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

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

  public int f(String p) {
    return 1;
  }
  public int f(int p) {
    return 2;
  }
  public int f(A p) {
    return 3;
  }
}
```

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

  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 4;
  }
  public int f(B p) {
    return 5;
  }
}
```

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

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

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

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

    System.out.println(a1.f("6"));                     //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      this.s = 0;
6    }
7
8    private boolean f(long p) {
9      return C.s != 0;
10   }
11
12   public static void main(String[] args) {
13     System.out.println(new A().f(7L));
14   }
15 }
```

Lösung:

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

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

      A ab = new B(5);
      System.out.println(((A)ab).s + " " + ((B)ab).s);//OUT: [ "b"] ["c5"]

      System.out.println(ab.getS());                     //OUT: ["c5"]

      System.out.println(a1.f("6"));                     //OUT: [  1 ]

      System.out.println(b1.f(ab));                      //OUT: [  4 ]

      System.out.println(ab.f(b1));                      //OUT: [  4 ]
    }
  }
```

b)

- Compilerfehler: Implicit `super` constructor `B()` is undefined. Must explicitly invoke another constructor (Line 4)
  Es wurde einer der folgenden Fehler erkannt:
  - Der Konstruktor in `C` muss explizit einen der Konstruktoren aus `B` aufrufen, da es in `B` keinen parameterlosen Konstruktor gibt (Line 4).
- Compilerfehler: Cannot make a `static` reference to the non-`static` field `C.s` (Line 9)
  Es wurde einer der folgenden Fehler erkannt:
  - Der Zugriff auf das Attribut `s` muss nicht-statisch erfolgen, etwa als `s` oder `this.s` (Line 9).
  - Das Attribut `s` in der Klasse `C` muss mit dem `static`-Keyword deklariert sein (Line 2).
- Compilerfehler: The method `f(String)` in the type `A` is not applicable for the arguments `(long)` (Line 13)
  Es wurde einer der folgenden Fehler erkannt:
  - Die Methode `f` kann nicht mit einem `long`-Wert aufgerufen werden (Line 13).
  - Ein `long`-Wert kann nicht implizit zu einem `int`-Wert gecastet werden (Line 13).
  - Die Methode `f` muss auf einer Instanz der Klasse `C` aufgerufen werden (Line 13).

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

Gegeben sei folgendes Java-Programm P über den `int`-Variablen `a` und `b`:

$\langle\, \texttt{b} \geq 0 \,\rangle$ (Vorbedingung)

```
a = b * b;
while (b > 0) {
    if (b % 2 == 0) {
        a = a - 4 * b + 4;
        b = b - 2;
    }
    else {
        a = a - 2 * b + 1;
        b = b - 1;
    }
}
```

$\langle\, \texttt{a} = 0 \,\rangle$ (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:

- 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) 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)

$\langle \mathtt{b} \geq 0 \rangle$
$\langle \mathtt{b} \geq 0 \land \mathtt{b} \cdot \mathtt{b} = \mathtt{b} \cdot \mathtt{b} \rangle$

```
a = b * b;
```

$\langle \mathtt{b} \geq 0 \land \mathtt{a} = \mathtt{b} \cdot \mathtt{b} \rangle$
$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \rangle$

```
while (b > 0) {
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \land \mathtt{b} > 0 \rangle$

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

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \land \mathtt{b} > 0 \land \mathtt{b} \mod 2 = 0 \rangle$
$\langle \mathtt{a} - 4 \cdot \mathtt{b} + 4 = (\mathtt{b} - 2)^2 \land (\mathtt{b} - 2) \geq 0 \rangle$

```
        a = a - 4 * b + 4;
```

$\langle \mathtt{a} = (\mathtt{b} - 2)^2 \land (\mathtt{b} - 2) \geq 0 \rangle$

```
        b = b - 2;
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \rangle$

```
    }
    else {
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \land \mathtt{b} > 0 \land \neg(\mathtt{b} \mod 2 = 0) \rangle$
$\langle \mathtt{a} - 2 \cdot \mathtt{b} + 1 = (\mathtt{b} - 1)^2 \land (\mathtt{b} - 1) \geq 0 \rangle$

```
        a = a - 2 * b + 1;
```

$\langle \mathtt{a} = (\mathtt{b} - 1)^2 \land (\mathtt{b} - 1) \geq 0 \rangle$

```
        b = b - 1;
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \rangle$

```
    }
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \rangle$

