Klausur Softwaretechnik

16.03.2005

Prof. Dr. Walter F. Tichy
Dipl.-Inform. T. Gelhausen
Dipl.-Inform. G. Malpohl

Hier das Namensschild aufkleben.

Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen. Die Bearbeitungszeit beträgt 60 Minuten. Die Klausur ist vollständig und geheftet abzugeben.

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**∑**
Maximal	8	11	11	13	8	9	**60**
K1							
K2							
K3							

Punkte:

Note:

Aufgabe 1: Aufwärmen (3+2+2+1=8P)

a.) Kreuzen Sie an, ob die Aussage Wahr oder Falsch ist. (3P)
Hinweis: Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz bewirkt 0,5 Punkte Abzug! Die Teilaufgabe wird mindestens mit 0 Punkten bewertet.

WAHR	FALSCH	Aussage
	X	Der einzige Nachteil eines Parallellaufs in der Einführungsphase sind die erhöhten Kosten für die doppelt benötigten Systemressourcen.
	X	In Java wird das Entwurfsmuster Null-Objekt durch das Konzept `null` realisiert.
	X	Im Harel-Automaten sind ε-Übergänge (Zustandsübergänge ohne auslösendes Ereignis) zulässig.
	X	Den kompletten Lebenszyklus eines Objektes, von der Speicherallokation bis zur –freigabe, nennt man ***α-λ-Zyklus*** .
	X	Walkthroughs, zumindest nach ANSI/IEEE 729-1983, sind formaler als Reviews.
	X	Das Verfahren, gegenseitige Benutzung von Modulen durch eine Verfeinerung der Modulstruktur aufzulösen, heißt ***Stacking*** .

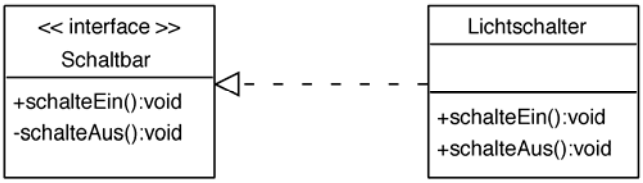
b.) Nehmen Sie an, Sie inspizieren das Klassendiagramm eines Kollegen. Auf der Prüfliste steht, Sie sollen für alle Klassen prüfen, ob diese tatsächlich existenzberechtigt sind. Nennen Sie 2 handfeste Indizien dafür, dass **keine** Klasse vorliegt. (2P)

- Es lassen sich weder Attribute noch Operationen identifizieren
- Eine Klasse enthält dieselben Attribute, Operationen, Restriktionen und Assoziationen/Aggregationen wie eine andere Klasse
- Eine Klasse enthält nur Operationen, die sich anderen Klassen zuordnen lassen
- Eine Klasse modelliert Implementierungsdetails

c.) **Warum** ist es bei der Implementierung des Einzelstücks in Java sinnvoll, das Schlüsselwort **`synchronized`** zu verwenden? **An welcher Stelle** muss der Code synchronisiert werden? (2P)

Der Code der prüft, ob die statische Variable null ist und ggf. eine neue Instanz erzeugt (1P) muss synchronisiert werden, um sicherzustellen, dass keine zwei Prozesse die „erste“ Instanz gleichzeitig anlegen und dann mit verschiedenen Instanzen arbeiten (1P).

d.) Warum ist folgende Modellierung falsch (mit **Begründung**)? (1P)



In einem Interface gibt es keine Implementierungen. Daher kann es keinen Benutzer für die Methode -schalteAus():void im Interface „Schaltbar“ geben. (Eine Implementierung kann es übrigens auch nicht geben.)

Hinweise: Der Stereotyp „Interface“ definiert die Methoden von Schaltbar automatisch als abstrakt. Entsprechend dem Substitutionsprinzip kann man Methoden der Oberklasse immer „sichtbarer“ machen.

