Klausur Softwaretechnik

04.09.2008

Prof. Dr. Walter F. Tichy
Dipl.-Inform. T. Gelhausen
Dipl.-Inform. A. Paar

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.
Mit Bleistift oder roter Farbe geschriebene Angaben werden nicht bewertet.

Aufgabe	**1**	**2**	**3**	**4**	**5**	∑
Maximal	17	12	13	12	6	**60**
K1						
K2						
K3						

Aufgabe 1: Aufwärmen (17P)

a.) Nennen Sie zwei der in der Vorlesung genannten Anforderungen an marktreife Software. (1P)

Funktionstreue,
Qualitätstreue,
Kostentreue, Termintreue

b.) Nennen Sie drei Prinzipien der Softwaretechnik, die die Entwicklung komplexer Software ermöglichen. (1,5P)

Hierarchisierung,
Modularisierung,
Strukturierung, (Abstraktion)

c.) Nennen Sie zwei Ergebnisse (Dokumente) der Planungsphase. (1P)

Durchführbarkeitsstudie,
Lastenheft,
Projektkalkulation, Projektplan

d.) Gegeben ist folgendes Java-Programm.

```
class A {
  public static void main(String[] args) {

    Object[] arrObjects = new String[5];

    arrObjects[0] = new A();

    System.out.println(arrObjects[0]);
  }
}
```

Kreuzen Sie die Ausgabe des obigen Java-Programms an. (1P)

❑ `null` ❑ `A` ❑ `java.lang.ArrayStoreException`

e.) Nennen Sie zwei Kriterien mit denen Anforderungen an Software validiert werden. (1P)

Korrektheit, Vollständigkeit,
Konsistenz, Realisierbarkeit, Rückverfolgbarkeit

f.) Vervollständigen Sie die folgende Definition für ein Modul. (1P)

Ein Modul ist eine Menge von Programmkomponenten, die nach dem Geheimnisprinzip

…… gemeinsam entworfen

…… und geändert werden

g.) Für welche vier Eigenschaften steht der Begriff „ACID"-Prinzip? (2P)

A: …….. Atomicity

C: …….. Consistency

I: …….. Durability

D: …….. Isolation

h.) Ein kontrollflussorientiertes Testverfahren für Kriterium x subsumiert ein Testverfahren für Kriterium y, wenn jede Menge von Pfaden, die Kriterium x erfüllt, auch Kriterium y erfüllt. Welche kontrollflussorientierten Teststrategien werden vom „Boundary-Interior Pfadtest" subsumiert, welche von der „Minimal-mehrfachen Bedingungsüberdeckung"? (2,5P)

Der „Boundary-Interior Pfadtest" subsumiert die Zweig-

und die Anweisungsüberdeckung .

Die „Minimal-mehrfache Bedingungsüberdeckung" subsumiert die

einfache Bedingungsüberdeckung, die

Zweig- und die

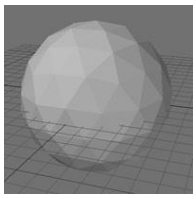
Anweisungsüberdeckung .

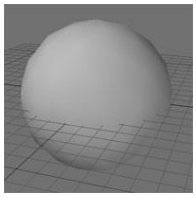
i.) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. (6P)

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	Software ist leichter zu ändern als ein physisches Produkt vergleichbarer Komplexität.
	X	Die Anforderungserhebungstechnik „Introspektion“ ist im Vergleich zur „Ethnographie“ besonders zeitaufwendig und teuer.
	X	Funktionale Attribute spezifizieren wie gut die Software ihre Funktionen erfüllt.
X		Durch Werkzeuge kann die Einhaltung von Methoden, Verfahren, Standards und Notationen erhöht werden.
X		In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht.
	X	Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll.
X		Bei Gleichheit 0. Stufe handelt es sich um identische Objekte.
	X	Das Entwurfsmuster „Abstrakte Fabrik“ konzentriert sich auf den **schrittweisen** Konstruktionsprozess komplexer Objekte.
	X	Im Gegensatz zu „Black Box“-Tests beweisen „White Box“-Tests die Korrektheit eines Programms.
X		Zusicherungen (z.B. mit dem Schlüsselwort **assert** in Java) werden **zur Laufzeit** eines Programs ausgeführt.
	X	In Java muss eine abstrakte Klasse, die eine Schnittstelle implementiert, alle in der Schnittstelle vorgegebenen Methoden implementieren.
X		Regressionstests helfen verhindern, dass alte Fehler wieder auftreten.