```
}
```

$\langle \mathtt{a} = \mathtt{b}^2 \land \mathtt{b} \geq 0 \land \neg(\mathtt{b} > 0) \rangle$
$\langle \mathtt{a} = 0 \rangle$

b) Eine gültige Variante ist $V = \mathtt{b}$.

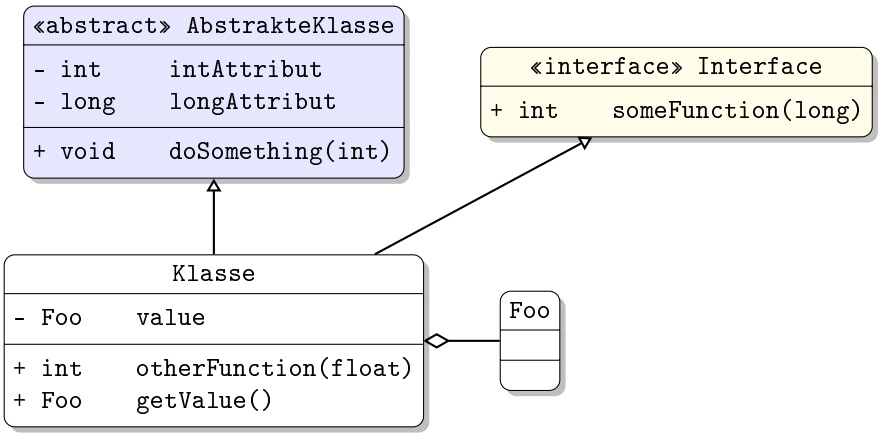
Aufgabe 3 (Klassenhierarchie): (11 + 8 = 19 Punkte)

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

- Ein Medium hat zwei Eigenschaften: Die *European Article Number*, kurz *EAN* und den herausgebenden Verlag. Die EAN kann führende Nullen enthalten und deshalb nicht als Zahl gespeichert werden.
- Ein analoges Medium ist ein Medium, für das das Gewicht und das Volumen von Interesse ist. Beide Werte sollen mit Nachkommastellen gespeichert werden.
- Ein Spiel ist ein analoges Medium. Es hat eine Dauer (in Minuten) und ein Mindestalter (in Jahren). Beide Werte sollen als ganze Zahlen gespeichert werden.
- Ein Buch ist ein analoges Medium. Es hat eine ganzzahlige Seitenzahl.
- Eine CD ist ein (nicht analoges) Medium. Sie hat ein Datenvolumen in GB, das eine Gleitkommazahl ist. Es ist interessant, ob es sich dabei um eine Musik-CD handelt oder nicht. Außerdem liegt einer CD eine Reihe von Handbüchern bei, die wiederum Bücher sind.
- Ein E-Book ist ein (nicht analoges) Medium. Es bietet eine Methode `ausdrucken()`, die ein Buch zurückliefert.
- Ein Medium ist stets ein Spiel, Buch, E-Book oder eine CD. Es soll nicht möglich sein, ein Medium zu erzeugen, das keine Instanz einer dieser vier Klassen ist.
- Spiele, Bücher und E-Books sind *erfunden*. Sie bieten alle eine Methode `istEinzelwerk()`, die zurückgibt, ob das Spiel, Buch oder E-Book nur einen oder mehrere Autoren hat.
- Eine Produktfamilie hat ein Array von Medien. (Dies sind die Produkte, die zur Produktfamilie gehören). Außerdem kann man mit der Methode `add(Medium... medien)` der Familie die übergebenen `medien` hinzufügen.

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 außerhalb der Klasse `Produktfamilie` eine Java-Methode `erzeugeUnterfamilie`, die eine Produktfamilie als Eingabeparameter bekommt. Der Aufruf `erzeugeUnterfamilie(f)` soll eine neue Produktfamilie zurückliefern, die aus all denjenigen *erfundenen* Medien in der Familie `f` besteht, die *Einzelwerke* sind.

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 einem Atrribut `int x` hätte also beispielsweise einen Konstruktor `A(int x)`.

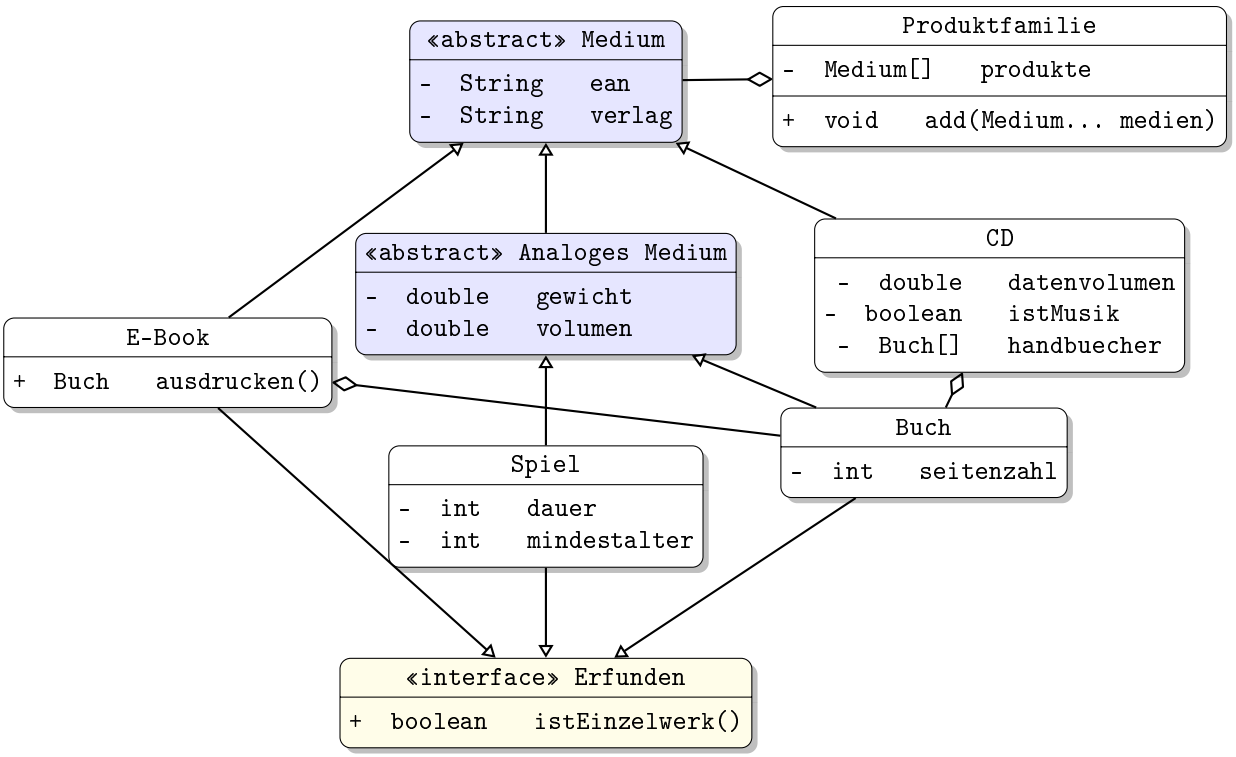
Hinweise:

- Gehen Sie davon aus, dass alle vorkommenden Arrays weder `null` sind noch `null` enthalten und dass außerdem `f` nicht `null` ist.
- Sie dürfen Hilfsmethoden schreiben.

