Aufgabe 1 (Programmanalyse): (12 + 6 = 18 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.

```
public class A {

    int x = 1;

    static double y = 3.0;

    public A() {
        x = (int)y;
    }

    public A(int x) {
        this.x = this.getX() + x;
    }

    int getX() {
        return this.x;
    }

    public void f(float z) {
        y *= z;
    }
}
```

```
public class B extends A {

    int x = 7;

    public B(int x) {
      super(x);
      this.x = x;
    }

    public B() {
      this.y += 1.0;
    }

    int getX() {
        return this.x;
    }

    public void f(long z) {
        y += z;
    }

    public void f(double z) {
      y -= z;
    }

}
```

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

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

        System.out.println(((A)b1).x);              // OUT: [    ]

        A ab = new B(5);
        System.out.println(ab.x + " " + A.y);       // OUT: [    ] [    ]

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

        A.y = 5.0;
        b1.f(2.0f);
        System.out.println(A.y);                    // OUT: [    ]

        ab.f(5);
        System.out.println(A.y);                    // 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 A {
2      int z;
3
4      public C(int x) {
5          this.z = x;
6          super(x);
7      }
8
9      int getX() {
10       return z;
11     }
12
13     public double f(float z) {
14       y *= z;
15       return y;
16     }
17
18     public void setX(double z) {
19       x = z;
20     }
21 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x + " " + A.y);       // OUT: [ 3  ] [ 3.0]

        B b1 = new B();
        System.out.println(b1.x + " " + A.y);       // OUT: [ 7  ] [ 4.0]

        System.out.println(((A)b1).x);              // OUT: [ 3  ]

        A ab = new B(5);
        System.out.println(ab.x + " " + A.y);       // OUT: [ 5  ] [ 4.0]

        System.out.println(((A)ab).x);              // OUT: [ 5  ]

        A.y = 5.0;
        b1.f(2.0f);
        System.out.println(A.y);                    // OUT: [10.0]

        ab.f(5);
        System.out.println(A.y);                    // OUT: [50.0]
    }
}
```

b)
- Der Aufruf von `super` muss die erste Anweisung im Konstruktor sein.
- Der Rückgabetyp einer Methode darf sich beim Überschreiben nicht ändern. Daher ist die Methode mit der Signatur `double f (float z)` unzulässig.

- Implizite Typumwandlung von `double` nach `int` ist nicht möglich. Da die rechte Seite von `x = z` der Parameter vom Typ `double` ist (das Attribut der Klasse wird verdeckt), führt diese Anweisung zu einem Fehler.

Aufgabe 2 (Hoare-Kalkül): (14 + 1 = 15 Punkte)

Gegeben sei folgendes Java-Programm P, das zu zwei ganzzahligen Eingaben `x` und `y` den Wert `y`$-$`x` berechnet.

$\langle\, true \,\rangle$ (Vorbedingung)

```
z = x;
res = 0;
while (z != y) {
    if (z < y) {
        z = z + 1;
        res = res + 1;
    } else {
        z = z - 1;
        res = res - 1;
    }
}
```

$\langle\, \texttt{res} = \texttt{y} - \texttt{x} \,\rangle$ (Nachbedingung)

Hinweise:

- Sie dürfen in beiden Teilaufgaben 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. Klammern dürfen und müssen Sie jedoch eventuell bei der Anwendung der Zuweisungsregel setzen.

a) Vervollständigen Sie die Verifikation des Algorithmus P auf der folgenden Seite 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.

b) Geben Sie eine gültige Variante an, mit deren Hilfe die Terminierung des Algorithmus P bewiesen werden kann. Sie brauchen den eigentlichen Terminierungsbeweis nicht durchzuführen – die Angabe einer geeigneten Variante für die `while`-Schleife genügt.

Lösung:

a)

$\langle true \rangle$
$\langle \texttt{x} = \texttt{x} \wedge 0 = 0 \rangle$

```
z = x;
```

$\langle \texttt{z} = \texttt{x} \wedge 0 = 0 \rangle$

```
res = 0;
```

$\langle \texttt{z} = \texttt{x} \wedge \texttt{res} = 0 \rangle$
$\langle \texttt{res} = \texttt{z} - \texttt{x} \rangle$

```
while (z != y) {
```

$\langle \texttt{res} = \texttt{z} - \texttt{x} \wedge \texttt{z} \neq \texttt{y} \rangle$

