

Vorname

Nachname

Matrikel-Nr

Diese Klausur ist mein letzter Prüfungsversuch (bitte ankreuzen): Ja ☐ Nein ☐

Schreiben Sie jede Lösung auf die Vorderseite eines *neuen Blattes* (und lassen Sie die Rückseiten Ihrer Lösungsblätter *leer*). Die Aufgaben 5 und 6 stehen auf der Rückseite dieses Blatts!

Aufgabe 1 (20 Punkte): Schreiben Sie eine Funktion, die der folgenden Spezifikation entspricht.

Wichtige Anforderung: Diese Funktion muss *rekursiv* arbeiten, nicht mit Schleifen!

```

1  static public int anzWege(int x, int y) {
2      // Sei ein rechtwinkliges Strassennetz gegeben, etwa so:
3      //
4      //   .
5      //   :
6      //   | | | | | | | | | |
7      // 4  +---+---+---+---+---+---+---+---+
8      //   | | | | | | | | | |
9      // 3  +---+---+---+---+---+---+---+---+
10     //   | | | | | | | | | |
11     // 2  +---+---+---*---+---+---+---+---+
12     //   | | | | | | | | | |
13     // 1  +---+---+---+---+---+---+---+---+
14     //   | | | | | | | | | |
15     // 0  #---+---+---+---+---+---+---+---+
16     //
17     //   y
18     //   x 0 1 2 3 4 4 6 7 8 ...
19     //
20     // Angenommen, wir befinden uns an einem Kreuzungspunkt (x, y) (z.B.
21     // an dem Punkt (3, 2), der oben mit einem * gekennzeichnet ist).
22     // Wie viele Wege gibt es von diesem Punkt zum Ursprungspunkt (0, 0)
23     // (oben mit # markiert)? Diese Funktion liefert die Antwort.
24     // Gezaehlt werden sollen nur Wege "ohne Umwege" (diese kuerzesten
25     // Wege haben alle die gleiche Laenge, naemlich x + y).
26     //
27     // Zwei Beispielwege vom Punkt (3, 2) zum Ursprungspunkt (0, 0),
28     // beschrieben als Folge von Punkten:
29     // (3, 2) (0, 2) (0, 0)
30     // (3, 2) (3, 1) (1, 1) (1, 0), (0, 0)
31     //
32     // Sie duerfen sich bei dieser Aufgabe darauf verlassen, dass die
33     // Werte der Parameter x und y nicht negativ sind.
34     //
35     // Beispiele fuer Aufrufe und Ergebnisse der Funktion anzWege:
36     // anzWege(2, 0) gleich 1
37     // anzWege(0, 3) gleich 1
38     // anzWege(0, 0) gleich 1 (sinnvolle Festlegung)
39     // anzWege(1, 1) gleich 2
40     // anzWege(2, 1) gleich 3
41     // anzWege(2, 2) gleich 6
42     // anzWege(3, 1) gleich 4
43     // anzWege(3, 2) gleich 10
44     ...
45 } // anzWege

```

Aufgabe 2 (20 Punkte): Schreiben Sie eine Funktion, die der folgenden Spezifikation entspricht:

```

1  static public int[] histogramm(String[] zeilen) {
2      // Wie viele Komponenten der Reihung zeilen haben die Laenge 0?
3      // Wie viele haben die Laenge 1? Die Laenge 2? ... Die Laenge 98?
4      // Eine groessere Laenge? Die Ergebnisreihung dieser Funktion
5      // enthaelt die 100 Antworten auf diese 100 Fragen.
6      //
7      // Mit anderen Worten: Sei erg die Ergebnisreihung, die diese
8      // Funktion fuer eine Parameterreihung zeilen liefert. Dann gilt:
9      //
10     // erg[0] viele Komponenten der Reihung zeilen haben die Laenge 0.
11     // erg[1] viele Komponenten der Reihung zeilen haben die Laenge 1.
12     // erg[2] viele Komponenten der Reihung zeilen haben die Laenge 2.
13     // ...
14     // erg[98] viele Komponenten der Reihung zeilen haben die Laenge 98.
15     // erg[99] viele Komponenten sind laenger als 98.
16     //
17     // Wichtige Anforderung: Eine Fallunterscheidung mit mehr als
18     // 5 Faellen ist hier nicht erlaubt.
19     ...
20 } // histogramm
```

