

Vorname

Nachname

Matrikel-Nr

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

Schreiben Sie jede Lösung auf die Vorderseite eines *neuen Blattes* (und lassen Sie die Rückseiten Ihrer Lösungsblätter *leer*).

Aufgabe 1 (15 Punkte): Schreiben Sie eine *rekursive Methode* (eine, welche sich unter bestimmten Bedingungen selbst aufruft und in der keine Schleifen vorkommen) entsprechend der folgenden Spezifikation:

```
1 static public int anzahlGeradeZiffern(long zahl) {
2     // Liefert die Anzahl der geraden Ziffern, die in einer Darstellung
3     // von zahl als Dezimalzahl vorkommen. Unnoetige fuehrende Nullen
4     // werden dabei nicht gezaehlt.
5     //
6     // Beispiele:
7     // anzahlGeradeZiffern(-123) ist gleich 1 (nur 2 ist gerade)
8     // anzahlGeradeZiffern(+123) ist gleich 1 (nur 2 ist gerade)
9     // anzahlGeradeZiffern(8226) ist gleich 4 (alle 4 Ziffern sind gerade)
10    // anzahlGeradeZiffern(-513) ist gleich 0 (keine Ziffer ist gerade)
11    // anzahlGeradeZiffern(0) ist gleich 1 (die Ziffer 0 ist gerade)
12    // anzahlGeradeZiffern(1000) ist gleich 3 (die Ziffer 1 ist ungerade)
13    // anzahlGeradeZiffern(0001) ist gleich 0 (unnoetige fuehrende Nullen
14    // werden nicht gezaehlt)
15    ...
16 }
```

Aufgabe 2 (15 Punkte): Schreiben Sie eine Methode entsprechend der folgenden Spezifikation:

```
1 static public void gibAus(String s1, String s2) {
2     // Je nachdem, ob s1 kleiner, gleich oder groesser s2 ist, wird
3     // der Text "KLEINER", "GLEICH" bzw. "GROESSER" ausgegeben.
4     // Dazu werden s1 und s2 moeglichst effizient miteinander verglichen.
5     ...
6 }
```

Aufgabe 3 (15 Punkte): Betrachten Sie die folgenden Vereinbarungen:

```
1 ArrayList<String> k1 = null;
2 ArrayList<String> k2 = new ArrayList<String>();
3 ArrayList<String> k3 = new ArrayList<String>();
4 ArrayList<ArrayList<String>> b = new ArrayList<ArrayList<String>>();
5
6 k3.add(new String("Anna"));
7 k3.add(new String("Bert"));
8
9 b.add(k1);
10 b.add(k2);
11 b.add(k3);
```

Es folgt eine Deutung dieser Vereinbarungen (mit Hilfe der relativ konkreten Begriffe *Buch*, *Kapitel* und *Abschnitt*): Das *Buch* **b** enthält die drei *Kapitel* **k1**, **k2** und **k3**. Die Variable **k1** zeigt noch nicht auf ein Kapitel-Objekt, sondern hat den Wert `null`. Die Variable **k2** zeigt auf ein noch leeres Kapitel-Objekt. Die Variable **k3** zeigt auf ein Kapitel-Objekt, welches zur Zeit zwei *Abschnitte* (d. h. *String*-Objekte) enthält.

Stellen Sie die Variablen **k1**, **k2**, **k3** und **b** als *Bojen* dar (nach Ausführung von Zeile 11).

Aufgabe 4 (15 Punkte): Schreiben Sie eine Methode entsprechend der folgenden Spezifikation:

```
1 static public void haengeAn(String t, ArrayList<ArrayList<String>> b) {
2     // Das Buch b ist
3     // eine Sammlung (vom Typ ArrayList<ArrayList<String>>)
4     // von Sammlungen (vom Typ ArrayList<String>)
5     // von Abschnitten (vom Typ String)
6     // Diese Methode haengeAn haengt an jeden Abschnitt des Buches b den
7     // Text t an.
8     //
9     // Beispiel1:
10    // Sei buch1 gleich [[Text1, Text2], [Text3, Text4, Text5]]
11    // Nach Ausfuehrung der Anweisung haengeAn("?!!", buch1);
12    // soll buch1 dann so aussehen:
13    // [[Text1?!!, Text2?!!], [Text3?!!, Text4?!!, Text5?!!]]
14    //
15    // Beispiel2:
16    // Sei buch2 gleich [null, [], [abc, def]].
17    // Nach Ausfuehrung der Anweisung haengeAn("xx", buch2);
18    // soll buch2 dann so aussehen:
19    // [null, [], [abctx, defxx]]
20    ...
21 }
```