```
    if (z < y) {
```

$\langle \texttt{res} = \texttt{z} - \texttt{x} \wedge \texttt{z} \neq \texttt{y} \wedge \texttt{z} < \texttt{y} \rangle$
$\langle \texttt{res} + 1 = \texttt{z} + 1 - \texttt{x} \rangle$

```
        z = z + 1;
```

$\langle \texttt{res} + 1 = \texttt{z} - \texttt{x} \rangle$

```
        res = res + 1;
                        ⟨res = z − x⟩
    } else {
                        ⟨res = z − x ∧ z ≠ y ∧ ¬(z < y)⟩
                        ⟨res − 1 = z − 1 − x⟩
        z = z - 1;
                        ⟨res − 1 = z − x⟩
        res = res - 1;
                        ⟨res = z − x⟩
    }
                        ⟨res = z − x⟩
}
                        ⟨res = z − x ∧ ¬(z ≠ y)⟩
                        ⟨res = y − x⟩
```

b) Eine gültige Variante für die Terminierung ist $V = |\mathtt{z} - \mathtt{y}|$.

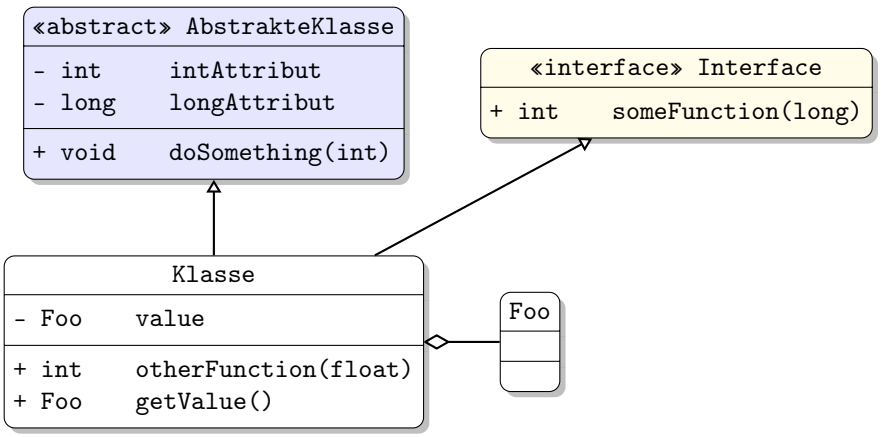
Aufgabe 3 (Klassenhierarchie): (9 + 8 = 17 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Klassenhierarchie zur Verwaltung von Kalendern.

- Ein Termin hat einen Startpunkt. Dieser wird als Zeitstempel in Form einer ganzen Zahl mit 64 bit gespeichert. Außerdem hat ein Termin noch eine Text-Bezeichnung.
- Alle Termine haben eine Methode ohne Eingabeparameter, um festzustellen, ob der Termin in der Vergangenheit liegt.
- Serientermine sind besondere Arten von Terminen. Der Startpunkt bezieht sich hierbei auf den ersten Termin der Serie. Sie haben eine Wiederholungsrate, die angibt, nach wie viel Tagen sich der Termin wiederholt.
- Ein Kalender hat eine Liste von Terminen.
- Erinnerungen können entweder ein Ton oder eine E-Mail sein. Weitere Arten von Erinnerungen gibt es nicht. Alle Erinnerungen haben eine Anzahl Sekunden, die angibt, wie lange vor einem Termin erinnert werden soll. Außerdem haben alle Erinnerungen eine Methode, um sie auszuführen.
