Klausur Softwaretechnik I

06.08.2009

Prof. Dr. Walter F. Tichy
Dipl.-Inform. A. Höfer
Dipl.-Inform. D. Meder

Musterlösung

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	16	13	10	13	8	60
K1						
K2						
K3						

Aufgabe 1: Aufwärmen (16 P)

a) Nennen Sie die in der Vorlesung besprochenen sechs Phasen der Softwareentwicklung. (3 P)

- Planungsphase — Je Phase 0,5 P
- Definitionsphase
- Entwurfsphase
- Implementierungsphase
- Test- und Abnahmephase
- Einsatz- und Wartungsphase

b) Nennen Sie **jeweils** zwei in der Vorlesung besprochene Entwurfsmuster der Kategorien Entkopplungsmuster und Variantenmuster und ordnen Sie die genannten Entwurfsmuster der entsprechenden Kategorie zu. (2 P)

Entkopplungsmuster **Variantenmuster**

Je Muster mit korrekter Zuordnung 0,5 P

- Entkopplungsmuster:
  Adapter, Beobachter, Brücke, Iterator, Stellvertreter, Vermittler
- Variantenmuster:
  Abstrakte Fabrik, Besucher, Erbauer, Fabrikmethode, Kompositum, Schablonenmethode, Strategie, Dekorierer

c) Nennen Sie die vier in der Vorlesung besprochenen Möglichkeiten, in Java Beobachter (Listener) zu implementieren. (2 P)

Pro genannter Möglichkeit 0,5 P

- Eigene Beobachter-Klasse
- Selbstbeobachtendes Steuerelement
- Adapter-Klassen
- Anonyme Klassen

d) Erklären Sie die beiden Begriffe „Striktes Ausbuchen“ und „Optimistisches Ausbuchen“ im Kontext einer Konfigurationsverwaltung. Nennen Sie **jeweils** einen Vor- sowie einen Nachteil. (4 P)

Pro genanntem Punkt 0,5 P

Striktes Ausbuchen
- Nur eine Ausbuchung gleichzeitig ist erlaubt
- Ausbucher hat exklusives Änderungsrecht
- Vorteil: kein Verschmelzungsaufwand beim Zurückschreiben
- Nachteil: immer nur einer kann eine Version ändern

Optimistisches Ausbuchen
- Mehrere Ausbuchungen gleichzeitig erlaubt
- Mehrere Entwickler Arbeiten an der gleichen Programmversion
- Vorteil: Mehrere Entwickler können eine Version ändern
- Nachteil: Aufwand beim Zusammenführen der Versionen (der Schnellere gewinnt)

e) Laut der Metastudie von Dybå et al. hat die Paarprogrammierung im Vergleich zur Einzelprogrammierung (einen) Vorteil(e) bei… (1 P)

Qualität und Dauer

und (einen) Nachteil(e) bei… (1 P)

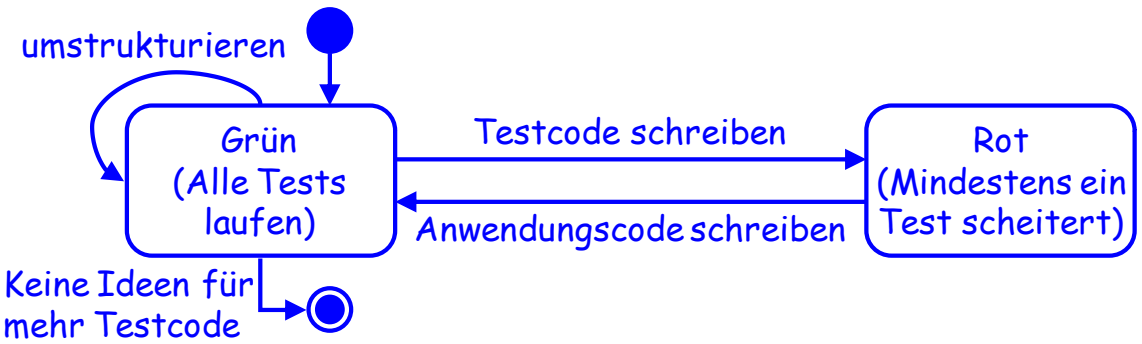
Aufwand (Kosten)

f) Erklären Sie den Unterschied zwischen den JUnit 4-Annotationen **`@Before`** und **`@BeforeClass`**. (1 P)

