Prof. Dr. J. Giesl
M. Hark, J. Hensel, D. Korzeniewski

Aufgabe 1 (Programmanalyse): (11.5 + 4.5 = 16 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. Achten Sie darauf, dass Gleitkommazahlen in der Form “5.0” und nicht als ”5” ausgegeben werden.

```
public class A {
  public static double x = 2;
  public int y;

  public A() {
    this.x++;
    this.y = 1;
  }

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

  public int f(long x) {
    return 2;
  }

  public int f(float x) {
    return 3;
  }
}
```

```
public class B extends A {
  public int x = 3;

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

  public int f(int x) {
    return 4;
  }

  public int f(long x) {
    return 5;
  }

  public int f(Float x) {
    return 7;
  }
}
```

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    public int z;
3    public C() {
4      z = 3;
5    }
6
7    private int f(long x) {
8      return 42;
9    }
10
11   public static int main(String[] args) {
12     Double x = 2.0;
13
14     A ab = new B();
15
16     int ret = ab.f(x);
17
18     for(Integer i = 0; i < 10; ++i) {
19       x += ab.f(i);
20     }
21     return ret;
22   }
23 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
      A aa = new A(1.5);
      System.out.println(aa.x + " " + aa.y);// OUT: [3.0] [ 1 ]

      int retAA = aa.f(1);
      System.out.println(retAA);            // OUT: [ 2 ]

      B bb = new B(6);
      A ab = bb;
      System.out.println(bb.x);             // OUT: [ 6 ]
      System.out.println(ab.x + " " + ab.y);// OUT: [9.0] [ 6 ]

      int retBB = bb.f(1f);
      System.out.println(retBB);            // OUT: [ 3 ]
      int retAB = ab.f(1);
      System.out.println(retAB);            // OUT: [ 5 ]

      retAB = ab.f(Float.valueOf(3.f));
      System.out.println(retAB);            // OUT: [ 3 ]

      retAB = bb.f(Integer.valueOf(1));
      System.out.println(retAB);            // OUT: [ 4 ]
    }
}
```

b)
- Die Zugriffsspezifikation darf beim Überschreiben von Methoden nicht stärker eingeschränkt sein. Daher ist `private` bei der Methode `f` nicht erlaubt.
- Die Klasse `B` hat keinen Konstruktor ohne Argumente, nur einen mit einem Argument. Daher führt der Aufruf von `new B()` zu einem Compilerfehler.
- Der Aufruf von `ab.f(x)` führt zu einem Fehler, da es in der Klasse `A` keine Methode `f` mit einem Argument von passendem Typ gibt. `Double` kann implizit weder zu `long` noch zu `float` umgewandelt werden.

Aufgabe 2 (Hoare-Kalkül): (13 + 3 = 16 Punkte)

Gegeben sei folgendes Java-Programm P, das zu zwei nicht–negativen Eingaben $\mathtt{x} = a$ und $\mathtt{y} = b$ den Wert $\max(0, a - b)$ berechnet.

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} \geq 0 \wedge \mathtt{x} = a \wedge \mathtt{y} = b\ \rangle$ (Vorbedingung)

```
while (y > 0) {
    if (x > 0) {
        x = x - 1;
    }
    y = y - 1;
}
```

$\langle\ \mathtt{x} = \max(0, a - b)\ \rangle$ (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.
- Beachten Sie, dass die Nachbedingung der `if`–Anweisung auch aus der Vorbedingung der `if`–Anweisung und der negierten `if`–Bedingung folgen muss. Sie müssen bei der Verwendung der Regel hierfür aber keinen expliziten Beweis angeben.

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) Beweisen Sie die Terminierung des Algorithmus P mit Hilfe des Hoare-Kalküls.

Lösung:

a)

$\langle \mathtt{x} \geq 0 \wedge \mathtt{y} \geq 0 \wedge \mathtt{x} = a \wedge \mathtt{y} = b \rangle$
$\langle \mathtt{x} = \max(0, a - b + \mathtt{y}) \wedge \mathtt{y} \geq 0 \rangle$

```
while (y > 0) {
```

$\langle \mathtt{x} = \max(0, a - b + \mathtt{y}) \wedge \mathtt{y} \geq 0 \wedge \mathtt{y} > 0 \rangle$
$\langle \mathtt{x} = \max(0, a - b + \mathtt{y}) \wedge \mathtt{y} > 0 \rangle$