```
public static Produktfamilie erzeugeUnterfamilie(Produktfamilie f) {
```

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)
```
public static Produktfamilie erzeugeUnterfamilie(Produktfamilie f) {
  Medium[] keineMedien = new Medium[]{};
  Produktfamilie result = new Produktfamilie(keineMedien);
  for (Medium m : f.getProdukte()) {
    if (m instanceof Erfunden) {
      if ((Erfunden)m.istEinzelwerk()) {
        result.add(m);
      }
    }
```

```
  }
  return result;
}
```

Alternative Lösung, in der die nicht-gewünschten Elemente auf `null` gesetzt werden:

```
public static Produktfamilie erzeugeUnterfamilie2(Produktfamilie f) {
  Medium[] neueMedien = new Medium[f.getProdukte().length];
  for (int i = 0; i < neueMedien.length; ++i) {
    Medium ac = f.getProdukte()[i];
    if (ac instanceof Erfunden && ((Erfunden)ac).istEinzelwerk()) {
      neueMedien[i] = ac;
    } else {
      neueMedien[i] = null;
    }
  }
  Produktfamilie result = new Produktfamilie(neueMedien);
  return result;
}
```

Aufgabe 4 (Programmierung in Java): (4 + 5 + 5 + 5.5 + 7.5 + 5 = 32 Punkte)

In dieser Aufgabe wollen wir ein Programm für eine Busgesellschaft schreiben, welches Informationen über eine Buslinie und deren Haltestellen verwalten kann. Dazu nutzen wir zwei Klassen: `Buslinie` und `Haltestelle`. Ein `Haltestelle`-Objekt hat eine Referenz auf die nächste Haltestelle (ihren *Nachfolger*) im Attribut `next` und es enthält den *Namen* der Haltestelle im Attribut `name`. Außerdem enthält es im Attribut `traffic` die Anzahl der Fahrgäste, welche im letzten Monat an dieser Haltestelle ein- oder ausgestiegen sind.

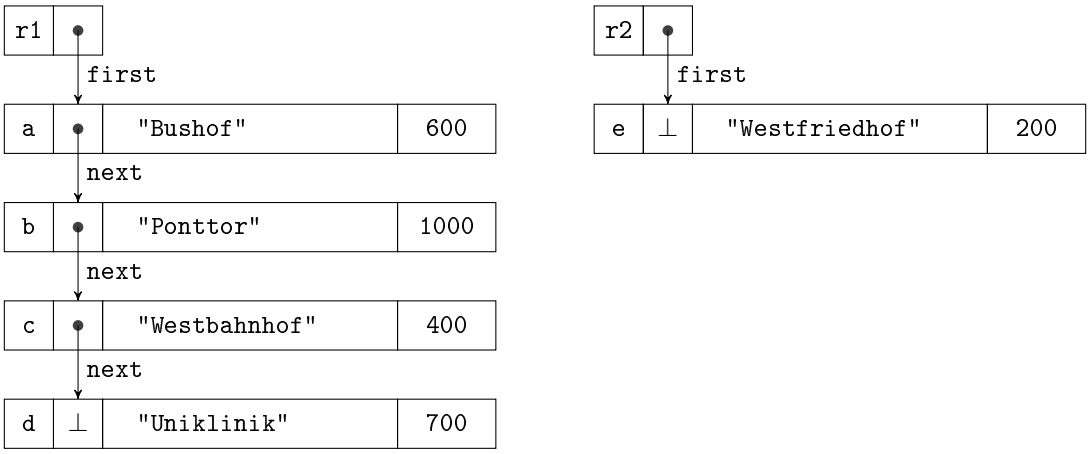
```
public class Haltestelle {
  private Haltestelle next;
  private final String name;
  private final int traffic;