Mit @BeforeClass annotierte Methoden werden einmalig für alle Testfälle einer Testklasse ausgeführt (0,5 P) - mit @Before annotierte Methoden vor jedem Testfall einer Testklasse (0,5 P). (Beide dienen zum Aufbau von Testressourcen.)

g) Zeichnen Sie das idealisierte Zustandsdiagramm der Testgetriebenen Entwicklung. (2 P)

Pro richtigem Zustand mit Übergängen 0,5 P.



Aufgabe 2: Kontrollflussorientierte Testverfahren (13 P)

a) Kreuzen Sie an, ob die Aussage wahr oder falsch ist. Jedes richtige Kreuz gibt einen halben Punkt, für jedes falsche Kreuz wird ein halber Punkt abgezogen. Die Aufgabe wird mindestens mit 0 Punkten bewertet. (2 P)

Aussage	Wahr	Falsch
Die Anweisungsüberdeckung subsumiert die einfache Bedingungsüberdeckung.		x
Die Zweigüberdeckung subsumiert die minimal-mehrfache Bedingungsüberdeckung.		x
Die Pfadüberdeckung subsumiert die Anweisungsüberdeckung.	x	
Bei der mehrfachen Bedingungsüberdeckung ist die Größe der minimalen Testfallmenge unabhängig davon, ob Kurzauswertung vorgenommen wird oder nicht.	x	

b) Beschreiben Sie, wie man eine **do-while**-Schleife in die strukturerhaltende Zwischensprache aus der Vorlesung überführen kann. Geben Sie zur Veranschaulichung das entstehende Zwischensprachprogramm für unten stehende **do-while**-Schleife an. (2 P)

```
do {
  statements;
} while (condition);
```

Zwischensprache:
(01: -)
02: statements;
03: if (condition) goto 02
1 P ganz oder gar nicht

1. do-Zeile entfernen
2. Befehle „statements" der Schleife übernehmen
3. "while (condition)" in „if (condition) goto 1. Befehl in der Schleife" umwandeln (Keine Negation der Bedingung).

