

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 zwei Methoden, die den folgenden Spezifikationen entsprechen. Die Methode `druckeAllePrimTeiler` muss rekursiv programmiert werden (in ihrem Rumpf sind *Schleifen nicht erlaubt*):

```

1  static int minTeiler(int n) {
2      // Verlaesst sich darauf, dass n groesser als 1 ist.
3      // Liefert die kleinste Zahl zwischen 2 und n (einschliesslich),
4      // durch die man n (ohne Rest) teilen kann.
5      // Beispiele:
6      // minTeiler( 2) ist gleich 2
7      // minTeiler( 3) ist gleich 3
8      // minTeiler( 4) ist gleich 2
9      // minTeiler(15) ist gleich 3
10     // minTeiler(77) ist gleich 7
11     //
12     // Hinweis: Das Ergebnis von minTeiler ist immer eine Primzahl.
13     ...
14 } // minTeiler
15
16 static void druckeAllePrimTeiler(int n) {
17     // Verlaesst sich darauf, dass n groesser als 1 ist.
18     // Gibt die Primzahlen aus, deren Produkt gleich n ist.
19     // Die Primzahlen werden durch je ein Blank voneinander getrennt.
20     // Beispiele:
21     // Aufruf                                     Ausgabe:
22     // druckeAllePrimTeiler( 2)                    2
23     // druckeAllePrimTeiler( 3)                    3
24     // druckeAllePrimTeiler( 4)                    2 2
25     // druckeAllePrimTeiler( 5)                    5
26     // druckeAllePrimTeiler( 8)                    2 2 2
27     // druckeAllePrimTeiler(12)                    2 2 3
28     // druckeAllePrimTeiler(15)                    3 5
29     // druckeAllePrimTeiler(17)                    17
30     // druckeAllePrimTeiler(30)                    2 3 5
31     // druckeAllePrimTeiler(90)                    2 3 3 5
32     // druckeAllePrimTeiler(150)                   2 3 5 5
33     ...
34 } // druckeAllePrimTeiler

```

**Aufgabe 2 (10 Punkte):** Schreiben Sie eine Funktion, die der folgenden Spezifikation entspricht:

```

1  static int anzahlChars(ArrayList<ArrayList<String>> aas) {
2      // Die elementaren Komponenten der Sammlung aas sind vom Typ String.
3      // Wieviele char-Komponenten enthalten alle in aas enthaltenen
4      // Strings zusammengenommen? Diese Funktion liefert die Antwort.
5      //
6      // Anforderung: Um die volle Punktzahl zu bekommen muessen Sie
7      // hier "einfache und sichere for-Schleifen" verwenden (und nicht
8      // die "unsicheren for-Schleifen mit Index").
9      ...
10 } // anzahlChars

```

**Aufgabe 3 (20 Punkte):** Schreiben Sie eine Methode, die der folgenden Spezifikation entspricht:

```

1  static void druckeMethodenMitZweiIntParams (Object ob) {
2      // Gibt die Namen aller Methoden aus fuer die gilt:
3      // 1. Die Methode wurde in der Klasse vereinbart, zu der
4      //    das Objekt ob gehoert (nicht in einer Oberklasse).
5      // 2. Die Methode hat genau 2 Parameter und beide gehoeren
6      //    zum (primitiven!) Typ int.
7      ...
8  } // druckeMethodenMitZweiIntParams

```

**Aufgabe 4 (15 Punkte):** Betrachten Sie die folgende Dokumenten Typ Definition (DTD) und die nachfolgenden sechs XML-Dateien:

```

1  Datei Anschrift.dtd:
2
3  <?xml version="1.0" encoding="UTF-8"?>
4
5  <!ELEMENT anschrift (name+, ort, land?)>
6  <!ELEMENT name      (#PCDATA)>
7  <!ELEMENT ort        (#PCDATA)>
8  <!ELEMENT land       (#PCDATA)>
9
10 <!ATTLIST ort
11   art (dorf | stadt | metropole) "dorf"
12 >

