Zur Erinnerung: Wichtige Prozeduren, Klassen und Interfaces:

```
int Integer.parseInt(String s){...}
double Double.parseDouble(String s){...}
```

```
public class StdIn {
  public static boolean isEmpty(){...}
  public static String readLine(){...}
  public static char readChar(){...}
  public static String readAll(){...}
  public static String readString(){...}
  public static int readInt() {...}
  public static double readDouble(){...}
  public static float readFloat(){...}
  public static long readLong(){...}
  public static boolean readBoolean(){...}
}
```

```
public class StdOut {
  public static void println() {...}
  public static void println(Object x){...}
  public static void print(Object x){...}
}
```

```
public interface Collection<E>
            extends Iterable<E> {
  boolean add(E o);
  boolean contains(Object o);
  boolean isEmpty();
  int size();
  boolean remove(Object o);
  boolean removeAll(Collection<?> c);
  ...
}
```

```
// Implementierungen: ArrayList, LinkedList
public interface List<T>
            extends Collection<T> {
  T get(int index);
  void add(int index, T o);
  T remove(int index);
  ...
}
```

```
// Implementierungen: HashSet, TreeSet
public interface Set<T>
            extends Collection<T> {
}
```

```
public interface Iterable<T> {
  Iterator<T> iterator();
}

public interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove();
}
```

```
// Implementierungen: HashMap, TreeMap
public interface Map<K, V> {
  public interface Entry<K, V> {
    K getKey();
    V getValue();
    ...
  }

  V get(Object key);
  V put(K key, V value);
  V remove(Object key);
  Set<K> keySet();
  Set<Map.Entry<K, V>> entrySet();
  Collection<V> values();
  ...
}
```

```
// Sortieren von Arrays und Listen:
Arrays.sort(Object[] a) {...}
Arrays.sort(T[] a, Comparator<T> c) {...}
Arrays.sort(int[] a) {...}
Collections.sort(List<T> l) {...}
Collections.sort(List<T> l, Comparator<T> c) {...}
```

```
// Junit Prozeduren zum Testen:
Assert.assertFalse(boolean condition) {...}
Assert.assertTrue(boolean condition) {...}
Assert.assertEquals(Object expected,
                    Object actual) {...}
Assert.assertEquals(long expected,
                    long actual) {...}
Assert.assertArrayEquals(int[] expecteds,
                         int[] actuals) {...}
Assert.assertArrayEquals(Object[] expecteds,
                         Object[] actuals) {...}
```

```
class SLNode {
  private int value;
  private SLNode next;

  SLNode(int value, SLNode next) {
      this.value = value;
      this.next = next;
  }

  int getValue() {
      return value;
  }

  SLNode getNext() {
      return next;
  }

  void setNext(SLNode n) {
      next = n;
  }
}
```

Aufgabe 1 Basiswissen zur Vorlesung (__ /8 Punkte)

Kreuzen Sie an, ob die folgenden Aussagen richtig oder falsch sind.
Bewertung: keine Antwort: 0 Punkte; richtige Antwort: $+0,5$ Punkte; falsche Antwort: $-0,5$ Punkte.
Fuer diese Aufgabe werden mindestens 0 Punkte vergeben.