Aufgabe 2: Entwurfsmuster (1+1+6+3=11P)

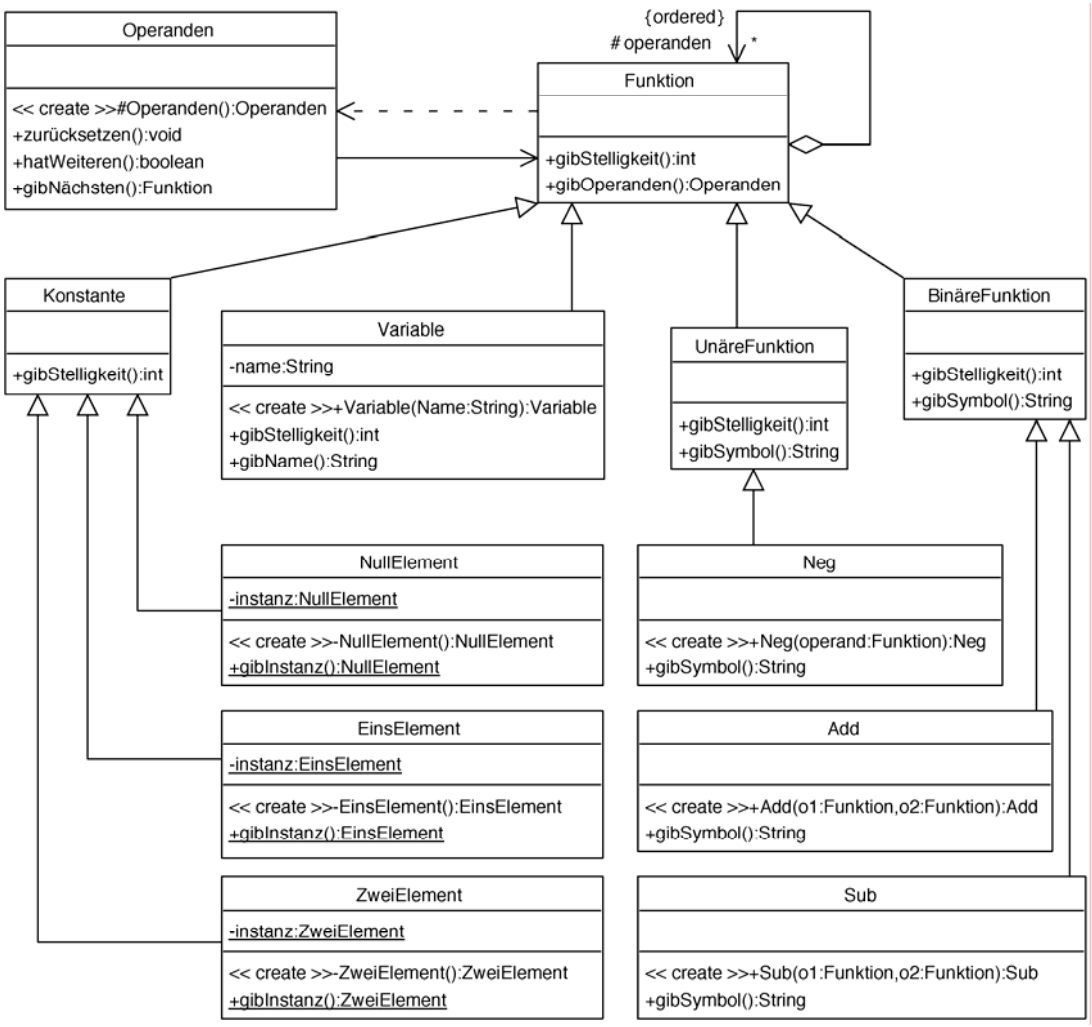
a.) Welchem **Zweck** dient das Besucher-Muster? (1P)

> Das Besuchermuster ermöglicht es, eine neue Operation zu definieren, ohne die Klassen der von ihr bearbeiteten Elemente zu verändern, externes Zusammenfassen gleichartiger Funktionalität.

b.) Welchem **Zweck** dient das Entwurfsmuster „Brücke“? (1P)

> Das Entwurfsmuster entkoppelt eine Abstraktion von ihrer Implementierung, so dass beide unabhängig voneinander variiert werden können.