Aufgabe 3 (15 Punkte): Schreiben Sie eine reflektive Methode entsprechend der folgenden Spezifikation:

```

1  static int summeAllerPrivatenIntAttribute(Object ob) throws Exception {
2      // Liefert die Summe aller privaten int-Attribute, die im Objekt
3      // ob enthalten sind. Summiert werden also alle
4      // in der Klasse von ob vereinbarten (privaten int-) Attribute und
5      // die von dieser Klasse geerbten (privaten int-) Attribute.
6      ...
7  } // summeAllerPrivatenIntAttribute
```

Tip 1: Aus der throws-Klausel in Zeile 1 folgt, dass Sie sich um Ausnahmen *nicht* zu kümmern brauchen.

Tip 2: Zur Lösung dieser Aufgabe braucht man die Methode `getSuperclass` der Klasse `Class`.

Aufgabe 4 (15 Punkte): Betrachten Sie die folgende Klasse `Knoten` und die Methode `aufgabe4` (die zu irgendeiner anderen Klasse gehört, die hier nicht dargestellt wird):

```

1  static class Knoten {
2      Knoten ref1;
3      Knoten ref2;
4      Long wert;
5
6      Knoten(Knoten ref1, Knoten ref2, Long wert) {
7          this.ref1 = ref1;
8          this.ref2 = ref2;
9          this.wert = wert;
10     }
11 }
12
13 static public void aufgabe4() {
14     Knoten k10 = new Knoten(null, null, 10L);
15     k10.ref1 = new Knoten(null, null, 20L);
16     k10.ref2 = new Knoten(null, null, 30L);
17
18     k10.ref1.ref2 = k10.ref2;
19     k10.ref2.ref1 = k10.ref1;
20     ...
21 } // aufgabe4
```

Angenommen, die Methode `aufgabe4` wird ausgeführt. Wie sieht die Variable `k10` aus (als Boje dargestellt), wenn die Zeile 19 fertig ausgeführt ist?

Aufgabe 5 (15 Punkte): Betrachten Sie die folgenden sechs Methoden:

```

1  static void zk01(int n) {
2      for (int i=0; i<3*n; i++) {machWas();}
3      for (int i=0; i<n*n/3; i++) {machWas();}
4  } // zk01
5
6  static void zk02(int n) {
7      for (int i=0; i<n*n; i++) {machWas();}
8      for (int i=0; i<n*n*n; i++) {machWas();}
9      for (int i=0; i<n*n; i++) {machWas();}
10 } // zk02
11
12 static void zk03(int n) {
13     for (int i=0; i<n*n; i++) {
14         for (int j=i; j<n*n; j++) {
15             machWas();
16         }
17     }
18 } // zk03
19
20 static void zk04(int n) {
21     for (int i=0; i<n*n*n; i++) {
22         for (int j=0; j<n*n; j++) {
23             machWas();
24         }
25     }
26 } // zk04
27
28 static void zk05(int n) {
29     for (int i=0; i<10; i++) {
30         for (int j=0; j<20; j++) {
31             for (int k=0; k<30; k++) {
32                 machWas();
33             }
34         }
35     }
36 } // zk05
37
38 static void zk06(int n) {
39     if (n>1) {
40         machWas();
41         zk06(n/2);
42     }
43 } // zk06

```

Ermitteln Sie für jede der Methoden die Zeitkomplexität und geben Sie sie in der üblichen groß-O-Notation an. Gehen Sie dabei von der Annahme aus, dass jeder Aufruf der Methode `machWas` als 1 Schritt gewertet werden kann, und dass der Zeitbedarf aller anderen Befehle *vernachlässigbar* ist.