- Ton-Erinnerungen haben eine Methode *klingeln* und E-Mail-Erinnerungen haben eine Methode *senden*.
- Jeder Termin kann genau eine Erinnerung haben. (Falls ein Termin keine Erinnerung hat, kann dies zum Beispiel durch die Erinnerung `null` dargestellt werden.) Eine Erinnerung kann aber mehreren Terminen zugeordnet sein.
- Termine und Kalender können geteilt werden. Alle Objekte, die geteilt werden können, haben hierzu eine entsprechende `void`-Methode ohne Eingabeparameter.

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 `String` hinzu.

b) Schreiben Sie eine Java-Methode `public void einzeltermineTeilen()` in der Klasse `Kalender`. Diese Methode soll alle Termine dieses Kalenders teilen, die nicht in der Vergangenheit liegen und keine Serientermine sind. Wenn ein Termin eine Erinnerung hat, soll jedoch eine Kopie des Termins ohne Erinnerung geteilt werden, statt des originalen Termins.

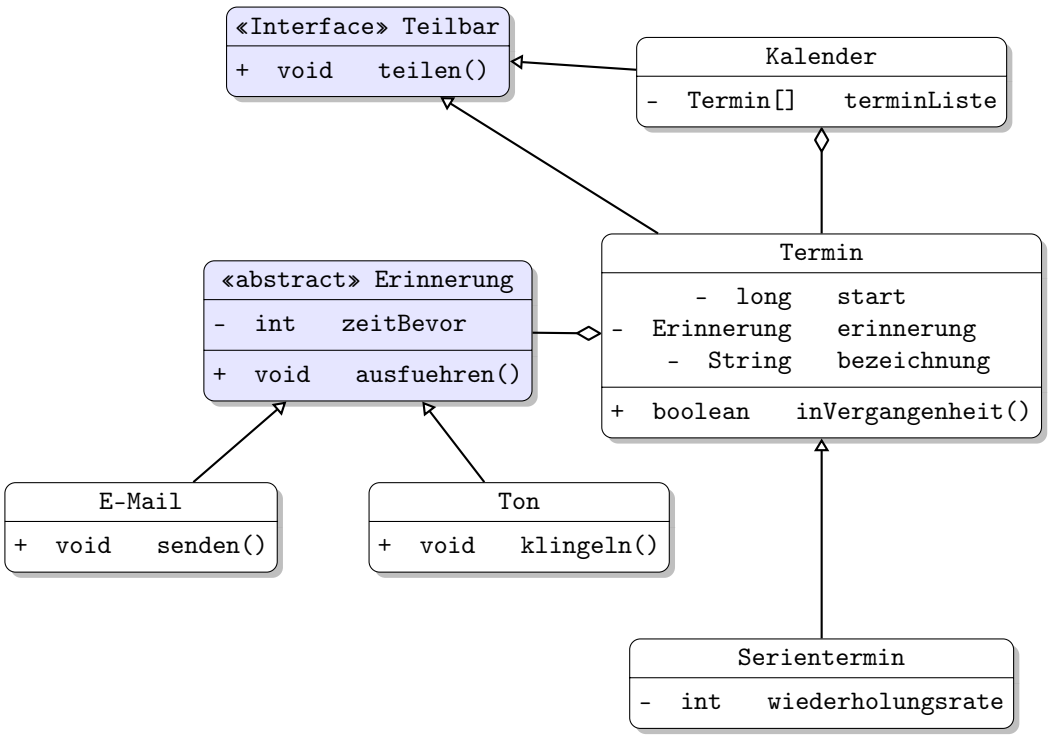
Gehen Sie davon aus, dass es zu jedem Attribut geeignete Selektoren gibt und alle Klassen einen Konstruktor besitzen, der für jedes Attribut einen Parameter hat. Eine Klasse `A` mit Attributen `int x` und `float y` hätte beispielsweise einen Konstruktor `A(int x, float y)`.

Hinweise:

- Beachten Sie, dass Termine und Listen von Terminen auch `null` sein können.

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)
```
public void einzeltermineTeilen() {
  if(this.terminListe != null) {
    for(Termin termin : this.terminListe) {
      if(termin == null || termin.inVergangenheit() || termin instanceof Serientermin) {
        continue;
      }

