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
X		Die Definition des Softwareproduktes in der Definitionsphase ist ein iterativer Prozess.
X		Nicht-funktionale Anforderungen sind sowohl Teil des Pflichtenhefts als auch des Lastenhefts.
X		Bei der Wahl einer Schnittstelle im Datenfluss-Diagramm wird von dem konkreten Gerät zur Ein- oder Ausgabe vollständig abstrahiert.
	X	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.
X		Gegenseitige Benutzung von Modulen kann oft durch eine Verfeinerung der Modulstruktur aufgelöst werden. Dieser Prozess heißt Sandwiching.
X		Klasse und Paket beim objektorientierten Entwurf sind Analoga zum Modul im modularen Entwurf.
	X	Regressionstests und Zusicherungen lassen sich zum Testen von Programmen nicht miteinander kombinieren.
X		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)

- Direkte Umstellung
- Parallellauf
- Versuchslauf

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

- Der Dekorierer bietet unter der gleichen Schnittstelle unterschiedliche Funktionalität, während der Adapter eine bestehende Funktionalität an eine neue Schnittstelle anpasst.
- Der Adapter kommt bei der Integration von unabhängigen (nicht-kompatiblen) Systemen zum Einsatz, während der Dekorierer als Entwurfselement innerhalb von einem System zum Einsatz kommt.

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

Ein Lasttest testet das System oder die Komponente auf Zuverlässigkeit und das Einhalten der Spezifikation innerhalb des erlaubten Grenzbereichs. Stresstests testen das Verhalten beim Überschreiten der definierten Grenzen.

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

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

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

- White: Bestimmen der Testfälle mit Kenntnis von Kontroll- und/oder Datenfluss.
- Black: Bestimmen der Testfälle ohne Kenntnis von Kontroll- und Datenfluss aus der Spezifikation heraus.

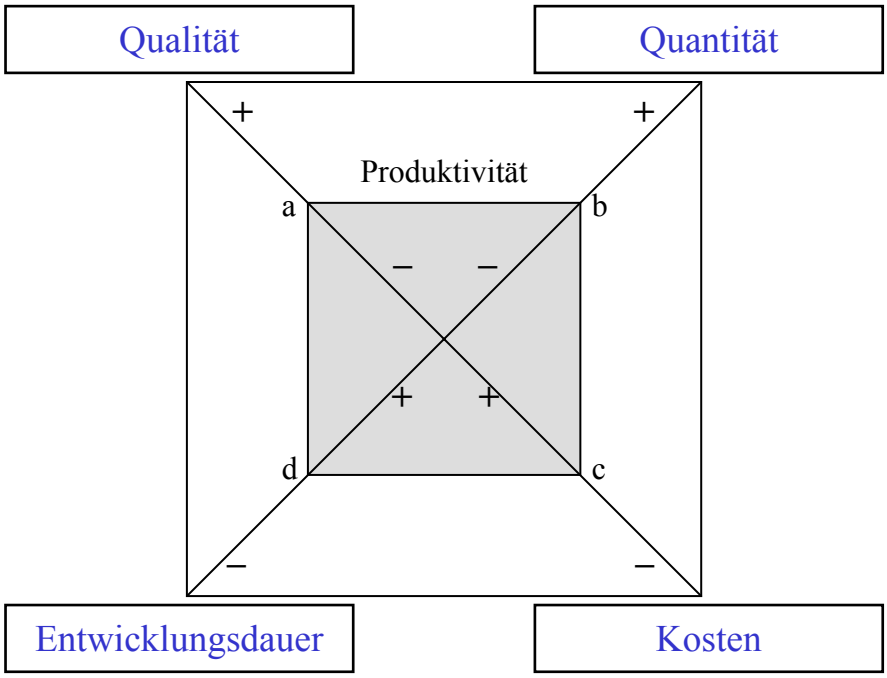
g.) Welche Programm-Bearbeitungszustände definiert das *V-Modell*? (1 P)

Geplant, in Bearbeitung, vorgelegt, akzeptiert

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

- Mehrfach-Instanzierung von Klassen
- Vererbung
- Polymorphie
- Variantenbildung durch Mehrfachimplementierung einer Schnittstelle

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



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

Das Diagramm illustriert die Abhängigkeit der Einflussfaktoren untereinander bei gleich bleibender Produktivität, welche durch die Fläche des Quadrates dargestellt wird.
Die Verschiebung eines Punktes („a“, „b“, „c“ oder „d“) auf der jeweiligen Achse muss auch ein Verschiebung eines anderen Punktes zur Folge haben.
Beispiel: Bei sinkender Entwicklungsdauer muss entweder die Qualität sinken, die Quantität sinken oder es müssen die Kosten steigen; oder eine Kombination dieser Möglichkeiten.