c.) Gegeben sei folgender Entwurf. Einer Ihrer Kommilitonen behauptet, er habe 3 Entwurfsmuster identifiziert: Iterator, Einzelstück und Kompositum. Hat er Recht? Begründen Sie Ihre Aussage, indem Sie jeweils **die Klassen** (3P), die er wohl gemeint hat, und **die Indizien** (3P), die dafür (oder dagegen) sprechen, angeben. Bedenken Sie, dass konkrete Implementierungen von den idealisierten Mustern abweichen können! (6P)



Iterator: ja, bei „Operanden“
Neben den typischen Funktionen +hatWeiteren():boolean und +gibNächsten():Funktion sind auch die typische Assoziation und „benutzt“-Relation vorhanden. Die Methode +zurücksetzen():void ist zusätzlich.

Kompositum: ja, bei „Funktion“
„Funktion“ ist Kompositum und Komponente in einem. Die Methoden +fügeHinzu(:Komponente):void und +entferne(:Komponente):void fehlen zwar, die gemeinsame Operation +gibStelligkeit():int und die Funktion zur Rückgabe der Kinder +gibOperanden(), sowie die entsprechende Assoziation sind vorhanden.

Singleton: ja, „NullElement“, „EinsElement“, „ZweiElement“
Typischer privater Konstruktor und statische +gibInstanz()-Methode.

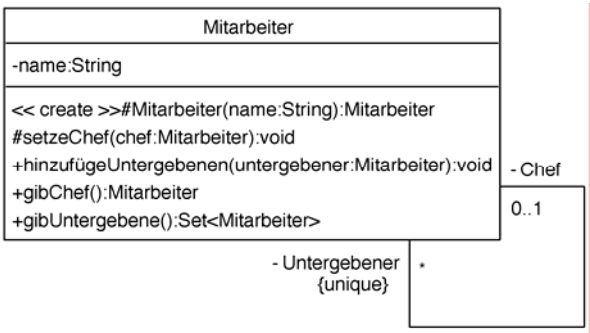
Je 1P wenn min. 1 Stelle angegeben, je 1P für die Begründung.

d.) Walter Zimmer beschreibt in „Relationships between design patterns, in Pattern languages of program design“, einen Gleichartigkeitsbegriff, wie er in deutscher Übersetzung auch auf Übungsblatt 4, Aufgabe 2 angegeben war. In dem angegebenen Artikel behauptet Herr Zimmer ferner, sowohl (Abstrakte Fabrik und Erbauer) als auch (Fassade und Mediator) seien gleichartig. Stimmen Sie dieser Aussage zu? Geben Sie die **Definition** an und **begründen** Sie Ihre Antwort auf deren Grundlage. (3P)

Definition: Die Entwurfsmuster X und Y sind gleichartig, wenn X und Y gleichartige Probleme lösen. Die Lösungsansätze brauchen dabei nicht gleichartig zu sein (1P).
Abstrakte Fabrik und Erbauer dienen beide der Konstruktion komplexer Objekte (1P).
Fassade und Mediator dienen beide zum Entkoppeln von Objekten (1P).

Aufgabe 3: Abbildung von UML auf Code (7+4=11P)

a.) Gegeben sei das nebenstehende UML-Diagramm. Implementieren Sie diese Klasse **in Java**. Schreiben Sie den Code für die Verwaltung der Assoziation in die angegebenen Methoden; Sie brauchen keine weiteren Verwaltungsmethoden anzugeben. Bedenken Sie, dass ein Mitarbeiter auch einen neuen Chef bekommen könnte. Sie dürfen das „Collection Framework“ aus java.util verwenden. Achten Sie auf Sichtbarkeiten und korrekte Syntax! (7P)




```
import java.util.*;

public class Mitarbeiter {
    private String name;
    private Mitarbeiter chef;
    private Set<Mitarbeiter> untergebene = new TreeSet<Mitarbeiter>();
    protected Mitarbeiter(String name) {
        this.name = name;
    }
    protected void setzeChef(Mitarbeiter chef) {
        if (this.chef!=chef) this.chef.untergebene.remove(this);
        this.chef = chef;
        if (!chef.gibUntergebene().contains(this))
            chef.hinzufügeUntergebenen(this);
    }
    public void hinzufügeUntergebenen(Mitarbeiter untergebener) {
        untergebene.add(untergebener);
        if (untergebener.gibChef()!=this)
            untergebener.setzeChef(this);
    }
    public Mitarbeiter gibChef() {
        return chef;
    }
    public Set<Mitarbeiter> gibUntergebene() {
        return untergebene;
    }
}
```