      if(termin.erinnerung == null) {
        termin.teilen();
      } else {
        Termin kopie = new Termin(
          termin.getStart(), null, termin.getBezeichnung());
        kopie.teilen();
      }
    }
```

```
        }
    }
```

Aufgabe 4 (Programmierung in Java): (4 + 5 + 2 + 9 + 4 + 6 = 30 Punkte)

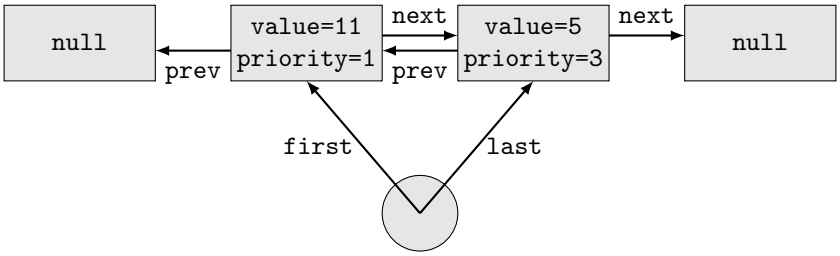
Die Klasse `PriorityList` dient zur Repräsentation von Prioritätslisten. Sie verfügt über ein Attribut `first`, welches auf das vorderste Element in der Prioritätsliste verweist und ein Attribut `last`, welches auf das letzte Element in der Prioritätsliste verweist. Jedes Element der Prioritätsliste wird als Objekt der Klasse `PriorityElement` dargestellt und enthält einen Wert vom Typ `int` (Attribut `value`), einen Verweis auf den direkten Nachfolger (Attribut `next`), einen Verweis auf den direkten Vorgänger (Attribut `prev`) und eine Priorität vom Typ `int` (Attribut `priority`).
Der Vorgänger des ersten PriorityElements einer Prioritätsliste ist immer `null`, ebenso der Nachfolger des letzten PriorityElements. Die leere Prioritätsliste wird repräsentiert durch ein `PriorityList`-Objekt, in dem `first = null` und `last = null` gilt.

```
public class PriorityList {
  private PriorityElement first;
  private PriorityElement last;
}

public class PriorityElement {
  private int value;
  private int priority;
  private PriorityElement next;
  private PriorityElement prev;

  public PriorityElement(int value, int priority,
                         PriorityElement next, PriorityElement prev) {
    this.value = value;
    this.priority = priority;
    this.next = next;
    this.prev = prev;
  }
}
```

In der folgenden Grafik ist dargestellt, wie eine Prioritätsliste mit den Werten `11` und `5` mit der Klasse `PriorityList` repräsentiert wird. Hierbei hat `11` die Priorität `1` und `5` die Priorität `3`.



Objekte der Klasse `PriorityList` sind als Kreise und Objekte der Klasse `PriorityElement` sind als Rechtecke dargestellt.
Sie dürfen in allen Teilaufgaben beliebige Hilfsmethoden zu den Klassen `PriorityList` und `PriorityElement` hinzufügen. Wenn Sie von dieser Möglichkeit Gebrauch machen, müssen Sie eindeutig kennzeichnen, welche Hilfsmethoden zu welchen Klassen gehören. Außerdem können Sie davon ausgehen, dass für alle Attribute jeder Klasse geeignete get– und set–Methoden bereitstehen.

a) Schreiben Sie in der Klasse `PriorityList` eine Methode `public int length()`. Der Rückgabewert ist die Länge der Prioritätsliste. Gehen Sie davon aus, dass die Liste keine Zyklen besitzt. Ein Aufruf der Methode auf der Beispielliste soll `2` als Rückgabewert liefern. Verwenden Sie in dieser Aufgabe **ausschließlich Rekursion**. Bei der Verwendung von Schleifen werden keine Punkte vergeben.

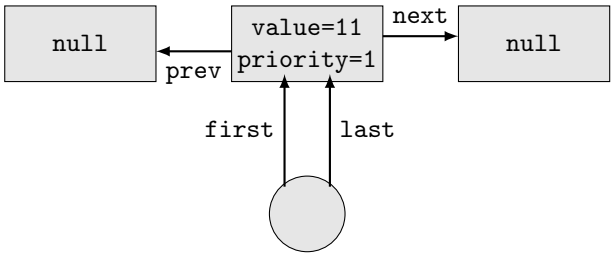
b) Implementieren Sie die Methode `public void enqueue(int val, int priority)` in der Klasse `PriorityList`. Diese Methode fügt ein Element mit dem Wert `val` und der Priorität `priority` am Ende der Prioritätsliste ein.