```
    if (x > 0) {
```

$\langle \mathtt{x} = \max(0, a - b + \mathtt{y}) \wedge \mathtt{y} > 0 \wedge \mathtt{x} > 0 \rangle$
$\langle \mathtt{x} - 1 = \max(0, a - b + \mathtt{y} - 1) \wedge \mathtt{y} - 1 \geq 0 \rangle$

```
        x = x - 1;
```

$\langle \mathtt{x} = \max(0, a - b + \mathtt{y} - 1) \wedge \mathtt{y} - 1 \geq 0 \rangle$

```
    }
```

$\langle \mathtt{x} = \max(0, a - b + \mathtt{y} - 1) \wedge \mathtt{y} - 1 \geq 0 \rangle$

```
    y = y - 1;
```

$\langle \mathtt{x} = \max(0, a - b + \mathtt{y}) \wedge \mathtt{y} \geq 0 \rangle$

```
}
```

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

b) Eine gültige Variante für die Terminierung ist $V = \texttt{y}$. Die Schleifenbedingung $B = \texttt{y} > 0$ impliziert $V \geq 0$. Es gilt:

$\langle \texttt{y} = m \wedge \texttt{y} > 0 \rangle$
$\langle \texttt{y} - 1 < m \rangle$
```
if (x > 0) {
```
$\langle \texttt{y} - 1 < m \wedge x > 0 \rangle$
$\langle \texttt{y} - 1 < m \rangle$
```
    x = x - 1;
```
$\langle \texttt{y} - 1 < m \rangle$
```
}
```
$\langle \texttt{y} - 1 < m \rangle$
```
y = y - 1;
```
$\langle \texttt{y} < m \rangle$

Damit ist die Terminierung der einzigen Schleife in P gezeigt.

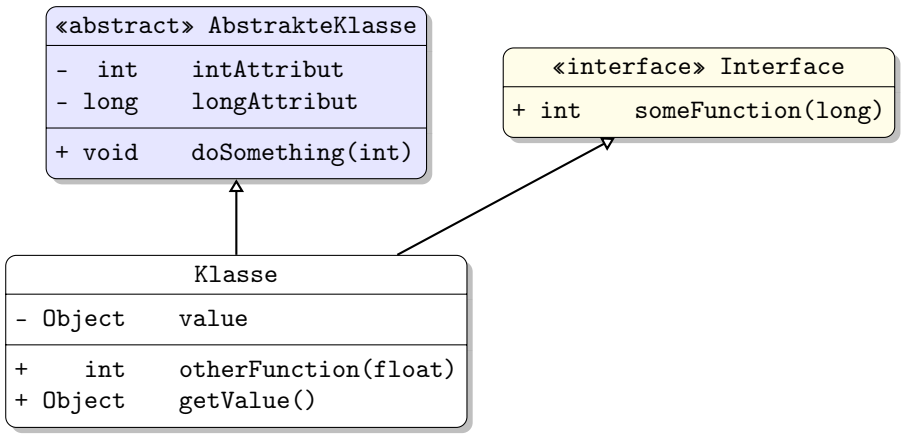
Aufgabe 3 (Klassenhierarchie): (6 + 4 = 10 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Hierarchie zur Verwaltung von Kunstwerken.

- Ein Kunstobjekt wurde von einen Künstler erschaffen, der durch einen `String` repräsentiert wird.
- Eine Malerei ist ein Kunstobjekt und zeichnet sich durch ihren Untergrund aus, welcher als `String` gespeichert wird.
- Ein Portrait ist eine Malerei, deren Alter (in Jahren) von Interesse ist.
- Eine Zeichnung ist ein Kunstobjekt, bei dem die Breite ihrer Linien wichtig ist, welche in Millimetern angegegeben wird.
- Eine Bildhauerei ist ein Kunstobjekt, das sich durch das verwendete Material auszeichnet, welches als `String` dargestellt wird.
- Eine Skulptur ist eine Bildhauerei, die mit einem Werkzeug herausgeschlagen wird. Hier ist der Name des Werkzeugs von Interesse.
- Eine Plastik ist eine Bildhauerei, die gegossen wird. Hier ist das Volumen (in Litern) der Plastik von Bedeutung.
- Portraits und Zeichnungen lassen sich an der Wand befestigen. Dazu stellen sie die Methode `void aufhaengen()` zur Verfügung.