1 P (0,5 P wenn 2 von 3 Punkten erwähnt wurden)

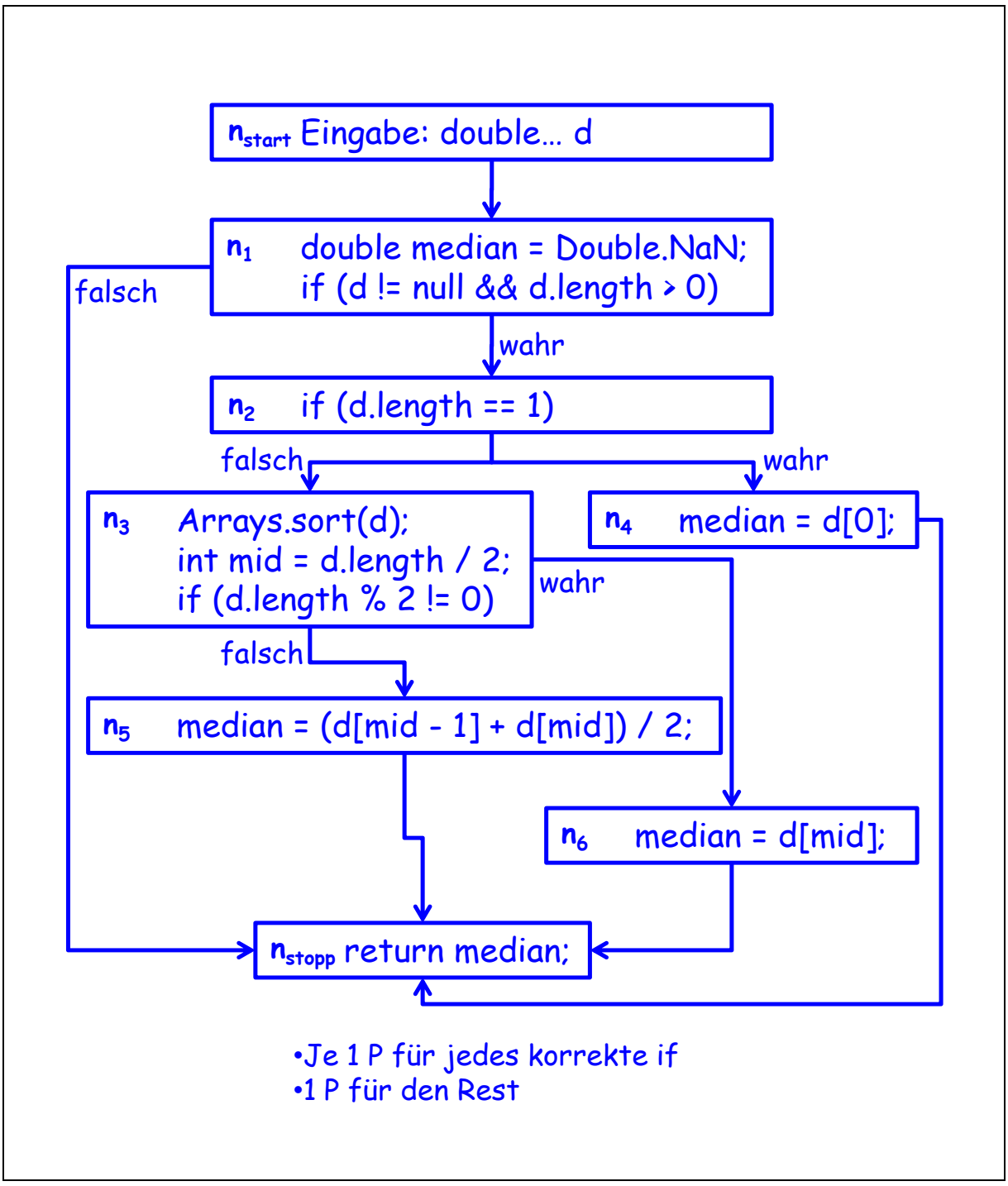
Gegeben sei folgende Java-Methode:

```
01 public static double median(double[] d) {
02   double median = Double.NaN;
03   if (d != null && d.length > 0)
04     if (d.length == 1) {
05        median = d[0];
06     } else {
07       Arrays.sort(d);    // sortiert d aufsteigend
08       int mid = d.length / 2;
09       if (d.length % 2 != 0) {
10         median = d[mid];
11       } else {
12         median = (d[mid - 1] + d[mid]) / 2;
13       }
14     }
15   }
16   return median;
17 }
```

c) Begründen Sie: Was wäre die Folge, wenn man das **&&** in Zeile 3 durch ein **&** ersetzt? (1 P)

Keine Kurzauswertung (0,5 P) → Bei null als Eingabe gäbe es eine NullPointerExeption bei d.length (0,5 P).

d) Erstellen Sie auf der folgenden Seite den Kontrollflussgraphen der Methode **median(…)**. Bitte schreiben Sie den Quelltext in die Kästchen, Verweise auf die Zeilennummern der Methode sind nicht ausreichend. (4 P)



e) Geben Sie eine minimale Testfallmenge an, welche für die Methode **`median(…)`** die Anweisungsüberdeckung erfüllt. Geben Sie die durchlaufenen Pfade an. (2 P)

