Aufgabe 1 (Programmanalyse): (9 + 6 = 15 Punkte)

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Tragen Sie hierzu jeweils die ausgegebenen Zeichen in die markierten Stellen hinter „`OUT:`“ ein. Schließen Sie dabei ausgegebene `Strings` in `""` ein, z.B. `"x"`.

```
public class A {
  public String s = null;

  public A(int s) {
    this.s = "a";
  }

  public A(String s) {
    this.s = "b";
  }

  public String getS() {
    return this.s;
  }

  public int f(int p) {
    return 1;
  }

  public int f(A p) {
    return 2;
  }
}
```

```
public class B extends A {
  public String s = "c";

  public B() {
    super(null);
  }
  public B(String s) {
    this(0);
    this.s = s;
  }
  public B(int x) {
    super("d");
    this.s = this.s + x;
  }

  public String getS() {
    return this.s;
  }
  public int f(A p) {
    return 3;
  }
  public int f(B p) {
    return 4;
  }
  private String g() {
    return "e";
  }
}
```

```
public class M {
  public static void main(String[] args) {
    A a1 = new A(5);
    System.out.println(a1.s);                          //OUT: [    ]

    B b1 = new B("g");
    System.out.println(((A)b1).s + " " + ((B)b1).s);//OUT: [    ] [    ]

    A ab = new B(6);
    System.out.println(((A)ab).s + " " + ((B)ab).s);//OUT: [    ] [    ]

    System.out.println(ab.getS());                     //OUT: [    ]

    System.out.println(a1.f(a1));                      //OUT: [    ]

    System.out.println(b1.f(ab));                      //OUT: [    ]

    System.out.println(ab.f(b1));                      //OUT: [    ]
  }
}
```

b) Wir schreiben zusätzlich zu `A` und `B` eine neue Klasse `C`. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
1  public class C extends B {
2    public int s;
3
4    public C() {
5      String g = g();
6      super(g);
7      this.s = 0;
8    }
9
10   public boolean f(int p) {
11     return super.g().length() != 0;
12   }
13
14   private String g() {
15     return "f";
16   }
17 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
      A a1 = new A(5);
      System.out.println(a1.s);                          //OUT: [ "a"]

      B b1 = new B("g");
      System.out.println(((A)b1).s + " " + ((B)b1).s);//OUT: [ "b"] [ "g"]

      A ab = new B(6);
      System.out.println(((A)ab).s + " " + ((B)ab).s);//OUT: [ "b"] ["c6"]

      System.out.println(ab.getS());                     //OUT: ["c6"]

      System.out.println(a1.f(a1));                      //OUT: [  2 ]

      System.out.println(b1.f(ab));                      //OUT: [  3 ]

      System.out.println(ab.f(b1));                      //OUT: [  3 ]
    }
  }
```

b)

- Compilerfehler: Constructor call must be the first statement in a constructor (Line 6)
  Es wurde einer der folgenden Fehler erkannt:
  - Die Methode `g` aus der Klasse `C` kann nicht vor dem `super` Konstruktor aufgerufen werden (Line 5).
  - Der Konstruktoraufruf muss die erste Anweisung des Konstruktors sein (Line 6).
- Compilerfehler: The return type is incompatible with `A.f(int)` (Line 10)
  Es wurde einer der folgenden Fehler erkannt:
  - Die Methode `f` aus der Klasse `C` muss den Rückgabetyp `int` haben (Line 10).
  - Die Methode `f` aus der Klasse `C` muss eine andere Signatur als die Methode `f` aus der Klasse `A` haben (Line 10).
- Compilerfehler: The method `g()` from the type `B` is not visible (Line 11)
  Es wurde einer der folgenden Fehler erkannt:
  - Die Methode `g` aus der Klasse `B` ist nicht sichtbar in der Klasse `C` und kann somit nicht aufgerufen werden (Line 11).
  - Die Methode `g` muss nicht auf `super` sondern auf `this` aufgerufen werden (Line 11).

Aufgabe 2 (Hoare-Kalkül): (14 + 3 = 17 Punkte)

Gegeben sei folgendes Java-Programm P über den Integer-Variablen `a`, `b`, `x`, `res` und `y`:

$\langle$ `b` ≥ 0 $\rangle$ (Vorbedingung)

```
x = b;
res = a;
y = 1;
while (x > 0) {
    if (x % 2 == 0) {
        x = x / 2;
    }
    else {
        res = res - y;
        x = (x - 1) / 2;
    }
    y = 2 * y;
}
```

$\langle$ `res` $=$ `a` $-$ `b` $\rangle$ (Nachbedingung)

a) Vervollständigen Sie die folgende Verifikation des Algorithmus P im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Der Programmausdruck `x / 2` berechnet das abgerundete Ergebnis der Ganzzahldivision, d.h. $\lfloor \frac{x}{2} \rfloor$.
- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x+1 = y+1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.
- Sie können die folgende Wertetabelle zum Finden der Schleifeninvariante benutzen.

Anzahl Schleifendurchläufe	`a`	`b`	`res`	`x`	`y`
0	8	5	8	5	1
1	8	5	7	2	2
2	8	5	7	1	4
3	8	5	3	0	8

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und unter Verwendung des Hoare-Kalküls die Terminierung bewiesen werden.

Lösung:

a)

$\langle \mathtt{b} \geq 0 \rangle$
$\langle \mathtt{b} \geq 0 \wedge \mathtt{b} = \mathtt{b} \wedge \mathtt{a} = \mathtt{a} \wedge 1 = 1 \rangle$

```
x = b;
```

$\langle \mathtt{x} \geq 0 \wedge \mathtt{x} = \mathtt{b} \wedge \mathtt{a} = \mathtt{a} \wedge 1 = 1 \rangle$

```
res = a;
```

$\langle \mathtt{x} \geq 0 \wedge \mathtt{x} = \mathtt{b} \wedge \mathtt{res} = \mathtt{a} \wedge 1 = 1 \rangle$

```
y = 1;
```

$\langle \mathtt{x} \geq 0 \wedge \mathtt{x} = \mathtt{b} \wedge \mathtt{res} = \mathtt{a} \wedge \mathtt{y} = 1 \rangle$
$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \rangle$

```
while (x > 0) {
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \wedge \mathtt{x} > 0 \rangle$

```
  if (x % 2 == 0) {
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \wedge \mathtt{x} > 0 \wedge \mathtt{x} \mod 2 = 0 \rangle$
$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \lfloor \frac{\mathtt{x}}{2} \rfloor \cdot (2 \cdot \mathtt{y}) \wedge \lfloor \frac{\mathtt{x}}{2} \rfloor \geq 0 \rangle$