c) Schreiben Sie eine Exception-Klasse `EmptyQueueException` ohne Attribute. Die einzige Methode dieser Klasse ist eine `toString`-Methode, die zurückgibt, dass es sich um eine leere Prioritätsliste handelt.

d) Implementieren Sie die Methode `public int dequeue()` in der Klasse `PriorityList`. Diese Methode gibt den Wert `value` des Elementes mit der **höchsten Priorität** in der Liste zurück und löscht dieses Element anschließend aus der Prioritätsliste. Gehen Sie davon aus, dass jede Priorität **höchstens einmal** vorkommt. Ist die Prioritätsliste leer, so wird eine `EmptyQueueException geworfen`. Verwenden Sie in dieser Aufgabe **ausschließlich Schleifen**. Bei Verwendung von Rekursion werden keine Punkte vergeben.

Gehen Sie davon aus, dass es in der Klasse `PriorityList` eine private Methode `private void delete(PriorityElement elem)` gibt, die aus einer gegebenen Prioritätsliste das Element `elem` löscht. Benutzen Sie diese Methode zur Lösung der Aufgabe.

Rufen wir die Methode `dequeue()` auf der Prioritätsliste aus der Aufgabenstellung auf, so ist der Rückgabewert `5` und die Prioritätsliste sieht danach wie folgt aus:



e) Passen Sie die Klassen `PriorityList` und `PriorityElement` sowie ihre Attribute so an, dass beliebige Objekte gleichen Typs als Werte gespeichert werden können. Erweitern Sie die Klassen zu diesem Zweck um einen generischen Typparameter `T`, der den Typ der in der Prioritätsliste gespeicherten Werte angibt. **Es ist nicht notwendig, die Konstruktoren und sonstige Methoden in den Klassen `PriorityList` und `PriorityElement` anzupassen!**

f) Implementieren Sie eine statische Methode `public static <T> LinkedList<T> toLinkedList(PriorityList<T> inputList)`. Diese soll die in der Prioritätsliste `inputList` gespeicherten Werte vom Typ `T` so in eine `LinkedList` einfügen, dass die Elemente nach Priorität geordnet sind, d.h. der Wert höchster Priorität steht am Beginn der Liste, der Wert niedrigster Priorität am Ende. Die Prioritätsliste ist nach dem Aufruf dieser Methode leer. Gehen Sie davon aus, dass jede Priorität **höchstens einmal** vorkommt.

Benutzen Sie hierzu die Methode `dequeue()` aus Aufgabenteil d), auch wenn Sie sie nicht implementiert haben.

Fangen Sie die unter Umständen auftretende `EmptyQueueException`.

Gehen Sie davon aus, dass es in der Klasse `LinkedList<T>` eine Methode `public boolean add(T value)` gibt, die ein Element am Ende der Liste einfügt und stets `true` zurückgibt.

Gehen Sie außerdem davon aus, dass das nötige Paket `java.util` zur Benutzung von `LinkedList<T>` importiert wurde.

Benutzen Sie **keine** Schleife, deren Bedingung immer `true` ist, d.h. in Ihrer Lösung darf **keine** Schleife wie `while(true){...}` vorkommen!

Hinweise:

- Auch wenn `dequeue()` nur für nicht–leere Prioritätslisten aufgerufen wird, so kann der Compiler trotzdem **nicht** feststellen, dass die Methode `dequeue` tatsächlich nie eine Exception wirft. Die `EmptyQueueException` muss daher dennoch in der Methode `toLinkedList` gefangen werden.

Lösung:

a) In der Klasse `PriorityList`