richtig	falsch	
☐	☒	Die erste Phase bei der Software Entwicklung ist immer die Implementierung.
☐	☒	Java Programme werden üblicherweise direkt in den Maschinencode eines Prozessors übersetzt.
☐	☒	Eine rekursive Methode ohne Schleifen terminiert immer abrupt.
☒	☐	In einem Stack werden die Daten nach dem LIFO-Prinzip (Last-in-First-out) verwaltet und in einer Queue nach dem FIFO-Prinzip (First-in-First-out).
☒	☐	Die Spezifikation "`modifies a`" sagt aus, dass die Variable `a` verändert werden kann, sie muss aber nicht unbedingt verändert werden.
☐	☒	Das Einfügen von Elementen in einen Suchbaum benötigt immer konstant viele Operationen, unabhängig von der aktuellen Anzahl der Knoten im Baum.
☒	☐	Interfaces können dazu verwendet werden, Implementierungsdetails vor dem Anwender zu verbergen.
☒	☐	In Java kann ein Objekt mehrere Typen haben.
☒	☐	Eine totale Java-Prozedur terminiert für alle Eingabewerte.
☒	☐	In Java sind Array-Typen immer Referenztypen.
☐	☒	Die Menge $\mathbb{Z}$ mit der Ordnung $<$ ist eine noethersche Ordnung.
☐	☒	Zum Übersetzen einer Datei "`Main.java`" kann der Befehl "`java Main.java`" verwendet werden.
☒	☐	Wenn ein Attribut mit **`private`** markiert ist, kann nur von der selben Klasse aus darauf zugegriffen werden
☒	☐	Der Ausdruck `new ArrayList<String>().get(0)` hat den Typ `String`, wenn `ArrayList` sich auf die Implementierung aus dem Java Collection Framework bezieht.
☒	☐	Eine kontextfreie Grammatik wird beschrieben durch ein Tupel mit 4 Elementen.
☐	☒	Eine mehrdeutige Grammatik definiert mehrere verschiedene Sprachen.

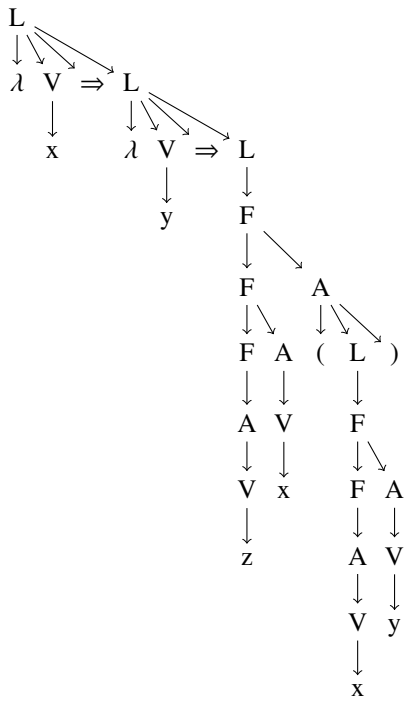
Aufgabe 2 Kontextfreie Grammatiken (__ /6 Punkte)

Gegeben ist die kontextfreie Grammatik $\Gamma = (N, T, \Pi, L)$ mit:

$$
\begin{aligned}
N &= \{L, F, A, V\} \\
T &= \{\lambda, \Rightarrow, (,), x, y, z\} \\
\Pi &= \left\{ \begin{array}{lcl}
L & \longrightarrow & \lambda\, V \Rightarrow L \\
L & \longrightarrow & F \\
F & \longrightarrow & F\, A \\
F & \longrightarrow & A \\
A & \longrightarrow & V \\
A & \longrightarrow & (\, L\,) \\
V & \longrightarrow & x \\
V & \longrightarrow & y \\
V & \longrightarrow & z
\end{array} \right\}
\end{aligned}
$$

Diese Grammatik beschreibt den sogenannten λ-Kalkül.

Geben Sie einen Syntaxbaum für das Wort "$\lambda x \Rightarrow \lambda y \Rightarrow z\ x\ (x\ y)$" an.



Aufgabe 3 Arrays (__ /13 Punkte)

a) Schreiben Sie eine Prozedur `swap`, welche ein **`int`**-Array `a` und zwei Zahlen `x`, `y` vom Typ **`int`** nimmt und im Array alle Vorkommen von `x` durch `y` ersetzt und alle Vorkommen von `y` durch `x`.

Beispiel: __ /3

```
int[] a = {7, 3, 5, 8, 3};
swap(a, 3, 8);
assertArrayEquals(new int[] {7, 8, 5, 3, 8}, a);

public static void swap(int[] ar, int x, int y) {
  for (int i = 0; i < ar.length; i++) {
    if (ar[i] == x) {
      ar[i] = y;
    } else if (ar[i] == y) {
      ar[i] = x;
    }
  }
}
```

b) Schreiben Sie eine Prozedur `seq`, welche ein **`int`**-Array nimmt und prüft, ob in diesem an irgend einer Stelle 4 mal hintereinander die gleiche Zahl vorkommt.

