Prof. Dr. J. Giesl
C. Aschermann, F. Frohn, J. Hensel, T. Ströder

Aufgabe 1 (Programmanalyse): (14 + 6 = 20 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 {

    public int x = 2;

    public A() {
        this.x++;
    }

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

    public void f(double x) {
        this.x = (int) (x + B.y);
    }
}
```

```
public class B extends A {

    public static double y = 3;

    public double x = 0;

    public B(double x) {
        y++;
    }

    public void f(int y) {
        this.x = y * 2;
        B.y = 0;
    }

    public void f(double y) {
        this.x = 2 * y + B.y;
    }
}
```

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

        B b = new B(2);
        System.out.println(b.x + " " + B.y);    // OUT: [    ]    [    ]

        A z = b;
        System.out.println(z.x);                // OUT: [    ]

        z.f(-5.0);
        System.out.println(b.x + " " + z.x);    // OUT: [    ]    [    ]

        z.f(-6);
        System.out.println(b.x + " " + B.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  import java.util.*;
2
3  public class C extends B {
4
5      private int z = 42;
6
7      public void f(LinkedList<Integer> integers) {
8          integers.addFirst(B.z);
9      }
10
11     public static void main(String[] args) {
12         C c = new C();
13         List<Integer> list = new LinkedList<>();
14         c.f(list);
15     }
16 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
        A a = new A((int) B.y);
        System.out.println(a.x);                // OUT: [ 5 ]

        B b = new B(2);
        System.out.println(b.x + " " + B.y);    // OUT: [0.0]    [4.0]

        A z = b;
        System.out.println(z.x);                // OUT: [ 3 ]

        z.f(-5.0);
        System.out.println(b.x + " " + z.x);    // OUT: [-6.0]   [ 3 ]

        z.f(-6);
        System.out.println(b.x + " " + B.y);    // OUT: [-8.0]   [4.0]
    }
}
```

b)

- In der Klasse `B` gibt es keinen Konstruktor ohne Parameter. Deshalb muss in der Klasse `C` ein Konstruktor definiert sein, der explizit einen anderen Konstruktor von `B` aufruft.
- `B.z`: `z` ist kein statisches Attribut der Klasse `B`.
- `c.f(list)`: Die Methode `f(LinkedList<Integer>)` der Klasse `C` ist ohne expliziten Cast nicht anwendbar für das Argument `list` vom Typ `List<Integer>`.

Aufgabe 2 (Hoare-Kalkül): (15 + 3 = 18 Punkte)

Gegeben sei folgendes Java-Programm P, das zu zwei Eingaben $\texttt{x} = a$ und $\texttt{y} = b$ den Wert $|a - b|$ berechnet.

$\langle\, \texttt{x} = a \wedge \texttt{y} = b \,\rangle$ (Vorbedingung)

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

$\langle\, \texttt{res} = |a - b| \,\rangle$ (Nachbedingung)

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.

Hinweise:

- Für alle $x, y \in \mathbb{Z}$ gilt:
$$x > y \implies |(x - 1) - y| = |x - y| - 1$$
und
$$x \neq y \wedge \neg(x > y) \implies |x - (y - 1)| = |x - y| - 1.$$
Achtung: Falls lediglich $x \geq y$ bzw. $x \leq y$ gilt, gelten beide Gleichungen **nicht** im Allgemeinen.
- 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. Klammern dürfen und müssen Sie jedoch eventuell bei der Anwendung der Zuweisungsregel setzen.
- Es empfiehlt sich oft, bei der Erstellung der Zusicherungen in der Schleife von unten (d. h. von der Nachbedingung aus) vorzugehen.