  public Haltestelle(Haltestelle next, String name, int traffic) {
    this.next = next;
    this.name = name;
    this.traffic = traffic;
  }

  public Haltestelle getNext() {
    return next;
  }

  public String getName() {
    return name;
  }

  public int getTraffic() {
    return traffic;
  }

  public void setNext(Haltestelle next) {
    this.next = next;
  }
}
```

Ein `Buslinie`-Objekt hat im Attribut `first` eine Referenz auf die erste Haltestelle dieser Buslinie.

```
public class Buslinie {
  private Haltestelle first;
}
```

Beispiel:

In der folgenden Grafik sind beispielhaft die Objekte `r1` und `r2` (vom Typ `Buslinie`) sowie die Objekte `a` bis `e` (vom Typ `Haltestelle`) dargestellt. Die Bezeichner `r1`, `r2` sowie `a` bis `e` 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 sieben Objekte, dargestellt durch je zwei bzw. vier nebeneinander liegende Rechtecke. Das linke Rechteck enthält dabei immer einen Bezeichner. Die Objekte `r1` und `r2` sind `Buslinie`-Objekte, deren `first`-Attribut auf das Objekt `a` bzw. `e` zeigt. Die Objekte `a` bis `c` sind `Haltestelle`-Objekte, deren `next`-Attribut je auf ein anderes `Haltestelle`-Objekt zeigt. Die Objekte `d` und `e` sind `Haltestelle`-Objekte, deren `next`-Attribut den Wert `null` (⊥) hat. Außerdem enthält jedes `Haltestelle`-Objekt im dritten Rechteck den Namen der Haltestelle und im vierten Rechteck den `traffic`-Wert der Haltestelle.

Hinweise:

- Sie dürfen in allen Teilaufgaben davon ausgehen, dass nur auf azyklischen Buslinien gearbeitet wird (d.h. man erreicht keinen Zyklus, wenn man nur den `next`-Referenzen folgt).
- Sie dürfen in allen Teilaufgaben davon ausgehen, dass innerhalb einer Buslinie verschiedene Haltestellen verschiedene `traffic`-Werte haben.
- Sie dürfen in allen Teilaufgaben davon ausgehen, dass das `first`-Attribut jedes `Buslinie`-Objekts nicht den Wert `null` hat.
- Sie dürfen in allen Teilaufgaben davon ausgehen, dass das `name`-Attribut jedes `Haltestelle`-Objekts nicht den Wert `null` hat.
- 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 `Haltestelle` die Methode `findStationWithMaxTraffic`. Beginnend bei der aktuellen Haltestelle gibt sie die Haltestelle mit dem höchsten `traffic`-Wert zurück. Sie gibt also entweder die aktuelle Haltestelle zurück oder, falls später auf der Buslinie eine Haltestelle mit einem höheren `traffic`-Wert liegt, die Haltestelle mit dem höchsten `traffic`-Wert auf der verbleibenden Buslinie.

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

Beispiele:

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

```
public Haltestelle findStationWithMaxTraffic() {
```

b) Die Busgesellschaft möchte die Haltestelle mit dem höchsten `traffic`-Wert entlasten.

Implementieren Sie in der Klasse `Buslinie` die Methode `createStationAfterStationWithMaxTraffic`. Sie findet zunächst in der aktuellen Buslinie die Haltestelle mit dem höchsten `traffic`-Wert. Dann fügt sie zwischen der gefundenen Haltestelle und ihrem Nachfolger eine neue Haltestelle ein. Der Name für die neue Haltestelle wird als Eingabeparameter übergeben. Der `traffic`-Wert der neuen Haltestelle ist 0.

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

Beispiele:

- Wenn beispielsweise