- Bildhauereien lassen sich an ihren Platz rücken und bieten dazu die Methode `void verruecken()` an.
- Jedes Kunstobjekt ist entweder eine Malerei, eine Zeichnung oder eine Bildhauerei.
- Eine Bildhauerei muss nicht zwangsläufig eine Skulptur oder Plastik sein.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Kunstwerken. 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:

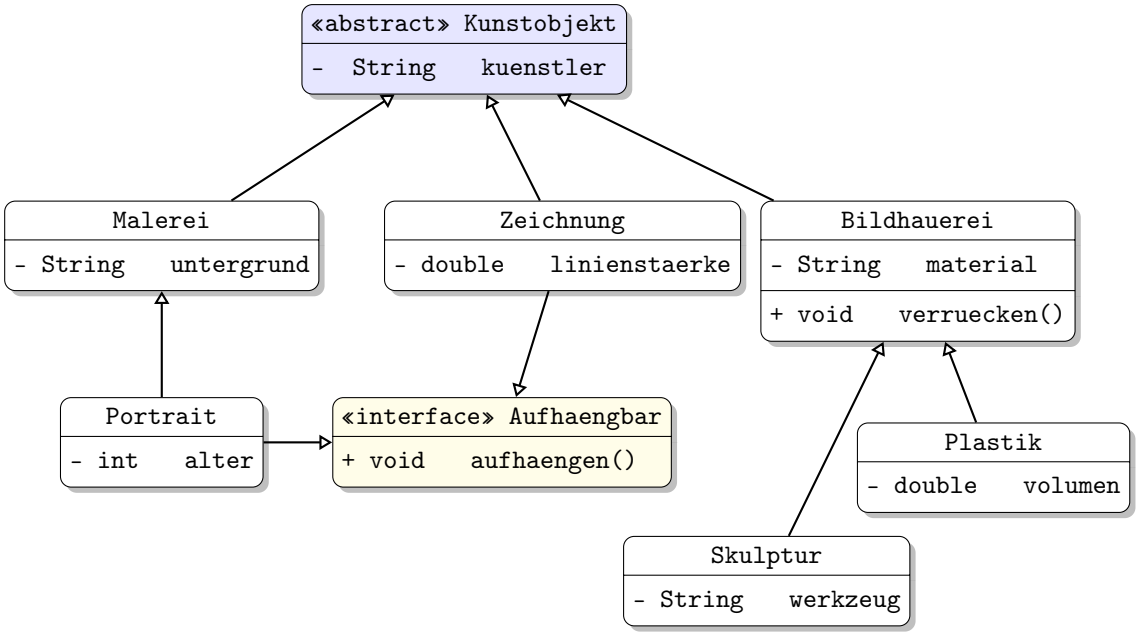


Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \twoheadrightarrow A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Benutzen Sie `-`, um `private` abzukürzen, und `+` für alle anderen Sichtbarkeiten (wie z. B. `public`).

b) Schreiben Sie eine Java-Methode mit der folgenden Signatur:
`public static void galerieEinrichten(Kunstobjekt[] kunstobjekte)`
Diese Methode soll alle Kunstobjekte, die sich an einer Wand befestigen lassen, aufhängen und alle Bildhauereien an ihren Platz rücken. Nehmen Sie dazu an, dass das übergebene Array `kunstobjekte` nicht `null` ist.

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



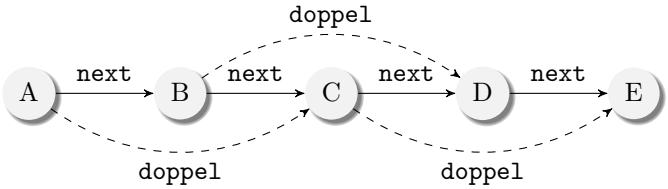
b)
```
public static void galerieEinrichten(Kunstobjekt[] kunstobjekte) {
    for (Kunstobjekt k : kunstobjekte) {
        if (k instanceof Aufhaengbar) {
            ((Aufhaengbar)k).aufhaengen();
        } else if (k instanceof Bildhauerei) {
            ((Bildhauerei)k).verruecken();
        }
    }
}
```

Aufgabe 4 (Programmieren in Java): (5 + 7 + 8 + 7 + 5 = 32 Punkte)

Die Klasse `DoppelListe<T>` dient zur Repräsentation von Listen. Jedes Listenelement wird als Objekt der Klasse `DoppelListe` dargestellt. Es enthält einen Wert vom generischen Typen `T`, einen Verweis auf den direkten Nachfolger und zusätzlich einen Verweis auf den **übernächsten** Nachfolger. Der Wert wird in dem Attribut `wert` gespeichert, das Attribut `next` zeigt auf das nächste Element der Liste und das Attribut `doppel` zeigt auf das übernächste Element der Liste.
Das letzte Element der Liste hat keinen Nachfolger, so dass dessen Attribut `next` auf `null` zeigt. Aus dem gleichen Grund zeigen die `doppel`-Attribute des letzten und vorletzten Listenelements ebenfalls auf `null`. Die leere Liste stellen wir als `null` dar.