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 beiden 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!

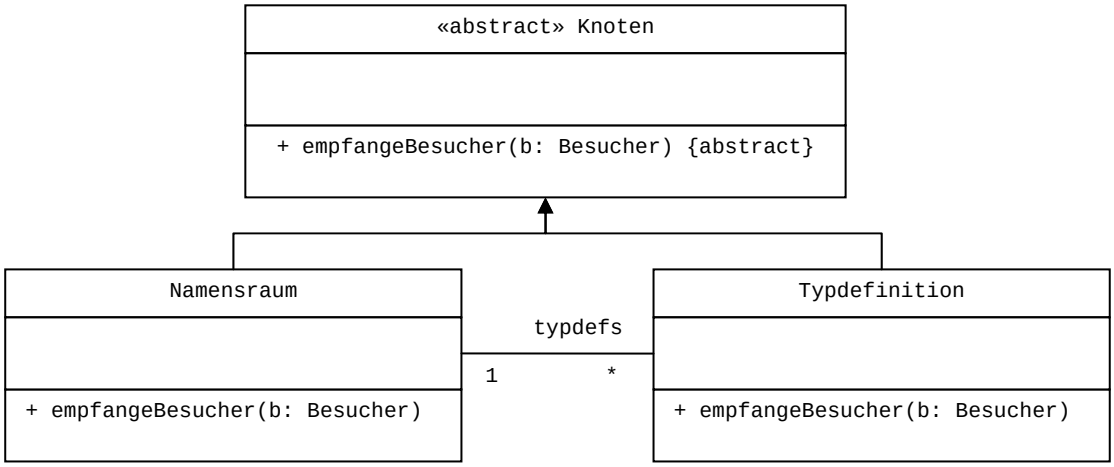
```
void erstelleWörterbuch(String sprache1, String sprache2)
void speichereWörterbuch(String dateiName)
void ladeWörterbuch(String dateiName)

void speichereKombination(String wortSprache1,
                          String wortSprache2)
void löscheKombination(String wortSprache1,
                       String wortSprache2)
String[] übersetze(String wort, int sprache)
int vokabular(int sprache)
String sprache(int sprache)
```

„übersetze“ erwartet im Parameter „sprache“ die Nummer der Sprache des „wort“-es („1“ oder „2“). Falls keine Übersetzung gefunden wurde, gibt die Funktion „übersetze“ ein leeres Feld zurück. Das gleiche gilt für den Fall, dass eine ungültige Sprach-Nummer im Parameter „sprache“ übergeben wurde.
Die Funktion „vokabular“ gibt bei einer ungültigen „sprache“ eine -1 zurück. Die Funktion „sprache“ gibt die String-Repräsentation der Sprache mit der Nummer des Parameters zurück und eine null-Referenz, falls diese ungültig war.

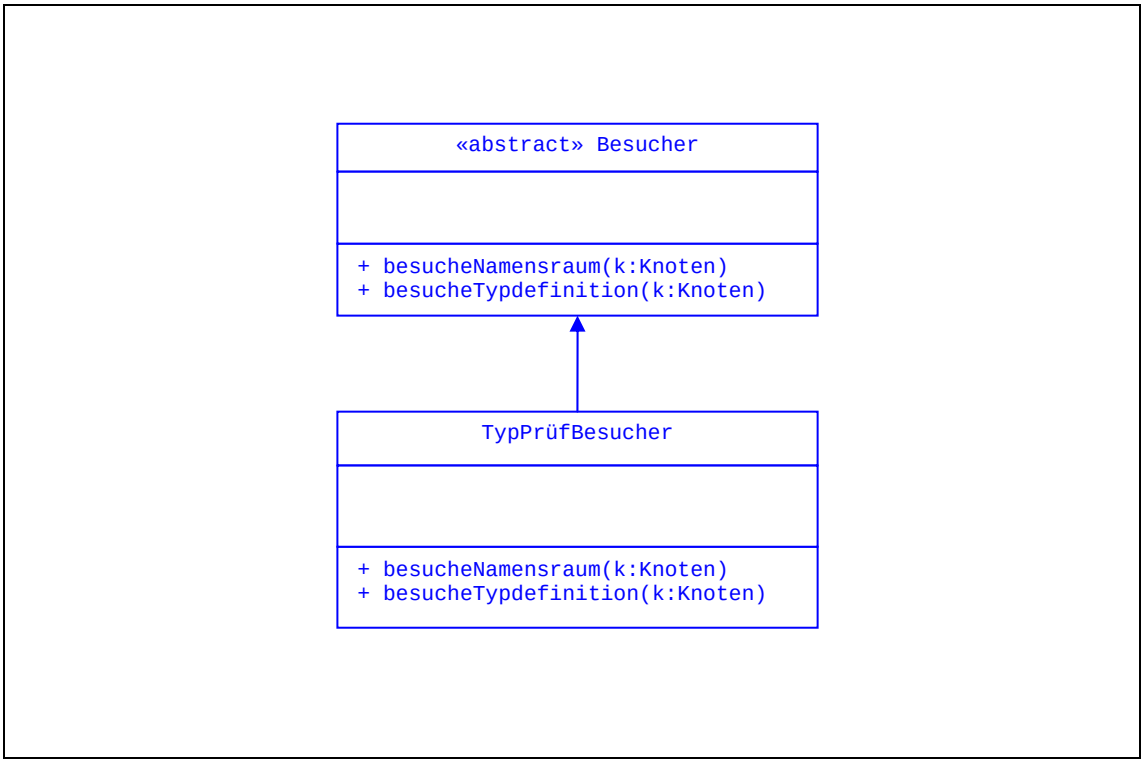
Aufgabe 3: Entwurfsmuster (2+3+3 = 8P)