  ```
  r1.createStationAfterStationWithMaxTraffic("Audimax");
  ```

  aufgerufen wird, so soll ein neues `Haltestelle`-Objekt `k` mit Namen `"Audimax"` und `traffic`-Wert 0 erstellt werden, dessen Nachfolger `c` ist. Außerdem soll der Nachfolger des Objekts `b` auf `k` gesetzt werden.
- Wenn beispielsweise

  ```
  r2.createStationAfterStationWithMaxTraffic("Audimax");
  ```

  aufgerufen wird, so soll ein neues `Haltestelle`-Objekt `k` mit Namen `"Audimax"` und `traffic`-Wert 0 erstellt werden, welches keinen Nachfolger hat. Außerdem soll der Nachfolger des Objekts `e` auf `k` gesetzt werden.

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `name` der Methode `createStationAfterStationWithMaxTraffic` nicht `null` ist.

```
public void createStationAfterStationWithMaxTraffic(String name) {
```

c) Implementieren Sie in der Klasse `Buslinie` die Methode `getTrafficOf`. Der Aufruf `getTrafficOf(n)` findet zunächst die erste Haltestelle mit dem Namen `n` in der aktuellen Buslinie. Anschließend wird der `traffic`-Wert dieser Haltestelle zurückgegeben. Falls die Buslinie keine Haltestelle mit dem übergebenen Namen `n` enthält, so soll eine `UnknownHaltestelleException` geworfen werden, welche wie folgt definiert ist:

```
public class UnknownHaltestelleException extends Exception {}
```

Beispiele:

- Wenn beispielsweise `r1.getTrafficOf("Westbahnhof")` aufgerufen wird, so soll 400 zurückgegeben werden.
- Wenn beispielsweise `r1.getTrafficOf("Ponttor")` aufgerufen wird, so soll 1000 zurückgegeben werden.
- Wenn beispielsweise `r1.getTrafficOf("Aachen Hbf")` aufgerufen wird, so soll eine `UnknownHaltestelleException` geworfen werden.

Hinweise:

- Sie dürfen davon ausgehen, dass der Parameter `name` der Methode `getTrafficOf` nicht `null` ist.

```
public int getTrafficOf(String name) throws UnknownHaltestelleException {
```

d) Implementieren Sie in der Klasse `Haltestelle` die Methode `findStationWithMinTraffic`. Beginnend bei der aktuellen Haltestelle gibt sie die Haltestelle mit dem niedrigsten `traffic`-Wert zurück. Sie gibt also entweder die aktuelle Haltestelle zurück oder, falls später auf der Buslinie eine Haltestelle mit einem niedrigeren `traffic`-Wert liegt, die Haltestelle mit dem niedrigsten `traffic`-Wert auf der verbleibenden Buslinie.

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

Beispiele:

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

Hinweise:

- Sie dürfen weitere private Hilfsmethoden schreiben.

```
public Haltestelle findStationWithMinTraffic() {
```

e) Die Busgesellschaft muss sparen und möchte deswegen die Haltestelle mit dem niedrigsten `traffic`-Wert schließen.

Implementieren Sie in der Klasse `Buslinie` die Methode `removeStationWithMinTraffic`, welche die Haltestelle mit dem niedrigsten `traffic`-Wert aus der Buslinie entfernt. Dabei sollen alle Haltestellen nach der zu entfernenden Haltestelle weiterhin Teil der Buslinie sein.

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

Beispiel:

Wenn beispielsweise `r1.removeStationWithMinTraffic()` aufgerufen wird, so soll `c` aus der Buslinie `r1` entfernt werden, sodass `r1` anschließend aus den Haltestellen `a`, `b` und `d` besteht. Der Nachfolger von `b` muss also auf `d` gesetzt werden.

```
public void removeStationWithMinTraffic() {
```

f) Implementieren Sie in der Klasse `Buslinie` die Methode `toList`, welche die aktuelle Buslinie in eine `Liste` von `String`s umwandelt. Achten Sie dabei darauf, die `String`-Darstellung der Haltestellen 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 `r1.toList()` aufgerufen wird, so soll eine Liste mit den folgenden vier `String`s zurückgegeben werden:
  - `"Bushof (600)"`
  - `"Ponttor (1000)"`
  - `"Westbahnhof (400)"`
  - `"Uniklinik (700)"`
- Wenn beispielsweise `r2.toList()` aufgerufen wird, so soll eine Liste mit dem folgenden `String` zurückgegeben werden:
  - `"Westfriedhof (200)"`

Hinweise:

- `LinkedList<T>` und `List<T>` bezeichnen Typen, welche sich im Paket `java.util` befinden. Gehen Sie daher davon aus, dass die Datei `Buslinie.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 Haltestelle findStationWithMaxTraffic() {
  Haltestelle max = this;
  Haltestelle current = next;
  while (current != null) {
    if (max.traffic < current.traffic) {
      max = current;
    }
    current = current.next;
  }
  return max;
}
```

