Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, N. Lommen, D. Meier, F. Meyer

Aufgabe 1 (Programmanalyse): (9 + 5 = 14 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 sealed class A permits B {
  public int n;
  public A v;

  public static A f(int n, A x) {
    x.n = n;
    x.v = x;
    return x;
  }

  public long getN() {
    return n;
  }
}
```

```
public non-sealed class B extends A {
  public long n;

  public B(int n) {
    this.v = new A();
    this.n = n;
  }

  public static A g(int n) {
    return new B(n);
  }

  public long getN() {
    return n;
  }
}
```

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 final unsigned n = 4;
3
4    public C() {
5      super(0);
6    }
7
8    public int C(int n) {
9      super(n);
10     return n;
11   }
12
13   public static C f() {
14     B b = new C();
15     return b;
16   }
17 }
```

Lösung:

a)
```
public class M {
  public static void main(String[] args) {

    A a = new A();
    System.out.println(a.n + " " + a.v);          // OUT: [ 0 ] [null]

    A.f(1,a);
```

```
        System.out.println(a.n + " " + a.v.n);          // OUT: [ 1 ] [ 1 ]

        B b = new B(2);
        System.out.println(b.n + " " + b.v.n);          // OUT: [ 2 ] [ 0 ]

        A ab = B.g(3);
        System.out.println(ab.n + " " + ab.getN());     // OUT: [ 0 ] [ 3 ]

        System.out.println(ab instanceof B);            // OUT: [true]
    }
}
```

In der Musterlösung werden die Werte `1`, `2` und `3` genutzt. Die Klausur war an diesen Stellen jedoch parametrisiert, sodass sie hier die drei zufällige Werte ihrer persönlichen Klausur einsetzen müssen.

b)

- Zeile 2: Es gibt keinen primitiven Datentyp `unsigned` in Java.
- Zeile 9: `super(...)` kann nur in Konstruktoren verwendet werden.
- Zeile 15: Der Rückgabewert der Methode ist `C`, es wird aber ein Objekt der Klasse `B` zurückgegeben.

Aufgabe 2 (Hoare-Kalkül): (12 + 3 = 15 Punkte)

Gegeben sei folgendes Java-Programm P über der `int[]`-Variablen `a` und den `int`-Variablen `res` und `i`:

$\langle$ `a.length` $> 0 \rangle$ (Vorbedingung)

```
i = a.length;
res = 0;
while (i > 0) {
    i = i - 1;
    if (a[i] > 0) {
     res = res + a[i];
    }
    else{
     res = res - a[i];
    }
}
```

$\langle$ `res` $= as(0)$ $\rangle$ (Nachbedingung)

Für alle $i \in \{0, \ldots,$ `a.length`$\}$ gilt $as(i) = \sum_{\texttt{j=i}}^{\texttt{a.length}-1} |\texttt{a[j]}|$, d.h. $as($`a.length`$) = 0$ und $as(0) = |\texttt{a[0]}| + \cdots + |\texttt{a[a.length} - 1\texttt{]}|$. Hierbei steht "$as$" für "absolute sum".

a) Vervollständigen Sie die folgende Verifikation des Programms 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:

- 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.

b) Untersuchen Sie das Programm 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)

$\langle \texttt{a.length} > 0 \rangle$
$\langle \texttt{a.length} = \texttt{a.length} \wedge 0 = 0 \wedge \texttt{a.length} > 0 \rangle$

```
i = a.length;
```

$\langle \texttt{i} = \texttt{a.length} \wedge 0 = 0 \wedge \texttt{a.length} > 0 \rangle$

```
res = 0;
```

$\langle \texttt{i} = \texttt{a.length} \wedge \texttt{res} = 0 \wedge \texttt{a.length} > 0 \rangle$
$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
while (i > 0) {
```

$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \wedge \texttt{i} > 0 \rangle$
$\langle \texttt{res} = as(\texttt{i} - 1 + 1) \wedge \texttt{i} - 1 \geq 0 \rangle$

```
    i = i - 1;
```

$\langle \texttt{res} = as(\texttt{i} + 1) \wedge \texttt{i} \geq 0 \rangle$

```
    if (a[i] > 0) {
```

$\langle \texttt{res} = as(\texttt{i} + 1) \wedge \texttt{i} \geq 0 \wedge \texttt{a}[\texttt{i}] > 0 \rangle$
$\langle \texttt{res} + \texttt{a}[\texttt{i}] = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
        res = res + a[i];
```

$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
    }
    else {
```

$\langle \texttt{res} = as(\texttt{i} + 1) \wedge \texttt{i} \geq 0 \wedge \neg \texttt{a}[\texttt{i}] > 0 \rangle$
$\langle \texttt{res} - \texttt{a}[\texttt{i}] = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
        res = res - a[i];
```

$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
    }
```

$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \rangle$

```
}
```

$\langle \texttt{res} = as(\texttt{i}) \wedge \texttt{i} \geq 0 \wedge \neg \texttt{i} > 0 \rangle$
$\langle \texttt{res} = as(0) \rangle$

b) Wir wählen als Variante $V = \texttt{i}$. Hiermit lässt sich die Terminierung von P beweisen, denn für die einzige Schleife im Programm (mit Schleifenbedingung $B = \texttt{i} > 0$) gilt:

- $B \Rightarrow V \geq 0$
- die folgende Ableitung ist korrekt:

$\langle \texttt{i} = m \wedge \texttt{i} > 0 \rangle$
$\langle \texttt{i} - 1 < m \rangle$

