Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Korzeniewski

Aufgabe 1 (Programmanalyse): (9.5 + 4.5 = 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. Achten Sie darauf, dass Gleitkommazahlen in der Form "5.0" und nicht als "5" ausgegeben werden.

```
public class A {
  public Integer a;
  public Float b;

  public A() {
    a = 1;
    b = 1.0f;
  }

  public A(Integer x) {
    a = 2;
    b = 2.0f;
  }

  public int f(long x) {
    return 3;
  }

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

```
public class B extends A {
  public int a;

  public B(float x) {
    a = 5;
  }

  public int f(int x) {
    return 6;
  }

  public int f(float x) {
    return 7;
  }

  public int f(double x) {
    return 8;
  }
}
```

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 z;
3    public C(float x) {
4      a = 5;
5    }
6
7    public int f(long x) {
8      return 42;
9    }
10
11   public void f() {
12     Double x = 2.0;
13
14     A ab = new B(x);
15
16     int ret = ab.f(x);
17
18     var val = true;
19
20     for(Integer i = 0; i < 10; ++i) {
21       x += ab.f(val);
22     }
23   }
```

```
24 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
      A aa = new A(1);
      System.out.println(aa.a + " " + aa.b);// OUT: [ 2 ] [2.0]

      int ret = aa.f(aa.b);
      System.out.println(ret);              // OUT: [ 4 ]

      B bb = new B(6);
      A ab = bb;
      System.out.println(bb.a);             // OUT: [ 5 ]
      System.out.println(ab.a + " " + ab.b);// OUT: [ 1 ] [1.0]

      ret = bb.f(1);
      System.out.println(ret);              // OUT: [ 6 ]

      ret = ab.f(1.0f);
      System.out.println(ret);              // OUT: [ 8 ]

      ret = ab.f(aa.a);
      System.out.println(ret);              // OUT: [ 3 ]

      ret = bb.f(aa.b);
      System.out.println(ret);              // OUT: [ 7 ]
    }
}
```

b)
- Im Konstruktor fehlt der Aufruf von `super`, da `B` keinen Konstruktor ohne Parameter besitzt.
- Es existiert in Zeile 14 kein passender Konstruktor für `B`, da `x` den Typ `Double` hat und nicht zu `float` umgewandelt werden kann.
- In Zeile 21 existiert keine passende Methode `f`, da `boolean` nicht in eine Zahl umgewandelt werden kann.

Aufgabe 2 (Hoare-Kalkül): (14 + 2 = 16 Punkte)

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

$\langle\ 0 \leq \texttt{a.length}\ \rangle$ (Vorbedingung)

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

$\langle\ \texttt{res} = |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} \leq 0\}|\ \rangle$ (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.
- Die Menge $\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}\}$ bezeichnet $\{0, 1, \ldots, \texttt{i} - 1\}$, falls $\texttt{i} > 0$ ist. Falls $\texttt{i} \leq 0$ ist, so gilt $\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}\} = \emptyset$.
- Der Ausdruck $|\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} \leq 0\}|$ ist die Differenz der Anzahl von positiven Arrayeinträgen und der Anzahl von nicht–positiven Arrayeinträgen. Im Array $\texttt{a} = [1, 5, -4, 10, 0]$ gibt es drei positive Einträge an den Positionen `0`, `1` und `3` und zwei nicht–positive Einträge an den Positionen `2` und `4`. Der Ausdruck ist also

$$\begin{aligned}|\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} \leq 0\}| &= |\{0, 1, 3\}| - |\{2, 4\}| \\ &= 3 - 2 = 1\end{aligned}$$

- Um Schreibarbeit zu sparen, können Sie häufiger vorkommenden Teilformeln einen Namen (wie z.B. φ oder $\varphi(\texttt{i})$) geben und dann diesen Namen anstelle der Teilformel verwenden.

b) Untersuchen Sie den Algorithmus 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) In der folgenden Lösung verwenden wir die Abkürzung $\varphi(\texttt{i}) = |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}, \texttt{a[j]} \leq 0\}|$ und entsprechend $\varphi(\texttt{i}+1) = |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}+1, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{i}+1, \texttt{a[j]} \leq 0\}|$.