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)

$\langle \texttt{x} = a \wedge \texttt{y} = b \rangle$
$\langle \texttt{x} = a \wedge \texttt{y} = b \wedge 0 = 0 \rangle$

```
res = 0;
```

$\langle \texttt{x} = a \wedge \texttt{y} = b \wedge \texttt{res} = 0 \rangle$
$\langle \texttt{res} = |a - b| - |\texttt{x} - \texttt{y}| \rangle$

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

$\langle \texttt{res} = |a - b| - |\texttt{x} - \texttt{y}| \wedge \texttt{x} \neq \texttt{y} \rangle$
$\langle \texttt{res} + 1 = |a - b| - (|\texttt{x} - \texttt{y}| - 1) \wedge \texttt{x} \neq \texttt{y} \rangle$

```
    if (x > y) {
```
$\langle \mathtt{res} + 1 = |a - b| - (|\mathtt{x} - \mathtt{y}| - 1) \wedge \mathtt{x} \neq \mathtt{y} \wedge \mathtt{x} > \mathtt{y} \rangle$
$\langle \mathtt{res} + 1 = |a - b| - |(\mathtt{x} - 1) - \mathtt{y}| \rangle$
```
        x = x - 1;
```
$\langle \mathtt{res} + 1 = |a - b| - |\mathtt{x} - \mathtt{y}| \rangle$
```
    } else {
```
$\langle \mathtt{res} + 1 = |a - b| - (|\mathtt{x} - \mathtt{y}| - 1) \wedge \mathtt{x} \neq \mathtt{y} \wedge \neg(\mathtt{x} > \mathtt{y}) \rangle$
$\langle \mathtt{res} + 1 = |a - b| - |\mathtt{x} - (\mathtt{y} - 1)| \rangle$
```
        y = y - 1;
```
$\langle \mathtt{res} + 1 = |a - b| - |\mathtt{x} - \mathtt{y}| \rangle$
```
    }
```
$\langle \mathtt{res} + 1 = |a - b| - |\mathtt{x} - \mathtt{y}| \rangle$
```
    res = res + 1;
```
$\langle \mathtt{res} = |a - b| - |\mathtt{x} - \mathtt{y}| \rangle$
```
}
```
$\langle \mathtt{res} = |a - b| - |\mathtt{x} - \mathtt{y}| \wedge \mathtt{x} = \mathtt{y} \rangle$
$\langle \mathtt{res} = |a - b| \rangle$

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

Der folgende Teil dieser Lösung ist nicht gefordert worden, aber wir geben ihn dennoch zu Übungszwecken an.

Die Schleifenbedingung $B = \mathtt{x} \neq \mathtt{y}$ impliziert $|\mathtt{x} - \mathtt{y}| > 0$ und damit $V \geq 0$. Es gilt:

$\langle |\mathtt{x} - \mathtt{y}| = m \wedge \mathtt{x} \neq \mathtt{y} \rangle$
$\langle |\mathtt{x} - \mathtt{y}| - 1 < m \wedge \mathtt{x} \neq \mathtt{y} \rangle$
```
if (x > y) {
```
$\langle |\mathtt{x} - \mathtt{y}| - 1 < m \wedge \mathtt{x} \neq \mathtt{y} \wedge \mathtt{x} > \mathtt{y} \rangle$
$\langle |(\mathtt{x} - 1) - \mathtt{y}| < m \rangle$
```
    x = x - 1;
```
$\langle |\mathtt{x} - \mathtt{y}| < m \rangle$
```
} else {
```
$\langle |\mathtt{x} - \mathtt{y}| - 1 < m \wedge \mathtt{x} \neq \mathtt{y} \wedge \neg(\mathtt{x} > \mathtt{y}) \rangle$
$\langle |\mathtt{x} - (\mathtt{y} - 1)| < m \rangle$
```
    y = y - 1;
```
$\langle |\mathtt{x} - \mathtt{y}| < m \rangle$
```
}
```
$\langle |\mathtt{x} - \mathtt{y}| < m \rangle$
```
res = res + 1;
```
$\langle |\mathtt{x} - \mathtt{y}| < m \rangle$

Damit ist die Terminierung der einzigen Schleife in P gezeigt.