{1}	nstart, n1, n2, n4, nstopp
{1, 2}	nstart, n1, n2, n3, n5, nstopp
{1, 2, 3}	nstart, n1, n2, n3, n6, nstopp

1 P wenn korrekt aber nicht minimal.

f) Ergänzen Sie die Testfallmenge aus e) so, dass Sie eine minimale Testfallmenge erhalten, welche die Zweigüberdeckung für die Methode **`median(…)`** erfüllt. Geben Sie für die neuen Testfälle die durchlaufenen Pfade an. (1 P)

null oder {} nstart, n1, nstopp
0,5 P wenn korrekt aber nicht minimal.

g) Erfüllt die minimale Testfallmenge aus f), welche die Zweigüberdeckung erfüllt, auch die Pfadüberdeckung für die Methode **`median(…)`**? Begründen Sie Ihre Antwort. (1 P)

Ja, es gibt in diesem Fall so viele Pfade wie Zweige (allgemein gilt das nicht). (1 P ganz oder gar nicht)

Aufgabe 3: Modellierung mit UML (10 P)

a) Erklären Sie den Unterschied zwischen einer Aggregation und einer Komposition im Kontext von UML. (1 P)

Aggregation:
- Teil-Ganzes-Beziehung (0,5 P)

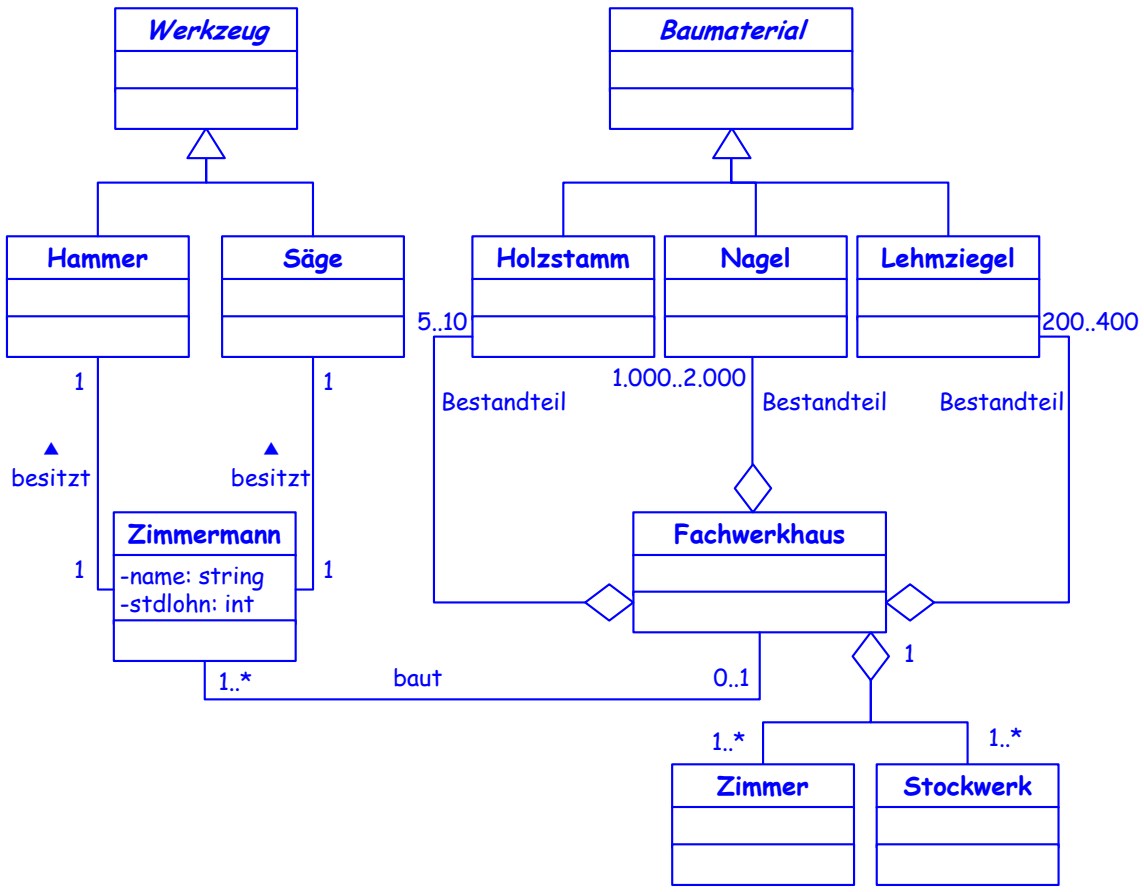
Komposition:
- strenger als Aggregation, Teile haben keine Daseinsberechtigung ohne das Ganze (0, 5 P)