8x0,5P für korrekte Signaturen (Attribute und Methoden),
bis zu 1P für korrekte Sichtbarkeit,
bis zu 2P für die Funktionalität von setzeChef() und hinzufügeUntergebenen()

b.) **Welche** Schwierigkeiten ergeben sich allgemein bei der Umsetzung von den Multiplizitäten „1“ und „1..*“ und **warum**? Machen Sie einen **Vorschlag**, wie man diese Schwierigkeiten umgehen kann. Was kann man tun, wenn an **beiden Enden** der Assoziation derartige Kardinalitäten zu finden sind? (4P, jedoch nur für ganze Sätze)

Die Kardinalitäten „1“ und „1..*“ garantieren dem Gegenüber mindestens einen Assoziationspartner (1P). Bei der Manipulation der Assoziation kann es zu Inkonsistenzen kommen (0,5P), der Assoziationen manipulierende Code muss diese verhindern (0,5P). Jedoch auch bei der Initialisierung kann es zu Inkonsistenzen kommen (0,5P). Wenn nur ein Ende solch eine Zusicherung macht, kann man über die Konstruktoren sicherstellen, dass die Initialisierung keine inkonsistenten Zustände zulässt (0,5P). Wenn beide Seiten derartige Zusicherungen machen, muss die initiale Konsistenz z.B. durch eine Fabrikmethode (o.ä.) sichergestellt werden (1P).

Aufgabe 4: Testen & Prüfen (1+5+2+2+3=13P)

```
double c = 0;
double TOL = 0.8;
double diff = 0;
do {
   c = (a + b)/2;
   double y = f(c)*f(a);
   if (y < 0)
      b = c;
   else
      a = c;
   diff = Math.abs(a-b);
} while (diff > TOL);
System.out.println(c);
```

Gegeben sei nebenstehendes Rahmenprogramm für einen Bisektionsalgorithmus*. Die Funktion f(x) mit der Signatur **`f(x:double):double`** sei verifiziert und

spezifiziert als $f(x) := \begin{cases} x^2, & \text{wenn } x > 0 \\ 0, & \text{wenn } x = 0 \\ -x^2, & \text{wenn } x < 0 \end{cases}$

**Das ist nur Kontextinformation. Für die Bearbeitung dieser Aufgabe brauchen Sie nicht zu wissen, wozu man einen Bisektionsalgorithmus braucht oder wie er funktioniert.*

a.) Was ist gemäß der Definition aus der Vorlesung unter „Die Funktion f ist verifiziert“ zu verstehen? (1P)

Die Korrektheit von f(x) ist bewiesen, sie braucht nicht mehr getestet zu werden.

b.) Wandeln Sie obiges Programm mit einer strukturerhaltenden Transformation in eine der Definition der Vorlesung entsprechenden Zwischensprache um. (5P)

```
 10:  double c = 0;
 20:  double TOL = 0.8;
 30:  double diff = 0;
 40:  c = (a + b)/2;                  // Schleifenanfang
 50:  double y = f(c)*f(a);
 60:  if (y < 0) goto 90;
 70:  a = c;
 80:  goto 100;
 90:  b = c;
100:  diff = Math.abs(a-b);
110:  if (diff > TOL) goto 40;
120:  System.out.println(c);
130:  -
```

1P für das korrekte Übertragen der unveränderten Teile

je 1P für das korrekte Umsetzen von Schleifenanfang, -ende, sowie if- und else-Zweig

Für KFG max. 2P: 1P für das korrekte Übertragen der unveränderten Teile und 1P für richtigen Kontrollfluss

c.) **Wie viele** initiale Belegungen für a und b braucht man, um die Schleife einem Boundary-Interior-Test zu unterziehen? **Erläutern** Sie Ihre Antwort kurz, damit wir sehen, dass Sie nicht geraten haben! (2P)

2 Grenztests (einen Schleifenquerer für jeden Pfad durch die if-Bedingung) und 2 Interieurtests (einen Schleifenquerer für jeden Pfad durch die if-Bedingung im 2. Durchlauf), analog Vorlesung

Punkte **nur** mit Begründung.

d.) **Welche Arten** von Überdeckung erreicht man mit (a,b) = (-1.0, 2.0)? **Erläutern** Sie Ihre Antwort kurz, damit wir sehen, dass Sie nicht geraten haben! (2P)
Tipp: Rechnen Sie mit Brüchen!