```
    x = x / 2;
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot (2 \cdot \mathtt{y}) \wedge \mathtt{x} \geq 0 \rangle$

```
  } else {
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \wedge \mathtt{x} > 0 \wedge \neg(\mathtt{x} \mod 2 = 0) \rangle$
$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{y} - \lfloor \frac{\mathtt{x}-1}{2} \rfloor \cdot (2 \cdot \mathtt{y}) \wedge \lfloor \frac{\mathtt{x}-1}{2} \rfloor \geq 0 \rangle$

```
    res = res - y;
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \lfloor \frac{\mathtt{x}-1}{2} \rfloor \cdot (2 \cdot \mathtt{y}) \wedge \lfloor \frac{\mathtt{x}-1}{2} \rfloor \geq 0 \rangle$

```
    x = (x - 1) / 2;
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot (2 \cdot \mathtt{y}) \wedge \mathtt{x} \geq 0 \rangle$

```
  }
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot (2 \cdot \mathtt{y}) \wedge \mathtt{x} \geq 0 \rangle$

```
    y = 2 * y;
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \rangle$

```
}
```

$\langle \mathtt{a} - \mathtt{b} = \mathtt{res} - \mathtt{x} \cdot \mathtt{y} \wedge \mathtt{x} \geq 0 \wedge \neg(\mathtt{x} > 0) \rangle$
$\langle \mathtt{res} = \mathtt{a} - \mathtt{b} \rangle$

b) Eine gültige Variante ist $V = \mathtt{x}$, denn aus der Schleifenbedingung `x > 0` folgt direkt $V \geq 0$.
Darüberhinaus ist folgende Ableitung korrekt.

$\langle \mathtt{x} = \mathtt{m} \wedge \mathtt{x} > 0 \rangle$

```
  if (x % 2 == 0) {
```

$\langle \mathtt{x} = \mathtt{m} \wedge \mathtt{x} > 0 \wedge \mathtt{x} \mod 2 = 0 \rangle$
$\langle \lfloor \frac{\mathtt{x}}{2} \rfloor < \mathtt{m} \rangle$

```
    x = x / 2;
```

$\langle \mathtt{x} < \mathtt{m} \rangle$

```
  } else {
```

$\langle \mathtt{x} = \mathtt{m} \wedge \mathtt{x} > 0 \wedge \neg(\mathtt{x} \mod 2 = 0) \rangle$
$\langle \lfloor \frac{\mathtt{x}-1}{2} \rfloor < \mathtt{m} \rangle$

```
    res = res - y;
```

$\langle \lfloor \frac{\mathtt{x}-1}{2} \rfloor < \mathtt{m} \rangle$