b) Gegeben sei folgendes Szenario:

Ein Fachwerkhaus besteht aus 5 bis 10 Holzstämmen, 200 bis 400 Lehmziegeln sowie 1.000 bis 2.000 Nägeln. Jedes Baumaterial, egal ob Holzstamm, Lehmziegel oder Nagel, ist Bestandteil in genau einem Fachwerkhaus. Jedes Fachwerkhaus hat eine bestimmte Anzahl an Zimmern und Stockwerken. Für den Bau eines Fachwerkhauses ist mindestens ein Zimmermann zuständig, welcher einen Namen sowie einen individuellen Stundenlohn besitzt. Zum Bau des Fachwerkhauses verwendet jeder Zimmermann sein eigenes Werkzeug, bestehend aus genau einem Hammer sowie genau einer Säge. Jeder Zimmermann kann an maximal einem Fachwerkhaus gleichzeitig bauen.

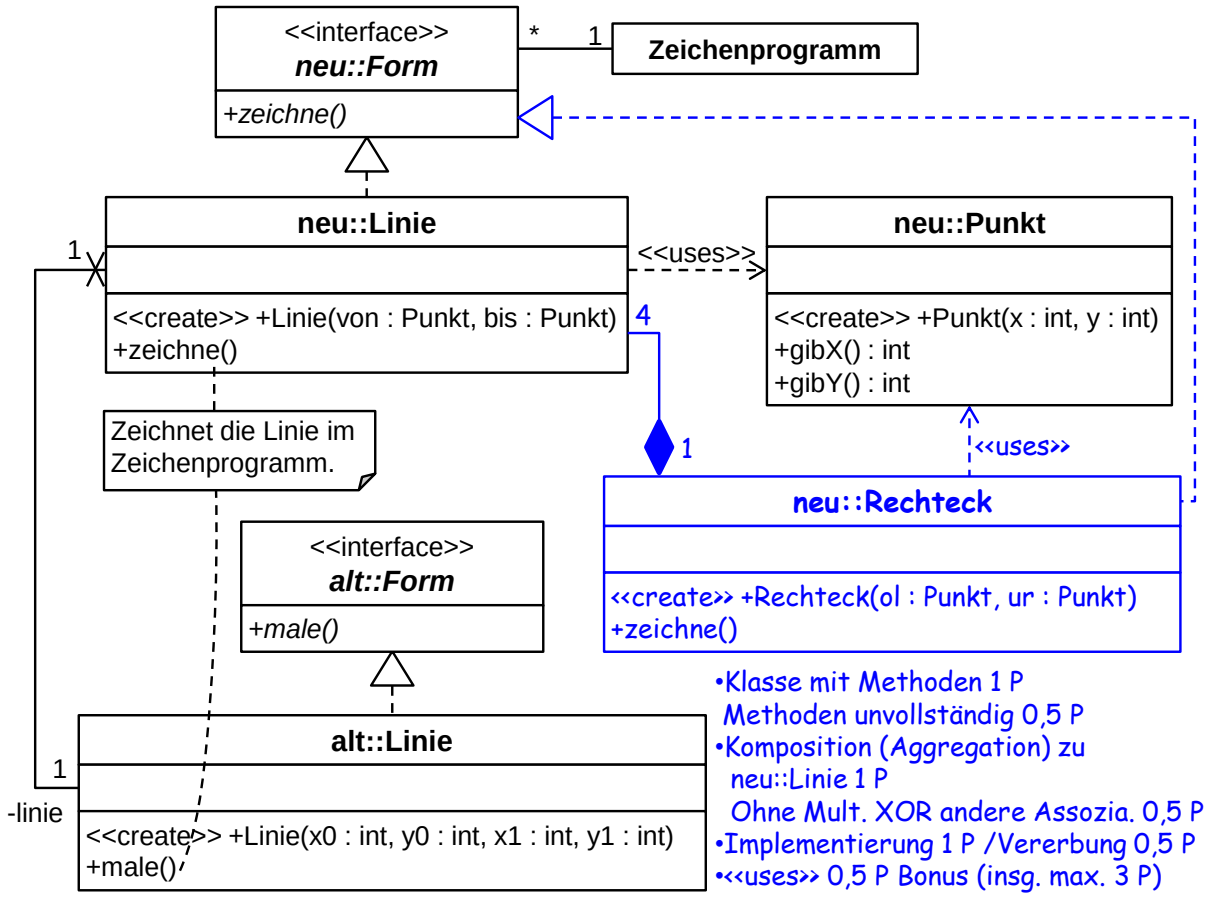
Modellieren Sie das Szenario möglichst vollständig als UML-Klassendiagramm. Modellieren Sie keine Methoden. Geben Sie Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen an. (9 P)

