Klausur Softwaretechnik

8.9.2005

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**	**∑**
Maximal	18	8	8	13	13	**60**
K1						
K2						
K3						

Aufgabe 1: Aufwärmen (4+1+2+2+1+2+1+2+1+2= 18P)

a.) Kreuzen Sie an, ob die Aussage wahr oder falsch ist. (4P)
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
		Die Definition des Softwareproduktes in der Definitionsphase ist ein iterativer Prozess.
		Nicht-funktionale Anforderungen sind sowohl Teil des Pflichtenhefts als auch des Lastenhefts.
		Bei der Wahl einer Schnittstelle im Datenfluss-Diagramm wird von dem konkreten Gerät zur Ein- oder Ausgabe vollständig abstrahiert.
		Bei der Definition von Vererbungsrelationen gilt folgender Leitsatz: Mache eine Klasse A erst dann zu einer Unterklasse einer Klasse B, wenn gezeigt werden kann, dass jede Instanz von B auch als eine Instanz von A gesehen werden kann.
		Gegenseitige Benutzung von Modulen kann oft durch eine Verfeinerung der Modulstruktur aufgelöst werden. Dieser Prozess heißt Sandwiching.
		Klasse und Paket beim objektorientierten Entwurf sind Analoga zum Modul im modularen Entwurf.
		Regressionstests und Zusicherungen lassen sich zum Testen von Programmen nicht miteinander kombinieren.
		Es gilt die Faustregel: Der Aufwand für Wartung und Pflege ist normalerweise größer als der Entwicklungsaufwand.

b.) Es gibt verschiedene Möglichkeiten, in der Einführungsphase von einem alten auf ein neues System umzustellen. Nennen Sie die drei Arten der Inbetriebnahme. (1 P)

c.) Beschreiben Sie den Unterschied zwischen den Entwurfsmustern *Adapter* und *Dekorierer*? (2 P)

d.) Was ist der Unterschied zwischen *Lasttests* und *Stresstests*? (2 P)

e.) Sortieren Sie die folgenden Testverfahren so, dass jedes Testverfahren alle zuvor genannten subsumiert: (1 P)

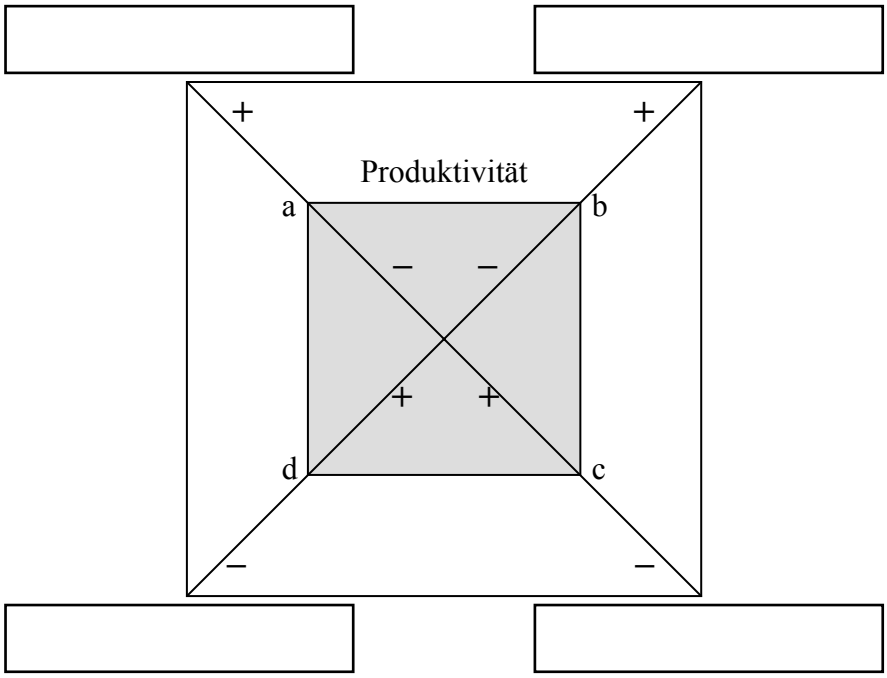
1. Mehrfache Bedingungsüberdeckung
2. Anweisungsüberdeckung
3. Minimal-mehrfache Bedingungsüberdeckung
4. Zweigüberdeckung

f.) Beschreiben Sie den Unterschied zwischen *Black-Box*- und *White-Box*-Testen. (2 P)

g.) Welche Programm-Bearbeitungszustände definiert das *V-Modell*? (1 P)

h.) Welche zusätzlichen Möglichkeiten hat man beim objektorientierten Entwurf im Vergleich zum modularen Entwurf? (2 P)

i.) Ergänzen Sie das unten abgebildete „Teufelsquadrat“ um die 4 Einflussfaktoren. (1 P)



j.) Welche Zusammenhänge illustriert dieses Diagramm? (2 P)