Beispiel: __ /4

```
int[] a = {7, 5, 5, 5, 3, 5, 8, 5, 3};
assertEquals(false, seq(a));
int[] b = {7, 3, 5, 5, 5, 5, 8, 3};
assertEquals(true, seq(b));

public static boolean seq(int[] ar) {
  for (int i = 0; i < ar.length-3; i++) {
    if (ar[i+1] == ar[i] && ar[i+2] == ar[i] && ar[i+3] == ar[i]) {
      return true;
    }
  }
  return false;
}
```

c) Schreiben Sie eine Prozedur `remZero`, welche ein zweidimensionales Array von **`int`**-Werten nimmt und ein neues zweidimensionales Array zurückgibt, in dem alle Spalten entfernt sind, welche nur aus Nullen bestehen. Sie können davon ausgehen, dass `ar` kein ungleichförmiges Array ist. Alle Elemente in `ar` sind also Arrays der gleichen Länge.

__/6

Beispiel:

```
int[][] a = {{0, 1, 2, 0, 0, 4},
             {0, 0, 0, 0, 0, 0},
             {0, 5, 6, 0, 7, 8}};
int[][] b = {{1, 2, 0, 4},
             {0, 0, 0, 0},
             {5, 6, 7, 8}};
assertArrayEquals(b, remZero(a));
```

```
public static int[][] remZero(int[][] ar) {
  if (ar.length == 0) {
    return new int[0][0];
  }

  boolean[] zeroCols = new boolean[ar[0].length];
  int newCols = 0;
  for (int col = 0; col < ar[0].length; col++) {
    boolean allZero = true;
    for (int row = 0; row < ar.length; row++) {
      if (ar[row][col] != 0) {
        allZero = false;
      }
    }
    zeroCols[col] = allZero;
    if (!allZero) {
      newCols++;
    }
  }

  int[][] result = new int[ar.length][];
  for (int row = 0; row < ar.length; row++) {
    result[row] = new int[newCols];
    int pos = 0;
    for (int col = 0; col < ar[row].length; col++) {
      if (!zeroCols[col]) {
        result[row][pos] = ar[row][col];
        pos++;
      }
    }
  }
  return result;
}
```

Aufgabe 4 Spezifikation und Testen (__ /7 Punkte)

```
class F {
  /*
   requires a != null
         && a.length > 0
         && (fuer alle int i in [0, a.length-2] gilt: a[i] < a[i+1])
   modifies \nothing
   ensures \result != null
        && \result.length == a.length
        && (fuer alle int i in [0, \result.length-1] gilt:
              \result[i] == (summe von j=0 bis i: a[j]))
  */
  static int[] f(int[] a) {
    // hier nicht gezeigt
  }
}
```

Dabei steht `(summe von j=0 bis i: a[j])` für den mathematischen Ausdruck $\sum_{j=0}^{i} \text{a[j]} = \text{a[0]} + \ldots + \text{a[i]}$.

Schreiben Sie zwei sinnvolle JUnit Testfälle, welche die Prozedur `f` bezüglich ihrer Spezifikation testen. Verwenden Sie für den ersten Testfall ein möglichst kurzes Array, das die Prozedur `f` als Eingabe akzeptiert. Für den zweiten Test verwenden Sie ein Array mit mindestens 5 Elementen.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertArrayEquals;
class FTest {

@Test
public void test1() {
  int[] ar = {1};
  int[] res = F.f(ar);
  int[] expected = {1};
  assertArrayEquals(expected, res);
}

@Test
public void test2() {
  int[] ar = {1, 3, 6, 8, 10};
  int[] res = F.f(ar);
  int[] expected = {1, 4, 10, 18, 28};
  assertEquals(expected, res);
}
```

Aufgabe 5 Terminierung (__ /7 Punkte)

Für diese Aufgabe ignorieren wir, dass es bei rekursiven Methoden in Java zu Stack-Überläufen (StackOverflow) kommen kann.

Zeigen Sie die Termierung der folgenden Methode `merge`. Dabei repräsentiert die Klasse `SLNode` die Knoten einer einfach verketteten Liste und ist wie in der Vorlesung definiert (und auf Seite 2 der Klausur nochmal abgedruckt). Sie können davon ausgehen, dass für die Methode **`int`** `size(SLNode ls)`, welche die Länge der Liste `ls` zurückliefert, die Terminierung bereits bewiesen wurde. Verwenden Sie für den Terminierungsbeweis für `merge` das Verfahren aus der Vorlesung!