Korrekturschema: Siehe Anhang



Aufgabe 4: Entwurfsmuster (13 P)

Betrachten Sie das folgende Klassendiagramm:



a) Welches Entwurfsmuster wird hier verwendet? (1 P)

(Objekt-) Adapter

b) Weisen Sie den Klassen und Methoden aus dem obigen Klassendiagramm ihre Entsprechungen im verwendeten Entwurfsmuster zu. (3 P)

Zeichenprogramm	= Klient	jeweils 0,5 P
neu::Linie	= Adapter	
alt::Linie	= AdaptierteKlasse	
neu::Form	= Ziel	
zeichne()	= operation()	
male	= spezifischeOperation()	

c) Erweitern Sie das obige Klassendiagramm so, dass das Zeichenprogramm auch mit Rechtecken umgehen kann. Ein Rechteck besteht aus vier Linien und soll über seine linke obere und rechte untere Ecke definiert werden. Vermeiden Sie Abhängigkeiten zum Paket **`alt`**. Geben Sie alle Multiplizitäten an. (3 P)

d) Vervollständigen Sie die folgenden Implementierungen der Klassen **`neu::Linie`** und **`neu::Rechteck`** unter Berücksichtigung Ihrer Erweiterung aus c). Achten Sie dabei auf korrekte Java-Syntax. (6 P)

```
package neu;

public class Linie implements Form {          inkl. Rechteck 0,5 P
  private alt.Linie linie;                                    0,5 P

  public Linie(Punkt von, Punkt bis) {                        0,5 P
    linie = new alt.Linie(von.gibX(), von.getY(),
      bis.gibX(), bis.getY());                                0,5 P
  }

  public void zeichne() {                                     0,5 P
    linie.male();                                             0,5 P
  }
}

package neu;

public class Rechteck implements Form {                       s.o.
  private Linie[] seiten = new Linie[4];                      0,5 P
  private static final int OBEN = 0;
  private static final int RECHTS = 1;
  private static final int UNTEN = 2;
  private static final int LINKS = 3;

  public Rechteck(Punkt obenLinks, Punkt untenRechts) {       0,5 P
    Punkt obenRechts = new Punkt(untenRechts.gibX(),
      obenLinks.getY());
    Punkt untenLinks = new Punkt(obenLinks.gibX(),
      untenRechts.getY());                                    0,5P
    seiten[OBEN] = new Linie(obenLinks, obenRechts);
    seiten[RECHTS] = new Linie(obenRechts, untenRechts);
```

```
    seiten[UNTEN] = new Linie(untenRechts, untenLinks);
    seiten[LINKS] = new Linie(untenLinks, obenLinks);        0,5P
  }

  public void zeichne() {                                    0,5 P
    for (Linie l : seiten) {
      l.zeichne();
    }                                                        0,5 P
  }
}
```