```
public class DoppelListe<T> {
  T wert;
  DoppelListe<T> next = null;
  DoppelListe<T> doppel = null;
}
```

In der folgenden Graphik ist dargestellt, wie eine Liste mit den Werten `A, B, C, D, E` mit der Klasse `DoppelListe` repräsentiert wird.



Objekte der Klasse `DoppelListe` sind als Kreise dargestellt, die den im `wert`-Attribut gespeicherten Wert enthalten. Die Elemente dieser Liste sind jeweils durch das Attribut `next` (durchgezogene Pfeile) mit dem direkten Nachfolger verbunden. Das Attribut `doppel` (gestrichelte Pfeile) verbindet jedes Listenelement mit dem übernächsten Listenelement. Attribute, deren Wert `null` ist, werden in der Graphik nicht gezeigt.
Die `doppel`-Attribute der `DoppelListe`-Objekte mit den Werten `D` und `E` zeigen beide auf `null`, da für die entsprechenden Listenelemente kein übernächstes Element existiert.

Wir definieren also, dass ein Objekt `x` vom Typ `DoppelListe<T>` eine korrekte `DoppelListe` ist, falls folgende beiden Eigenschaften gelten:

- Wenn `x.next == null`, dann auch `x.doppel == null`.
- Wenn `x.next != null`, dann `x.doppel == x.next.next`.

Hinweis: Sie dürfen in **allen Teilaufgaben** davon ausgehen, dass nur auf **azyklischen Listen** gearbeitet wird.

a) Implementieren Sie einen Konstruktor für die Klasse `DoppelListe`, der als erstes Argument einen Wert vom Typ `T` und als zweites Argument eine weitere `DoppelListe<T>` bekommt.

Der Konstruktor soll ein neues `DoppelListe`-Objekt erzeugen, dessen `wert`-Attribut auf den Wert des ersten Arguments gesetzt wird und dessen direkter Nachfolger die im zweiten Argument übergebene `DoppelListe` ist. Schreiben Sie den Konstruktor so, dass auch das `doppel`-Attribut entsprechend obiger Erklärung richtig gesetzt wird. Beachten Sie hierbei bitte, dass das zweite Argument auch `null` sein kann!

b) Implementieren Sie die nicht-statische Methode `finde` in der Klasse `DoppelListe<T>`, die ein Argument vom Typ `T` übergeben bekommt. Der Rückgabewert der Methode ist das erste `DoppelListe`-Objekt in der aktuellen `DoppelListe`, dessen Wert gleich dem übergebenen Wert ist. Falls kein solches Element in der `DoppelListe` existiert, soll `null` zurückgegeben werden.

Verwenden Sie dazu **ausschließlich Schleifen** und keine Rekursion. Verändern Sie die `DoppelListe` nicht durch Schreibzugriffe. Verwenden Sie für den Vergleich die in der Klasse `Object` vordefinierte

Methode
`boolean equals(Object obj)`.

Hinweise:

- Sie dürfen davon ausgehen, dass sowohl die in der `DoppelListe` gespeicherten Werte als auch der übergebene Wert nicht `null` sind.
- Für diese Aufgabe brauchen Sie das `doppel`-Attribut nicht zu verwenden!

c) Implementieren Sie die nicht-statische Methode `pruefe` in der Klasse `DoppelListe<T>` ohne Argumente und ohne Rückgabewert, die überprüft, ob die aktuelle `DoppelListe` korrekt ist (wie am Anfang der Aufgabe definiert). Falls sie nicht korrekt ist, soll eine Exception der Klasse `StrukturFehlerException` mit der folgenden Deklaration geworfen werden.