Aufgabe 2: Modularer Entwurf (8 P)

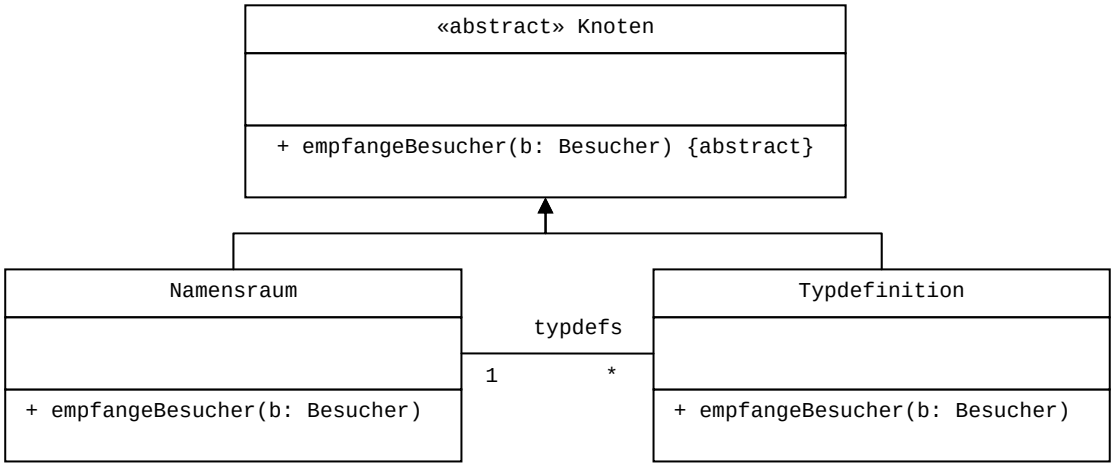
Entwerfen Sie eine Modulschnittstelle für eine Wörterbuchbibliothek:
Die Bibliothek soll den Aufbau und die Verwaltung eines zweisprachigen Wörterbuches erlauben. Das „Geheimnis“ des Moduls ist die interne Datenstruktur und die verwendeten Dateiformate.
Die Schnittstelle soll insbesondere folgende Aktionen unterstützen:

1. Initialisieren des Wörterbuchs mit Festlegung der beiden Sprachen.
2. Paarweises Einspeichern und Löschen von Wortkombinationen (=Wörtern mit gleicher Bedeutung in den beiden Sprachen).
   Hinweis: Im Allgemeinen kann es zu einem Wort in einer Sprache mehrere Übersetzungen in der anderen Sprache geben. Beispiel für Wortpaare: „gehen“↔„go“, „gehen“↔„walk“, „starten“↔„go“
3. Abfragen aller Übersetzungen eines Wortes von einer Sprache in die andere. (Dies soll in beidenz Richtungen möglich sein.)
4. Abfrage der Anzahl der gespeicherten Wörter für beide Sprachen.
5. Abfrage der Sprachen.
6. Speichern und Laden des gesamten Wörterbuches.

Geben Sie eine genaue Beschreibung der von ihrem Modul zur Verfügung gestellten Typen und Funktionen in Java-ähnlicher Pseudonotation an. Beschreiben Sie ihre Schnittstellen ausreichend, so dass klar ist, wozu sie dienen. Beachten Sie, dass Sie keinen objektorientierten Entwurf erstellen sollen!

Aufgabe 3: Entwurfsmuster (2+3+3 = 8P)

Eine Firma hat für die neue Programmiersprache *Choky#* einen abstrakten Syntaxbaum implementiert. Ein Teil der Datenstruktur ist nachfolgend für die Knoten `Namensraum` und `Typdefinition` dargestellt.