Man erreicht Zweigüberdeckung und damit automatisch Anweisungsüberdeckung, die Schleife wird 1x wiederholt und beide if-Zweige jeweils 1x durchschritten: bei der ersten Iteration der Code aus Zeile 90, beim der zweiten Iteration der Code aus Zeile 70
Variablenbelegung über die Zeit: a=-1, b=2, c=1/2, y=-1/4, b=1/2, diff=3/2, c=-1/4, y=1/16, a=-1/4, diff=3/4, Ausgabe: -1/4

„Bedingungsüberdeckung“ oder „Zweigüberdeckung“: bis zu 2P
Nur „Anweisungsüberdeckung“: bis zu 1P
Punkte **nur** mit Begründung.

e.) Für diese Teilaufgabe sei $f(x) = x^2$. **Welche Probleme** ergeben sich dann für das Testen? Wie lassen sich solche Probleme im Alltag finden? **Nennen** Sie hierfür je ein **dynamisches** und ein **statisches** Verfahren oder Werkzeug! (3P)

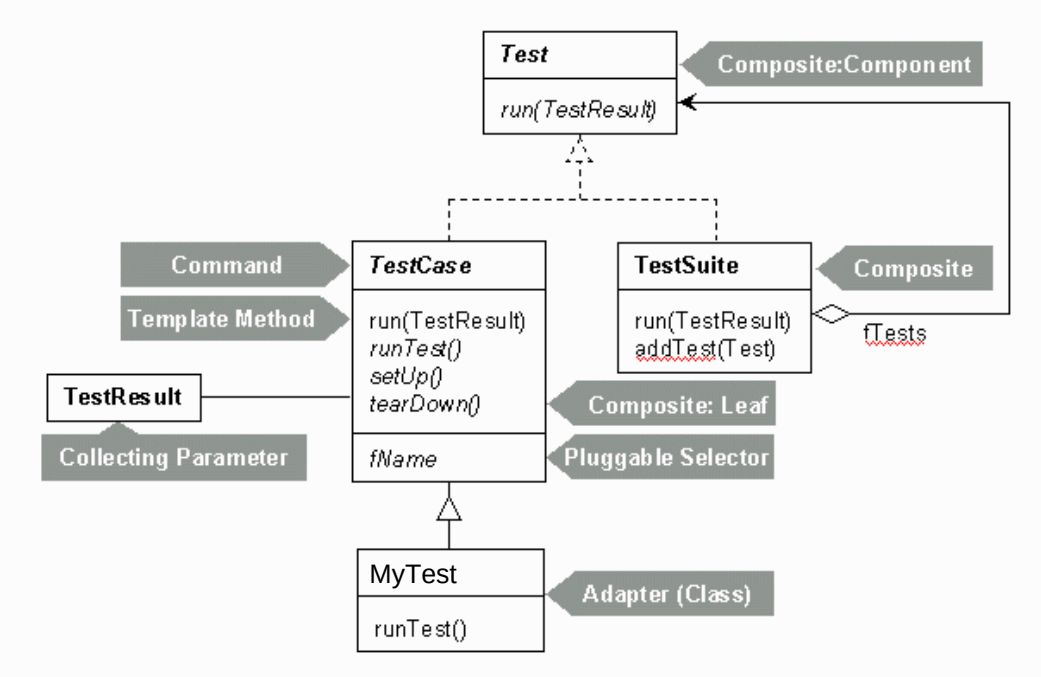
Problem: Es kann kein x geben, so dass f(x)<0. Damit ist in Zeile 60 immer y>=0 und keine Anweisungsüberdeckung möglich. (1P)
Dynamische Methode: Emma (Codeüberdeckungswerkzeug) (1P)
Statische Methode: Inspektion (1P)