```
    x = (x - 1) / 2;
```

$\langle \mathtt{x} < \mathtt{m} \rangle$

```
  }
```

$\langle \texttt{x} < \texttt{m} \rangle$

```
y = 2 * y;
```

$\langle \texttt{x} < \texttt{m} \rangle$

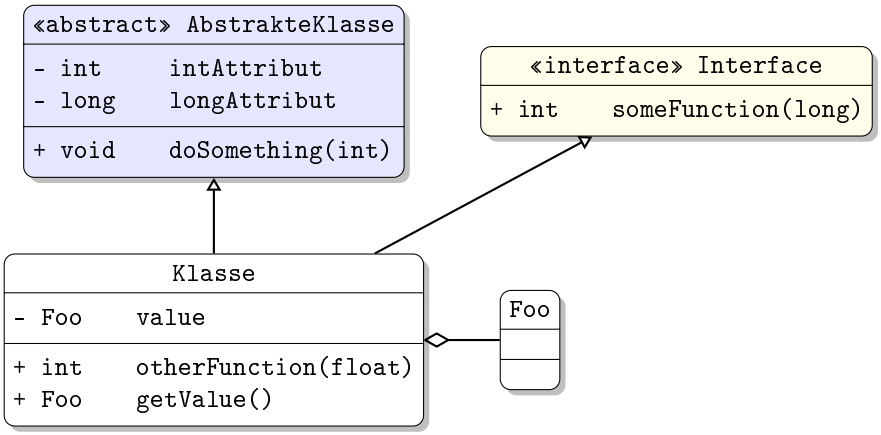
Aufgabe 3 (Klassenhierarchie): (7 + 11 = 18 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Klassenhierarchie zur Verwaltung von Buslinien. Alle Zeiten, die in dieser Aufgabe vorkommen, sollen vom Typ `LocalTime` sein, den Sie als gegeben annehmen können. Ein Objekt vom Typ `LocalTime` beinhaltet eine Uhrzeit, aber kein Datum.

- Eine Linie hat ein Array von Haltestellen, eine Nummer und einen Fahrplan (den Linienfahrplan). Die Methode `boolean istNachtlinie()` gibt zurück, ob diese Linie nur nachts bedient wird.
- Eine Haltestelle hat einen Namen und einen Fahrplan (den Aushangfahrplan).
- Eine Kreishaltestelle ist eine Haltestelle, die auf einem Kreis von Haltestellen liegt. Daher hat jede Kreishaltestelle einen Nachfolger, der wiederum eine Kreishaltestelle ist.
- Linien und Haltestellen sind abfragbar. Die Methode `Fahrplan getAbfahrten` nimmt zwei Uhrzeiten entgegen und gibt den Fahrplan zurück, der alle Abfahrten zwischen diesen beiden Uhrzeiten enthält.
- Ein Fahrplan hat eine Reihe von Einträgen.
- Ein Eintrag hat eine Haltestelle, eine Abfahrtszeit und eine Linie.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Sachverhalte. Notieren Sie keine Konstruktoren oder Selektoren. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden und markieren Sie alle Klassen als abstrakt, bei denen dies sinnvoll ist.

Verwenden Sie hierbei die folgende Notation:



Eine Klasse wird hier durch einen Kasten dargestellt, in dem der Name der Klasse sowie alle in der Klasse definierten bzw. überschriebenen Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \multimap\!\triangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Der Pfeil $B \multimap\!\diamond A$ bedeutet, dass A ein Objekt vom Typ B benutzt. Benutzen Sie `-`, um `private` abzukürzen, und `+` für alle anderen Sichtbarkeiten (wie z. B. `public`). Fügen Sie Ihrem Diagramm keine Kästen für vordefinierte Klassen wie `LocalTime` hinzu.

b) Schreiben Sie außerhalb der zuvor modellierten Klassen eine Java-Methode `pruefeKreislinie`, die eine Linie als Eingabeparameter bekommt. Der Aufruf `pruefeKreislinie(linie)` soll einen Wahrheitswert zurückliefern, der angibt, ob `linie` eine strukturell korrekte Kreisline ist. Dazu müssen die Haltestellen im Haltestellen-Array von `linie` alle Kreishaltestellen sein. Außerdem müssen die Haltestellen im Array genau in der Reihenfolge vorkommen, wie sie über die Nachfolger-Attribute verknüpft sind. Insbesondere muss die erste Haltestelle im Array Nachfolger der letzten sein.

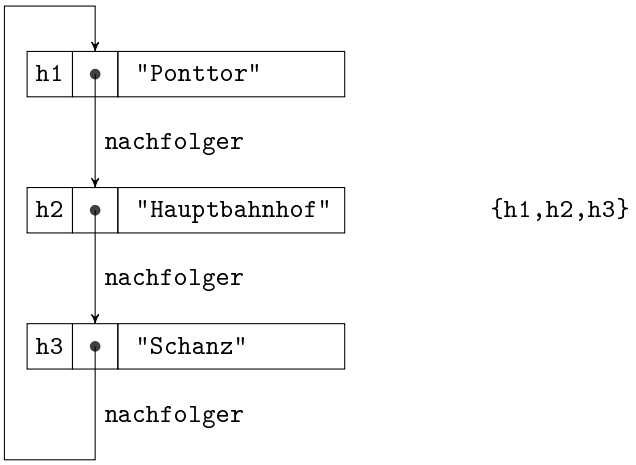
Ein Beispiel für eine strukturell korrekte Kreislinie sehen Sie unten auf der Seite.

Gehen Sie davon aus, dass es zu jedem Attribut geeignete Selektoren gibt und alle Klassen einen Konstruktor besitzen, der für jedes Attribut einen Parameter hat.

Hinweise:

- Gehen Sie davon aus, dass alle vorkommenden Arrays und Attribute weder `null` sind noch `null` enthalten und dass außerdem `l` nicht `null` ist.
- Gehen Sie davon aus, dass das Haltestellen-Array der Linie `l` nicht leer ist.
- Sie dürfen Hilfsmethoden schreiben.

Beispiel:

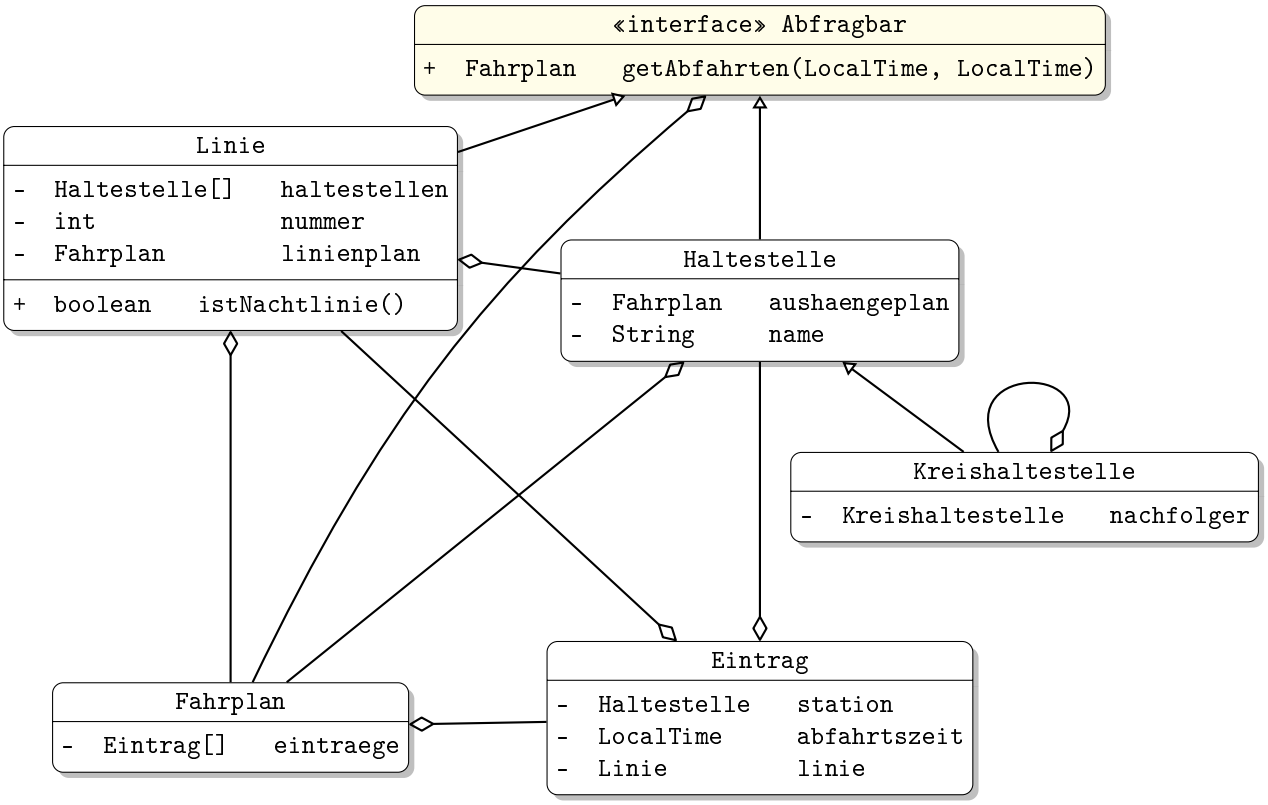


Oben links sehen Sie drei Objekte, dargestellt durch je drei nebeneinander liegende Rechtecke. Das linke Rechteck enthält dabei immer einen Bezeichner. Die Objekte `h1` bis `h3` sind `Kreishaltestelle`-Objekte, deren `nachfolger`-Attribut wieder auf ein `Kreishaltestelle`-Objekt zeigt. Außerdem enthält jedes `Kreishaltestelle`-Objekt im dritten Rechteck den Wert seines `name`-Attributs aus der Oberklasse `Haltestelle`. Oben rechts sehen Sie ein Array mit diesen drei Objekten, sodass eine `Linie` mit diesem Haltestellen-Array eine strukturell korrekte Kreislinie wäre.

