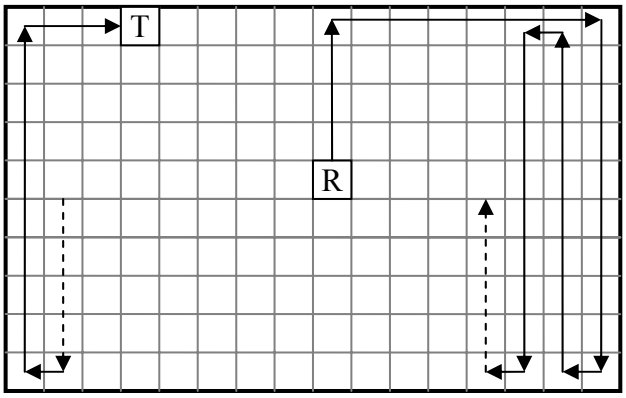
- Die Hardware des tragbaren Computers informiert den **`LocationManager`** über einen Ortswechsel.
- Auf **`eineNetzwerkVerbindung`** wird **`setzeNetzwerkSchnittstelle`** aufgerufen.
- **`eineEthernetSchnittstelle`** wird geschlossen.
- **`eineUMTSSchnittstelle`** wird geöffnet.
- Auf **`eineAnwendung`** wird **`sende()`** aufgerufen.
- **`eineAnwendung`** versendet Daten via **`eineUMTSSchnittstelle`**.

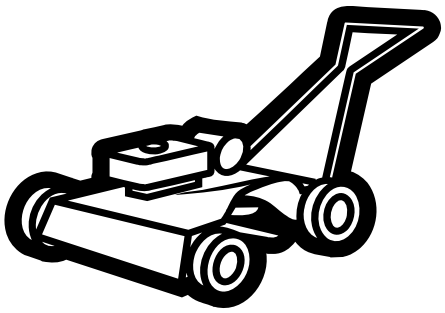


Aufgabe 5 (Harelautomaten) **(11 Punkte)**

Sommerzeit ist auch die Zeit des Rasenmähens. Als Informatiker verwendet man dafür natürlich einen vollautomatischen Rasenmäher. Sie sollen deshalb mittels Harelautomaten ein solches Gerät entwerfen. Einmal im Garten plaziert, soll der Rasenmäher selbstständig alle Teile des Gartens besuchen und, nachdem der Rasen überall gemäht ist, wieder zurück zur „Stromtankstelle“ fahren.

Wie die folgende Abbildung (nur ein Beispiel!) veranschaulicht wird davon ausgegangen, dass das Gartengrundstück eine rechteckige Fläche ist. Es gibt keine Hindernisse. Der einzige markante Punkt ist die Stromtankstelle. Die Tankstelle befindet sich immer direkt an einer Wand! Der Garten ist in Planquadrate unterteilt, die alle die gleiche Größe wie die Stromtankstelle (T) bzw. der Rasenmäher (R) haben.





Der Rasenmäher kann folgende Bewegungen durchführen:

1. Ein Quadrat weiter geradeaus fahren (*fahre_geradeaus* oder *FG*)
2. Sich um 90° nach links drehen (*drehe_nach_links* oder *DL*)
3. Sich um 90° nach rechts drehen (*drehe_nach_rechts* oder *DR*)

Zwei Sensoren sind die einzige Möglichkeit für den Rasenmäher, Informationen über seine aktuelle Position abzufragen:

1. Ein „Wandsensor“ an der Vorderseite des Gerätes liefert kontinuierlich die Werte *Wand* (oder *W*) wenn der Rasenmäher vor einer Wand steht bzw. *FreieFahrt* (oder *FF*) sonst.
2. Ein Sensor an der Unterseite zeigt ständig an, ob sich der Rasenmäher über der Stromtankstelle befindet (*Tankstelle* oder *T*) oder über Rasen (*Rasen* oder *R*).

Der ganze Arbeitsvorgang besteht aus drei aufeinanderfolgenden Schritten:

1. Der Rasenmäher wird irgendwo im Garten plaziert und soll daraufhin eine Ausgangsposition in einer beliebigen Ecke der Rasenfläche einnehmen (*Fahre_in_Ausgangsposition*).
2. Der Rasen wird in Bahnen von Wand zu Wand gemäht (*Rasen_mähen*).
3. Nachdem das gesamte Grundstück gemäht wurde fährt der Rasenmäher so lange an der Wand entlang bis er auf der Stromtankstelle steht (*Auftanken*).

Name: _____ Matrikelnummer: _____

Entwerfen Sie unter Verwendung der folgenden Zustände einen Harelautomaten gemäß der oben gegebenen Systembeschreibung.