```
public int length() {
   return length(this.getFirst());
 }

 private static int length(PriorityElement inputEl){
   if(inputEl == null){
     return 0;
   }
   return 1 + length(inputEl.getNext());
 }
```

b)

```
public void enqueue(int value, int priority) {
    PriorityElement newElem = new PriorityElement(value, priority, null, null);

   if(this.getFirst() == null){
     this.setFirst(newElem);
   }else{
     PriorityElement last = this.getLast();
     last.setNext(newElem);
     newElem.setPrev(last);
   }
   this.setLast(newElem);
 }
```

c)

```
public class EmptyQueueException extends Exception {
   public String toString(){
     return "Queue is empty";
   }
 }
```

d)

```
public int dequeue() throws EmptyQueueException{
   if(this.getFirst() == null){
     throw new EmptyQueueException();
   }

   PriorityElement el = this.getFirst();
   PriorityElement maxElement = el;
   int maxValue = el.getValue();
   int maxPriority = el.getPriority();

   while(el.getNext() != null){
     el = el.getNext();
     if(el.getPriority() > maxPriority){
       maxElement = el;
       maxValue = el.getValue();
       maxPriority = el.getPriority();
```

```
      }
    }

    this.delete(maxElement);

    return maxValue;
  }
```

e)
```
  public class PriorityElement<T> {
    private T value;
    private int priority;
    private PriorityElement<T> next;
    private PriorityElement<T> prev;
  }

  public class PriorityList<T> {
    private PriorityElement<T> first;
    private PriorityElement<T> last;
  }
```

f)
```
  public static <T> LinkedList<T> toLinkedList(PriorityList<T> inputList) {
    LinkedList<T> resultList = new LinkedList<T>();
    while(inputList.getFirst()!=null){
      try{
        resultList.add(inputList.dequeue());
      }
      //this case never occurs but it is necessary for the compiler
      catch(EmptyQueueException e){
        System.out.println(e);
      }
    }
    return resultList;
  }
```

Aufgabe 5 (Haskell): (4 + 4 + 2 + 5 + 5 = 20 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 0 _ _ = []
f n False xs = if n > 0 then (n:xs) else []

g x y z = if (x == z) then g y x z else []
```

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

```
i :: Int
i = (\f y z -> length (f y z)) (\xs ys -> xs ++ ys) [1,2,3] [4,5,6]
```

Hinweise:

- Die vordefinierte Funktion `length::[a] -> Int` gibt die Anzahl der Elemente einer Liste zurück. So ist `length [] = 0` und `length [7,11,14,9] = 4`.

```
j :: [[Int]]
j = map (filter odd) [[0],[2,4,6],[12,14]]
```

Hinweise:

- Für die vordefinierte Funktion `odd::Int -> Bool` ist `odd x = True` genau dann, wenn `x` eine ungerade ganze Zahl ist.

c) Wir verwenden die folgende Datenstruktur `Tree a` zur Repräsentation binärer Bäume in Haskell.

```
data Tree a = Node (Tree a) a (Tree a) | Leaf a
```

Werte des Typs `a` können sowohl in den Blättern eines Baumes als auch in inneren Knoten gespeichert werden. Ein Beispiel für einen solchen Baum ist in der folgenden Abbildung gegeben. Dabei repräsentieren Quadrate Blätter und Kreise stellen innere Knoten dar.

In einem Binärbaum ist ein Knoten also entweder ein Blatt, das einen Wert speichert, oder ein innerer Knoten, der einen Wert speichert und genau zwei Kinder hat.

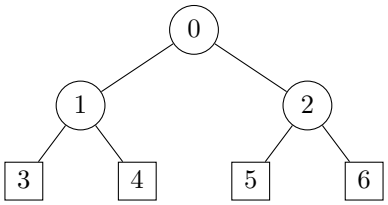


Abbildung 1: Vollständiger Binärbaum des Typs `Tree Int`

Schreiben Sie eine Haskell–Funktion `innerNodes::Tree a -> Int`, die die Anzahl der inneren Knoten eines Baumes zurückgibt. Wird `innerNodes` auf dem Binärbaum in Abbildung 1 aufgerufen, so ist das Ergebnis `3`.

d) Schreiben Sie eine Haskell–Funktion `rootLeaf::Tree a -> [[a]]`, die für einen gegebenen Baum eine Liste aller Pfade von der Wurzel zu einem Blatt zurückgibt. Ein Pfad wird hier als Liste des Typs `[a]` dargestellt, wobei das erste Element der Liste der Wert des Wurzelknotens ist und danach die Werte der auf dem Pfad liegenden inneren Knoten folgen. Der letzte Eintrag der Liste ist der Wert des Blatts, zu dem der Pfad führt.

Wird `rootLeaf` auf den Baum aus Abbildung 1 angewendet, so ergibt sich `[[0,1,3],[0,1,4],[0,2,5],[0,2,6]]`.

Hinweise:

- Sie dürfen hierbei beliebige vordefinierte Funktionen verwenden, wie z.B. `(++)` und `map`. Beispielsweise ergibt `map (0:)[[1,3],[1,4]]` das Resultat `[[0,1,3],[0,1,4]]`.

e) Schreiben Sie eine Haskell–Funktion `isComplete::Tree a -> Bool`, die genau dann `True` zurückgibt, wenn der eingegebene Binärbaum **vollständig** ist. Hierbei heißt ein Binärbaum vollständig, wenn alle Pfade von der Wurzel zu einem Blatt dieselbe Länge haben, d.h. wenn die Anzahl der vorkommenden Knoten auf jedem dieser Pfade gleich ist.

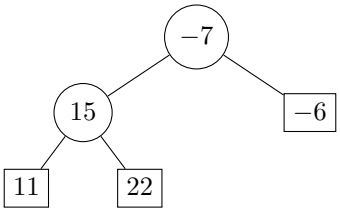


Abbildung 2: Unvollständiger Binärbaum des Typs `Tree Int`

So ist zum Beispiel der Binärbaum aus Abbildung 1 vollständig, da alle Pfade die Länge 3 haben. Der Baum aus Abbildung 2 hingegen ist nicht vollständig, da es sowohl einen Pfad der Länge zwei, `[-7,-6]`, als auch einen Pfad der Länge drei, `[-7,15,22]`, von der Wurzel zu einem Blatt gibt.