```
public static boolean pruefeKreislinie(Linie linie) {
```

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)

```
public static boolean pruefeKreislinie(Linie linie) {
  for (Haltestelle h : linie.getHaltestellen()) {
    if (!(h instanceof Kreishaltestelle)) {
      return false;
    }
  }

  Haltestelle h2 = linie.getHaltestellen()[linie.getHaltestellen().length - 1];
  Kreishaltestelle r2 = (Kreishaltestelle)h2;
  for (Haltestelle h : linie.getHaltestellen()) {
    Kreishaltestelle r = (Kreishaltestelle)h;
    if (r2.getNachfolger() != r) {
      return false;
    }
    r2 = r;
  }
  return true;
}
```

Alternative Lösung:

```
public static boolean pruefeKreislinie(Linie linie) {
  Haltestelle[] haltestellen = linie.getHaltestellen();
  Haltestelle first1 = haltestellen[0];
  if (!(first1 instanceof Kreishaltestelle)) {
    return false;
  }

  Kreishaltestelle first = (Kreishaltestelle)first1;
  Kreishaltestelle current = first.getNachfolger();
  int index = 1;
```

```
  while (current != first) {
    if (index >= haltestellen.length) {
      return false;
    }
    if (current != haltestellen[index]) {
      return false;
    }
    current = current.getNachfolger();
    ++index;
  }
  return index == haltestellen.length;
}
```

Aufgabe 4 (Programmierung in Java): (4 + 4 + 5 + 11 + 7 = 31 Punkte)

In dieser Aufgabe wollen wir uns mit Kreislisten beschäftigen. Eine Kreisliste ist eine einfach verkettete Liste, welche eine Kreisstruktur bildet. Dazu nutzen wir die Klasse `Kreisliste`.
Eine Kreisliste besteht aus einem oder mehreren `Kreisliste`-Objekten, deren `next`-Attribute (*Nachfolger*) eine Kreisstruktur bilden. Die in der Liste gespeicherten *Werte* sind in den `value`-Attributen der `Kreisliste`-Objekte abgelegt.

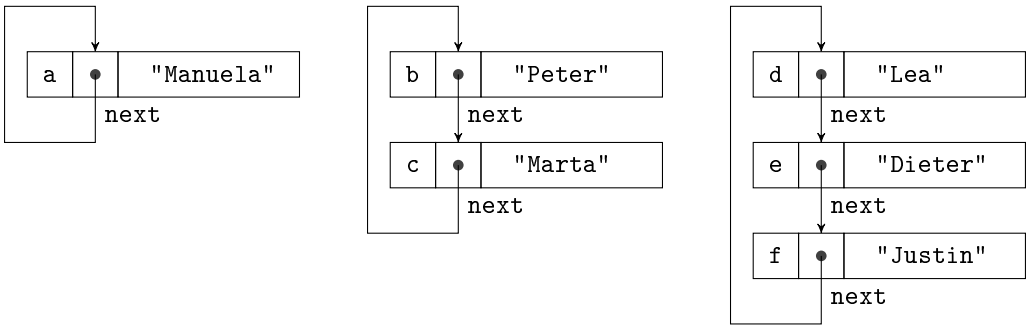
```
public class Kreisliste {
  private Kreisliste next;
  private final String value;

  public Kreisliste(Kreisliste next, String value) {
    this.next = next;
    this.value = value;
  }

  public Kreisliste getNext() {
    return next;
  }
  public String getValue() {
    return value;
  }
}
```

Beispiel:

In der folgenden Grafik sind beispielhaft die Objekte `a` bis `f` vom Typ `Kreisliste` dargestellt. Die Bezeichner `a` bis `f` sind nur dazu da, damit wir in den Beispielen der folgenden Teilaufgaben auf die hier dargestellten Objekte verweisen können. Insbesondere sollte Ihre Lösung die hier genannten Beispielobjekte nicht explizit erwähnen.



Wir sehen sechs Objekte, dargestellt durch je drei nebeneinander liegende Rechtecke. Das linke Rechteck enthält dabei immer einen Bezeichner. Die Objekte `a` bis `f` sind `Kreisliste`-Objekte, deren `next`-Attribut wieder auf ein `Kreisliste`-Objekt zeigt. Außerdem enthält jedes `Kreisliste`-Objekt im dritten Rechteck den Wert seines `value`-Attributs.

Hinweise:

- Sie dürfen in allen Teilaufgaben davon ausgehen, dass nur auf zyklischen Kreislisten gearbeitet wird. Wenn man nur den `next`-Referenzen folgt, so erreicht man garantiert irgendwann erneut das `Kreisliste`-Objekt bei dem man gestartet ist. (Dies ist anders als nachher in Aufgabe 5). Insbesondere erreicht man nie den Wert `null`, wenn man nur den `next`-Referenzen folgt.
- Sie dürfen in allen Teilaufgaben davon ausgehen, dass das `value`-Attribut jedes `Kreisliste`-Objekts nicht den Wert `null` hat.
- Sie dürfen in allen Teilaufgaben weitere private Hilfsmethoden schreiben.
- Achten Sie darauf, bei allen Teilaufgaben nicht nur die Beispiele korrekt zu behandeln, sondern den allgemeinen Fall zu lösen.

a) Implementieren Sie in der Klasse `Kreisliste` die Methode `singleton`. Sie erstellt eine neue `Kreisliste` mit genau einem Wert. Der Nachfolger ist die neu erstellte `Kreisliste`. Der Wert ist der `String`, welcher der Methode im `value`-Parameter übergeben wurde.

Beispiel:

Wenn beispielsweise `Kreisliste.singleton("Manuela")` aufgerufen wird, so soll eine Referenz auf eine `Kreisliste` wie `a` zurückgegeben werden.

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `value` der Methode `singleton` nicht `null` ist.