```

<pre> 1  +-----+ 2  Datei <b>A.xml</b>: 3 4  &lt;anschrift&gt; 5    &lt;name&gt; Anna      &lt;/name&gt; 6    &lt;ort art="metropole"&gt; 7      Berlin 8    &lt;/ort&gt; 9    &lt;land&gt; Deutschland &lt;/land&gt; 10 &lt;/anschrift&gt; </pre>	<pre> 1  +-----+ 2  Datei <b>D.xml</b> 3 4  &lt;anschrift&gt; 5    &lt;name&gt; Anna      &lt;/name&gt; 6    &lt;ort art="grossstadt"&gt; 7      Berlin 8    &lt;/ort&gt; 9    &lt;land&gt; USA &lt;/land&gt; 10 &lt;/anschrift&gt; </pre>
<pre> 12 Datei <b>B.xml</b> 13 14 &lt;anschrift&gt; 15   &lt;name&gt; Anna      &lt;/name&gt; 16   &lt;name&gt; Bert      &lt;/name&gt; 17   &lt;ort&gt; Berlin     &lt;/ort&gt; 18 &lt;/anschrift&gt; 19 20 21 22 </pre>	<pre> 12 Datei <b>E.xml</b>: 13 14 &lt;anschrift&gt; 15   &lt;name&gt; Anna      &lt;/name&gt; 16   &lt;name&gt; Bert      &lt;/name&gt; 17   &lt;name&gt; Anna      &lt;/name&gt; 18   &lt;ort art="metropole"&gt; 19     Gransee 20   &lt;/ort&gt; 21   &lt;land&gt; ab123cde?? &lt;/land&gt; 22 &lt;/anschrift&gt; </pre>
<pre> 24 Datei <b>C.xml</b>: 25 26 &lt;anschrift&gt; 27   &lt;name&gt; Anna      &lt;/name&gt; 28   &lt;name&gt; Anna      &lt;/name&gt; 29   &lt;ort&gt; Berlin     &lt;/ort&gt; 30   &lt;ort&gt; Berlin     &lt;/ort&gt; 31 &lt;/anschrift&gt; 32 33 </pre>	<pre> 24 Datei <b>F.xml</b> 25 26 &lt;anschrift&gt; 27   &lt;name&gt; Anna Blume &lt;/name&gt; 28   &lt;name&gt; Bert August &lt;/name&gt; 29   &lt;ort art="stadt"&gt; 30     Gransee 31   &lt;/ort&gt; 32   &lt;land kuerzel="B"&gt; Berlin &lt;/land&gt; 33 &lt;/anschrift&gt; </pre>

Welche der sechs XML-Dokumente sind **gültig** (relativ zur Anschrift.dtd) und welche nicht? Geben Sie für jedes nicht-gültige Dokument eine kurze **Begründung** an, **warum** es nicht gültig ist.

**Aufgabe 5 (10 Punkte):**

5.1. Fügen Sie die folgenden Schlüssel in einen (anfangs leeren) binären Baum ein. Wie sieht der Baum aus, nachdem Sie alle Schlüssel eingefügt haben?

32, 41, 34, 38, 24, 46, 15, 51, 20.

Im Folgenden sollen Sie unterschiedliche Sammlungsstrukturen miteinander vergleichen:

5.2. Geben Sie einen Vorteil von *binären Bäumen* im Vergleich zu *sortierten Listen* an.

5.3. Geben Sie einen Vorteil von *unsortierten Listen* im Vergleich zu *sortierten Listen* an.

5.4. Geben Sie einen Vorteil von *sortierten Listen* im Vergleich zu *unsortierten Listen* an.

**Aufgabe 6: (15 Punkte)**

6.1. Beschreiben Sie ein Beispiel für einen Nebenläufigkeitsfehler ("Fadenfehler"). Welche "Zutaten" braucht man, um einen solchen Fehler zu ermöglichen? Was muss im Einzelnen passieren, damit der Fehler eintritt?