Hinweise:

- Sie können die Funktion `rootLeaf` aus der vorherigen Teilaufgabe benutzen, auch wenn Sie diese Methode nicht implementiert haben.
- Sie dürfen wieder beliebige vordefinierte Funktionen verwenden, wie den vordefinierten Ungleich–Operator `(/=)::Int -> Int -> Bool`, oder die Funktion `length::[a] -> Int`, die die Anzahl der Elemente einer Liste berechnet. Außerdem dürfen Sie Funktionen höherer Ordnung wie `map` und `filter` benutzen.

Lösung:

a) i)
```
f :: Int -> Bool -> [Int] -> [Int]
```
ii)
```
g :: a -> a -> a -> [b]
```

b)
```
i = 6
j = [[],[],[]]
```

c)
```
innerNodes :: (Tree a) -> Int
innerNodes (Leaf _) = 0
innerNodes (Node t1 _ t2) = 1 + (innerNodes t1) + (innerNodes t2)
```

d)
```
rootLeaf :: (Tree a) -> [[a]]
rootLeaf (Leaf x) = [[x]]
rootLeaf (Node t1 x t2) = map (x:) ((rootLeaf t1) ++ (rootLeaf t2))
```

e)
```
isComplete t = (filter (/= x) xs) == []
  where (x:xs) = map length (rootLeaf t)
```

Aufgabe 6 (Prolog): (2 + 6 + 4 + 5 + 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) $f(X, s(Y), Z), f(s(Z), Z, Z)$

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

b) Gegeben sei folgendes Prolog-Programm P.

```
p(0,s(Y)) :- p(s(0),Y).
p(s(X),s(Y)) :- p(X,s(Y)).
p(s(0),s(0)).
```

Erstellen Sie für das Programm P den Beweisbaum zur Anfrage "`?- p(s(0),R).`" bis zur Höhe 4 (die Wurzel hat dabei die Höhe 1). Markieren Sie Pfade, die zu einer unendlichen Auswertung führen, mit ∞ und geben Sie alle Antwortsubstitutionen zur Anfrage "`?- p(s(0),R).`" an, die im Beweisbaum bis zur Höhe 4 enthalten sind. Geben Sie außerdem zu jeder dieser Antwortsubstitutionen an, ob sie von Prolog gefunden wird. Welche der beiden Regeln muss ans Ende des Programms verschoben werden, damit Prolog bei obiger Anfrage alle Antwortsubstitutionen bis zur Höhe 4 findet?

c) Natürliche Zahlen lassen sich mit Hilfe der Funktionssymbole `0` und `s` in sogenannter *Peano-Notation* darstellen (d. h., der Term `s(0)` stellt die Zahl 1 dar, `s(s(0))` stellt 2 dar, etc.). Implementieren Sie ein Prädikat `filterNat` mit Stelligkeit 2 in Prolog, sodass beim Aufruf von `filterNat(`t_1`,Res)` für eine (möglicherweise leere) Liste t_1 ohne Variablen alle Elemente aus der Liste entfernt werden, die nicht einer natürlichen Zahl in Peano-Notation entsprechen. Beispielsweise soll `filterNat([0,q(0),s(s(0)),2018], Res)` als erste Antwort `Res = [0,s(s(0))]` liefern.

d) Wie in Aufgabe 5 betrachten wir wieder binäre Bäume, wobei Werte sowohl in Blättern als auch in inneren Knoten gespeichert werden können. Solche binären Bäume lassen sich in Prolog mit Hilfe der Funktionssymbole `leaf` und `node` als Terme darstellen. Dabei repräsentiert `leaf(`t`)` ein Blatt mit dem Wert t und `node(`l`,`t`,`r`)` repräsentiert einen Baum mit dem Wert t im Wurzelknoten, der den Teilbaum l als linkes Kind und den Teilbaum r als rechtes Kind hat. Implementieren Sie ein Prädikat `dec` mit Stelligkeit 2 in Prolog, wobei `dec(`t_1`,`t_2`)` für einen binären Baum t_1 genau dann gilt, wenn alle Werte in t_1 größer als 0 sind und t_2 der Baum ist, der sich aus t_1 ergibt, indem jeder Wert, der in einem Knoten oder in einem Blatt steht, um eins verringert wird. Nehmen Sie hierbei an, dass die Werte vordefinierte Prolog-Zahlen sind. Beispielsweise soll `dec(node(leaf(6),4,node(leaf(7),3,leaf(5))), Res)` die einzige Antwort `Res = node(leaf(5),3,node(leaf(6),2,leaf(4)))` liefern.