Aufgabe 6: (15 Punkte) Beantworten Sie die folgenden Fragen möglichst kurz und genau:

1. Ist die folgende Methode eine *vernünftige rekursive Methode*, oder verstößt sie gegen grundlegende Regeln, die jede rekursive Methode einhalten sollte?

```

1  static int berechneA(int n) {
2      pln("n: " + n);
3      if (n % 10 > 0) {
4          return berechneA(n-1);
5      } else if (n % 10 < 0) {
6          return berechneA(n+1);
7      } else {
8          return n;
9      }
10 }

```

2. Betrachten Sie folgenden Befehl:

```

1  String          s1  = new String("ABC");
2  Integer         n1  = new Integer(17);
3  ArrayList<String> als = new ArrayList<String>();
4
5  als.add(s1);
6  als.add(n1);
7  als.remove(s1);
8  als.remove(n1);
9  if (als.contains(s1)) pln("A");
10 if (als.contains(n1)) pln("B");

```

Die Befehle in den Zeilen 1 bis 3 *sind* korrekt.

Welche der Befehle in den Zeilen 5 bis 10 sind ebenfalls korrekt (d.h. werden vom Ausführer akzeptiert)? Geben Sie nur die Zeilen-Nrn der korrekten Befehle an.

3. Wenn man Algorithmen untersucht, um ihre Zeitkomplexität zu ermitteln, was versteht man dann unter einem *Schritt*?

4. Angenommen, eine Klasse `Orange` implementiert die Schnittstelle `Comparable<String>`. Wird dadurch eine natürliche Ordnung für `Orange`-Objekte definiert? Begründen Sie Ihre Antwort kurz.

5. In der Klasse `Class` werden unter anderem die folgenden beiden parameterlosen Methoden vereinbart:

```

1  public Method[] getDeclaredMethods()
2  public Method[] getMethods()

```

Beschreiben Sie kurz den Unterschied zwischen diesen beiden Methoden.

6. Was ist der Unterschied zwischen einer *Sammlung* (engl. collection) und einem *Behälter* (engl. container)?

Beurteilung dieser Klausur:

A1	
A2	
A3	
A4	
A5	
A6	
Summe	
Note	

Korrigierte Beurteilung:

A1	
A2	
A3	
A4	
A5	
A6	
Summe	
Note	

Datum	
-------	--

Datum	
-------	--

Lösung 1 (20 Punkte): Schreiben Sie eine Funktion, die der folgenden Spezifikation entspricht.

Wichtige Anforderung: Diese Funktion muss *rekursiv* arbeiten, nicht mit Schleifen!

```
3 static public int anzWege(int x, int y) {
4
5     // Einfache Faelle:
6     if (x == 0 || y == 0) return 1;
7
8     // Rekursiver Fall (x und y sind beide groesser als 0):
9     return anzWege(x-1, y) + anzWege(x, y-1);
10
11 }
```

Lösung 2 (20 Punkte): Schreiben Sie eine Funktion, die der folgenden Spezifikation entspricht:

```
1 static public int[] histogramm(String[] zeilen) {
2
3     int[] erg = new int[100];
4
5     for (String z : zeilen) {
6         int len = Math.min(erg.length-1, z.length());
7         erg[len]++;
8     }
9
10    return erg;
11 }
```

Lösung 3 (15 Punkte): Schreiben Sie eine reflektive Methode entsprechend der folgenden Spezifikation:

```
1 static int summeAllerPrivatenIntAttribute(Object ob) throws Exception {
2     // Liefert die Summe aller privaten int-Attribute, die im Objekt
3     // ob enthalten sind. Summiert werden also alle
4     // in der Klasse von ob vereinbarten (privaten int-) Attribute und
5     // die von dieser Klasse geerbten (privaten int-) Attribute.
6
7     int erg = 0;
8     Class<?> kob = ob.getClass();
9
10    while (true) {
11        Field[] fr = kob.getDeclaredFields();
12
13        for (Field f : fr) {
14            if (f.getType() != Integer.TYPE) continue;
15            int modifiers = f.getModifiers();
16            if (!Modifier.isPrivate(modifiers)) continue;
17            f.setAccessible(true);
18            erg += f.getInt(ob);
19        }
20
21        kob = kob.getSuperclass();
22        if (kob==null) break;
23    }
24
25    return erg;
26 }
```