Aufgabe 2: Entwurfsmuster (Brücke und Beobachter) (12P)

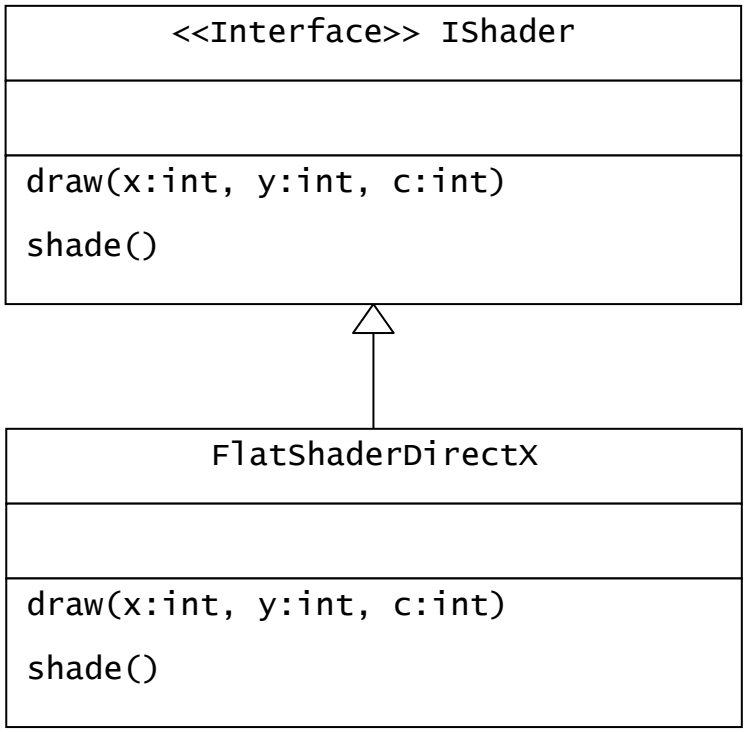




Der Begriff **Shading** bezeichnet in der 3D-Computergrafik im allgemeinen Sinne die Simulation der Oberfläche eines Objekts. Für die Berechnung der Oberfläche gibt es viele verschiedene Verfahren – unter anderem das **Flat Shading** (Abb. links oben) und das **Gouraud Shading** (Abb. links unten).

Sie haben einen Flat Shader für die DirectX-Programmierschnittstelle entsprechend dem folgenden UML-Klassendiagramm entworfen.