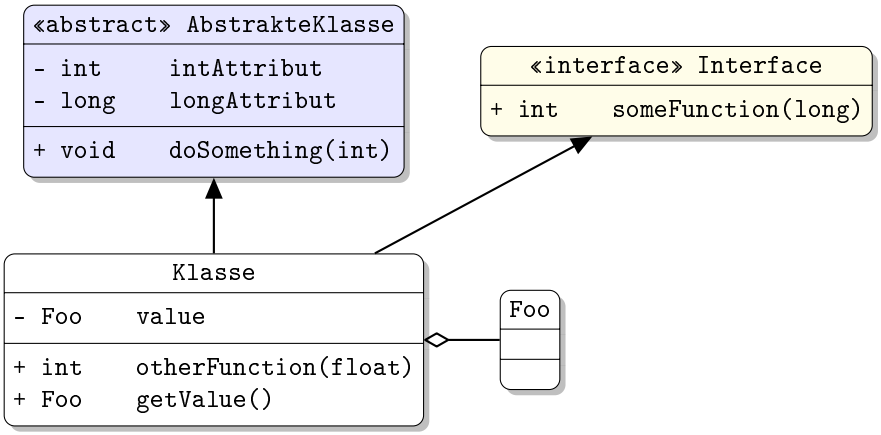
Aufgabe 3 (Klassenhierarchie): (7 + 9 = 16 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Hierarchie zur Verwaltung von "Häusern, die Verrückte machen" (HDVM)[1].

- Ein HDVM enthält eine Reihe (d.h. ein Array) von beliebig vielen Sachbearbeitern.
- Jeder Sachbearbeiter kann genau eine Sorte Formulare ausstellen. Dazu stellt er die Methode `Formular formularAustellen()` zur Verfügung.
- Jeder Sachbearbeiter kann zu diesem Zweck nach dem Namen des Antragstellers fragen. Dazu stellt er die Methode `String nameErfragen()` zur Verfügung.
- Bürodrachen sind Sachbearbeiter, die besonders laut und unfreundlich reden (dazu implementieren sie ebenfalls die Methode `String nameErfragen()`).
- Manche Formulare sind Passierscheine. In dem Fall ist die Nummer des Passierscheins von Interesse.
- Rundschreiben sind Formulare, in denen ein anderes (angehängtes) Formular beschrieben wird. Bei ihnen ist das angehängte Formular von Interesse.
- Alle Formulare haben eine offizielle Bezeichnung, welche in einem Attribut `String bezeichnung` gespeichert wird, sowie eine Get-Methode (d.h. einen Selektor) für dieses Attribut.
- Es gibt keine anderen Formulare als Passierscheine und Rundschreiben.

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 beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \blacktriangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Der Pfeil $B \diamond\!\!-\!\!\blacktriangleright A$ 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 mit der folgenden Signatur:

[1] vgl. Asterix erobert Rom

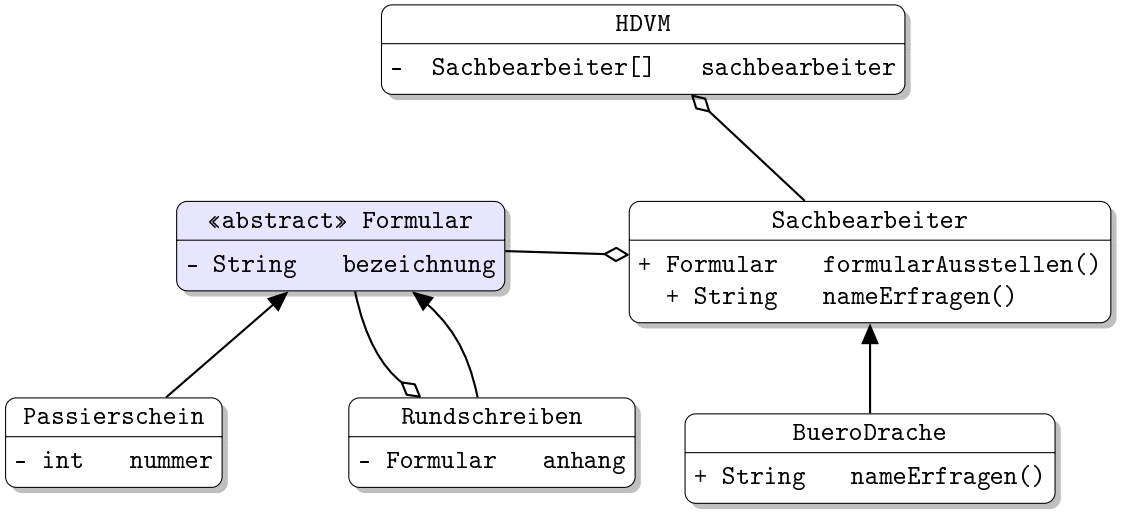
```
public static Sachbearbeiter zustaendigenFinden(
    Sachbearbeiter[] sachbearbeiterArray,
    String beschreibung
)
```