In weiser Voraussicht enthalten die Klassen eine Methode `empfangeBesucher(b: Besucher)`, die bereits einen Teil der Implementierung eines Besuchermusters vorgibt

a.) Geben Sie für die Klassen `Namensraum` und `Typdefinition` eine abstrakte Besucher-Klasse in UML-Notation an. Definieren Sie hierzu eine Spezialisierung namens `TypPrüfBesucher`. Der Zweck des `TypPrüfBesucher` ist die Typprüfung von Choky# Programmen. Spezifizieren Sie in beiden Klassen die vollständigen Methodensignaturen (Sichtbarkeit, Namen, Parameter, Rückgabetyp). (2 P)

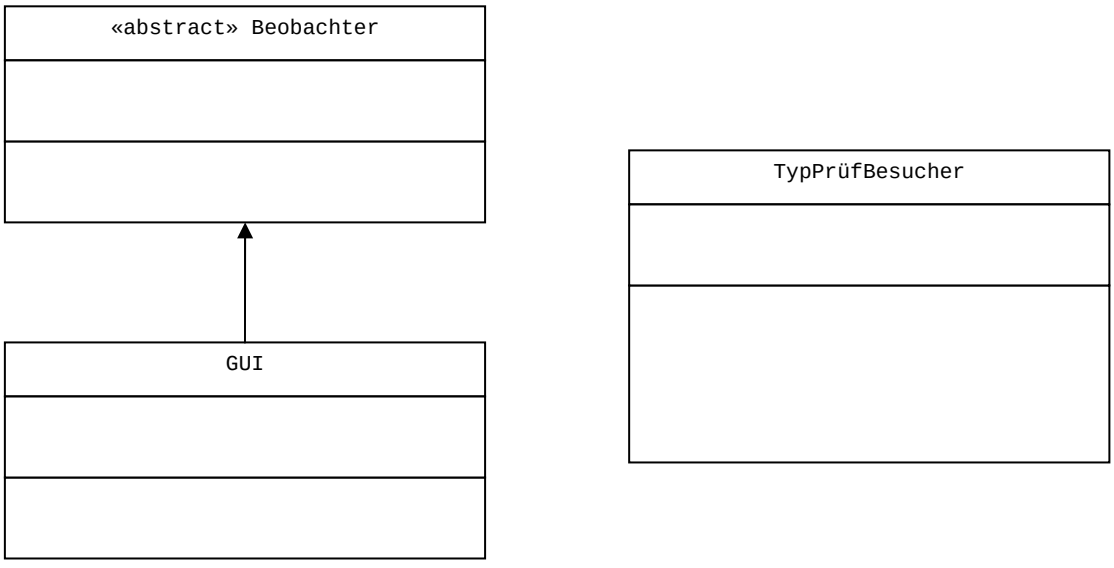
Ein Besucher, der eine Instanz der Knoten-Klasse `Namensraum` besucht, muss, nachdem er seine Aufgaben erledigt hat, auch die in dem `Namensraum` enthaltenen `Typdefinitionen` besuchen.

b.) Implementieren Sie den für die Traversierung der `Typdefinition`en notwendige Methode `empfangeBesucher` in der Klasse `Namensraum`. Gehen Sie davon aus, dass `Typdefs` vom Typ `java.util.Vector` ist. Achten Sie auf Konsistenz mit den UML-Spezifikationen. (3 P)

In der GUI der Choky#-Entwicklungsumgebung sollen alle `TypDefinition`en, die die Typprüfung durchlaufen haben, angezeigt werden.

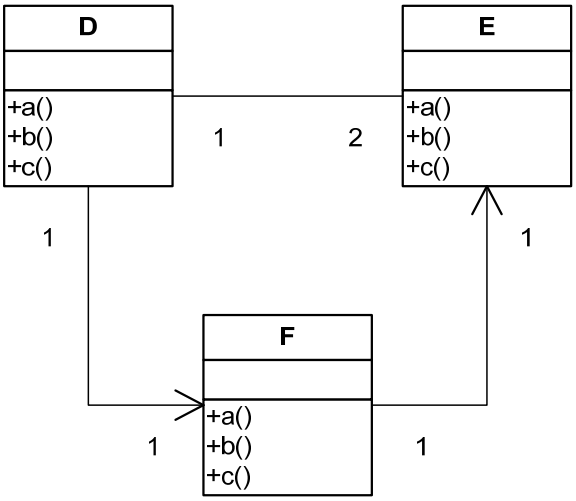
c.) Erweitern Sie unter Verwendung des Entwurfsmusters *Beobacher* die nachfolgend gegebenen Klassen `Beobachter`, `GUI` und `TypPrüfBesucher` um die für das Entwurfsmuster *Beobachter* notwendigen Methoden und Assoziationen. Spezifizieren Sie dabei die vollständigen Methodensignaturen (Sichtbarkeit, Namen, Parameter, Rückgabetyp) und die Kardinalitäten der Assoziationen. (3 P)