Aufgabe 5: Parallelität (8 P)

a) Gegeben sei folgender Algorithmus zur Multiplikation zweier **n** × **n**-Matrizen **A** und **B**.

```
for (int k = 0; k < n; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Dieser Algorithmus soll unter Verwendung von Java-Threads parallelisiert werden. Dazu müssen die Matrizen unter den verfügbaren **z** Fäden aufgeteilt werden. Nehmen Sie an, Sie würden die Matrix **A** spaltenweise und die Matrix **B** zeilenweise aufteilen (Index **k**).

Welches Problem kann auftreten, wenn nur die äußere Schleife (Index **k**) parallelisiert wird? Wie kann dieses Problem behoben werden, wenn auch weiterhin nur die äußere Schleife parallelisiert werden soll? (2 P)

Gleichzeitiger Zugriff mehrerer Fäden auf ein einziges Element der Matrix C. → Wettlauf/Race Condition (1 P)
Sicherstellen, dass nur ein Faden zu einem Zeitpunkt den kritischen Abschnitt betreten kann. (1 P)

b) Geben Sie an, wie der **anfangs-** und **endIndex** für den Faden **f** sowie die Blockgröße berechnet werden, in Abhängigkeit von der Anzahl der verwendeten Fäden **z**. (1,5 P)

```
int blockgroesse   = (int) Math.ceil((double) n / z);
int anfangsIndex   = f * blockgroesse;        //pro korrekter Zeile 0,5 P
int endIndex       = Math.min((f + 1) * blockgroesse, n);
```

c) Ergänzen Sie folgende Methode an den durch Strichen gekennzeichneten Stellen so, dass das Produkt der **n** × **n**-Matrizen **A** und **B** unter Verwendung von **z** Fäden (**z > 1**) das korrekte Ergebnis liefert. Nehmen Sie an, dass die Matrix **A** spaltenweise und die Matrix **B** zeilenweise aufgeteilt wird (Index **k**). (2,5 P)

```
private int[][] matrixMult(int[][] A, int[][] B,
                           int anfangsIndex, int endIndex, int n) {
  int[][] C =  new int[n][n]; //pro korrekter Zeile 0,5 P
  for (int k =  anfangsIndex  ; k <  endIndex  ; k++) {
    for (int i =  0  ; i <  n  ; i++) {
      for (int j =  0  ; j <  n  ; j++) {
      synchronized (this) {
        C[i][j] += A[i][k] * B[k][j];
      } // keine Punkte
      }
    }
  }
  return c;
}
```

d) Gegeben sei ein Mehrkernsystem mit 6 Prozessorkernen. Die Beschleunigung einer Anwendung im Vergleich zu einem Einkernsystem betrage S(6) = 8. Auf einem Einkernsystem betrage die Ausführungszeit T(1) = 32 Zeiteinheiten. Geben Sie die zur Berechnung der Effizienz E(n) und der Ausführungszeit T(n) verwendeten Formeln an und berechnen Sie die Effizienz E(6) sowie die Ausführungszeit T(6). (2 P)

Je korrekter Formel 0,5 P
Je korrekter Berechnung 0,5 P
E(n) = S(n) / n → E(6) = S(6) / 6 = 8 / 6 = 4/3
S(n) = T(1) / T(n) → T(6) = T(1) / S(6) = 32 / 8 = 4