```
    i = i - 1;
```

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

```
    if (a[i] > 0) {
```

$\langle \texttt{i} < m \wedge \texttt{a}[\texttt{i}] > 0 \rangle$
$\langle \texttt{i} < m \rangle$

```
        res = res + a[i];
```

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

```
    }
    else {
```

$\langle \texttt{i} < m \wedge \neg \texttt{a}[\texttt{i}] > 0 \rangle$

```
                        ⟨i < m⟩
    res = res - a[i];
                        ⟨i < m⟩
}
                        ⟨i < m⟩
```

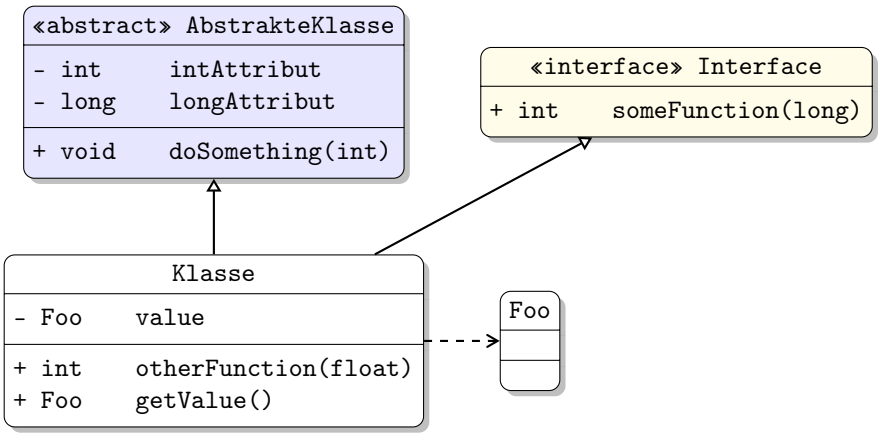
Aufgabe 3 (Klassenhierarchie): (8 + 7 = 15 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Hierarchie zum Umgang mit verschiedenen Produkten, die von einem Verlag verlegt werden, den sogenannten Verlagsprodukten.

- Verlagsprodukte haben einen Namen, der ein Attribut vom Typ `String` ist. Jedes Verlagsprodukt ist stets auch von mindestens einem weiteren (Unter-)Typ.
- Ein Buch ist ein Verlagsprodukt, das innerhalb einer Buchreihe erscheint. Daher hat ein Buch ein Attribut für seinen Vorgänger, welcher wiederum ein Buch ist.
- Ein Krimi ist ein Buch. Er hat ein Attribut, um den Namen des Täters (aus dem Krimi) zu speichern.
- Ein Gesellschaftsspiel ist ein Verlagsprodukt, das die maximale Anzahl an Spielern als Attribut hat. Jedes Gesellschaftsspiel ist stets auch von mindestens einem weiteren (Unter-)Typ.
- Ein Kartenspiel ist ein Gesellschaftsspiel. Man kann ein Kartenspiel mischen, wofür keine weiteren Informationen benötigt werden. Es wird auch nichts zurückgegeben.
- Ein Krimispiel ist ein Gesellschaftsspiel. Es hat ein Attribut, um den Namen des Täters (aus dem Krimispiel) zu speichern.
- Bei Krimis und Krimispielen ist der *Täter ratbar*. Die Methode `boolean raten(String taeter)` gibt zurück, ob der Täter richtig geraten wurde.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Klassen. 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 Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Überschriebene Methoden müssen nicht angegeben werden. Weiterhin bedeutet der Pfeil $B \longrightarrow\!\triangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Der Pfeil $A \dashrightarrow B$ bedeutet, dass A den Typ B in den Typen seiner Attribute oder in den Ein- oder Ausgabeparametern seiner Methoden verwendet. 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) Sarah plant eine große Spieleveranstaltung, für die sie auf ihre Spielesammlung zurückgreifen möchte. Schreiben Sie außerhalb der in Teilaufgabe (a) modellierten Hierarchie eine Java-Methode `vorbereiten`. Beim Aufruf von `vorbereiten(arr)` werden alle Verlagsprodukte im Parameter `arr` durchlaufen. Um

festzustellen, wie viele Eintrittskarten Sarah verkaufen kann, möchte sie wissen, wie viele Personen höchstens gleichzeitig spielen können, wenn all ihre Gesellschaftsspiele zum Einsatz kommen. Diese Zahl soll von der Methode zurückgegeben werden. Außerdem möchte sie den Besuchern etwas Arbeit abnehmen: Bei jedem Kartenspiel in `arr` sollen die Karten gemischt werden.

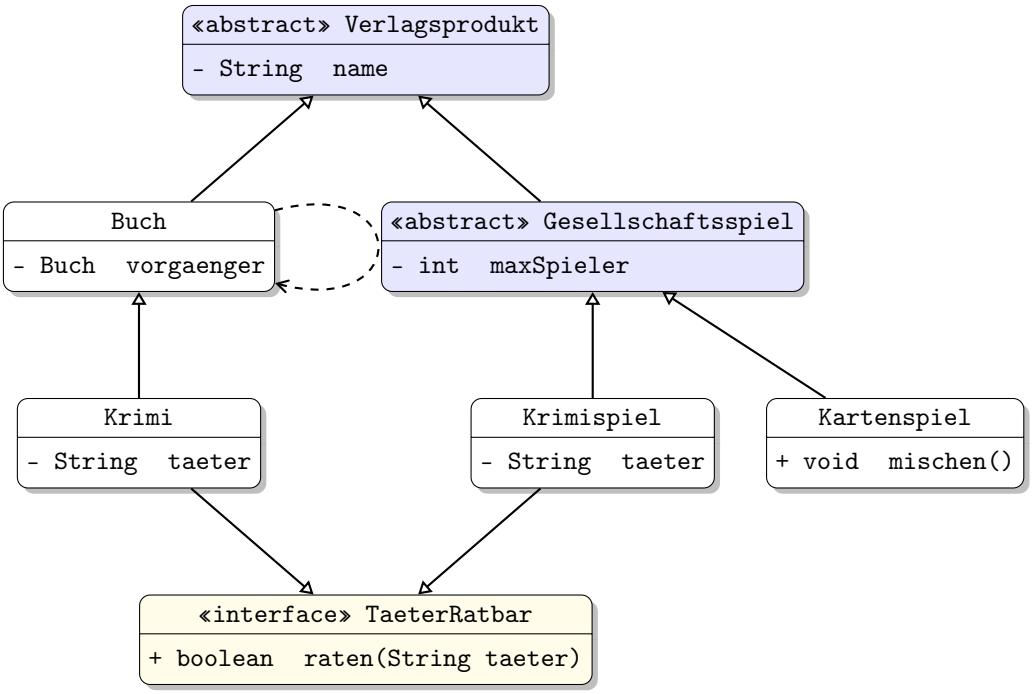
Hinweise:

- Nehmen Sie an, dass der übergebene Parameter `arr` nicht `null` ist.
- Gehen Sie davon aus, dass es für alle Attribute geeignete Selektoren gibt.