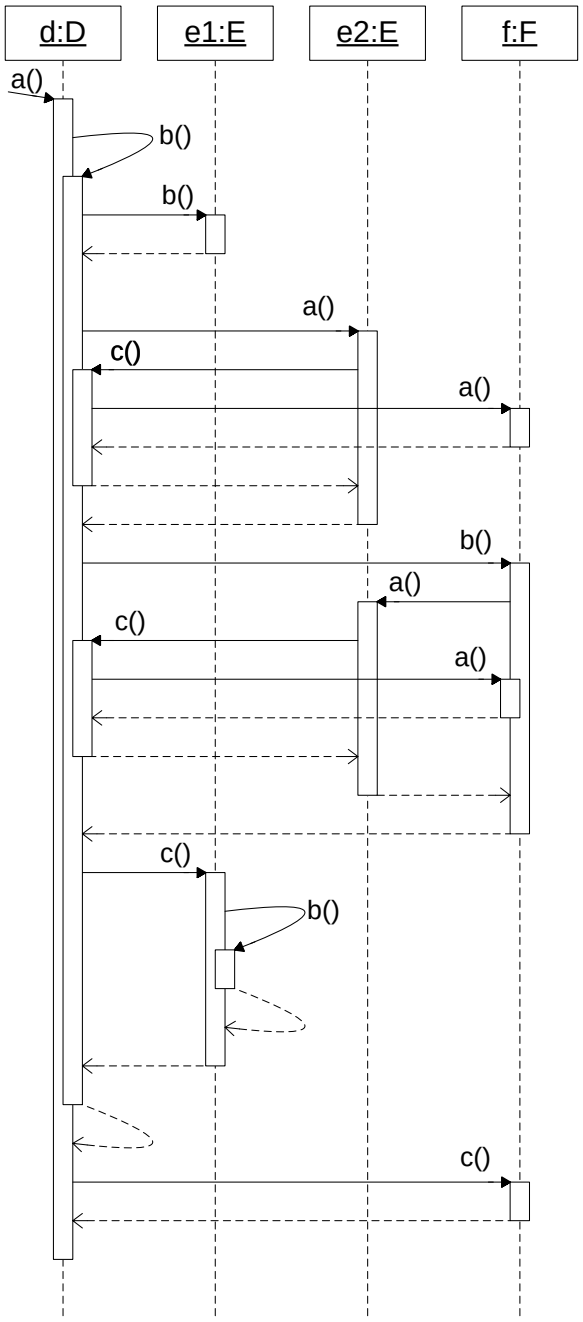
Hinweis: Sie brauchen die Methodensignaturen aus Aufgabenteil a) nicht erneut angeben.



Aufgabe 4: Abbildung von UML auf Code (13 P)

Schreiben Sie Java-Code für die in den beiden UML-Diagrammen angegebenen Klassen. Implementieren Sie die Klassen so, dass sie möglichst genau das durch die UML-Diagramme spezifizierte Verhalten an den Tag legen. Verwenden Sie hierfür die Lücken der Vorlage auf dieser und der folgenden Seite. Schreiben Sie „// nicht spezifiziert“ in die Lücken, für die die UML-Diagramme keinen Inhalt spezifizieren.
Achten Sie auf korrekte Java Syntax!
Hinweis: Initialisieren Sie alle Verweise auf Objekte im Konstruktor über Parameter.

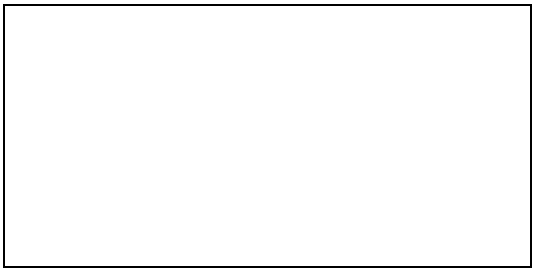




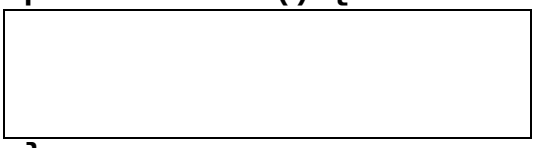

```
class D {
    // Instanzvariablen
    // Konstruktor
    // Methoden
    public void a() {
    }
    public void b() {
    }
    public void c() {
    }
}
```

```
class E {
    // Instanzvariablen
```