```
public static Kreisliste singleton(String value) {
```

b) Wenn `x` und `y` `Kreisliste`-Objekte sind und `y` der Nachfolger von `x` ist, dann ist `x` der *Vorgänger* von `y`.

Implementieren Sie in der Klasse `Kreisliste` die Methode `getPrevious`. Sie gibt den Vorgänger des aktuellen `Kreisliste`-Objekts zurück. Dazu folgt sie solange den `next`-Referenzen, bis sie ein `Kreisliste`-Objekt findet, dessen Nachfolger das `Kreisliste`-Objekt ist, auf dem die `getPrevious`-Methode aufgerufen wurde.

Verwenden Sie in dieser Teilaufgabe keine Rekursion, sondern **ausschließlich Schleifen**.

Beispiele:

- Wenn beispielsweise `a.getPrevious()` aufgerufen wird, so soll eine Referenz auf `a` zurückgegeben werden.
- Wenn beispielsweise `b.getPrevious()` aufgerufen wird, so soll eine Referenz auf `c` zurückgegeben werden.
- Wenn beispielsweise `c.getPrevious()` aufgerufen wird, so soll eine Referenz auf `b` zurückgegeben werden.
- Wenn beispielsweise `d.getPrevious()` aufgerufen wird, so soll eine Referenz auf `f` zurückgegeben werden.
- Wenn beispielsweise `e.getPrevious()` aufgerufen wird, so soll eine Referenz auf `d` zurückgegeben werden.
- Wenn beispielsweise `f.getPrevious()` aufgerufen wird, so soll eine Referenz auf `e` zurückgegeben werden.

```
public Kreisliste getPrevious() {
```

c) In dieser Aufgaben wollen wir zwei Kreislisten zusammenfügen.

Implementieren Sie in der Klasse `Kreisliste` die Methode `merge`. Sie fügt die aktuelle `Kreisliste` mit der `Kreisliste` zusammen, welche im Parameter `other` übergeben wurde. Dazu ermittelt sie zunächst die Vorgänger beider Kreislisten und setzt anschließend den Nachfolger des Vorgängers der aktuellen `Kreisliste` auf `other` und den Nachfolger des Vorgängers von `other` auf die aktuelle `Kreisliste`.

Sie dürfen in dieser Teilaufgabe die Methode `getPrevious` aus der vorherigen Teilaufgabe verwenden, unabhängig davon, ob Sie die vorherige Teilaufgabe bearbeitet haben.

Beispiel:

Wenn beispielsweise `b.merge(d)` aufgerufen wird, so soll der Nachfolger von `c` auf `d` gesetzt werden und der Nachfolger von `f` auf `b` gesetzt werden.

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `other` der Methode `merge` nicht `null` ist.
- Sie dürfen davon ausgehen, dass die aktuelle Kreisliste und `other` nicht Teil der selben Kreisliste sind. Wenn man nur den `next`-Referenzen folgt, ist es also nicht möglich, von der aktuellen Kreisliste die Kreisliste `other` zu erreichen (und umgekehrt). Der Aufruf `b.merge(c)` ist also ungültig.

```
public void merge(Kreisliste other) {
```

d) Wenn `x` und `y` `Kreisliste`-Objekte sind, dann ist der *Index von* `y` *in* `x` die Anzahl der `next`-Referenzen, denen mindestens gefolgt werden muss, um von `x` nach `y` zu gelangen.

Implementieren Sie in der Klasse `Kreisliste` die Methode `indexOf`. Der Aufruf `indexOf(w)` findet zunächst das erste `Kreisliste`-Objekt mit dem Wert `w` in der aktuellen Kreisliste. Anschließend wird der Index dieses `Kreisliste`-Objekts in der aktuellen `Kreisliste` zurückgegeben. Falls die Kreisliste kein `Kreisliste`-Objekt mit dem übergebenen Wert `w` enthält, so soll eine `UnknownValueException` geworfen werden, welche wie folgt definiert ist:

```
public class UnknownValueException extends Exception {}
```

Verwenden Sie in dieser Teilaufgabe keine Schleifen, sondern **ausschließlich Rekursion**.

Beispiele:

- Wenn beispielsweise `a.indexOf("Manuela")` aufgerufen wird, so soll 0 zurückgegeben werden.
- Wenn beispielsweise `b.indexOf("Marta")` aufgerufen wird, so soll 1 zurückgegeben werden.
- Wenn beispielsweise `c.indexOf("Marta")` aufgerufen wird, so soll 0 zurückgegeben werden.
- Wenn beispielsweise `d.indexOf("Dieter")` aufgerufen wird, so soll 1 zurückgegeben werden.
- Wenn beispielsweise `e.indexOf("Dieter")` aufgerufen wird, so soll 0 zurückgegeben werden.
- Wenn beispielsweise `f.indexOf("Dieter")` aufgerufen wird, so soll 2 zurückgegeben werden.
- Wenn beispielsweise `a.indexOf("Dieter")` aufgerufen wird, so soll eine `UnknownValueException` geworfen werden.

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `value` der Methode `indexOf` nicht `null` ist.

```
public int indexOf(String value) throws UnknownValueException {
```

e) Implementieren Sie in der Klasse `Kreisliste` die Methode `toList`, welche die aktuelle Kreisliste in eine `Liste` von `String`s umwandelt. Achten Sie dabei darauf, die `String`-Darstellung der Kreislisten so zu generieren, wie es in den Beispielen gezeigt wird. Orientieren Sie sich auch für die Reihenfolge der `String`s in der zurückgegebenen `Liste` an den Beispielen.

Beispiele:

- Wenn beispielsweise `a.toList()` aufgerufen wird, so soll eine Liste mit dem folgenden `String` zurückgegeben werden:
  - `"Manuela"`
- Wenn beispielsweise `b.toList()` aufgerufen wird, so soll eine Liste mit folgenden `String`s zurückgegeben werden:
  - `"Peter"`
  - `"Marta"`
- Wenn beispielsweise `e.toList()` aufgerufen wird, so soll eine Liste mit folgenden `String`s zurückgegeben werden:
  - `"Dieter"`
  - `"Justin"`
  - `"Lea"`

Hinweise:

- `LinkedList<T>` und `List<T>` bezeichnen Typen, welche sich im Paket `java.util` befinden. Gehen Sie daher davon aus, dass die Datei `Kreisliste.java` mit `import java.util.*;` beginnt.
- Sie dürfen die Methode `boolean add(T t)` aus dem Interface `List<T>` benutzen, um hinten an eine Liste weitere Elemente anzuhängen.