```
public int vorbereiten(Verlagsprodukt[] arr) {
```

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)
```
public int vorbereiten(Verlagsprodukt[] arr) {
    int res = 0;
    for (Verlagsprodukt p : arr) {
        if (p instanceof Gesellschaftsspiel s) {
            res += s.getMaxSpieler();
            if (s instanceof Kartenspiel k) {
                k.mischen();
            }
        }
    }
    return res;
}
```

Aufgabe 4 (Programmierung in Java): (9 + 8 + 12 + 7 = 36 Punkte)

In einer Pandemie ist es hilfreich, das Infektionsgeschehen nachzuverfolgen. In dieser Aufgabe entwerfen wir eine Datenstruktur, um zu erfassen, wer wen angesteckt hat. Dazu nutzen wir die Klasse `Spreader`.

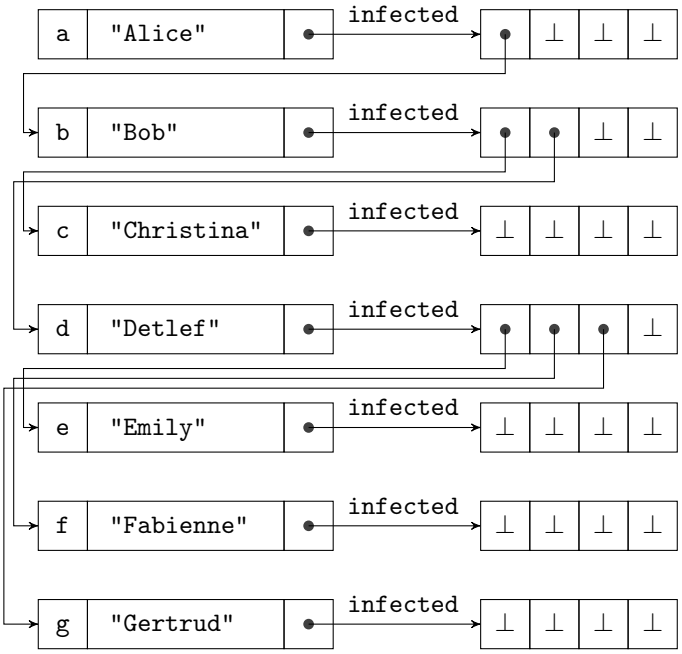
```
public class Spreader {
  private String info;
  private Spreader[] infected;