Aufgabe 5 (15 Punkte): Die *numerischen Literale* einer neuen Programmiersprache sollen aus *Dezimalziffern* bestehen. Außerdem dürfen sie einen *Dezimalpunkt* enthalten und mit einem *Vorzeichen* beginnen. Hier ein paar Beispiele für *erlaubte* und *nicht erlaubte* Literale:

22	Erlaubt:	Nicht erlaubt:
23	-----	-----
24	123	12A // 'A' ist keine Ziffer
25	0	FF0 // 'F' ist keine Ziffer
26	00999876	123+ // Vorzeichen muss vorne stehen
27	12.34	.5 // Vor dem Punkt mindestens eine Ziffer
28	0.0	3. // Nach dem Punkt mindestens eine Ziffer
29	+123	
30	-123	
31	+12.34	
32	-12.34	

Beschreiben Sie die (unendliche) Menge all dieser numerischen Literale durch eine (kontextfreie) Grammatik. Die Menge der Endsymbole dieser Grammatik ist gleich `{ '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.', '+', '-' }`. Die Zwischensymbole können Sie frei wählen. Empfehlung: Wählen Sie in Zweifelsfällen lieber längere, aber verständliche Zwischensymbole an Stelle von kürzeren, unverständlichen (z. B. eher Vorzeichen als nur V).

Aufgabe 6: (15 Punkte)

1. Nennen Sie zwei (allgemeine und wichtige) *Unterschiede* zwischen einem *konventionellen Prozessor* (z. B. Pentium) und der *virtuellen Java-Maschine* (JVM).
2. Mit welchen Ausdrücken kann man auf die *Class*-Objekte der folgenden drei Typen zugreifen: `char`, `Integer` und `char[]`? Ein Ausdruck pro Typ genügt, auch wenn es mehrere gibt.
3. Was leistet ein *Lexer*? Was liest er ein und was macht er daraus?
4. Nennen Sie vier *Klassen*, die zur *Reflexionsschnittstelle* von Java gehören.
5. Was ist ein *binärer Baum* (Definition)?
6. Wann ist ein binärer Baum *sortiert* (Definition)?

Lösung 1 (15 Punkte): Die Funktion `anzahlGeradeZiffern`:

```

1  static public int anzahlGeradeZiffern(long zahl) {
2      // Liefert die Anzahl der geraden Ziffern, die in einer Darstellung
3      // von zahl als Dezimalzahl vorkommen. Unnoetige fuehrende Nullen
4      // werden dabei nicht gezaehlt.
5
6      long ziff = zahl%10;
7      int anz = (ziff%2 == 0) ? 1 : 0;
8
9      if (Math.abs(zahl)<=9) {
10         return anz; // Einfacher Fall
11     } else {
12         return anz + anzahlGeradeZiffern(zahl/10); // Rekursiver Fall
13     }
14 } // anzahlGeradeZiffern

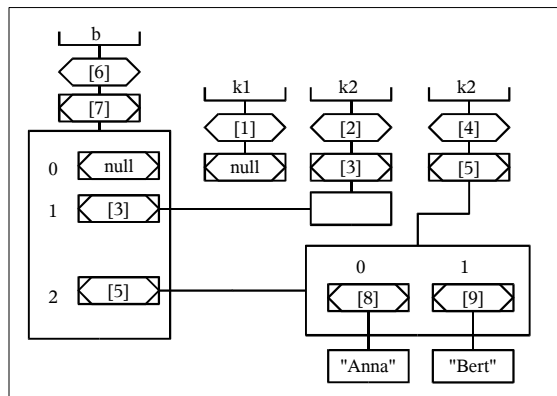
```