Lösung 4 (15 Punkte): Betrachten Sie die folgende Klasse Knoten und die Methode aufgabe4 (die zu irgendeiner anderen Klasse gehört, die hier nicht dargestellt wird):

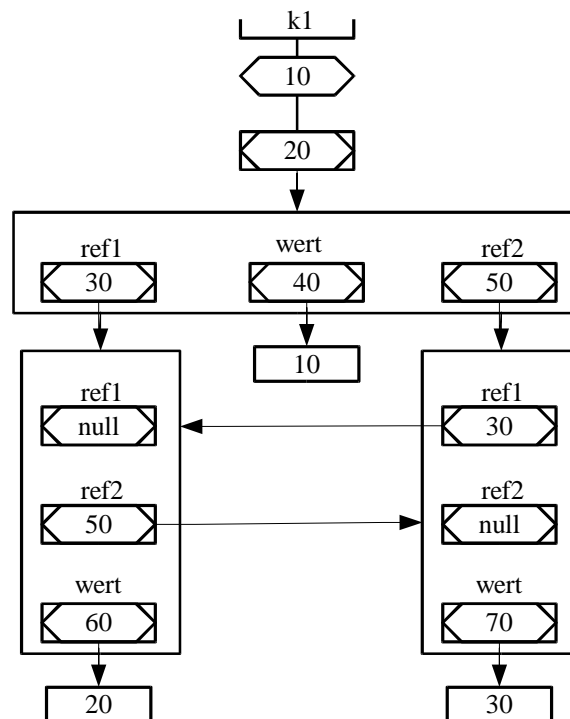
```
1 static class Knoten {
2     Knoten ref1;
3     Knoten ref2;
```

```

4      Long   wert;
5
6      Knoten(Knoten ref1, Knoten ref2, Long wert) {
7          this.ref1 = ref1;
8          this.ref2 = ref2;
9          this.wert = wert;
10     }
11 }
12
13 static public void aufgabe4() {
14     Knoten k10 = new Knoten(null, null, 10L);
15     k10.ref1    = new Knoten(null, null, 20L);
16     k10.ref2    = new Knoten(null, null, 30L);
17
18     k10.ref1.ref2 = k10.ref2;
19     k10.ref2.ref1 = k10.ref1;
20     ...
21 } // aufgabe4

```

Angenommen, die Methode `aufgabe4` wird ausgeführt. Wie sieht die Variable `k10` (als Boje dargestellt) aus, wenn die Zeile 19 fertig ausgeführt ist?



Lösung 5 (15 Punkte): Betrachten Sie die folgenden sechs Methoden:

```

1  static void zk01(int n) {
2      for (int i=0; i<3*n; i++) {machWas();}
3      for (int i=0; i<n*n/3; i++) {machWas();}
4  } // zk01
5
6  static void zk02(int n) {
7      for (int i=0; i<n*n; i++) {machWas();}
8      for (int i=0; i<n*n*n; i++) {machWas();}
9      for (int i=0; i<n*n; i++) {machWas();}
10 } // zk02
11
12 static void zk03(int n) {
13     for (int i=0; i<n*n; i++) {
14         for (int j=i; j<n*n; j++) {
15             machWas();

```



```

16     }
17 }
18 } // zk03
19
20 static void zk04(int n) {
21     for (int i=0; i<n*n*n; i++) {
22         for (int j=0; j<n*n; j++) {
23             machWas();
24         }
25     }
26 } // zk04
27
28 static void zk05(int n) {
29     for (int i=0; i<10; i++) {
30         for (int j=0; j<20; j++) {
31             for (int k=0; k<30; k++) {
32                 machWas();
33             }
34         }
35     }
36 } // zk05
37
38 static void zk06(int n) {
39     if (n>1) {
40         machWas();
41         zk06(n/2);
42     }
43 } // zk06

```

Gehen Sie von der Annahme aus, dass jeder Aufruf der Methode `machWas` als 1 Schritt gewertet werden kann, und dass der Zeitbedarf aller anderen Befehle *vernachlässigbar* ist. Ermitteln Sie für jede der Methoden die Zeitkomplexität und geben Sie sie in der üblichen groß-O-Notation an.