  public String getInfo() {
    return info;
  }
}
```

Ein `Spreader`-Objekt stellt eine infizierte Person dar. Es enthält die Personendaten dieser infizierten Person im Attribut `info`. Im Attribut `infected` werden die *Ansteckungen* erfasst. Es enthält ein Array mit jenen `Spreader`-Objekten, die von dem aktuellen `Spreader`-Objekt angesteckt worden sind.

In dieser Aufgabe soll das Array im Attribut `infected` immer die Länge 4 haben. Es können also pro Person höchstens 4 Ansteckungen erfasst werden. Falls weniger Ansteckungen erfasst werden sollen, so sollen die ungenutzten Arrayeinträge den Wert `null` haben.

Beispiel:

In der folgenden Grafik ist beispielhaft ein Infektionsgeschehen dargestellt.



Wir sehen sieben `Spreader`-Objekte `a` bis `g`, dargestellt durch je drei nebeneinander liegende Rechtecke. Das linke Rechteck enthält dabei immer einen Bezeichner, 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. Das mittlere Rechteck enthält den Wert des Attributs `info`. Das rechte Rechteck enthält den Wert des Attributs `infected`. Wir sehen außerdem sieben Arrays mit Einträgen vom Typ `Spreader`, dargestellt durch je vier nebeneinander liegende Quadrate. Dabei enthalten die Quadrate von links nach rechts die Werte der Arrayeinträge an Index 0 bis 3. Der Wert `null` wird durch ⊥ dargestellt. In diesem Beispiel enthält das Attribut `info` den Namen des `Spreader`s.

Der `Spreader` `a` hat den Namen `"Alice"` und hat den `Spreader` `b` angesteckt. Der `Spreader` `b` hat den Namen `"Bob"` und hat die `Spreader` `c` und `d` angesteckt. Der `Spreader` `c` hat den Namen `"Christina"` und hat bisher niemanden angesteckt. Der `Spreader` `d` hat den Namen `"Detlef"` und hat die `Spreader` `e`, `f` und `g` angesteckt. Der `Spreader` `e` hat den Namen `"Emily"`, der `Spreader` `f` hat den Namen `"Fabienne"` und der `Spreader` `g` hat den Namen `"Gertrud"`. Die `Spreader` `e`, `f` und `g` haben bisher niemanden angesteckt.

Hinweise:

- Sie dürfen in allen Teilaufgaben davon ausgehen, dass nur auf azyklischen Infektionsgeschehen gearbeitet wird (d.h. man erreicht keinen Zyklus, wenn man jeweils einem beliebigen Eintrag des `infected`-Attributs folgt). Außerdem dürfen Sie davon ausgehen, dass es für jede Person höchstens eine Person gibt, von der diese angesteckt wurde. Daher existiert zwischen zwei `Spreader`-Objekten höchstens ein Pfad.
- Achten Sie darauf, bei allen Teilaufgaben nicht nur die Beispiele korrekt zu behandeln, sondern den allgemeinen Fall zu lösen.
- Sie dürfen in allen Teilaufgaben davon ausgehen, dass die Attribute `info` und `infected` jedes `Spreader`-Objekts nicht den Wert `null` haben.
- Sie dürfen in allen Teilaufgaben Klassen und Methoden aus vorigen Teilaufgaben verwenden, auch wenn Sie diese nicht implementiert haben.
- Sie dürfen eigene Hilfsmethoden implementieren und nutzen.
- Es ist nicht erlaubt, von `Java` vorgegebene Methoden zu nutzen. Ausnahmen dieser Regel finden sich in den Hinweisen der Teilaufgaben.

a) Implementieren Sie in der Klasse `Spreader` einen Konstruktor mit den Parametern `info` und `infected`. Dieser soll das Attribut `info` mit dem Parameter `info` initialisieren. Außerdem soll er das Attribut `infected` mit einem Array der Länge vier initialisieren. Enthält der Parameter `infected` höchstens vier Einträge, so sollen diese ins Array im Attribut `infected` kopiert werden. Enthält der Parameter `infected` mehr als vier Einträge, so sollen nur die ersten vier Einträge ins Array im Attribut `infected` kopiert werden. Im Attribut `infected` sollen die Einträge dabei die gleichen Index-Positionen bekommen wie im Parameter `infected`.

Beispiele:

- Beispielsweise kann das Objekt `d` durch folgenden Aufruf erzeugt werden:
  `new Spreader("Detlef", e, f, g)`
- Beispielsweise kann das Objekt `f` durch folgenden Aufruf erzeugt werden:
  `new Spreader("Fabienne")`

Hinweise:

- Sie dürfen die Methode `Math.min(i1, i2)` nutzen, welche zwei `int`-Werte erhält und den kleineren `int`-Wert zurückgibt.
- Sie dürfen davon ausgehen, dass der Parameter `info` des Konstruktors nicht `null` ist.
- Sie dürfen davon ausgehen, dass der Parameter `infected` des Konstruktors nicht `null` ist und keine `null`-Werte enthält.

```
public Spreader(String info, Spreader... infected) {
```

b) Implementieren Sie in der Klasse `Spreader` die Methode `addInfected`, welche für den aktuellen `Spreader` eine neue Ansteckung erfassen soll. Dazu werden die Personendaten der angesteckten Person im Parameter `infectedInfo` übergeben. Die Methode sucht dann den ersten freien Platz des Arrays im Attribut `infected` (d.h. den ersten `null`-Eintrag in diesem Array). Existiert ein solcher freier Platz, so wird ein neues `Spreader`-Objekt mit `infectedInfo` als Personendaten erstellt und an diesem ersten freien Platz im Array abgelegt. Dieses neue `Spreader`-Objekt hat selbst noch keine weiteren Personen angesteckt. Existiert kein freier Platz mehr im Array im Attribut `infected`, so wird eine `CannotTrackException` geworfen, welche wie folgt definiert ist:

```
public class CannotTrackException extends Exception {}
```

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

Beispiele:

- Beispielsweise kann durch folgenden Aufruf erfasst werden, dass Emily von Detlef angesteckt wurde:
  `d.addInfected("Emily")`

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `infectedInfo` der Methode `addInfected` nicht `null` ist.