Lösung 2 (15 Punkte):

```

1  static public void gibAus(String s1, String s2) {
2      // Je nachdem, ob s1 kleiner, gleich oder groesser s2 ist, wird
3      // der Text "KLEINER", "GLEICH" bzw. "GROESSER" ausgegeben.
4      // Dazu werden s1 und s2 moeglichst effizient miteinander verglichen.
5
6      int erg = s1.compareTo(s2);
7
8      if (erg < 0) {
9          pln("KLEINER");
10     } else if (erg == 0) {
11         pln("GLEICH");
12     } else {
13         pln("GROESSER");
14     }
15 } // gibAus

```

Lösung 3 (15 Punkte): Die Variablen `k1`, `k2`, `k3` und `b` als Bojen dargestellt:**Lösung 4 (15 Punkte):** Die Prozedur `haengeAn`:

```

1  static public void haengeAn(String t, ArrayList<ArrayList<String>> b) {
2      // Haengt an jeden String-Abschnitt des Buches b den Text t an.
3
4      for (int kapitelNr=0; kapitelNr<b.size(); kapitelNr++) {
5          ArrayList<String> kap = b.get(kapitelNr);
6          if (kap == null) continue;
7          for (int abschnittNr=0; abschnittNr<kap.size(); abschnittNr++) {
8              String abschnitt = kap.get(abschnittNr);
9              if (abschnitt == null) continue;
10             kap.set(abschnittNr, abschnitt + t);
11         } // for kapitelNr
12     } // for abschnittNr
13 } // haengeAn

```

Lösung 5 (15 Punkte): Eine Grammatik für numerische Literale**Endsymbole:** {'0', '1', '2', ..., '9', '.', '+', '-'}**Zwischensymbole:** {Ziff, Vorzeichen, ZiffFo, NumLitOV, NumLit}**Startsymbol:** NumLit

```

R01: Ziff      -> '0'           // Ziffer
R02: Ziff      -> '1'
...
R10: Ziff      -> '9'
R11: Vorzeichen -> '+'           // Vorzeichen
R12: Vorzeichen -> '-'
R13: ZiffFo     -> Ziff          // Ziffern-Folge
R14: ZiffFo     -> Ziff ZiffFo
R15: NumLitOV   -> ZiffFo        // Numerisches Literal ohne Vorzeichen
R16: NumLitOV   -> ZiffFo '.' ZiffFo
R17: NumLit     -> NumLitOV      // Numerisches Literal
R18: NumLit     -> Vorzeichen NumLitOV

```

Lösung 6 (15 Punkte):

6.1. Nennen Sie zwei (allgemeine und wichtige) *Unterschiede* zwischen einem *konventionellen Prozessor* (z. B. Pentium) und der *virtuellen Java-Maschine (JVM)*.

Die JVM ist typsicher und objektorientiert.

6.2. Mit welchen Ausdrücken kann man auf die Class-Objekte der folgenden drei Typen zugreifen: `char`, `Integer` und `char[]`? *Ein* Ausdruck pro Typ genügt, auch wenn es mehrere gibt.

Character.TYPE, **Class.forName("java.lang.Integer")** (oder: **Integer.class**), **Class.forName("[C]")** (oder: **char[][].class**).

6.3. Was leistet ein *Lexer*? Was liest er ein und was macht er daraus?

Ein Lexer macht aus einer Folge von Zeichen eine Folge von Token.

6.4. Nennen Sie vier *Klassen*, die zur Reflexionsschnittstelle von Java gehören.

Class, Constructor, Method, Field

6.5. Was ist ein *binärer Baum* (Definition)?

Entweder ein leerer Baum oder ein Knoten, an dem zwei binäre Bäume hängen.

6.6. Wann ist ein binärer Baum *sortiert* (Definition)?

Wenn für jeden Knoten K mit Schlüssel S gilt:

Alle Schlüssel im linken Unterbaum von K sind kleiner als S und alle Schlüssel im rechten Unterbaum von K sind größer als S.