b)
```
public void createStationAfterStationWithMaxTraffic(String name) {
  Haltestelle max = first.findStationWithMaxTraffic();
  max.setNext(new Haltestelle(max.getNext(), name, 0));
}
```

c)
```
public int getTrafficOf(String name) throws UnknownHaltestelleException {
  Haltestelle current = first;
  while (current != null) {
    if (current.getName().equals(name)) {
```

```
      return current.getTraffic();
    }
    current = current.getNext();
  }
  throw new UnknownHaltestelleException();
}
```

d)

```
public Haltestelle findStationWithMinTraffic() {
  return doFindStationWithMinTraffic(this);
}

private Haltestelle doFindStationWithMinTraffic(Haltestelle currentMin) {
  Haltestelle nextMin = this.traffic < currentMin.traffic ? this : currentMin;
  if (next != null) {
    return next.doFindStationWithMinTraffic(nextMin);
  } else {
    return nextMin;
  }
}
```

e)

```
public void removeStationWithMinTraffic() {
  Haltestelle min = first.findStationWithMinTraffic();
  if (min == first) {
    first = first.getNext();
  } else {
    Haltestelle current = first;
    while (current != null) {
      if (min == current.getNext()) {
        current.setNext(current.getNext().getNext());
      }
      current = current.getNext();
    }
  }
}
```

Alternative Lösung ohne Aufruf von `findStationWithMinTraffic`

```
public void removeStationWithMinTraffic() {
  Haltestelle minPredecessor = null;
  Haltestelle min = first;
  Haltestelle predecessor = first;
  while (predecessor.getNext() != null) {
    if (min.getTraffic() < predecessor.getNext().getTraffic()) {
      minPredecessor = predecessor;
      min = predecessor.getNext();
    }
    predecessor = predecessor.getNext();
  }
  if (minPredecessor == null) { // min == first
    first = first.getNext();
  } else {
    minPredecessor.setNext(min.getNext());
```

```
    }
  }
```

f)
```
public List<String> toList() {
  List<String> result = new LinkedList<>();
  Haltestelle current = first;
  while (current != null) {
    result.add(current.getName() + " (" + current.getTraffic() + ")");
    current = current.getNext();
  }
  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 z | f z y x = f x (x:z:y) z
        | otherwise = f (f x y x) y x

g [x] _ = (g [] []) ++ ([x] : [])
g (x:xs) y = g (2*x : xs) y
```

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

```
i :: [Int]
i = take 4 (let x = 1 : map (+2) x in x)
```

Hinweise:

- Die vordefinierte Funktion `take:: Int -> [a] -> [a]` liefert die ersten n Elemente der übergebenen Liste zurück. Beispielsweise ist `take 5 [1..10] = [1,2,3,4,5]`.

```
j :: Int
j = (\(x,y) -> x (x y)) ((2+),1)
```

c) Schreiben Sie eine Haskell–Funktion `removeLast :: [a] -> [a]`. Die von `removeLast` zurückgegebene Liste soll bis auf das letzte Element alle Elemente der Eingabeliste in derselben Reihenfolge enthalten. Der Ausdruck `removeLast [1,2,3]` soll also zu `[1,2]` evaluieren, der Ausdruck `removeLast []` zu `[]`.

Hinweise:

- In dieser Aufgabe dürfen Sie keine vordefinierten Funktionen (außer Datenkonstruktoren) verwenden.

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(a) durch folgenden Haskell–Code dargestellt werden:

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

Die Cyclist `c2` aus Abb. 1(b) kann durch folgenden Haskell–Code dargestellt werden:

```
c2 :: Cyclist Int
let lastElement = Element 3 lastElement in Element 1 (Element 2 lastElement)
```

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

(a) Cyclist `c1` des Typs `Cyclist Int`

(b) Cyclist `c2` des Typs `Cyclist Int`

Abbildung 1: Zwei beispielhafte Cyclists

Sie dürfen im Folgenden stets davon ausgehen, dass in einer Liste nur *unterschiedliche* Werte des Typs `a` vorkommen. Beachten Sie, dass sich mit dem Datentyp `Cyclist a` keine leere Liste darstellen lässt.

Schreiben Sie unter Nutzung dieses Datentyps eine Haskell–Funktion `toCyclist :: [a] -> Cyclist a`, die aus einer Haskell–Liste `[x1,x2,...,xn]` die Cyclist in Abb. 2 erzeugt. Die zurückgegebene Cyclist soll also alle Elemente der Eingabeliste in der gleichen Reihenfolge enthalten. Der Nachfolger des letzten Elements soll wiederum das letzte Element sein. Auf einer leeren Eingabeliste darf sich die Funktion `toCyclist` beliebig verhalten.