```
public void addInfected(String infectedInfo) throws CannotTrackException {
```

c) Alleine der Name der infizierten Person genügt nicht, um eine effektive Erfassung der Ansteckungen zu gewährleisten. Um möglichst flexibel zu sein, welche Informationen die Personendaten umfassen, soll der Typ der Personendaten nun nicht mehr `String` sein, sondern ein generischer Typparameter. Es ist nicht notwendig, die Klasse `Spreader` anzupassen.

Schreiben Sie die Klasse `PopulationImpl<T>`, welche das nachfolgende Interface `Population<T>` implementiert. Jedes Objekt der Klasse soll ein Attribut `infos` vom Typ `java.util.List<T>` haben, welches eine Liste mit Personendaten enthält. Dieses Attribut soll durch einen Konstruktor initialisiert werden, welcher einen Parameter des Typs `java.util.List<T>` erhält und diesen dem Attribut `infos` zuweist. Die Methode `removeRandomPerson` der Klasse `PopulationImpl<T>` soll dann zufällig einen der Listeneinträge im Attribut `infos` auswählen, löschen und zurückgeben. Dabei soll bei jedem Aufruf der Methode erneut zufällig entschieden werden, welcher Listeneintrag zurückgegeben wird. Im Prinzip soll es möglich sein, dass jeder Listeneintrag zufällig gewählt wird.

```
public interface Population<T> {
    T removeRandomPerson();
}
```

Beispiel:

Beispielsweise soll folgender Aufruf zufällig einen der `T`-Werte `t1`, `t2` oder `t3` aus der Liste `infos` löschen und in die Variable `result` schreiben:

```
List<T> infos = new ArrayList<>();
infos.add(t1); infos.add(t2); infos.add(t3);
T result = new PopulationImpl(infos).removeRandomPerson();
```

Hinweise:

- Sie dürfen die Methode `Math.random()` nutzen, welche bei jedem Aufruf einen zufällig gewählten `double`-Wert zurückgibt, welcher im Bereich zwischen `0.0` (inklusive) und `1.0` (exklusive) liegt.
- Sie dürfen für eine Liste `xs` vom Typ `List<T>` die Methode `xs.size()` nutzen, welche die Anzahl ihrer Elemente zurückgibt.
- Sie dürfen für eine Liste `xs` vom Typ `List<T>` und $0 \leq$ `i` $<$ `xs.size()` die Methode `xs.remove(i)` nutzen, welche das Element an der Position `i` entfernt und den Index `j` aller Listeneinträge mit `j` $>$ `i` um eins verringert. Die Methode `remove` liefert das entfernte Element vom Typ `T` zurück.
- Gehen Sie davon aus, dass die Datei `PopulationImpl.java` mit folgender Zeile beginnt:
  `import java.util.List;`
- Sie dürfen davon ausgehen, dass der Parameter des Konstruktors nicht `null` ist und keine `null`-Werte enthält.

d) Implementieren Sie in der Klasse `Spreader` die Methode `spread` mit dem Parameter `population`, welche einen Ausbreitungsschritt der Pandemie darstellen soll. Dazu wird für den aktuellen `Spreader`, für alle direkt von ihm angesteckten Personen und für alle indirekt von ihm angesteckten Personen jeweils das im folgenden Abschnitt beschriebene Vorgehen ausgeführt. Eine indirekte Ansteckung zwischen Person p_1 und Person p_n liegt vor, wenn Person p_i jeweils Person p_{i+1} angesteckt hat (für alle $1 \leq i < n$ und $3 \leq n$).

Jeder dieser `Spreader` steckt, jeweils mit einer Wahrscheinlichkeit von 70%, eine weitere Person an, welche zufällig aus der Bevölkerung ausgewählt wird. Die Personendaten der neu infizierten Personen sollen also mithilfe der Methode `removeRandomPerson` des Parameters `population` gewonnen werden. Beachten Sie, dass es bei der Krankheit eine Inkubationszeit gibt. Personen, welche gerade erst infiziert worden sind, stecken also im selben Ausbreitungsschritt keine weiteren Personen an. Nutzen Sie zum Erfassen der neuen Ansteckungen die Methode `addInfected`.

Zum Durchlaufen des Arrays im Attribut `infected` darf eine Schleife verwendet werden. Ansonsten dürfen Sie in dieser Teilaufgabe keine weiteren Schleifen, sondern **nur Rekursion** benutzen.

Beispiel:

Folgender Aufruf führt dazu, dass die `Spreader` `b` bis `g` jeweils mit einer Wahrscheinlichkeit von 70% eine weitere Person aus der Bevölkerung `population` anstecken:
`b.spread(population)`

Hinweise:

- Sie dürfen die Methode `Math.random()` nutzen, welche bei jedem Aufruf einen zufällig gewählten `double`-Wert zurückgibt, welcher im Bereich zwischen `0.0` (inklusive) und `1.0` (exklusive) liegt.
- Sie dürfen davon ausgehen, dass der Parameter `population` der Methode `spread` nicht `null` ist.
- Wir betrachten hier eine `population` vom Typ `Population<String>`. Die Methode `removeRandomPerson` liefert daher einen `String` zurück, d.h. ihr Resultat hat den gleichen Typ wie der Parameter `infectedInfo` der Methode `addInfected`. Es ist also nicht notwendig, die Methode `addInfected` anzupassen.

- Da die Methode `addInfected` für die angesteckte Person ein *neues* `Spreader`-Objekt erzeugt, ist sicher gestellt, dass es auch weiterhin für jede Person höchstens eine Person gibt, von der diese angesteckt wurde.