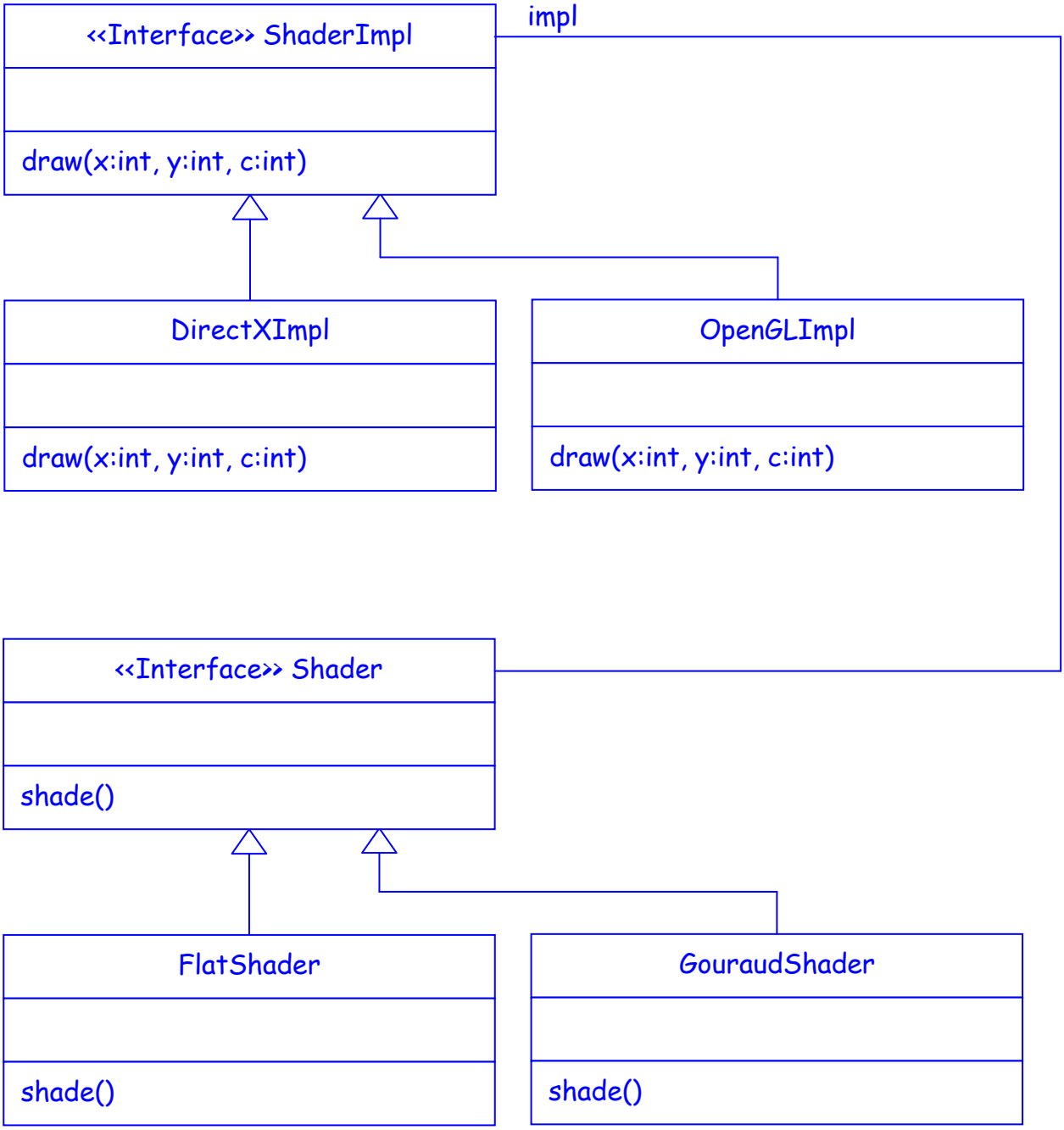
Die Methode `shade()` in der Klasse `FlatShaderDirectX` implementiert den Flat Shading-Algorithmus und verwendet die Methode `draw(x:int, y:int, c:int)`, um einen Punkt unter Verwendung der DirectX-Programmierschnittstelle zu zeichnen.



Neben Flat Shading möchten Sie zusätzlich noch das Gouraud Shading-Verfahren anbieten. Außerdem möchten Sie beide Shading-Verfahren sowohl für die DirectX-API als auch für die OpenGL-Programmierschnittstelle implementieren.

a.) Verwenden Sie das Entwurfsmuster **Brücke** um die Abstraktion (Methode `shade`) von der Implementierung (Methode `draw`) zu trennen.

Hinweis: Trennen Sie die obige Schnittstelle `IShader` geeignet auf und erweitern Sie Ihr UML-Klassendiagramm um die konkreten Klassen `DirectXImpl` und `OpenGLImpl`, `FlatShader` und `GouraudShader`. Tragen Sie Vererbungsbeziehungen und Assoziationen entsprechend dem Entwurfsmuster Brücke ein. (7P)