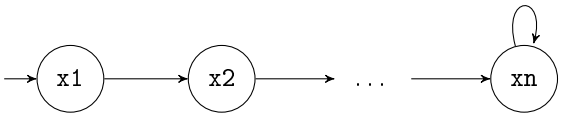


Abbildung 2: Allgemeine Form der Ausgabe-Cyclist der Funktion `toCyclist`

e) Schreiben Sie eine Haskell–Funktion `toList :: Cyclist Int -> [Int]`, die aus einer Eingabe-Cyclist `c` eine Haskell–Liste `xs` erzeugt, sodass `xs` jeden Eintrag aus `c` enthält. Die Auflistung soll den Nachfolgern so lange folgen, bis sich eine Wiederholung ergibt, und diese als Letztes angeben. Die Sortierung der Elemente soll beibehalten werden. So ergibt der Aufruf `toList c1` die Liste `[1,2,3,1]` und der Aufruf `toList c2` die Liste `[1,2,3,3]`.

Für diese und die folgende Teilaufgabe gelten die folgenden Hinweise:

Hinweise:

- Gehen Sie davon aus, dass alle Funktionen nur auf Cyclists mit endlich vielen Elementen aufgerufen werden.
- Sie dürfen beliebige vordefinierte Funktionen aus der Standardbibliothek verwenden, wie die vordefinierte Funktion `elem :: Int -> [Int] -> Bool`, die prüft, ob die übergebene Zahl in der übergebenen Liste vom Typ `[Int]` enthalten ist. Außerdem dürfen Sie Hilfsfunktionen schreiben.

f) Schreiben Sie eine Haskell–Funktion `normalize :: Cyclist Int -> Cyclist Int`. Die Rückgabe des Ausdrucks `normalize c` soll eine Cyclist `d` mit denselben Werten wie in `c` sein, aber das letzte Element von `d` soll sein eigener Nachfolger sein. Beim Aufruf von `toList c` und `toList d` erhält man also Listen, die bis auf das letzte Element gleich sind. Der letzte und vorletzte Wert in `toList d` sind identisch. Beispielweise ergibt `normalize c1` die Cyclist `c2`.

Hinweise:

- Sie können `removeLast`, `toCyclist` und `toList` verwenden, unabhängig davon, ob Sie diese implementiert haben.

Lösung:

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

b)
```
i = [1,3,5,7]
j = 5
```

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

d)
```
toCyclist :: [a] -> Cyclist a
toCyclist [x] = let xs = Element x xs in xs
toCyclist (x:xs) = Element x (toCyclist xs)
```

e)
```
toList :: Cyclist Int -> [Int]
toList c = toListH [] c

toListH :: [Int] -> Cyclist Int -> [Int]
toListH xs (Element x c) | elem x xs = [x]
                         | otherwise = x : toListH (x:xs) c
```

f)
```
normalize :: Cyclist Int -> Cyclist Int
normalize c = toCyclist (removeLast (toList c))
```

Aufgabe 6 (Prolog): (2 + 9 + (1 + 2 + 4 + 2) = 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) $f(X, s(Y), X), f(s(Z), Z, s(s(a)))$

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

b) Gegeben sei folgendes Prolog-Programm P.

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

- Erstellen Sie für das Programm P den Beweisbaum zur Anfrage "`?- q(s(0),R).`" 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 "`?- q(s(0),R).`" 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 geht es darum, einen Sortieralgorithmus für Listen von natürlichen Zahlen in Prolog zu implementieren. Wir benutzen die vordefinierten Listen aus Prolog. Für natürliche Zahlen benutzen wir die aus der Vorlesung bekannte Darstellung mit den Funktionssymbolen `0` und `s`.

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.

i) Implementieren Sie ein Prädikat `leq` mit Stelligkeit 2 in Prolog. Wenn `leq` auf zwei natürliche Zahlen in der oben genannten Darstellung angewendet wird, soll es genau dann wahr sein, wenn das erste Argument kleiner oder gleich dem zweiten Argument ist. Beispielsweise soll also `leq(0,0)` wahr sein, wohingegen `leq(s(0),0)` falsch ist.

ii) Implementieren Sie ein Prädikat `split` mit Stelligkeit 3 in Prolog, das genau dann wahr ist, wenn das zweite Argument die Liste der Elemente ist, die an *geraden* Indizes des ersten Arguments stehen und das dritte Argument die Liste der Elemente ist, die an *ungeraden* Indizes stehen. Hierbei zählen wir die Elemente einer Liste ab dem Index `0`.

Also soll `split([s(0),0,s(s(0)),s(0)],XS,YS)` genau dann wahr sein, wenn `XS = [s(0),s(s(0))]` und `YS = [0,s(0)]` gilt, denn das erste `s(0)` hat den Index 0, `0` hat den Index 1, `s(s(0))` hat den Index 2 und das zweite `s(0)` hat den Index 3 in der ursprünglichen Liste.