```
public List<String> toList() {
```

Lösung:

a)
```
public static Kreisliste singleton(String value) {
  Kreisliste result = new Kreisliste(null, value);
  result.next = result;
  return result;
}
```

b)
```
public Kreisliste getPrevious() {
  Kreisliste current = this;
  while (current.next != this) {
    current = current.next;
  }
  return current;
}
```

c)
```
public void merge(Kreisliste other) {
  Kreisliste thisPrevious = this.getPrevious();
  Kreisliste otherPrevious = other.getPrevious();
  thisPrevious.next = other;
  otherPrevious.next = this;
}
```

d)
```
public int indexOf(String value) throws UnknownValueException {
  return doIndexOf(value, this);
}

private int doIndexOf(String value, Kreisliste first)
                                    throws UnknownValueException {
  if (this.value.equals(value)) {
    return 0;
  } else if (this.next != first) {
    return this.next.doIndexOf(value, first) + 1;
  } else {
    throw new UnknownValueException();
  }
}
```

Alternative Lösung mit Tail-Recursion

```
public int indexOfWithTailRecursion(String value)
                                    throws UnknownValueException {
  return doIndexOfWithTailRecursion(value, this, 0);
}

private int doIndexOfWithTailRecursion(String value, Kreisliste first,
               int currentIndex) throws UnknownValueException {
```

```
    if (this.value.equals(value)) {
      return currentIndex;
    } else if (this.next != first) {
      return this.next.doIndexOfWithTailRecursion(value, first,
                                                  currentIndex + 1);
    } else {
      throw new UnknownValueException();
    }
  }
```

e)
```
public List<String> toList() {
    List<String> result = new LinkedList<>();
    Kreisliste current = this;
    do {
      result.add(current.value);
      current = current.next;
    } while (current != this);
    return result;
  }
```

Aufgabe 5 (Haskell): (4 + 3 + 2 + 2 + 6 + 2 = 19 Punkte)

a) Geben Sie zu den folgenden Haskell–Funktionen `f` und `g` jeweils den allgemeinsten Typ an. Gehen Sie hierbei davon aus, dass alle Zahlen den Typ `Int` haben.

```
f x (y:ys) = (f x [elem y ys]) ++ (f x (x [1]))
```

Hinweise:

- Die vordefinierte Funktion `elem:: a -> [a] -> Bool` gibt zurück, ob das übergebene Element Teil der übergebenen Liste ist. Beispielsweise ist `elem 7 [6,7,8] = True`.

```
g (x,y) z | x = g z z
          | otherwise = g (y > 0, y) z
```

b) Bestimmen Sie, zu welchem Ergebnis die Ausdrücke `i` und `j` jeweils auswerten.

```
i :: Int
i = (\x y z -> y (x z)) (\u -> 2 * u) (\v -> v + 7) 5

j :: [Bool]
j = map (\x -> not x) (filter (\x -> not x) [False,True,False,True,False])
```

c) Schreiben Sie eine Haskell–Funktion `reverselist :: [a] -> [a]`, die als Eingabe eine endliche Haskell–Liste erhält. Die von `reverselist` zurückgegebene Liste soll alle Elemente der Eingabeliste in umgekehrter Reihenfolge enthalten. Der Ausdruck `reverselist [1,2,3]` soll also zu `[3,2,1]` evaluieren.

Für diese und alle folgenden Teilaufgaben in dieser Aufgabe gelten die folgenden Hinweise:

Hinweise:

- Sie dürfen nur dann vordefinierte Funktionen verwenden, wenn diese explizit in den Hinweisen zur Aufgabe erwähnt sind.
- In dieser Aufgabe dürfen Sie die vordefinierte Funktion `++` verwenden.
- Sie dürfen Hilfsfunktionen schreiben.

d) Wir verwenden die folgende Datenstruktur `Cyclist`, um zyklische, einfach verkettete Listen mit Elementen eines bestimmten Typs `a` zu repräsentieren.

```
data Cyclist a = Element a (Cyclist a)
```

Beispielsweise kann die Cyclist `c1` aus Abb. 1 durch folgenden Haskell–Code dargestellt werden:

```
c1 :: Cyclist Int
c1 = let lastElement = Element 3 lastElement in Element 1 (Element 2 lastElement)
```

Dabei bedeutet der auf den linken Knoten zeigende Pfeil in den Abbildungen, dass die Liste mit diesem Element beginnt. Die restlichen Pfeile geben jeweils den Nachfolger des Listenelements an.

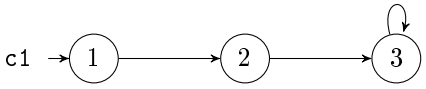


Abbildung 1: Cyclist `c1` des Typs `Cyclist Int`

Gehen Sie im Folgenden stets davon aus, dass in einer Liste nur *unterschiedliche* Werte des Typs `a` auftreten. Wir betrachten in dieser Aufgabe nur solche Cyclists, bei denen ein Element vorkommt, das sich selbst zum Nachfolger hat (wie der Knoten 3 in Abb. 1). Dies ist anders als bei den Kreislisten aus Aufgabe 4, wo der Zyklus stets die ganze Liste umfasst. Nur einelementige Kreislisten und Cyclists sehen gleich aus. Beachten Sie, dass sich mit dem Datentyp `Cyclist a` keine leere Liste darstellen lässt.

Schreiben Sie eine Haskell–Funktion `singleton`, die aus einem Wert `n` vom Typ `a` eine Cyclist `c` macht, sodass `c` nur aus dem Element `n` besteht, das sich selbst zum Nachfolger hat. Geben Sie auch die Typdeklaration der Funktion an.

e) Schreiben Sie eine Haskell–Funktion `append :: Cyclist Int -> [Int] -> Cyclist Int`. Die zurückgegebene Cyclist soll alle Elemente der Eingabe-Cyclist enthalten und anschließend alle Elemente der Eingabe–Haskell–Liste. Dabei soll die Reihenfolge der Elemente unverändert bestehen bleiben. Der Ausdruck `append c1 [4,5]` wertet also zu der Cyclist `c2` aus, die in Abb. 2 dargestellt ist.

Hinweise:

- Sie dürfen die Funktion `singleton` aus der vorherigen Teilaufgabe verwenden, unabhängig davon, ob Sie diese implementiert haben.

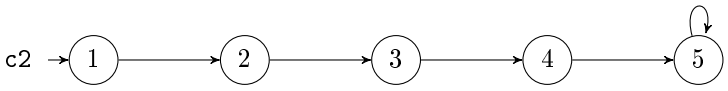


Abbildung 2: Cyclist `c2` des Typs `Cyclist Int`

f) Schreiben Sie nun eine Haskell–Funktion `toReversedCyclist :: [Int] -> Cyclist Int`, die aus einer Haskell–Liste `[x1,x2,...,xn]` die Cyclist `c` in Abb. 3 erzeugt. Die zurückgegebene Cyclist soll also alle Elemente der Eingabeliste in der umgekehrten Reihenfolge enthalten. Das letzte Element der Cyclist soll sein eigener Nachfolger sein. Auf einer leeren Eingabeliste darf sich die Funktion `toCyclist` beliebig verhalten.

Hinweise:

- Die Funktion `last :: [a] -> a` gibt das letzte Element der übergebenen Liste zurück. Die Funktion `init :: [a] -> [a]` gibt die Eingabeliste ohne das letzte Element zurück. Zum Beispiel wertet `last [1,2,3]` zu 3 und `init [1,2,3]` zu `[1,2]` aus.
- Sie dürfen die Funktionen aus den vorherigen Teilaufgaben verwenden, unabhängig davon, ob Sie diese implementiert haben.

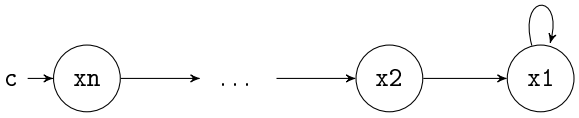


Abbildung 3: Allgemeine Form der Ergebnis-Cyclist der Funktion `toReversedCyclist`

Lösung:

a)
```
f :: ([Int] -> [Bool]) -> [Bool] -> [e]
g :: (Bool,Int) -> (Bool,Int) -> d
```

b)
```
i = 17
j = [True,True,True]
```

c)
```
reverselist :: [a] -> [a]
reverselist [] = []
reverselist (x:xs) = reverselist xs ++ [x]
```

d)
```
singleton :: a -> Cyclist a
singleton n = let x = Element n x in x
```

e)
```
append :: Cyclist Int -> [Int] -> Cyclist Int
append c [] = c
append (Element a (Element b s)) (x:xs) | a == b = Element a (append (singleton x) xs)
                                        | otherwise = Element a (append (Element b s) (x:xs))
```

f)
```
toReversedCyclist :: [Int] -> Cyclist Int
toReversedCyclist xs = append (singleton (last xs)) (reverselist (init xs))
```

Alternative Lösung