```
public void spread(Population<String> population)
    throws CannotTrackException {
```

Lösung:

a)
```
public Spreader(String info, Spreader... infected) {
    this.info = info;
    this.infected = new Spreader[4];
    int length = Math.min(this.infected.length, infected.length);
    // int length = this.infected.length < infected.length
    //             ? this.infected.length : infected.length;
    for (int i = 0; i < length; i++) {
      this.infected[i] = infected[i];
    }
  }
```

b)
```
public void addInfected(String infectedInfo) throws CannotTrackException {
  for (int i = 0; i < infected.length; i++) {
    if (infected[i] == null) {
      infected[i] = new Spreader(infectedInfo);
      return;
    }
  }
  throw new CannotTrackException();
}
```

c)
```
public class PopulationImpl<T> implements Population<T> {
  private List<T> infos;

  public PopulationImpl(List<T> infos) {
    this.infos = infos;
  }

  @Override
  public T removeRandomPerson() {
    return infos.remove(
        (int) (
        Math.random() * infos.size()));
  }
}
```

d)
```
public void spread(Population<String> population) throws CannotTrackException {
  for (Spreader spreader : infected) {
    if (spreader != null) {
      spreader.spread(population);
    }
  }
  if (Math.random() < 0.7) { // Hier kann auch <= genutzt werden.
    addInfected(population.removeRandomPerson());
  }
}
```

Aufgabe 5 (Haskell): (2 + 3 + 4 + 8 + 3 = 20 Punkte)

a) Geben Sie zur folgenden Haskell–Funktion `f` den allgemeinsten Typ an.

```
f w (x:y:_) | x w = [x w]
            | y w = [w, w]
```

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

```
i :: [Int]
i = filter (\x -> any (>x) [3,4]) [2,3,4,5]
```

Hinweise:

- Für jede Zahl `x` vom Typ `Int` ist `(>x) :: Int -> Bool` die Funktion, die herausfindet, ob ihr Argument größer als `x` ist. Es gilt also `(>3) 1 == False` (denn $1 \not> 3$) und `(>3) 4 == True` (denn $4 > 3$).
- Die vordefinierte Funktion `any :: (a -> Bool) -> [a] -> Bool` wertet genau dann zu `True` aus, wenn die übergebene Funktion auf mindestens einem der Elemente der übergebenen Liste zu `True` auswertet. Beispielsweise gilt `any (>2) [1,2,3] == True` und `any (>3) [1,2,3] == False`.

```
j :: [Bool]
j = map (\flag -> ((not flag) || flag) && (not flag) && flag) [False,True]
```

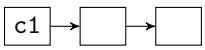
c) Wir verwenden die folgende Datenstruktur `Cell`, um zweidimensionale Rastergitter zu repräsentieren.

```
data Cell = C Cell Cell | N
```

Der Datenkonstruktor `C` (kurz für *Center*) erzeugt eine neue Zelle, deren rechter Nachbar im ersten Argument und deren unterer Nachbar im zweiten Argument steht. Diese Argumente sind dann wieder vom Typ `Cell`. Ihren linken und oberen Nachbarn kennt eine Zelle dagegen nicht. Der Datenkonstruktor `N` (kurz für *Nil*) erzeugt eine Stelle, die nicht zum Raster gehört, wo sich also keine Zelle befindet.

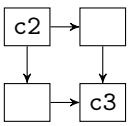
Der folgende Haskell–Ausdruck beschreibt ein Raster mit Länge 1 und Breite 3:

```
c1 :: Cell
c1 = C (C (C N N) N) N
```



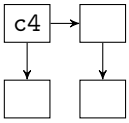
Der folgende Haskell–Ausdruck beschreibt ein Raster mit Länge 2 und Breite 2:

```
c2 :: Cell
c2 = C (C N c3) (C c3 N)
    where c3 = C N N
```



Darüber hinaus sind auch unvollständige Raster darstellbar:

```
c4 :: Cell
c4 = C (C N (C N N)) (C N N)
```



Außerdem verwenden wir eine zweite Datenstruktur `Movement`, um eine Bewegung nach rechts (Right) oder nach unten (Down) zu repräsentieren.