Diese Methode soll einen Sachbearbeiter aus dem Array `sachbearbeiterArray` zurückgeben, der Formulare mit der übergebenen Beschreibung `beschreibung` ausstellt. Ist kein solcher Sachbearbeiter vorhanden, soll `null` zurückgegeben werden. Wenn sowohl Bürodrachen als auch normale Sachbearbeiter ein solches Formular ausstellen, soll immer ein normaler Sachbearbeiter zurückgegeben werden (niemand möchte zu den Bürodrachen). Unter den dann zur Auswahl stehenden Sachbearbeitern soll immer der erste Sachbearbeiter im Array ausgegeben werden.

Nehmen Sie dazu an, dass das übergebene Array `sachbearbeiterArray` nicht `null` ist. Ein Sachbearbeiter selbst kann hingegen durchaus eine `null` sein.

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)
```
public static Sachbearbeiter zustaendigenFinden(
    Sachbearbeiter[] sachbearbeiterArray,
    String beschreibung
) {
    Sachbearbeiter erster_drache = null;
    for (Sachbearbeiter s : sachbearbeiterArray) {
        if (s == null) { continue; }
        if (s.formularAusstellen().getBezeichnung().equals(beschreibung)) {
            if (s instanceof BueroDrache) {
                if (erster_drache == null ) { erster_drache = s; }
              } else {
                  return s;
              }
         }
    }
    return erster_drache;
}
```

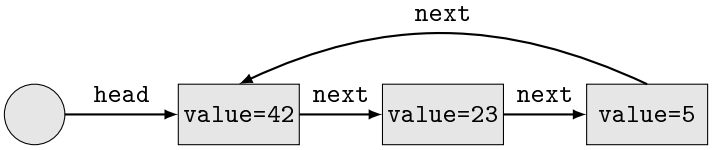
Aufgabe 4 (Listen in Java): (7 + 7 + 3 + 3 + 8 + 8 = 36 Punkte)

Die Klasse `List` dient zur Repräsentation von zyklischen Listen. Sie verfügt über ein Attribut `head`, welches auf das erste Listenelement verweist. Jedes Listenelement wird als Objekt der Klasse `Element` dargestellt und enthält einen Wert vom Typ `int` (Attribut `value`) und einen Verweis auf den direkten Nachfolger (Attribut `next`). Die Klassen `List` und `Element` sind im gleichen Paket.
Die leere Liste ist eine Instanz der Klasse `List`, deren Attribut `head` den Wert `null` hat. Da es sich um eine zyklische Liste handelt, ist der Nachfolger des letzten Listenelements das erste Listenelement, wenn die Liste nicht leer ist.

```
public class List {
  Element head = null;
}

public class Element {
  int value;
  Element next;
}
```

In der folgenden Grafik ist dargestellt, wie eine Liste mit den Werten `42, 23, 5` mit der Klasse `List` repräsentiert wird.



Objekte der Klasse `List` sind als Kreise und Objekte der Klasse `Element` sind als Rechtecke dargestellt.
Sie dürfen in allen Teilaufgaben beliebige Hilfsmethoden zu den Klassen `List` und `Element` hinzufügen. Wenn Sie von dieser Möglichkeit Gebrauch machen, müssen Sie eindeutig kennzeichnen, welche Hilfsmethoden zu welchen Klassen gehören.

a) Implementieren Sie die Methode `Element last()` in der Klasse `List`. Diese Methode gibt das letzte Element der Liste zurück. Wenn die Liste leer ist, wird `null` zurückgegeben. **Verwenden Sie zur Lösung dieser Teilaufgabe keine Rekursion!**

b) Implementieren Sie die Methode `void add(int i)` in der Klasse `List`. Diese Methode fügt den Wert `i` am Ende der Liste ein. Dabei soll keine neue Liste erzeugt werden. Stattdessen muss die aktuelle Liste entsprechend verändert werden. Selbstverständlich muss die Liste nach dem Ausführen der Methode `add` immer noch zyklisch sein.

c) Passen Sie die Deklarationen der Klassen `List` und `Element` 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 Liste gespeicherten Werte angibt.