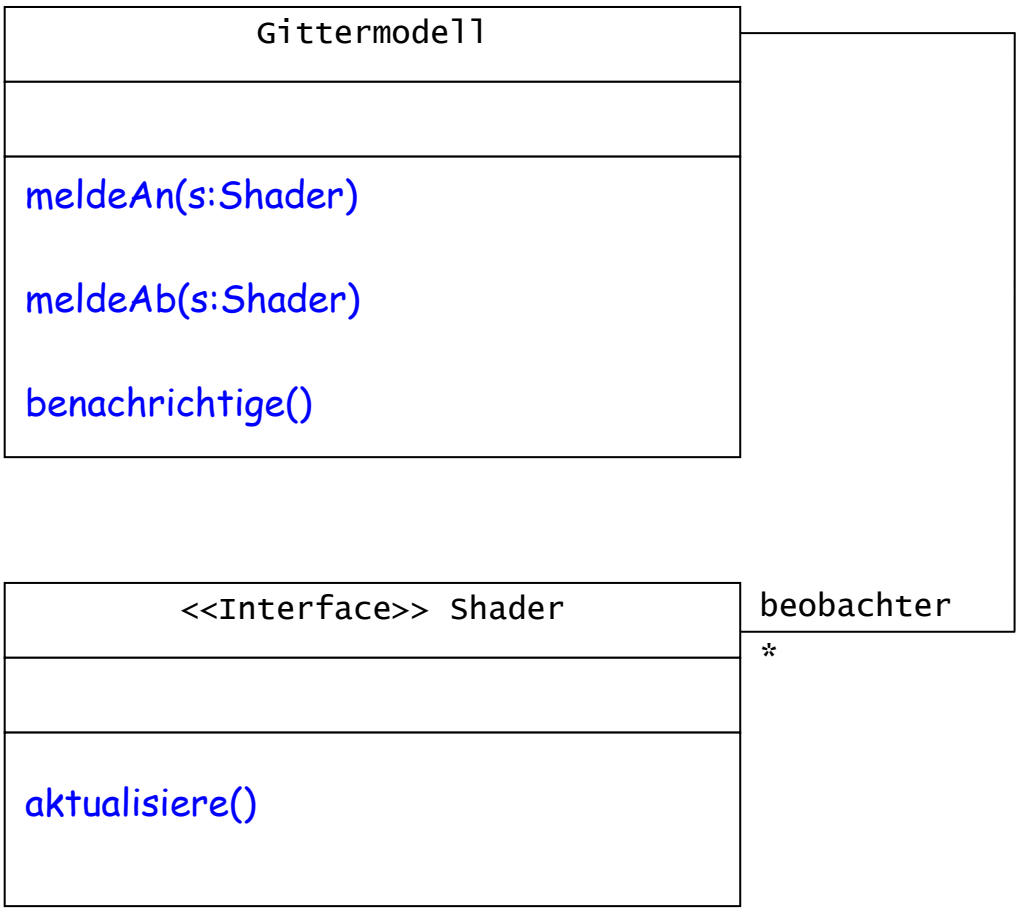


b.) Geben Sie nachfolgend in Java-Notation an wie – entsprechend dem geforderten Entwurf mit dem Entwurfsmuster **Brücke** – die Methode `shade` im `FlatShader` einen schwarzen Punkt (c = 0) mit den Koordinaten x=23 und y=42 zeichnen kann. (1P)

```
impl.draw(23, 42, 0);
```

c.) Das Gittermodell, für das Ihre Shader-Klassen die Darstellungen der Oberflächen berechnen, wird von der Klasse **`Gittermodell`** verwaltet. Bei jeder Änderung des Gittermodells (Subjekt) soll der verwendete **`Shader`** (Beobachter) benachrichtigt werden. Verwenden Sie das Entwurfsmuster **Beobachter** und erweitern Sie das folgende UML-Klassendiagramm um Methoden-Signaturen für das An- und Abmelden, Benachrichtigen und Aktualisieren des Beobachters. (3P)

Hinweis: Definieren Sie keine zusätzlichen abstrakten Subjekt- und Beobachterklassen sondern verwenden und erweitern Sie lediglich die gegebene Schnittstelle **`Shader`** und die Klasse **`Gittermodell`**.



d.) Geben Sie eine Java-Datenstruktur für die **`beobachter`**-Assoziation in obigem UML-Klassendiagramm an. Geben Sie für diese Datenstruktur in Java-Notation die Implementierung für das Benachrichtigen angemeldeter **`Shader`** an. (1P)

```
Vector beobachter;
for (Shader s : beobachter) {
  s.aktualisiere();
}
```

Aufgabe 3: Abbildung von Zustandsautomaten (13P)

a) Vergleichen Sie die implizite und explizite Speicherung des Zustandes eines Objektes. Geben Sie Definitionen und drei Vor- bzw. Nachteile der Verfahren an. (5P)

Definition der impliziten Speicherung:

Implizit: Zustand wird (aus Attributwerten) berechnet. (1P)

Definition der expliziten Speicherung:

Explizit: Zustand wird in einer dedizierten Instanzvariablen gespeichert. (1P)

Vor-/Nachteile:

Vorteile Implizit (=Nachteile Explizit):
Keine zusätzlichen Instanzvariablen nötig (spart Speicherplatz)
Nachteile Implizit (=Vorteile Explizit):
Zustand muss jedes Mal neu berechnet werden (kostet ggf. Rechenzeit), Übergangsfunktion implizit, weniger offensichtlich, Vergessen bei Änderungen, Ggf. komplizierter, Methode nicht immer möglich