```
data Movement = R | D
```

Für alle folgenden Teilaufgaben gelten die folgenden Hinweise:

Hinweise:

- Sie können jeweils Funktionen aus vorherigen Teilaufgaben verwenden, unabhängig davon, ob Sie diese implementiert haben.
- Sie dürfen beliebige vordefinierte Funktionen aus der Standardbibliothek verwenden, wie z.B. `map`.
- Sie dürfen eigene Hilfsfunktionen implementieren und nutzen.

Schreiben Sie eine Haskell–Funktion `move :: Cell -> [Movement] -> Cell`, die von der übergebenen Zelle aus die Bewegungen der übergebenen Liste ausführt und die dadurch erreichte Zelle zurückgibt. Wenn die Bewegungen nicht vollständig ausführbar sind, weil vor dem Ende der Liste `N` erreicht wird, soll `N` zurückgegeben werden. Der Ausdruck `move c1 [D,D]` soll also zu `N` auswerten und die Ausdrücke `move c2 [R,D]` und `move c2 [D,R]` sollen zu `C N N` auswerten.

d) Schreiben Sie eine Haskell–Funktion `movements :: Cell -> [[Movement]]`, die für die übergebene Zelle die Liste aller möglichen Bewegungslisten zurückgibt, die *nicht* in `N` enden. Dabei stellt die leere Liste `[]` die leere Bewegung dar, die von jeder Zelle (außer `N`) möglich ist. Der Ausdruck `movements c1` soll zu `[[R,R],[R],[]]` auswerten und der Ausdruck `movements c2` soll zu `[[R,D],[R],[D,R],[D],[]]` auswerten. Die Reihenfolge der Listen in der Ergebnis-Bewegungsliste ist unerheblich.

Hinweise:

- Für einen Wert `x` vom Typ `a` ist `(x:) :: [a] -> [a]` die Funktion, die das Element `x` vorne in die Argument-Liste einfügt. Der Ausdruck `(R:) [D,R]` steht also für `[R,D,R]`. Dies ist ähnlich zu der Funktion `(>x)` in Teilaufgabe (b).

e) Gegeben sei die folgende Funktion:

```
foo :: Cell -> [Cell]
foo c = map (move c) (movements c)
```

Beschreiben Sie in eigenen Worten, in welcher Beziehung die Eingabezelle von `foo` zur Ausgabeliste von `foo` steht. Geben Sie außerdem an, zu welchem Ergebnis `foo c1` auswertet. Die Reihenfolge der Zellen in der Ergebnisliste ist dabei unerheblich.

Lösung:

a)
```
f :: Bool -> [Bool -> Bool] -> [Bool]
```

b)
```
i = [2,3]
j = [False,False]
```

c) In den folgenden Aufgaben ist zu beachten, dass Werte vom Typ `Movement` und `Cell` nicht mittels `(==)` verglichen werden können. Stattdessen ist zum Unterscheiden der jeweiligen Werte ein Patternmatching notwendig.

```
move :: Cell -> [Movement] -> Cell
move N _ = N
move c [] = c
move (C r d) (R:xs) = move r xs
move (C r d) (D:xs) = move d xs
```

d)
```
movements :: Cell -> [[Movement]]
movements N = []
movements (C r d) = (map (R :) mr) ++ (map (D :) md) ++ [[]]
                          where mr = movements r
                                md = movements d
```

e) Die Funktion `foo` berechnet für eine Zelle `c` die Liste der erreichbaren Zellen, (wobei eine Zelle auch stets von sich selbst aus erreichbar ist.) (Wenn eine Zelle auf mehreren Wegen erreichbar ist, wird sie auch mehrfach ausgegeben.)

Der Ausdruck `foo c1` wertet zu `[C N N,C (C N N) N,C (C (C N N) N) N]` aus.

Aufgabe 6 (Prolog): (2 + 7 + (3 + 3 + 5) = 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) $f(g(X), Y, Z), f(Y, g(h(X)), Y)$

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

b) Gegeben sei folgendes Prolog-Programm P.

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

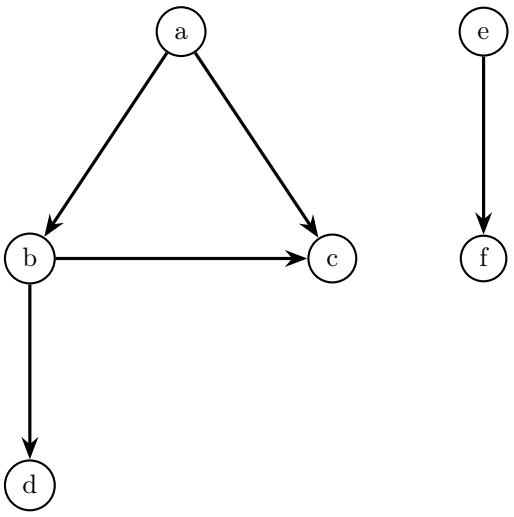
- Erstellen Sie für das Programm P den Beweisbaum zur Anfrage "`?- t(q(0),Y).`" bis zur Höhe 3 **(die Wurzel hat dabei die Höhe 1)**. Die Pfade haben also maximal die Länge 2.
- Markieren Sie die Knoten in Höhe 3, die zu einer unendlichen Auswertung führen können, mit ∞.
- Geben Sie alle Antwortsubstitutionen zur Anfrage "`?- t(q(0),Y).`" 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 gerichtete Graphen in Prolog. Dabei wird ein Graph durch eine Sammlung von `knoten`-Fakten definiert. Für jeden Knoten des Graphen enthält das Programm ein Fakt `knoten(Knotenname,Nachfolgerliste)`. Für einen Knoten `a` mit den Nachfolgern `b` und `c` ergäbe sich also `knoten(a,[b,c])`.

Die Fakten