Methode	Zeitkomplexität
zk01	$O(n^2)$
zk02	$O(n^3)$
zk03	$O(n^4)$
zk04	$O(n^5)$
zk05	$O(1)$
zk06	$O(\log(n))$

Lösung 6: (15 Punkte) Beantworten Sie die folgenden Fragen möglichst kurz und genau:

1. Ist die folgende Methode eine *vernünftige rekursive Methode*, oder verstößt sie gegen grundlegende Regeln, die jede rekursive Methode einhalten sollte?

```

1  static int berechneA(int n) {
2      println("n: " + n);
3      if (n % 10 > 0) {
4          return berechneA(n-1);
5      } else if (n % 10 < 0) {
6          return berechneA(n+1);
7      } else {
8          return n;
9      }
10 }

```

Ja, sie ist vernünftig rekursiv.

2. Betrachten Sie folgenden Befehle:

```
1  String          s1  = new String("ABC");
2  Integer         n1  = new Integer(17);
3  ArrayList<String> als = new ArrayList<String>();
4
5  als.add(s1);
6  als.add(n1);
7  als.remove(s1);
8  als.remove(n1);
9  if (als.contains(s1)) pln("A");
10 if (als.contains(n1)) pln("B");
```

Die Befehle in den Zeilen 1 bis 3 *sind* korrekt.

Welche der Befehle in den Zeilen 5 bis 10 sind ebenfalls korrekt (d.h. werden vom Ausführer akzeptiert)? Geben Sie nur die Zeilen-Nrn der korrekten Befehle an.

Korrekt sind die Befehle in den Zeilen 5, 7, 8, 9 und 10 (nur 6 ist nicht korrekt).

3. Wenn man Algorithmen untersucht, um ihre Zeitkomplexität zu ermitteln, was versteht man dann unter einem *Schritt*?

Eine beliebige Befehlsfolge, von der es plausibel ist anzunehmen, dass sie von einem Ausführer in einer bestimmten, festen Zeit ausgeführt werden kann. Dieses Zeit darf insbesondere nicht von der Größe des gerade bearbeiteten Problems abhängen.

4. Angenommen, eine Klasse *Orange* implementiert die Schnittstelle *Comparable<String>*. Wird dadurch eine natürliche Ordnung für *Orange*-Objekte definiert? Begründen Sie Ihre Antwort kurz.

Nein, es wird keine natürliche Ordnung für *Orange*-Objekte definiert, weil man Orangen nicht mit Orangen vergleichen kann (nur Orangen mit Strings).

5. In der Klasse *Class* werden unter anderem die folgenden beiden paramterlosen Methoden vereinbart:

```
1  public Method[] getDeclaredMethods()
2  public Method[] getMethods()
```

Beschreiben Sie kurz den Unterschied zwischen diesen beiden Methoden.

Die erste liefert eine Reihung aller Methoden, die in der betreffenden Klasse vereinbart wurden.

Die zweite liefert eine Reihung aller öffentlichen (public) Methoden.

6. Was ist der Unterschied zwischen einer *Sammlung* (engl. collection) und einem *Behälter* (engl. container)?

Eine Sammlung ist ein Objekt, in dem man andere Objekte (eines bestimmten Typs) sammeln kann (d.h. hineintun, dort suchen und von dort wieder entfernen kann).

Ein Behälter ist ein Grabo-Objekt, in das man andere Grabo-Objekte hineintun kann.