Aufgabe 5: JUnit (0,5+4+3,5=8P)

a.) Wozu verwendet man JUnit? (0,5P)

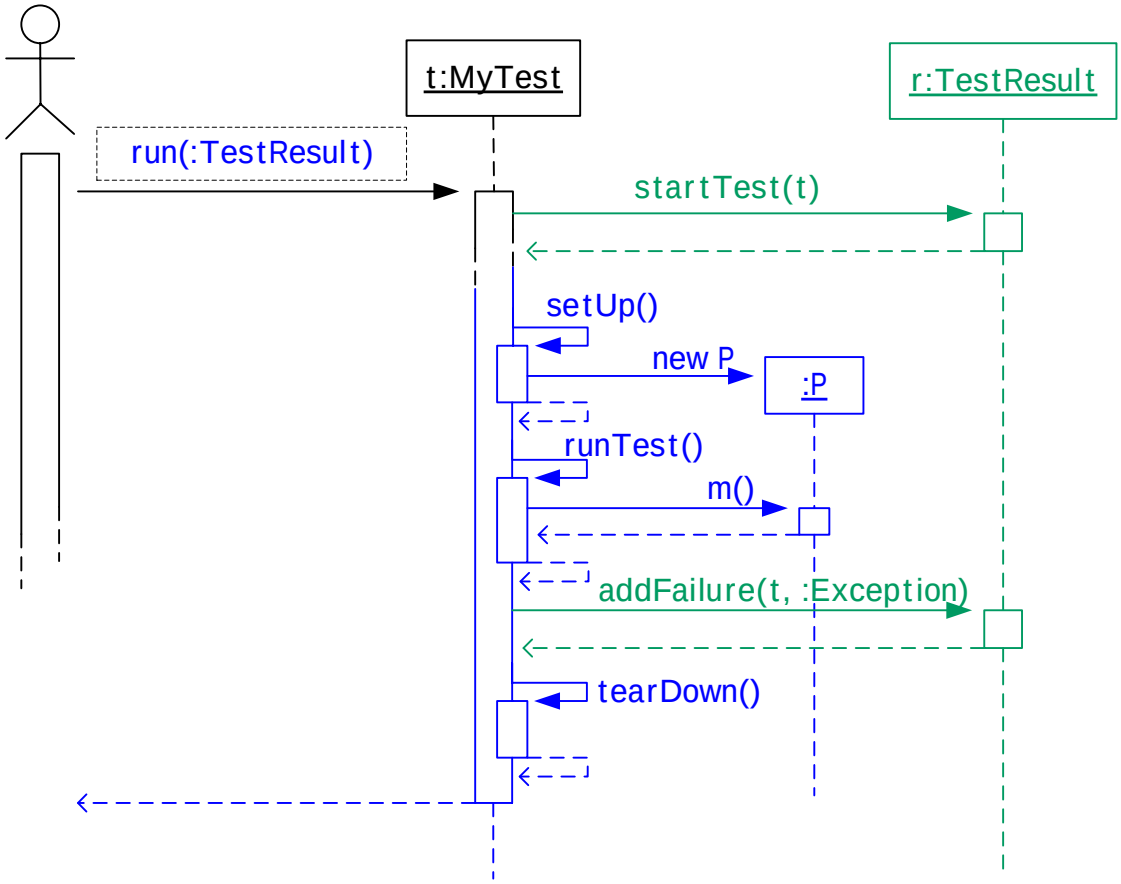
Regressionstest

b.) Wie ist JUnit aufgebaut? Geben Sie ein Klassendiagramm an, das die Typen **`Test`**, **`TestResult`**, **`TestSuite`**, **`TestCase`** und **`MyTest`** in Beziehung setzt und ihre wichtigsten Methodennamen angibt. **`MyTest`** sei dabei Ihre eigene Implementierung eines Tests (entsprechend den JUnit-Konventionen). (4P)



Vererbungsbeziehungen 1P
Kompositum 1P
run() in irgendeiner Form und in MyTest ausführbar 1P
setUp() + tearDown() in MyTest ausführbar 1P