```
knoten(a,[b,c]).
knoten(b,[c,d]).
knoten(c,[]).
knoten(d,[]).
knoten(e,[f]).
knoten(f,[]).
```

entsprechen also dem folgenden Graphen:



In den folgenden Teilaufgaben dürfen sie **keine** vordefinierten Prädikate von Prolog verwenden. Gehen Sie für die folgenden Aufgaben davon aus, dass das Programm bereits Fakten für einen beliebigen wie oben beschriebenen gerichteten Graph enthält.

Hinweise:

- Sie dürfen Prädikate aus vorigen Teilaufgaben verwenden, auch wenn Sie diese Prädikate nicht implementiert haben.
- Sie dürfen eigene Hilfsprädikate implementieren und nutzen.

i Implementieren Sie ein Prädikat `nachfolger` mit Stelligkeit 2 in Prolog. Wenn `X` und `Y` Knoten des gerichteten Graphen sind, dann soll `nachfolger` genau dann wahr sein, wenn `Y` einer der direkten Nachfolger von `X` ist. Bei dem oben angegebenen Graphen ist beispielsweise `nachfolger(a,b)` wahr und `nachfolger(a,d)` falsch.

ii Implementieren Sie ein Prädikat `pfad` mit Stelligkeit 3 in Prolog. Wenn `X` und `Y` Knoten des gerichteten Graphen sind, so soll `pfad(X,Y,ZS)` genau dann wahr sein, wenn es einen Pfad im Graphen von `X` nach `Y` gibt und die Liste `ZS` aus den Knoten des Pfades in der richtigen Reihenfolge besteht. Sie dürfen davon ausgehen, dass der Graph keine Zyklen enhält. Der ausgegebene Pfad soll sowohl den Start- als auch den Zielknoten enthalten. Bei dem obigen Graphen hätte die Anfrage `?-pfad(a,c,ZS).` also die beiden Antwortsubstitutionen `ZS = [a,b,c]` und `ZS = [a,c].` Die Anfrage `?-pfad(a,a,ZS)` hätte die einzige Antwortsubstitution `ZS = [a].`

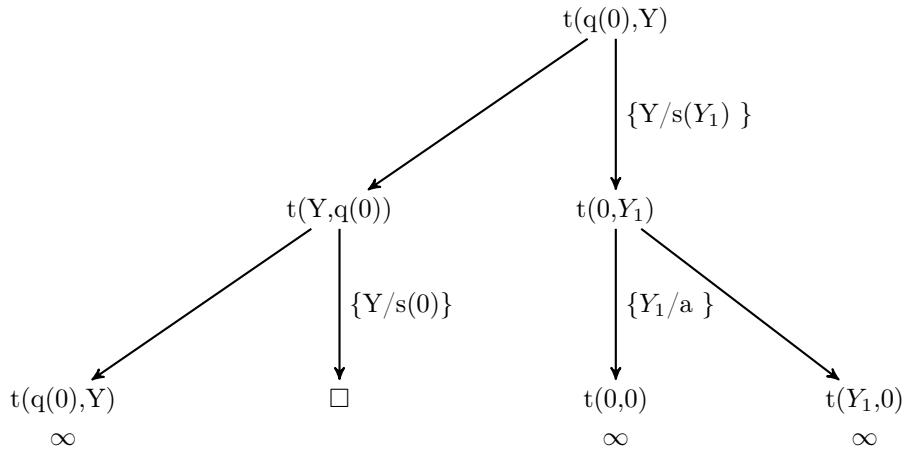
iii Implementieren Sie ein Prädikat `vollstaendig` mit Stelligkeit 1 in Prolog. Wenn `XS` eine Liste von Knoten des gerichteten Graphen ist, so soll `vollstaendig(XS)` genau dann wahr sein, wenn zwischen jedem Paar verschiedener Knoten in `XS` mindestens eine Kante existiert. Dabei soll die Richtung der Kante keine Rolle spielen. Beispielsweise ist bei dem obigen Graphen also `vollstaendig([a,b,c])` wahr, aber `vollstaendig([a,b,c,d])` ist falsch, weil es keine Kante zwischen `a` und `d` gibt.

Lösung:

a) i) $f(g(X), Y, Z), f(Y, g(h(X)), Y)$ kann nicht unifiziert werden, da nach der Unifikation des ersten Arguments das zweite Argument $g(X)$ bzw. $g(h(X))$ ist. Es liegt ein Occur Failure vor.

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

b)



Die einzige Antwortsubstitution innerhalb des Beweisbaums ist $\{Y/s(0)\}$. Diese wird von Prolog nicht gefunden.

Die zweite Regel muss hinter das Faktum verschoben werden, damit Prolog die Antwortsubstitutionen bis zur Höhe 3 findet. Eine Möglichkeit wäre z.B.:

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

c)

```
mitglied(X,[X|_]).
mitglied(X,[_|Ys]):-mitglied(X,Ys).

nachfolger(A,B):- knoten(A,Ys),mitglied(B,Ys).

pfad(A,A,[A]).
pfad(A,B,[A|Xs]):- nachfolger(A,Y),pfad(Y,B,Xs).


verbunden(X,[]).
verbunden(X,[Y|Ys]):-nachfolger(X,Y), verbunden(X,Ys).
verbunden(X,[Y|Ys]):-nachfolger(Y,X), verbunden(X,Ys).

vollstaendig([]).
vollstaendig([V|Vs]):-verbunden(V,Vs), vollstaendig(Vs).
```