Sie können davon ausgehen, dass bereits die folgenden Fakten gegeben sind:

```
split([],[],[]).
split([X],[X],[]).
```

iii) Implementieren Sie ein Prädikat `merge` mit Stelligkeit 3 in Prolog. Für Listen `XS`, `YS` und `ZS` soll `merge(XS,YS,ZS)` genau dann wahr sein, wenn `ZS` die sortierte Verkettung von `XS` und `YS` ist. Sie können davon ausgehen, dass die ersten beiden Argumente stets aufsteigend sortierte Listen sind. So ist also z.B. `merge([s(0),s(s(0))],[0,s(0)],ZS)` genau dann wahr, wenn `ZS = [0,s(0),s(0),s(s(0))]` ist.

Sie können die beiden folgenden Fakten als gegeben ansehen:

```
merge([],YS,YS).
merge(XS,[],XS).
```

iv) Implementieren Sie ein Prädikat `sort` mit Stelligkeit 2 in Prolog. Hierbei soll `sort(XS,YS)` genau dann wahr sein, wenn sowohl `split(XS,AS,BS)` als auch `merge(SAS,SBS,YS)` wahr ist, wobei `SAS` bzw. `SBS` die Listen sind, die entstehen, wenn `AS` bzw. `BS` sortiert werden.

So soll also `sort([s(0),0,s(s(0)),s(0)],XS)` genau dann wahr sein, wenn `XS = [0,s(0),s(0),s(s(0))]` gilt.

Sie können die beiden folgenden Fakten als gegeben ansehen:

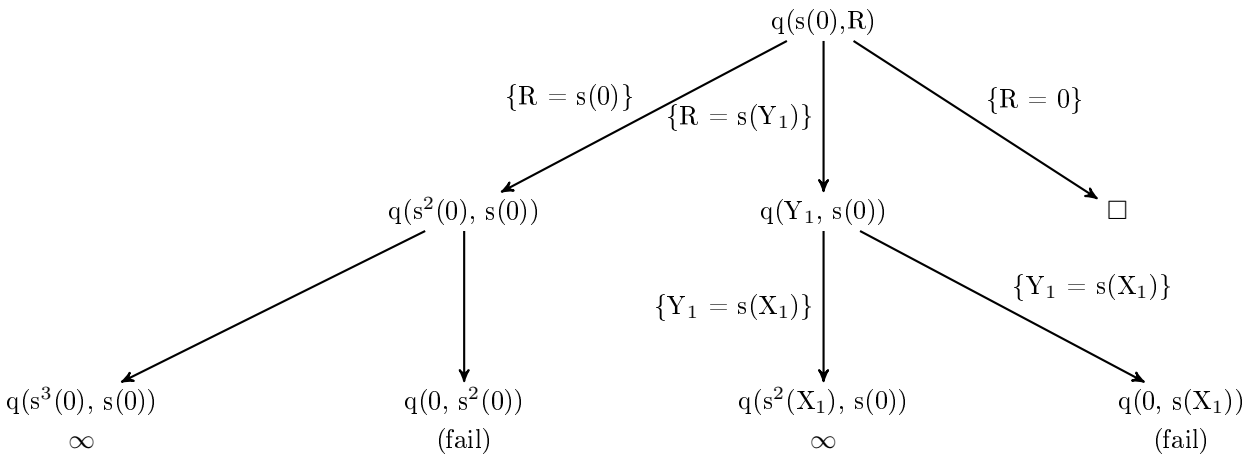
```
sort([],[]).
sort([X],[X]).
```

Lösung:

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

ii) $g(X, s(X), Y), g(s(Z), Y, X)$ führt zu einem Occur Failure, da nach der Unifikation der ersten beiden Argumente das dritte Argument $s(Z)$ bzw. $s(s(Z))$ ist.

b)



Die einzige Antwortsubstitution innerhalb des Beweisbaums ist $\{R = 0\}$. Diese wird von Prolog nicht gefunden.

Die dritte Regel muss an den Anfang des Programms verschoben werden, damit Prolog die Antwortsubstitution bis zur Höhe 3 findet:

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

c)

```
% i)
leq(0,_).
leq(s(X),s(Y)) :- leq(X,Y).

% ii)
split([],[],[]).
split([X],[X],[]).
split([X,Y|ZS], [X|XS], [Y|YS]) :- split(ZS,XS,YS).
```

```
% iii)
merge([],YS,YS).
merge(XS,[],XS).
merge([X|XS],[Y|YS],[X|ZS]) :- leq(X,Y), merge(XS,[Y|YS],ZS).
merge([X|XS],[Y|YS],[Y|ZS]) :- leq(Y,X), merge([X|XS],YS,ZS).

% iv)
sort([],[]).
sort([X],[X]).
sort([X|XS],YS) :- split([X|XS],AS,BS), sort(AS,SAS), sort(BS,SBS),
                   merge(SAS,SBS,YS).
```