e) Implementieren Sie ein Prädikat `trees` mit Stelligkeit 2 in Prolog. Für einen gegebenen mit Hilfe der Funktionssymbole `leaf` und `node` dargestellten Binärbaum t_1 soll `trees(`t_1`,`t_2`)` für all diejenigen Bäume t_2 gelten, die sich aus t_1 ergeben, wenn man alle in t_1 enthaltenen Werte um eine Zahl n mit $1 \leq n \leq m$ verringert, wobei m der kleinste in t_1 auftretende Wert ist. Beispielsweise soll die Anfrage `trees(node(leaf(6),4,node(leaf(7),3,leaf(5))), T)` folgende Antworten liefern.

```
T = node(leaf(5),3,node(leaf(6),2,leaf(4))) ;
T = node(leaf(4),2,node(leaf(5),1,leaf(3))) ;
T = node(leaf(3),1,node(leaf(4),0,leaf(2))) ;
false.
```

Die Anfrage `trees(node(leaf(0),4,node(leaf(7),3,leaf(5))), T)` liefert die einzige Antwort `false`.

Hinweise:

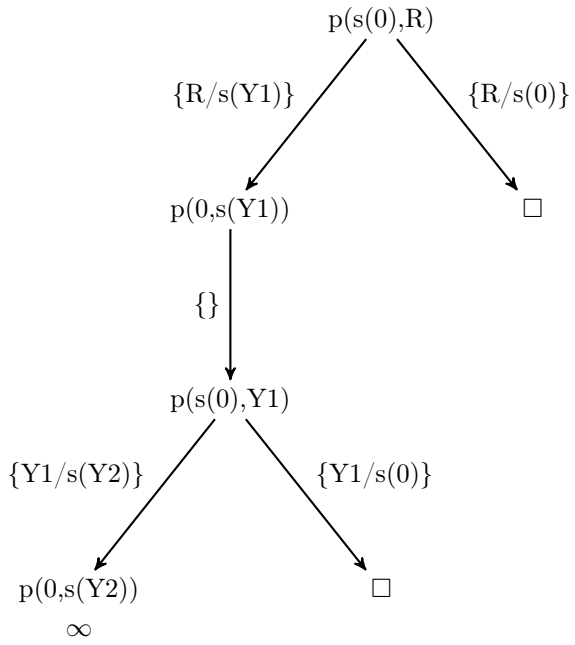
- Sie dürfen das Prädikat `dec` aus Teilaufgabe **d)** verwenden, auch wenn Sie dieses Prädikat nicht implementiert haben.

Lösung:

a) i) $f(X, s(Y), Z), f(s(Z), Z, Z)$: $X/s(s(Y)), Z/s(Y)$

ii) $g(X, Y, Z), g(s(Y), s(Z), s(X))$: occur failure X in $s(s(s(X)))$

b)



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

Die zweite Regel muss ans Ende des Programms verschoben werden, damit Prolog alle Antwortsubstitutionen bis zur Höhe 4 findet:

```
p(0,s(Y)) :- p(s(0),Y).
p(s(0),s(0)).
p(s(X),s(Y)) :- p(X,s(Y)).
```

c)
```
isNumber(0).
isNumber(s(X)) :- isNumber(X).

filterNat([],[]).
filterNat([X|XS], [X|Res]) :- isNumber(X), filterNat(XS, Res).
filterNat([_|XS], Res) :- filterNat(XS, Res).
```

d)
```
dec(leaf(X), leaf(DecX)) :- X > 0, DecX is X - 1.
dec(node(L,X,R), node(DecL,DecX,DecR)) :- dec(L,DecL), dec(R,DecR),
                                          X > 0, DecX is X - 1.
```

e)
```
trees(T, Res) :- dec(T, Res).
trees(T, Res) :- dec(T, Dec), trees(Dec, Res).
```