```
1  public static SLNode merge(SLNode l1, SLNode l2) {
2    if (l1 == null) {
3        return l2;
4    } else if (l2 == null) {
5        return l1;
6    } else {
7        if (l1.getValue() <= l2.getValue()) {
8            SLNode rem = merge(l1.getNext(), l2);
9            return new SLNode(l1.getValue(), rem);
10       } else { // l1.getValue() > l2.getValue()
11           SLNode rem = merge(l1, l2.getNext());
12           return new SLNode(l2.getValue(), rem);
13       }
14   }
15 }
```

1. Als Vorbedingung wählen wir: l1 und l2 sind gültige und somit zyklenfreie Listen.
2. Gültige Listen haben als Restliste wieder eine gültige Liste. Somit sind die Parameter `l1.getNext()` in Zeile 8 und `l2.getNext()` in Zeile 11 wiederum gültige Listen. Die Vorbedingung gilt weiterhin.
3. Als Abstiegsfunktion wählen wir nun
   $h(l1, l2) = size(l1) + size(l2)$
   Diese bildet in die natürlichen Zahlen ab, da $size(l1) \geq 0$ und $size(l2) \geq 0$ gilt.
4. Fall Zeile 8:
   Zu zeigen: $h(l1, l2) > h(l1.getNext(), l2)$
   Es gilt `l1 != null` wegen if in Zeile 2. Somit ist
   $h(l1, l2) = size(l1) + size(l2) > size(l1) - 1 + size(l2) = h(l1.getNext(), l2)$
   Da die Länge von `l1.getNext()` um eins kleiner ist als die Länge von `l1`.

   Fall Zeile 11:
   Zu zeigen: $h(l1, l2) > h(l1, l2.getNext())$
   Es gilt `l2 != null` wegen if in Zeile 4. Somit ist
   $h(l1, l2) = size(l1) + size(l2) > size(l1) + size(l2) - 1 = h(l1, l2.getNext())$
   Da die Länge von `l2.getNext()` um eines kleienr ist als die Länge von `l2`.

Aufgabe 6 Bäume (__ /10 Punkte)

Gegeben sind die folgenden Klassen, welche binäre Bäume repräsentieren:

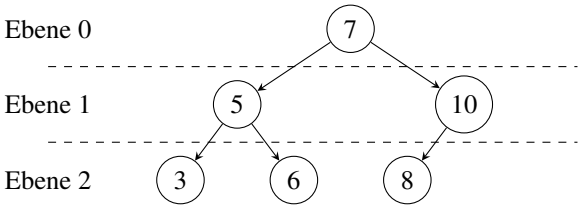
```
public class Tree {
    private TreeNode root;

    public void add(int x) { ... }
}

class TreeNode {
    int mark;
    TreeNode left, right;
    TreeNode(TreeNode left, TreeNode right, int mark) {
        this.left = left; this.right = right; this.mark = mark;
    }
}
```

a) Schreiben Sie eine Methode `List<Integer> levelorder()` in der Klasse `Tree`, welche einen Breitendurchlauf durch den Baum macht und die Elemente in dieser Reihenfolge in eine Liste schreibt. Das heißt, dass in der Ergebnis-Liste erst die Markierung des Wurzelknotens, dann die Markierungen der ersten Ebene von links nach rechts und so weiter für die restlichen Ebenen des Baums stehen soll.

__ /6



.levelorder() $\Longrightarrow$ `[7,5,10,3,6,8]`