6.2. Beschreiben Sie ein Beispiel für eine Verklemmung (engl. deadlock). Welche "Zutaten" braucht man, um einen solchen Fehler zu ermöglichen? Was muss im Einzelnen passieren, damit der Fehler eintritt?

6.3. Skizzieren Sie ein einfaches Verfahren, mit dem man Verklemmungen grundsätzlich vermeiden kann. Ihre Beschreibung kann ziemlich kurz und Stichpunktartig sein, soll aber einen Leser, der das Verfahren kennt, davon überzeugen, dass Sie es auch kennen.



**Lösung 1 (20 Punkte):**

```
1  static int minTeiler(int n) {
2      // Verlaesst sich darauf, dass n groesser als 1 ist.
3      // Liefert die kleinste Zahl zwischen 2 und n (einschliesslich),
4      // durch die man n (ohne Rest) teilen kann.
5
6      if (n <= 3) return n;
7
8      for (int tk=2; tk <= n/2; tk++) {
9          if (n%tk == 0) return tk;
10     }
11     return n;
12 } // minTeiler
13
14 static void druckeAllePrimTeiler(int n) {
15     // Verlaesst sich darauf, dass n groesser als 1 ist.
16     // Gibt die Primzahlen aus, deren Produkt gleich n ist.
17     // Die Primzahlen werden durch je ein Blank voneinander getrennt.
18     //
19     // Anforderung: Diese Methode muss rekursiv programmiert werden
20     // (d.h. sie darf keine Schleife enthalten)
21
22     int minT = minTeiler(n);
23     p(minT + " ");
24     if (minT != n) {
25         druckeAllePrimTeiler(n/minT);
26     }
27 } // druckeAllePrimTeiler
```

**Lösung 2 (10 Punkte):**

```
1  static int anzahlChars(ArrayList<ArrayList<String>> aas) {
2      // Die elementaren Komponenten der Sammlung aas sind vom Typ String.
3      // Wieviele char-Komponenten enthalten alle in aas enthaltenen
4      // Strings zusammengenommen? Diese Funktion liefert die Antwort.
5      //
6      // Anforderung: Einfache und sichere for-Schleifen!
7
8      int anzahl = 0;
9      for (ArrayList<String> as : aas) {
10         for (String s : as) {
11             anzahl += s.length();
12         }
13     }
14     return anzahl;
15 } // anzahlChars
```

**Lösung 3 (20 Punkte):**

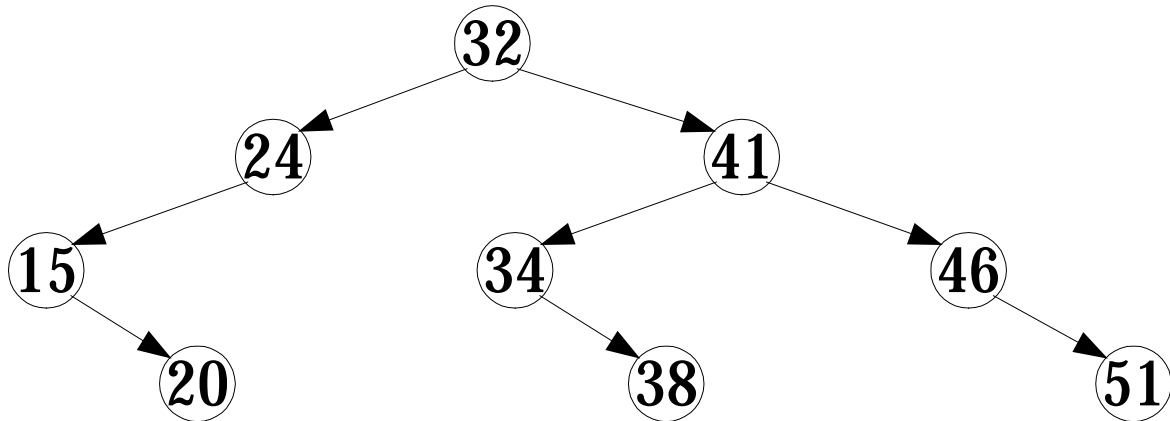
```
1  static void druckeMethodenMitZweiIntParams(Object ob) {
2      // Gibt die Namen aller Methoden aus fuer die gilt:
3      // 1. Die Methode wurde in der Klasse vereinbart, zu der
4      //    das Objekt ob gehoert
5      // 2. Die Methode hat genau 2 Parameter und beide gehoeren
6      //    zum (primitiven!) Typ int.
7
8      Class<?> kob = ob.getClass();
9      Method[] mr = kob.getDeclaredMethods();
10     for (Method m : mr) {
11         Class<?>[] tr = m.getParameterTypes();
12         if (tr.length != 2) continue;
13         if (tr[0] == Integer.TYPE && tr[1] == Integer.TYPE) {
14             p(m.getName() + " ");
15         }
16     }
17     pln();
18 } // druckeMethodenMitZweiIntParams
```