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.

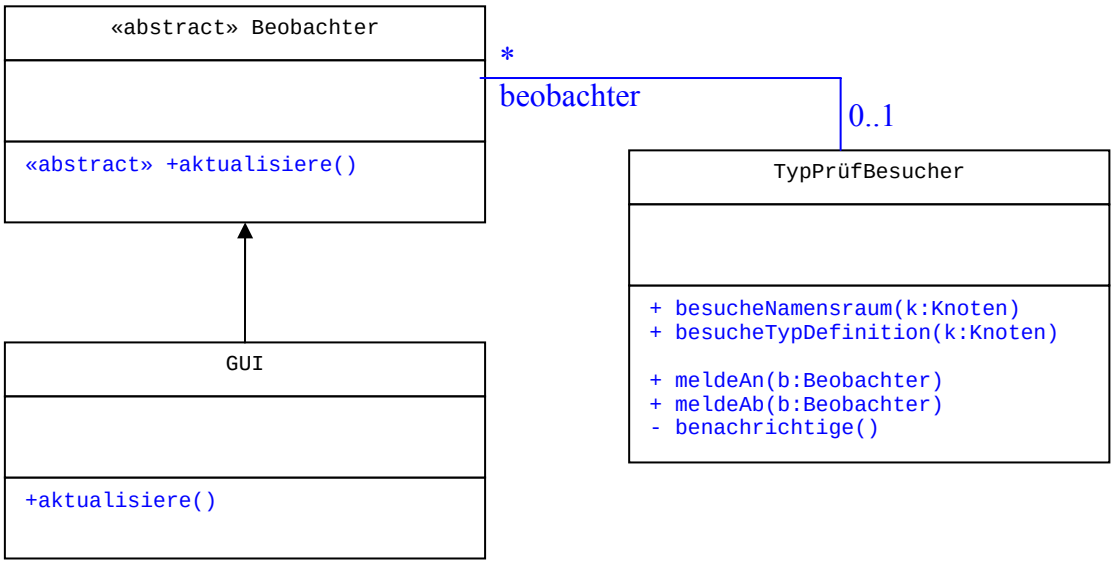
b.) Implementieren Sie die für die Traversierung der `Typdefinitionen` notwendige Methode `empfangeBesucher` in der Klasse `Namensraum`. Gehen Sie davon aus, dass `Typdefs` vom Typ `java.util.Vector` ist. Achten Sie auf Konsistenz mit den UML-Spezifikationen. (3 P)

```
public void empfangeBesucher(Besucher besucher) {
    besucher.besucheNamensraum(this);
    // Traversiere typedefs
    Iterator i = typedefs.iterator();
    while (i.hasNext())
        ((TypDefinition)i.next()).empfangeBesucher(besucher);
}
```