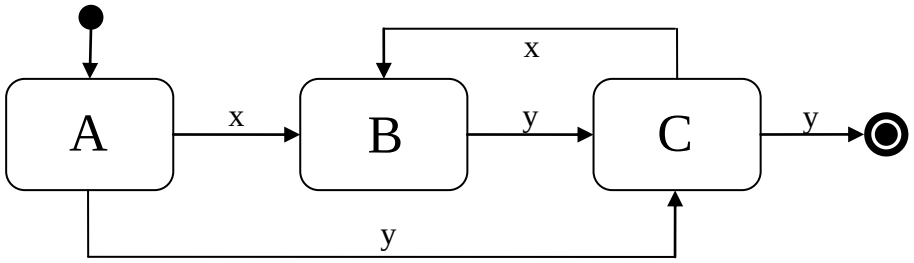
(je Vorteil/Nachteil 1P, max. 3P)

b) Wie realisieren wir in der Vorlesung die **ausgelagerte explizite** Speicherung des Zustands eines Objektes? Skizzieren Sie grob die Idee. (2P)

„State Pattern"

Objekt kennt nur seinen Zustand (1P) und delegiert alle Nachrichten an das jeweilige Zustands-Objekt (1P)

c) Realisieren Sie folgenden Zustandsautomaten für die Klasse **P** als **eingebettete explizite Speicherung** mit Java-Code. (6P)



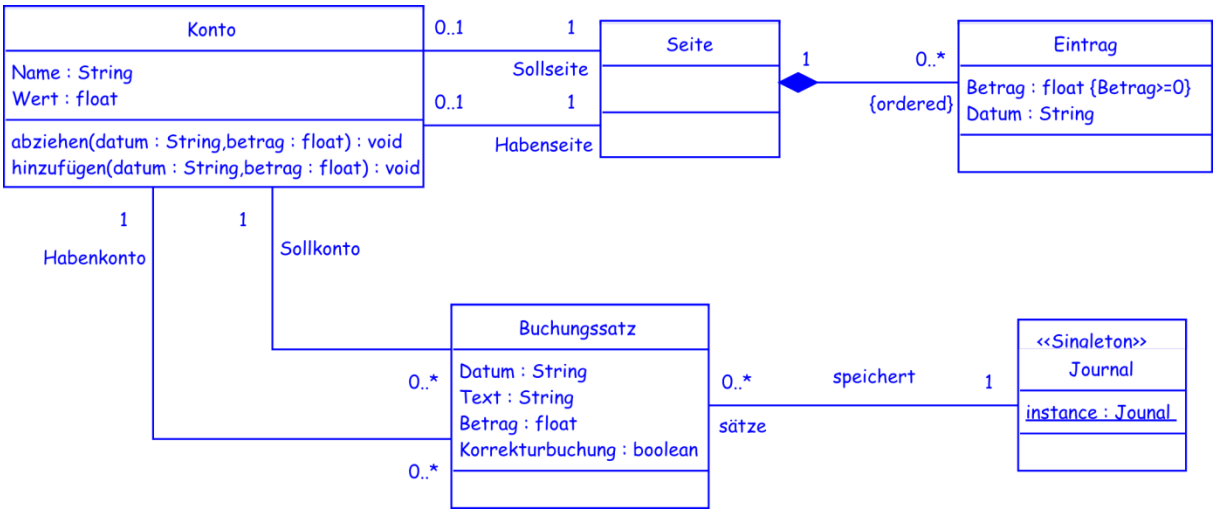
Instanzvariable für den Zustand (0,5P)
Zustandsvariable mit Zustand A initialisiert (0,5P)
Ein Endzustand wurde eingeführt (0,5P)
Methoden x und y (diese Namen!) angelegt (0,5P)
Korrekte Behandlung (Abfrage und Übergang) von A in x (0,5P)
Korrekte Behandlung (Abfrage und Fehler) von B in x (0,5P)
Korrekte Behandlung (Abfrage und Übergang) von C in x (0,5P)
Korrekte Behandlung (Abfrage und Fehler) vom Endzustand in x (0,5P)
Korrekte Behandlung (Abfrage und Übergang) von A in y (0,5P)
Korrekte Behandlung (Abfrage und Übergang) von B in y (0,5P)
Korrekte Behandlung (Abfrage und Übergang) von C in y (0,5P)
Korrekte Behandlung (Abfrage und Fehler) vom Endzustand in y (0,5P)

Aufgabe 4: Objektorientierte Analyse & Entwurf (12P)

a) Modellieren Sie folgende Domänenbeschreibung der doppelten Buchführung möglichst genau in einem einzigen UML-Klassendiagramm. (9P)

Hinweise: Jeder Satz erzeugt mindestens ein Modellelement. Modellieren Sie nichts, das nicht im Text erwähnt wird. Modellieren Sie keine Sichtbarkeiten. Geben Sie aber überall explizit Multiplizitäten und Assoziations- und Rollennamen an. Für die Deklaration von Attributen dürfen Sie nur die in Java verfügbaren Typen verwenden.