```
in Tree:
public List<Integer> levelorder() {
  List<TreeNode> worklist = new LinkedList<>();
  worklist.add(root);
  List<Integer> res = new LinkedList<>();
  while(!worklist.isEmpty()) {
    TreeNode next = worklist.remove(0);
    res.add(next.mark);
    if (next.left != null) {
      worklist.add(next.left);
    }
    if (next.right != null) {
      worklist.add(next.right);
    }
  }

  return res;
}
```

b) Gegeben ist das folgende Interface für Iteratoren über int-Werte:

```
interface IntIterator {
  public int next();
  public boolean hasNext();
}
```

Implementieren Sie einen `TreeIterator` als Subtyp von `IntIterator`, der in Level-Order die Markierungen des Baumes besucht. Der Iterator soll dabei analog zu den Iteratoren auf Listen funktionieren, die wir in der Vorlesung kennen gelernt haben. Implementieren Sie dazu die Methode `IntIterator iterator()` in der Klasse `Tree`.

__/4

```
class TreeIterator implements IntIterator {
  private Iterator<Integer> liter;
  public TreeIterator(Iterator<Integer> liter) {
    this.liter = liter;
  }
  public int next() {
    return this.liter.next();
  }
  public boolean hasNext() {
    return this.liter.hasNext();
  }
}

public TreeIterator iterator() {
  return new TreeIterator(levelorder().iterator());
}
```

Aufgabe 7 Programmverständnis (__/7 Punkte)

a) Welche Methoden müssen von der Klasse `E` in folgendem Beispiel implementiert werden? __/2

```
1  interface A {
2    int f();
3  }
4
5  interface B {
6    int g();
7  }
8
9  interface C extends A, B {
10   int h();
11 }
12
13 interface D extends C {
14   int i();
15 }
16
17 class E implements A, C {
18   // ...
19 }
```

f, g und h

b) Wenn die Typen wie in Teil a) definiert sind, welche der folgenden Zuweisungen werden vom Java-Compiler akzeptiert? __/2

```
1  A a = null;
2  C c = null;
3  a = a;
4  a = c;
5  c = a;
```

3: ja (gleicher Typ), 4: ja (C ist Subtyp von A), 5: nein (A ist kein Subtyp von C)

c) Was wird von folgendem Programm ausgegeben? Erklären Sie dazu, was dynamische Methodenbindung ist und wo sie in diesem Beispiel auftritt.

__/3

```
1  interface Producer {
2    void produce(Consumer c);
3  }
4
5  interface Consumer {
6    void consume(int x);
7  }
8
9  class A implements Producer, Consumer {
10   private int x = 0;
11   public void produce(Consumer c) {
12     x = x + 1;
13     c.consume(2*x);
14   }
15   public void consume(int x) {
16     System.out.println("A " + x);
17   }
18 }
19
20 class B implements Consumer {
21   public void consume(int x) {
22     System.out.println("B " + x);
23   }
24 }
25
26 public class Main {
27   public static void main(String[] args) {
28     A a = new A();
29     Consumer[] consumers = {
30       a,
31       new B(),
32       new A()
33     };
34     Producer prod = a;
35     for (int i=0; i<consumers.length; i++) {
36       prod.produce(consumers[i]);
37     }
38   }
39 }
```

Die Ausgabe ist `A 2, B 4, A 6`. Dynamische Methodenbindung (oder Methodenauswahl) bedeutet, dass zur Laufzeit abhängig vom dynamischen Typs des Zielobjekts die Implementierung ausgewählt wird, die bei einem Methodenaufruf ausgeführt wird. Im Beispiel passiert dies in Zeile 13 (für die anderen Methoden gibt es im Beispiel nur eine Implementierung, Java verwendet aber trotzdem intern die dynamische Methodenbindung). Beim ersten Aufruf ist `c` vom Typ `A`, also wird die Implementierung in der Klasse `A` aufgerufen. Beim zweiten Aufruf dann vom Typ `B` und dann wieder `A`.

Aufgabe 8 Objektorientierte Modellierung (__ /15 Punkte)

Gegeben ist die folgende Beschreibung eines Fuhrunternehmens, welches die Planung seiner Touren teilweise automatisieren will:

> Es gibt verschiedene Arten von Fahrzeugen: Zugfahrzeuge und Anhänger. Ein Zugfahrzeug ist entweder ein LKW oder ein Kleintransporter (PKW). An einen LKW kann ein Anhänger angekoppelt werden, an einen PKW jedoch nicht. Jedes Fahrzeug hat ein maximales Ladegewicht (in Kilogramm). Identifiziert werden die Fahrzeuge über ihr Kennzeichen.
>
> Zur Planung der Touren muss zusätzlich noch für jedes Fahrzeug gespeichert werden, welche Waren ihm zugeordnet sind. Dabei haben alle Waren ein Gewicht (in Kilogramm). Außerdem muss für jeden LKW gespeichert werden, ob und welchen Anhänger er ziehen sollen.

a) Implementieren Sie Klassen, welche wie oben beschrieben die Fahrzeuge des Unternehmens und die Zuweisung von Waren an Fahrzeuge modellieren. Jedes Attribut soll über einen Konstruktor oder eine setter-Methode initialisierbar und über eine getter-Methode lesbar sein. Es soll nicht möglich sein, ein Fahrzeug zu erstellen, ohne ein Kennzeichen anzugeben. Achten Sie auf gute Kapselung.

Hinweis: Achten Sie auch darauf, dass Aufgabe b) mit Ihrer Modellierung gut lösbar ist. **__ /10**