Nehmen Sie an, Sie seien Softwaretechnik-Tutor und wollen Ihren Studenten erläutern, wie JUnit funktioniert. Geben Sie ein einfaches Sequenzdiagramm an, das den Kontrollfluss eines Testlaufs für einen Prüfling P mit der Methode m() verdeutlicht. (3,5P)



0,5P für korrekten Aufruf von run()
0,5P für korrekten Aufruf von setUp()
0,5P für korrektes Erzeugen der Instanz von Prüfling p
0,5P für korrekten Aufruf von runTest()
0,5P für korrekten Aufruf von m()
0,5P für korrekten Aufruf von tearDown()
0,5P für korrekte (oder überdurchschnittlich gute) Syntax
1P Bonus für den grünen Teil (startTest() und addFailure(), nicht verlangt!)
Wenn statt run(:TestResult) jedoch runTest() aufgerufen wird, muss eine Instanz von TestResult erzeugt werden!

Aufgabe 6a: Prozessmodelle (1+6+2=9P) ***(nur für Informatiker)***

a.) In der Vorlesung wurden zwei unterschiedliche Konzepte mit dem Namen „V-Modell“ vorgestellt. Worin unterscheiden sich diese bzw. wie hängen sie zusammen? (1P)

Das Prozessmodell mit der „üblichen“ Darstellung als V ist eine auf die Bedürfnisse der Softwaretechnik spezialisierte Version und nur ein Teilaspekt des gleichnamigen Standards der Öffentlichen Hand – eine mögliche Ausprägung auf Basis des Wasserfallmodells.

b.) Beim Wasserfallmodell spricht man von einem „**dokumentgetriebenen Modell**“. Was versteht man hierunter? **Welche Dokumente** kommen in dem in der Vorlesung vorgestellten 6-stufigen Wasserfallmodell **an welchen Stellen** vor? (6P)

„Dokumentgetriebenes Modell“ bedeutet, dass jede Phase mit einem definierten Dokument/Satz von Dokumenten abgeschlossen wird. 1P
Folgende Dokumente beenden die angegebenen Phasen:
Planung
→ Lastenheft, Projektplan, Kalkulation, Glossar 1P
Definition
→ Pflichtenheft, GUI-Beschreibung, B.-Handbuch, Glossar 1P
Entwurf
→ Architektur, UML, Modulführer 1P
Implementierung
→ Komponenten, Doku, Testdaten, Code 1P
Testen
→ fertiges System, Testprotokoll 1P
Einsatz/Wartung
Punkte für mehr richtig als falsch zugeordnete Dokumente, keine Punkte für die ausschließliche Nennung der Phasen

c.) Beim Iterativen Modell unterscheiden wir den „inkrementellen“ und den „evolutionären“ Ansatz. Worin unterscheiden sich die beiden Vorgehensweisen? (2P)

Während bei der inkrementellen Vorgehensweise erst alles geplant und dann sukzessive implementiert wird (1P), wird bei der evolutionären Vorgehensweise jeder neue Teilaspekt erst dann geplant, wenn die vorhergehenden implementiert sind (1P).

Aufgabe 6b: Objektorientierte Analyse (9P) ***(nur Informationswirte)***

Beschreiben Sie, wie man bei der objektorientierten Analyse die Assoziationen in einem UML-Klassendiagramm erhält. (Die Klassen seien schon vorhanden.) Beachten Sie bei Ihrer Antwort folgende Aspekte:

a.) **Wie** sucht man nach möglichen Assoziationen in einem textuellen Anforderungsdokument? *(max. 3P)*
b.) Was sind die **Voraussetzungen** für die Übernahme gefundener Beziehungen in das Klassendiagramm? *(max. 3P)*
c.) Gehen Sie auf die verschiedenen Ausprägungen und Formen von Assoziationen ein: Erläutern Sie, anhand welcher **Kriterien** man welche **Auswahl** trifft! *(max. 6P)*

(Um volle Punktzahl erhalten zu können, muss auf jeden Aspekt eingegangen werden. Die Angabe der Maxima dient nur zu Ihrer Orientierung, die Aufgabe wird insgesamt mit maximal 9P bewertet!)