Die Konten der doppelten Buchführung haben jeweils einen Namen und einen aktuellen Wert. Jedes Konto hat genau eine Soll- und eine Habenseite und kann sowohl als Sollkonto und als auch als Habenkonto verwendet werden. Die Soll- und die Habenseite sind geordnete Listen von Einträge aus jeweils einem Betrag und einem Datum. Dieser Betrag muss immer positiv sein! Jeder Geschäftsvorfall erzeugt einen Buchungssatz, der jeweils aus einem Datum, einem Buchungstext, dem Sollkonto, dem Habenkonto und dem Betrag besteht. Sollkonto ist das Konto, bei dem der Betrag zum angegebenen Datum abgezogen wird, Habenkonto das, bei dem der Betrag zum angegebenen Datum hinzugefügt wird. Alle Buchungssätze werden im selben, zentralen Journal gespeichert.



b) Für die Korrektur eines Buchungsfehlers erzeugt das Programm einen neuen, zusätzlichen Buchungssatz im Journal, der dem zu korrigierenden Buchungssatz exakt entspricht, außer dass Sollkonto und Habenkonto vertauscht sind. Geben Sie einen OCL-Ausdruck an, der sicherstellt, dass zu jeder Korrekturbuchung im Journal (mindestens) eine normale Buchung existiert und die zuvor genannten Bedingungen gelten. Ihr OCL-Ausdruck muss zu ihrem Modell passen – erweitern Sie daher Ihr Modell aus Aufgabenteil a) nach Bedarf. (3P)

```
context Journal inv:
  self.sätze->forAll(k | k.Korrekturbuchung=true implies
    self.sätze->exists( b|b.Korrekturbuchung=false
    and b.Datum=k.Datum and b.Text=k.Text and b.Betrag=k.Betrag
    and b.Habenkonto=k.Sollkonto and b.Sollkonto=k.Habenkonto))
```

Aufgabe 5: Kontrollflussorientierte Testverfahren (6P)

Gegeben sei nachfolgender Code:

```
void drawLineH( int x0, int x1, int y0, int y1 ) {
   int dy = y1 - y0;
   int dx = x1 - x0;
   int stepx = 1;
   int stepy = 1;
   if( dy < 0 ) { dy = -dy; stepy = -1; };
   if( dx < 0 ) { dx = -dx; stepx = -1; };
   dy = dy*2;
   dx = dx*2;
   int fraction = dy - (dx/2);
   while( x0 != x1 ) {
      if( fraction >= 0 ) { y0 += stepy; fraction -= dx; }
      x0 += stepx;
      fraction += dy;
      setPixel( x0, y0 );
   }
}
```

Hinweis: Wenn man der Methode `setPixel()` ungültige Werte übergibt, passiert nichts, sie ignoriert den Aufruf einfach (keine Ausnahme).

a) Bestimmen Sie die Anzahl der Äquivalenzklassen die benötigt werden, um die Schleife boundary-interior zu testen (2P, keine Punkte ohne Rechenweg/Begründung!)

4 Pfade außerhalb der Schleife, 2 für Grenztest vs. Interieurtest, 2 Schleifenquerer, 1 Weg, die Schleife zu verlassen (da weder setPixel() noch eine der Rechenoperationen in der Schleife Ausnahmen produzieren) ⇒ 4 x 2 x 2 x 1 = 16 Äquivalenzklassen

b) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. (4P)

Hinweis: Jedes korrekte Kreuz zählt 1 Punkt, jedes falsche Kreuz bewirkt 1 Punkt Abzug. Die Teilaufgabe wird mindestens mit 0 Punkten bewertet.

Wahr	Falsch	Aussage
	X	Die Werte x_0=1, x_1=4, y_0=1, y_1=4 führen zu Anweisungsüberdeckung.
	X	Die Werte x_0=5, x_1=1, y_0=4, y_1=1 führen zu Zweigüberdeckung.
X		Die Werte x_0=1, x_1=2, y_0=1, y_1=1 führen zu einem Grenztest der Schleife.
X		Die Werte x_0=1, x_1=3, y_0=2, y_1=1führen zu einem Interieurtest der Schleife.