```
public abstract class Fahrzeug {
  private int maximalGewicht;
  private String kennzeichen;
  private List<Ware> waren;

  public Fahrzeug(String kennz, int maxGewicht) {
    kennzeichen = kennz;
    maximalGewicht = maxGewicht;
  }
  public String getKennzeichen() { return kennzeichen; }
  public int getMaxGewicht() { return maximalGewicht; }
  public List<Ware> getWaren() { return waren; }
  public void setWaren(List<Ware> w) { waren = w; }
}

public abstract class Zugfahrzeug extends Fahrzeug {
  public Zugfahrzeug(String kennz, int maxGewicht) {
    super(kennz, maxGewicht);
  }
  public abstract Anhaenger getAnhaenger();
}

public class LKW extends Zugfahrzeug {
  private Anhaenger anhaenger;
  public LKW(String kennz, int maxGewicht) {
    super(kennz, maxGewicht);
  }
  public Anhaenger getAnhaenger() { return anhaenger; }
  public void setAnhanger(Anhaenger a) { anhaenger = a; }
}
```

```
public class Kleintransporter extends Zugfahrzeug {
  public Kleintransporter(String kennz, int maxGewicht) {
    super(kennz, maxGewicht);
  }
  public Anhaenger getAnhaenger() { return null; }
}

public class Anhaenger extends Fahrzeug {
  public Anhaenger(String kennz, int maxGewicht) {
    super(kennz, maxGewicht);
  }
}

public class Ware {
  private int gewicht;
  public int getGewicht() { return gewicht; }
  public void setGewicht(int g) { gewicht = g; }
}
```

b) Implementieren Sie eine statische Methode `checkCapacity` in einer Klasse `Tourenplanung`, welche eine Liste von Zugfahrzeugen einschließlich der geplanten Warenverteilung nimmt und für diese Planung prüft, ob das maximale Ladegewicht für jedes Fahrzeug eingehalten ist. Denken Sie daran, dass auch die Anhänger der Zugfahrzeuge überprüft werden müssen.

Hinweis: Ergänzen Sie ggf. Ihre Klassen aus a) um weitere Methoden, um b) zu lösen.

—/5

```
public class Tourenplanung {
  public static boolean checkCapacity(List<Zugfahrzeug> fahrzeuge) {
    for (Zugfahrzeug zfahrzeug : fahrzeuge) {
      if (!checkFahrzeug(zfahrzeug)) {
        return false;
      }
      if (zfahrzeug.getAnhaenger() != null
        && !checkFahrzeug(zfahrzeug.getAnhaenger())) {
          return false;
      }
    }
    return true;
  }

  private static boolean checkFahrzeug(Fahrzeug fahrzeug) {
    int gewicht = 0;
    for (Ware ware : fahrzeug.getWaren()) {
      gewicht += ware.getGewicht();
    }
    return gewicht <= fahrzeug.getMaxGewicht();
  }
}
```