**Lösung 4 (15 Punkte):**

- 1 A.xml: **Gültig**
- 2 B.xml: **Gültig**
- 3 C.xml: **Ungültig**: 2 **ort**-Elemente
- 4 D.xml: **Ungültig**: Attribut **art** hat unzulässigen Wert "grossstadt"
- 5 E.xml: **Gültig**:
- 6 F.xml: **Ungültig**: Das **land**-Element hat ein unerlaubtes Attribut **kuerzel**

**Lösung 5 (10 Punkte)**

5.1. Wie sieht der sortierte binäre Baum aus, nachdem alle 9 Knoten eingefügt wurden:



5.2. Ein Vorteil von *binären Bäumen* im Vergleich zu *sortierten Listen*:

Das Suchen geht (erheblich) schneller.

5.3. Ein Vorteil von *unsortierten Listen* im Vergleich zu *sortierten Listen*:

Das Einfügen geht schneller.

5.4. Ein Vorteil von *sortierten Listen* im Vergleich zu *unsortierten Listen*:

Das Suchen im negativen Fall geht schneller.

**Lösung 6 (15 Punkte):**

6.1. Beschreiben Sie ein Beispiel für einen Nebenläufigkeitsfehler ("Fadenfehler").

Man braucht mindestens 2 Fäden F1 und F2 und eine Variable v, z.B. eine int-Variable mit dem Wert 17. Beide Fäden wollen v um 1 erhöhen. Ein Fehler tritt z.B. bei der folgenden Aktionsfolge ein:

1. F1 liest v und berechnet  $17 + 1$  gleich 18.
2. F2 liest v und berechnet  $17 + 1$  gleich 18.
3. F1 schreibt sein Ergebnis nach v.
4. F2 schreibt sein Ergebnis nach v.

6.2. Beschreiben Sie ein Beispiel für eine Verklemmung (engl. deadlock).

Man braucht mindestens 2 Fäden F1 und F2 und 2 Betriebsmittel b1 und b2. Jeder der beiden Fäden will beide Betriebsmittel für sich reservieren. Ein Fehler tritt z.B. bei der folgenden Aktionsfolge ein:

1. F1 reserviert b1 (in Java: `synchronized(b1)`).
2. F2 reserviert b2 (in Java: `synchronized(b2)`).
3. F1 wartet darauf, dass b2 frei wird.
4. F2 wartet darauf, dass b1 frei wird.

6.3. Skizzieren Sie ein einfaches Verfahren, mit dem man Verklemmungen grundsätzlich vermeiden kann.

Man ordnet jedem Betriebsmittel, welches irgendwann für Fäden reserviert werden soll, eine eindeutige Nummer (natürliche Zahl) zu.

Wenn ein Faden mehrere Betriebsmittel gleichzeitig benutzen will, muss er sie in aufsteigender Reihenfolge reservieren (z.B. zuerst b17 und dann b25, nicht umgekehrt).