```
public class StrukturFehlerException extends Exception {
}
```

Verwenden Sie keine Schleifen, sondern **ausschließlich Rekursion**.

d) Implementieren Sie die nicht-statische Methode `rueckwaerts` ohne Argumente in der Klasse `DoppelListe<T>`, die eine `DoppelListe<T>` zurückgibt. Das Ergebnis der Methode soll eine `DoppelListe` sein, die die Elemente der aktuellen `DoppelListe` in umgekehrter Reihenfolge enthält. Verändern Sie die aktuelle `DoppelListe` nicht durch Schreibzugriffe, sondern erzeugen Sie für die Rückgabe bei Bedarf eine neue `DoppelListe`.

Hinweise:

- Falls Sie Ihre Lösung mit einer Schleife implementieren, so empfiehlt es sich, die Ergebnisliste schrittweise zu erzeugen, während Sie die vorderen Elemente der Eingabeliste abarbeiten.
- Verwenden Sie den Konstruktor aus Aufgabenteil a).

e) Implementieren Sie eine nicht-statische Methode `filtern` in der Klasse `DoppelListe<T>`, die eine `LinkedList<T>` aus dem Java-Collections Framework als Argument bekommt und eine ganze Zahl zurück gibt.

Die Methode soll einen Iterator der `LinkedList` nutzen, um die `LinkedList` zu durchlaufen. Dabei sollen alle Elemente aus der `LinkedList` entfernt werden, die auch in der aktuellen `DoppelListe` vorhanden sind. Zum Entfernen der Elemente muss die Methode `remove` des Iterators genutzt werden. Anschließend soll die Methode zurück geben, wie viele Elemente entfernt wurden.

Hinweise:

- Verwenden Sie die Methode `finde` aus Aufgabenteil b).

Lösung:

a)
```
public DoppelListe(T wert, DoppelListe<T> next) {
  this.wert = wert;
  if (next != null) {
    this.next = next;
    this.doppel = next.next;
  }
}
```

b)
```
public DoppelListe<T> finde(T wert) {
  DoppelListe<T> cur = this;
  while (cur != null) {
```

```
    if (cur.wert.equals(wert)) {
      return cur;
    }
    cur = cur.next;
  }
  return null;
}
```

c)
```
public void pruefe() throws StrukturFehlerException {
  if (this.next == null) {
    if (this.doppel != null) {
      throw new StrukturFehlerException();
    }
  } else {
    if (this.next.next != this.doppel) {
      throw new StrukturFehlerException();
    }
    this.next.pruefe();
  }
}
```

d) Iterativ:

```
public DoppelListe<T> rueckwaerts() {
  DoppelListe<T> cur = this;
  DoppelListe<T> res = null;
  while (cur != null) {
    res = new DoppelListe<T>(cur.wert, res);
    cur = cur.next;
  }
  return res;
}
```

Rekursiv, nicht-statisch:

```
public DoppelListe<T> rueckwaerts() {
  return rueckwaerts(null);
}

private DoppelListe<T> rueckwaerts(DoppelListe<T> rest) {
  if (this.next == null) {
    return new DoppelListe<T>(this.wert, rest);
  }
  return this.next.rueckwaerts(new DoppelListe<T>(this.wert, rest));
}
```

Rekursiv, statisch:

```
public DoppelListe<T> rueckwaerts() {
  return rueckwaerts(this, null);
}

private static <T> DoppelListe<T> rueckwaerts(DoppelListe<T> liste,
```

```
                                                    DoppelListe<T> rest) {
    if (liste.next == null) {
      return new DoppelListe<T>(liste.wert, rest);
    }
    return rueckwaerts(liste.next, new DoppelListe<T>(liste.wert, rest));
  }
```

e)
```
public int filtern(LinkedList<T> ll) {
    int anzahl = 0;
    T wert;
    Iterator<T> iter = ll.iterator();
    while(iter.hasNext()) {
      wert = iter.next();
      if(this.finde(wert) != null) {
        iter.remove();
        anzahl += 1;
      }
    }
    return anzahl;
  }
```