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

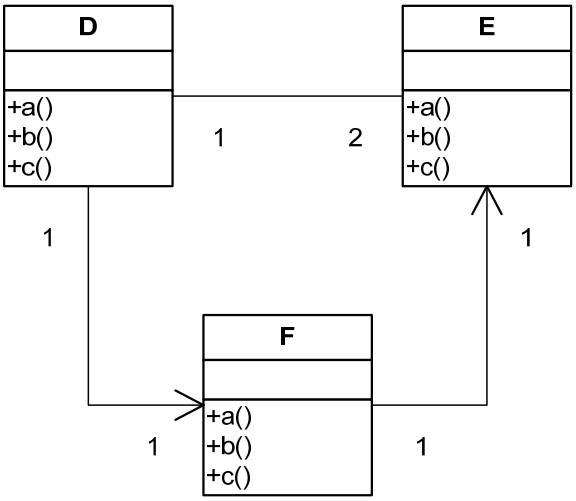
c.) Erweitern Sie unter Verwendung des Entwurfsmusters *Beobacher* die nachfolgend gegebenen Klassen `Beobachter`, `GUI` und `TypPrüfBesucher` um die für das Entwurfsmuster *Beobachter* notwendigen Methoden und Assoziationen. Spezifizieren Sie dabei die vollständigen Methodensignaturen (Sichtbarkeit,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

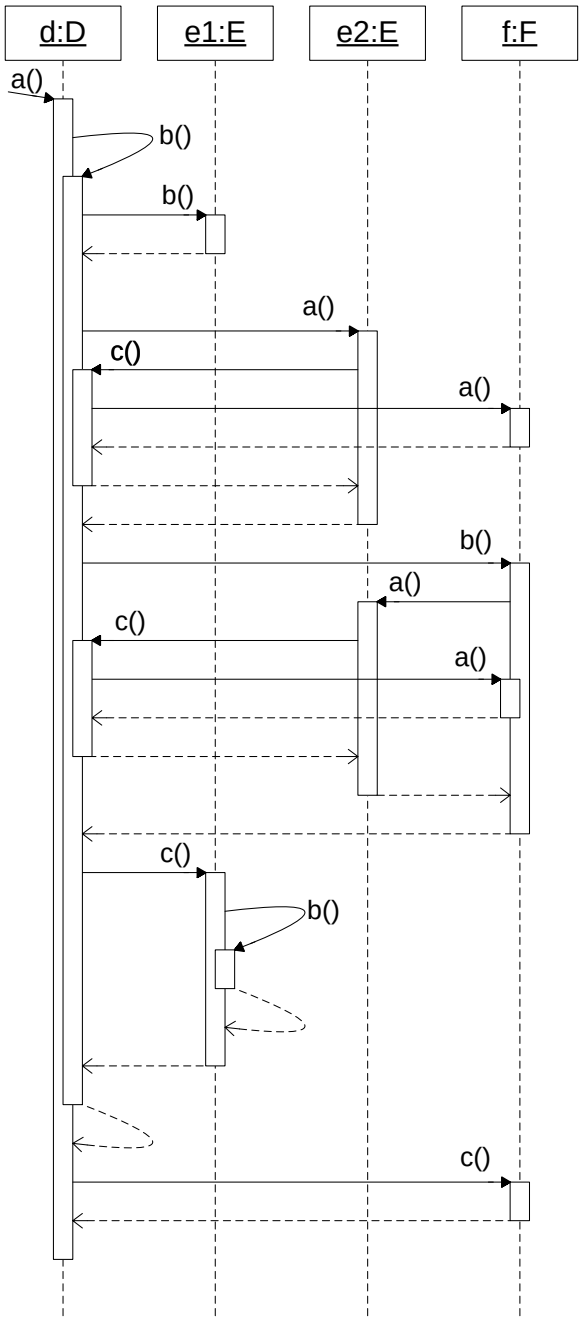
Hinweis: Sie brauchen die Methodensignaturen aus Aufgabenteil a) nicht erneut angeben.



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
    E e1;
    E e2;
    F f;
    // Konstruktor
    public D() {
        e1 = new E(this);
        e2 = new E(this);
        f = new F(e2);
    }
    // Methoden
    public void a() {
        b();
        f.c();
    }
    public void b() {
        e1.b();
        e2.a();
        f.b();
        e1.c();
    }
    public void c() {
        f.a();
    }
}
```

```
class E {
    // Instanzvariablen
    D d;

    // Konstruktor
    public E(D d) {
        this.d = d;
    }

    // Methoden
    public void a() {
        d.c();
    }

    public void b() {
        // nicht spezifiziert
    }

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

```
class F {
    // Instanzvariablen
    E e;

    // Konstruktor
    public F(E e) {
        this.e = e;
    }

    // Methoden
    public void a() {
        // nicht spezifiziert
    }

    public void b() {
        e.a();
    }

    public void c() {
        // nicht spezifiziert
    }
}
```