Folgende Aspekte werden z.B. bepunktet

a) Quellen (je 1P, max. 3P)
- Verben in der Problembeschreibung (räumliche Nähe (in der Nähe von), Aktionen (fährt), Kommunikation (redet mit), Besitz (hat), allgemeine Beziehungen (verheiratet mit)
- Müssen die Objekte der Klassen miteinander kommunizieren?

b) Voraussetzungen (je 1P, max. 3P)
- Liegen zwischen Objekten permanente Beziehungen vor?/Existieren sie für einen längeren Zeitraum?
- Sind die Assoziationen problemrelevant?
- Existiert eine Beziehung unabhängig von allen nicht beteiligten Klassen?
- Assoziationen, die keine neue Information hinzufügen, vermeiden

c) Mögliche Ausprägungen (normale Assoziationen) (je 1P, max. 3P)
- Richtung: Ist der Informationsfluss uni- oder bidirektional?
- Mehrere Assoziationen zwischen zwei Klassen?
- Rollennamen, wenn die Assoziation zwischen Objekten derselben Klasse existiert
- Rollennamen, wenn eine Klasse in verschiedenen Assoziationen auch verschiedene Rollen spielt
- Rollennamen, wenn durch den Rollennamen die Bedeutung der Klasse in der Assoziation genauer spezifiziert werden kann
- Kardinalitäten nur für binäre Assoziationen und Aggregationen.
- Kardinalitäten: Liegt eine Muss-Beziehung vor? (Sobald das Objekt erzeugt ist, muss auch die Beziehung zu dem anderen Objekt aufgebaut

werden)/Liegt eine Kann-Beziehung vor? (Die Beziehung kann zu einem beliebigen Zeitpunkt nach dem Erzeugen des Objekts aufgebaut werden.)
- Kardinalitäten: Ist die Obergrenze fest oder variabel?
- Ist eine Obergrenze vom Problem her zwingend vorgegeben? (Im Zweifelsfall mit variablen Obergrenzen arbeiten)
- Gelten besondere Bedingungen? Beispiele: Gerade Anzahl mindestens 2, maximal 6.

Mögliche Ausprägungen (Aggregationen und Kompositionen) (je 1P, max. 3P)
- Lässt sich die Beziehung durch »besteht aus« oder »ist Teil von« beschreiben?
- Kann problemlos angegeben werden, ob eine Klasse Aggregat oder Teil in der Beziehung ist?
- Gehören die beteiligten Klassen in ein Subsystem? Wenn der Zugriff auf die Teilobjekte ausschließlich über das Aggregat-Objekt erfolgt, dann liegt eindeutig eine Aggregation vor.
- Lässt sich die Aggregation in eine Kategorie einordnen? Das Ganze und seine Teile (Pkw und Motor), Behälter und Inhalt (Feld und seine Elemente, Flugzeug und Mannschaft), Gruppe/Mitglied (Hotel besteht aus Zimmern), Konfiguration von Teilen in einem Ganzen (Szenen sind Teile eines Films), Invariante Konfiguration von Teilen in einem Ganzen (aus was ist ein Objekt gemacht), Teil-Objekte dürfen nicht entfernt werden (Baum besteht teilweise aus Holz), Teile sind im Prinzip dasselbe wie das Ganze (Meter ist Teil eines Kilometers), Teile können nicht vom Ganzen getrennt werden (München ist Teil von Bayern), Kollektion von Teilen in einem Ganzen (Schiff ist Teil einer Flotte)
- Wird ein Mitglied entfernt, dann wird auch das Ganze zerstört (Lässt sich ein Ehepartner scheiden, dann ist auch die Ehe zerstört)
- Jedes Teil-Objekt trägt zur Funktionalität des Aggregat-Objekts bei (Direktor einer Schule, Motor eines Autos)
- Attribute einer Klasse nicht mit der Aggregation verwechseln!

Notizen:

2+2=5

$e=mc^2$

Softwaretechnik ist das tollste Fach der Welt

ELI+OLI=EhE

XML ist DOOF