```
toReversedCyclist :: [Int] -> Cyclist Int
toReversedCyclist (x:xs) = rec xs (singleton x)
  where rec [] tail = tail
        rec (x:xs) tail = rec xs (Element x tail)
```

Aufgabe 6 (Prolog): (2 + 10 + (2 + 3 + 3) = 20 Punkte)

a) Geben Sie zu den folgenden Termpaaren jeweils einen allgemeinsten Unifikator an oder begründen Sie, warum sie nicht unifizierbar sind. Hierbei werden Variablen durch Großbuchstaben dargestellt und Funktionssymbole durch Kleinbuchstaben.

i) $f(g(X), X, Y), f(Y, Z, g(a))$

ii) $f(g(g(X)), X, Y), f(Y, Z, g(Z))$

b) Gegeben sei folgendes Prolog-Programm P.

```
a(s(X),s(Y),s(Z)) :- a(s(Z),s(s(Y)),s(X)).
a(s(X),s(Y),s(Z)) :- a(s(Z),Y,X).
a(X,0,s(0)).
```

- Erstellen Sie für das Programm P den Beweisbaum zur Anfrage "`?- a(s(s(0)),R,s(0)).`" bis zur Höhe 3 (die Wurzel hat dabei die Höhe 1).
- Markieren Sie die Knoten in Höhe 3, die zu einer unendlichen Auswertung führen können, mit ∞.
- Geben Sie alle Antwortsubstitutionen zur Anfrage "`?- a(s(s(0)),R,s(0)).`" an, die im Beweisbaum bis zur Höhe 3 enthalten sind.
- Geben Sie außerdem zu jeder dieser Antwortsubstitutionen an, ob sie von Prolog gefunden wird.
- Knoten, von denen keine weitere Ausführung aus möglich ist, sollen mit *fail* markiert werden.
- Wie muss das Programm durch Verschiebung von Klauseln abgeändert werden, damit Prolog alle Antwortsubstitutionen bis zur Höhe 3 findet?

c) In dieser Aufgabe betrachten wir binäre Bäume von natürlichen Zahlen in Prolog. Wir stellen binäre Bäume durch die Funktionssymbole `nil` und `node` dar. Hierbei repräsentiert `nil` den leeren Baum und `node(L,V,R)` steht für den Baum, in dem die natürliche Zahl `V` in der Wurzel gespeichert ist und `L` der linke bzw. `R` der rechte Teilbaum ist. Die folgenden Abbildungen zeigen binäre Bäume. Hierbei bedeutet eine Linie, die von einem Knoten ausgeht, aber nicht in einem Knoten endet, dass dieses Kind des Knotens der leere Baum ist. Sind beide Kinder eines Knotens leer, so zeichnen wir keine Linien ein.

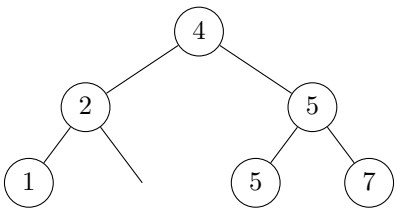


Abbildung 4: Binärer Suchbaum

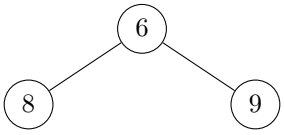


Abbildung 5: Binärbaum, der kein Suchbaum ist

Der Binärbaum in Abb. 4 wird somit durch den Ausdruck `node(node(node(nil,1,nil),2,nil),4,node(node(nil,5,nil),5,node(nil,7,nil)))` dargestellt, wohingegen der Binärbaum in Abb. 5 durch den Ausdruck `node(node(nil,8,nil),6,node(nil,9,nil))` repräsentiert wird.

Ein binärer *Suchbaum* ist ein spezieller Binärbaum. Der leere Baum `nil` ist ein binärer Suchbaum. Ein Baum `node(L,V,R)` ist ein binärer Suchbaum, wenn alle Werte, die in `L` gespeichert sind, kleiner oder gleich `V` sind und alle Werte, die in `R` gespeichert sind, echt größer als `V` sind. Der binäre Baum in Abb. 4 ist somit ein binärer Suchbaum. Der binäre Baum in Abb. 5 ist kein binärer Suchbaum, da `8 > 6` gilt.

Hinweise:

- Sie dürfen in allen Teilaufgaben Prädikate aus den vorherigen Teilaufgaben verwenden, auch wenn Sie diese Prädikate nicht implementiert haben.
- Sie dürfen beliebig viele Hilfsprädikate definieren.
- Sie dürfen beliebige vordefinierte Prädikate für Arithmetik in Prolog benutzen, wie z.B. das zweistellige Prädikat `=<` für *kleiner gleich*. So wertet `3 =< 4` zu `true` aus, `6 =< 5` zu `false`.

i) Implementieren Sie ein Prädikat `maxT` mit Stelligkeit 2 in Prolog. Wenn `X` ein binärer Suchbaum ist, dann soll `maxT(X,M)` wahr sein, wenn `M` der größte Wert ist, der in `X` gespeichert ist. Wenn `X` *kein* binärer Suchbaum ist oder *nicht nur* natürliche Zahlen enthält, darf sich das Prädikat beliebig verhalten.

Sie können das folgende Faktum als gegeben ansehen:

```
maxT(nil,0).
```

ii) Implementieren Sie ein Prädikat `isSearchTree` mit Stelligkeit 1 in Prolog. Wenn `X` ein binärer Baum ist, dann soll `isSearchTree(X)` genau dann wahr sein, wenn `X` ein binärer Suchbaum ist.

So soll also der Aufruf von `isSearchTree` auf dem binären Baum aus Abb. 4 zu `true` auswerten. Rufen wir `isSearchTree` hingegen auf dem binären Baum aus Abb. 5 auf, so soll das Ergebnis `false` sein.

Gehen Sie davon aus, dass es analog zum Prädikat `maxT/2` auch ein Prädikat `minT/2` gibt, welches zur Bestimmung des Minimums in einem binären Suchbaum benutzt werden kann.

Sie können das folgende Faktum als gegeben ansehen:

```
isSearchTree(nil).
```

iii) Implementieren Sie ein Prädikat `containsN` mit Stelligkeit 3 in Prolog. Wenn `X` einen binären Suchbaum repräsentiert und `V` bzw. `W` natürliche Zahlen sind, dann soll der Aufruf `containsN(X,V,W)` genau dann wahr sein, wenn die Zahl `V` in `X` genau `W`-mal vorkommt.

Wenn `X1` also den binären Suchbaum in Abb. 4 repräsentiert, dann soll der Aufruf `containsN(X1,5,W)` also genau dann wahr sein, wenn `W = 2` ist und `containsN(X1,11,W)` soll genau dann wahr sein, wenn `W = 0` gilt.

Sie können das folgende Faktum als gegeben ansehen:

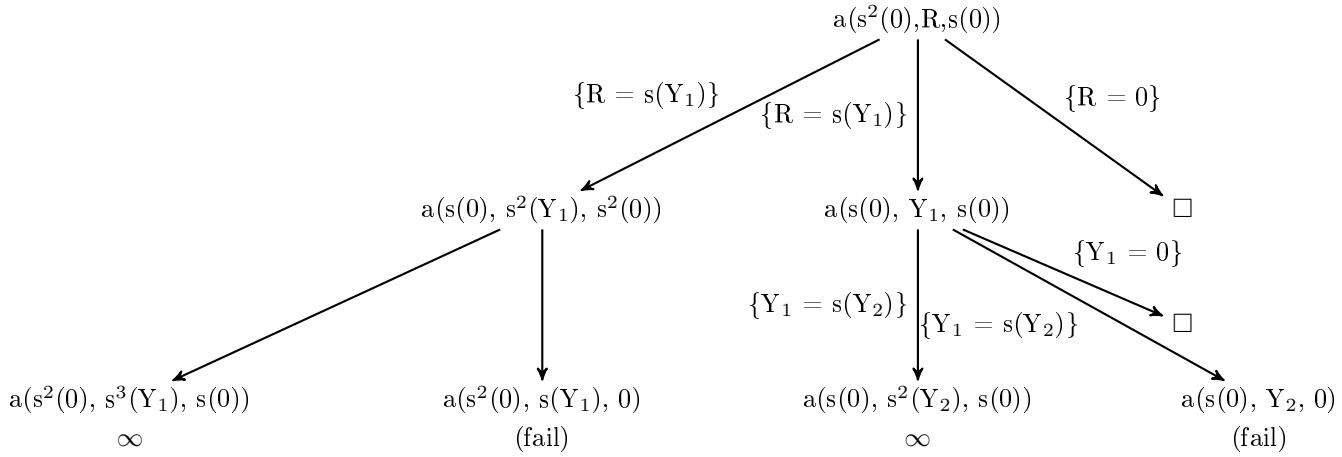
```
containsN(nil,_,0).
```

Lösung:

a) i) $f(g(X), X, Y), f(Y, Z, g(a))$ hat als MGU $\sigma = \{X = a, Y = g(a), Z = a\}$

ii) $f(g(g(X)), X, Y), f(Y, Z, g(Z))$ kann nicht unifiziert werden, da nach der Unifikation der ersten beiden Argumente das dritte Argument $g(X)$ bzw. $g(g(X))$ ist. Es liegt ein Occur Failure vor.

b)



Die Antwortsubstitutionen innerhalb des Beweisbaums sind $\{R = 0\}$ und $\{R = s(0)\}$. Diese werden von Prolog nicht gefunden.

Die dritte Regel muss mit der ersten vertauscht werden, damit Prolog die Antwortsubstitutionen bis zur Höhe 3 findet:

```
a(X,0,s(0)).
a(s(X),s(Y),s(Z)) :- a(s(Z),Y,X).
a(s(X),s(Y),s(Z)) :- a(s(Z),s(s(Y)),s(X)).
```

c)
```
% i)
maxT(nil,0).
maxT(node(_,V,nil),V).
maxT(node(_,_,node(L,V,R)),W) :- maxT(node(L,V,R),W).

%Alternative Loesung mit Arithmetik

maxT(nil,0).
maxT(node(_,V,R),V) :- maxT(R,W), W =< V.
maxT(node(_,V,R),W) :- maxT(R,W), W > V.

% ii)
isSearchTree(nil).
isSearchTree(node(L,V,R)) :- isSearchTree(L), isSearchTree(R), maxT(L,X),
                             minT(R,Y), X =< V, Y > V.

% iii)
containsN(nil,_,0).
containsN(node(L,V,_),V,Z) :- containsN(L,V,ZZ), Z is ZZ + 1.
containsN(node(L,V,_),W,Z) :- W < V, containsN(L,W,Z).
containsN(node(_,V,R),W,Z) :- W > V, containsN(R,W,Z).
```