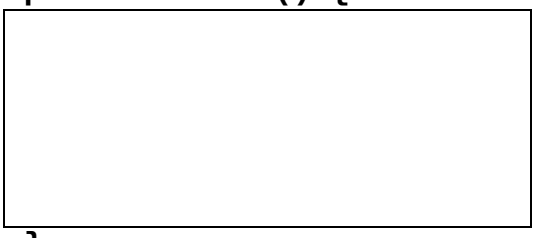
```
    // Konstruktor
```



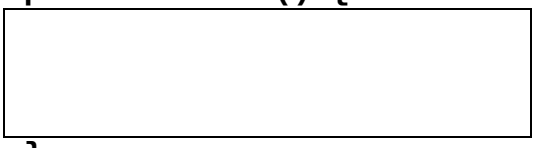
```
    // Methoden
    public void a() {
```



```
    }
    public void b() {
```



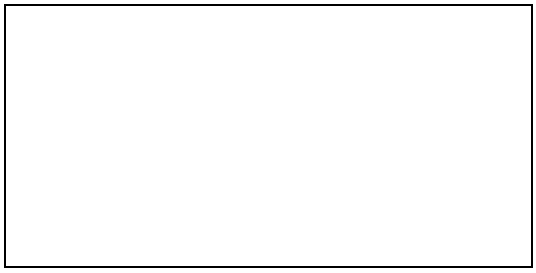
```
    }
    public void c() {
```



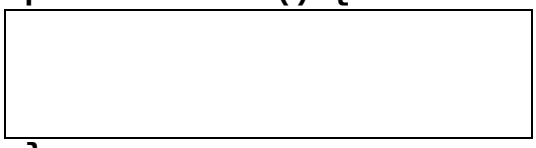
```
    }
}
```

```
class F {
    // Instanzvariablen
```

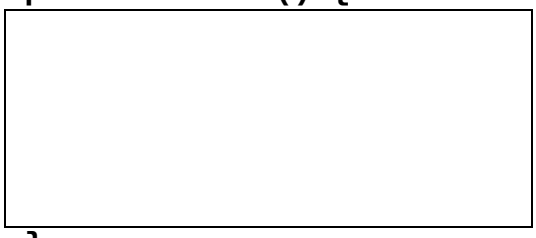
```
    // Konstruktor
```



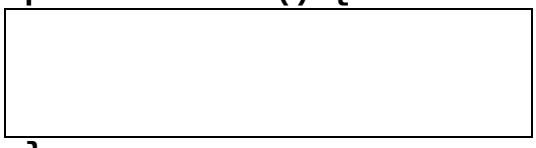
```
    // Methoden
    public void a() {
```



```
    }
    public void b() {
```



```
    }
    public void c() {
```



```
    }
}
```

Aufgabe 5: Testen (5+3+2+3 = 13P)

Betrachten Sie die folgende Funktion zur Berechnung des größten gemeinsamen Teilers:

```
int ggT(int a, int b) {
    for (int i=1; a != b; i++) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
        System.out.println(i+": a="+a+" b="+b);
    }
    return a;
}
```

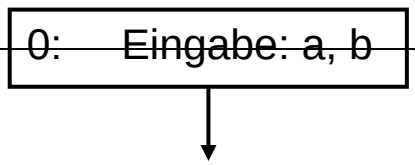
a.) Wandeln Sie obiges Programm mit einer strukturerhaltenden Transformation in eine der Definition der Vorlesung entsprechenden Zwischensprache um. (5 P)

Eingabeparameter: a, b

b.) Benutzen Sie die in Aufgabenteil a.) erstellte Repräsentation des Programms in der Zwischensprache, um auf der folgenden Seite einen Kontrollflussgraphen der Funktion `ggT` zu erstellen. Wenden Sie dabei das aus der Vorlesung bekannte Verfahren an. (3 P)

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! (2 P)

Kontrollflussgraph



d.) Nennen Sie eine Menge von Belegungen der Eingabewerte (a, b), mit der das Kriterium der Zweigeüberdeckung erfüllt wird. Geben Sie weiterhin den vollständigen Pfad (bzw. die vollständigen Pfade) an, die mit ihren Eingabedaten im Kontrollflussgraphen durchlaufen werden, und begründen Sie, warum das Kriterium der Zweigüberdeckung erfüllt wird. (3 P)