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

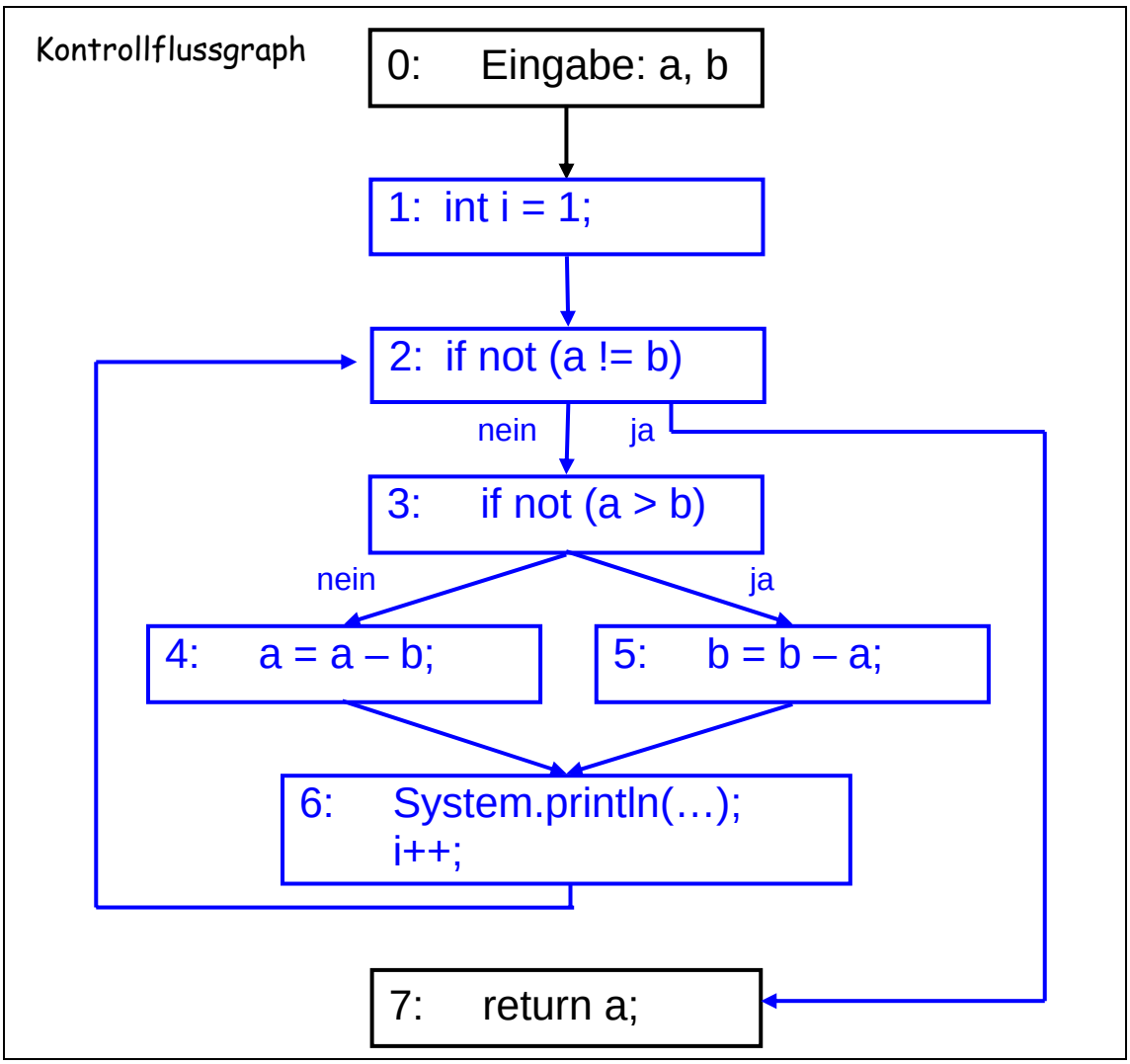
```
10:  int i = 1;
20:  if not (a != b) goto 100;
30:  if not (a > b) goto 60;
40:  a = a - b;
50:  goto 70;
60:  b = b - a;
70:  System.out.println(i+": a="+a+" b="+b);
80:  i++;
90:  goto 20;
100: return a;
```

Grob: 2P für "for", 2P für "if" und 2P für den Rest

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)

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



d.) Nennen Sie eine Menge von Belegungen der Eingabewerte (a, b), mit der das Kriterium der Zweigüberdeckung 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)

(2 ,3) [oder (2, 1), (1, 2)]

Pfad: 0, 1, 2, 3, 5, 6, 2, 3, 4, 6, 2, 7
Begründung (Beispiel): Alle Verzweigungen: [(2, 3), (2, 7) und (3, 4), (3, 5)] werden durchlaufen, die entsprechenden Teilpfade sind in dem oben angegeben Pfad enthalten.

Punkte **nur** mit Begründung.