Es ist nicht notwendig, die Methoden `last` oder `add` anzupassen!

d) Wir geben das folgende Interface vor:

```
public interface Updater<T> {
  T update(T value);
}
```

Schreiben Sie eine nicht-abstrakte Klasse `Doubler`, welche das Interface `Updater<Integer>` implementiert. Die Methode `update` in der Klasse `Doubler` soll für eine Zahl x als Eingabe die Zahl $2 \cdot x$ als Ergebnis zurückgeben.

e) Implementieren Sie die Methode `void apply(Updater<T> updater)` in der Klasse `List<T>`. Die Methode ersetzt die Werte der aktuellen Liste mit Hilfe des übergebenen `Updater`-Objekts. Hierbei wird jedes `value`-Attribut mit dem Wert überschrieben, den die `update`-Methode des Arguments zurückgibt, wenn man den ursprünglichen Wert des `value`-Attributs als Eingabe übergibt.

Wenn `a.apply(new Doubler())` für die Integer-Liste `a` mit den Werten `42, 23, 5` aufgerufen wird, enthält diese also anschließend die Werte `84, 46, 10`.

Gehen Sie davon aus, dass das `updater`-Argument nie `null` ist.

Verwenden Sie zur Lösung dieser Teilaufgabe keine Schleifen!

f) Dem Josephus-Problem liegt folgende Situation zugrunde: n Objekte werden im Kreis angeordnet. Dann wird reihum jedes x-te Objekt entfernt, wobei der Kreis jedes Mal wieder geschlossen wird, bis nur noch ein Objekt übrig bleibt. Dieses Objekt nennen wir das Josephus-Element.

Als Beispiel betrachten wir die Zahlen von 1 bis 5 und entfernen jedes zweite Objekt. Dann werden nacheinander die Zahlen 2, 4, 1 und 5 entfernt, sodass die Zahl 3 übrig bleibt und damit das Josephus-Element ist.

Schreiben Sie eine Methode `T josephus(int x)` in der Klasse `List<T>`, welche das Josephus-Element der aktuellen Liste zurückliefert, wenn jedes `x`-te Element entfernt wird. Die aktuelle Liste darf dabei verändert werden. Sie dürfen annehmen, dass die Liste nicht leer und $\texttt{x} > 0$ ist.

Lösung:

a)
```
Element last() {
    if (head == null) {
        return null;
    } else {
        Element cur = head;
        while (cur.next != head) {
            cur = cur.next;
        }
        return cur;
    }
}
```

b)
```
void add(int i) {
    Element toAdd = new Element();
    toAdd.value = i;
    if (head == null) {
        head = toAdd;
        head.next = head;
    } else {
        Element last = last();
        last.next = toAdd;
        toAdd.next = head;
    }
}
```

c)
```
public class List<T> {
    Element<T> head = null;
}

public class Element<T> {
    T value;
    Element<T> next;
}
```

d)

```
public class Doubler implements Updater<Integer> {
    public Integer update(Integer value) {
        return 2 * value;
    }
}
```

e)

```
// in Klasse List
void apply(Updater<T> updater) {
    if (head != null) {
        head.apply(updater, head);
    }
}

// in Klasse Element
void apply(Updater<T> updater, Element<T> head) {
    this.value = updater.update(value);
    if (this.next != head) {
        this.next.apply(updater, head);
    }
}
```

f)

```
T josephus(int x) {
    Element<T> current = last();
    while (current.next != current) {
        for (int i = 0; i < x - 1; i++) {
            current = current.next;
        }
        current.next = current.next.next;
    }
    return current.value;
}
```