$\langle 0 \leq \texttt{a.length} \rangle$
$\langle 0 \leq \texttt{a.length} \wedge 0 = 0 \wedge 0 = 0 \rangle$
`i = 0;`

$\langle 0 \leq \texttt{a.length} \wedge \texttt{i} = 0 \wedge 0 = 0 \rangle$
```
res = 0;
```
$\langle 0 \leq \texttt{a.length} \wedge \texttt{i} = 0 \wedge \texttt{res} = 0 \rangle$
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \rangle$
```
while (i < a.length) {
```
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \wedge \texttt{i} < \texttt{a.length} \rangle$
```
    if (a[i] > 0) {
```
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \wedge \texttt{i} < \texttt{a.length} \wedge \texttt{a[i]} > 0 \rangle$
$\langle \texttt{res} + 1 = \varphi(\texttt{i} + 1) \wedge \texttt{i} + 1 \leq \texttt{a.length} \rangle$
```
        res = res + 1;
```
$\langle \texttt{res} = \varphi(\texttt{i} + 1) \wedge \texttt{i} + 1 \leq \texttt{a.length} \rangle$
```
    }
    else {
```
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \wedge \texttt{i} < \texttt{a.length} \wedge \neg(\texttt{a[i]} > 0) \rangle$
$\langle \texttt{res} - 1 = \varphi(\texttt{i} + 1) \wedge \texttt{i} + 1 \leq \texttt{a.length} \rangle$
```
        res = res - 1;
```
$\langle \texttt{res} = \varphi(\texttt{i} + 1) \wedge \texttt{i} + 1 \leq \texttt{a.length} \rangle$
```
    }
```
$\langle \texttt{res} = \varphi(\texttt{i} + 1) \wedge \texttt{i} + 1 \leq \texttt{a.length} \rangle$
```
    i = i + 1;
```
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \rangle$
```
}
```
$\langle \texttt{res} = \varphi(\texttt{i}) \wedge \texttt{i} \leq \texttt{a.length} \wedge \neg(\texttt{i} < \texttt{a.length}) \rangle$
$\langle \texttt{res} = |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} > 0\}| - |\{\texttt{j} \mid 0 \leq \texttt{j} < \texttt{a.length}, \texttt{a[j]} \leq 0\}| \rangle$

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

- $B \Rightarrow V \geq 0$, denn $B = \texttt{i} < \texttt{a.length} \Rightarrow \texttt{a.length} - \texttt{i} \geq 0$ und
- die folgende Ableitung ist korrekt:

$\langle \texttt{a.length} - \texttt{i} = \texttt{m} \wedge \texttt{i} < \texttt{a.length} \rangle$
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
    if (a[i] > 0 ) {
```
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \wedge \texttt{a[i]} > 0 \rangle$
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
        res = res + 1;
```
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
    }
    else {
```
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \wedge \neg(\texttt{a[i]} > 0) \rangle$
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
        res = res - 1;
```
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
    }
```
$\langle \texttt{a.length} - (\texttt{i} + 1) < \texttt{m} \rangle$
```
     i = i + 1;
```
$\langle \texttt{a.length} - \texttt{i} < \texttt{m} \rangle$

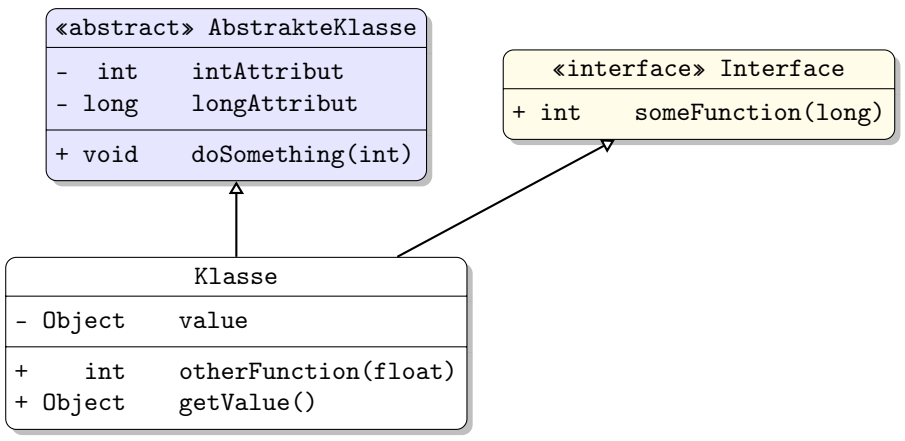
Aufgabe 3 (Klassenhierarchie): (6 + 4 = 10 Punkte)

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

- Stoffe haben ein Gewicht pro Quadratmeter und eine Länge. Ein Stoff könnte z.B. das Gewicht $120\frac{g}{m^2}$ und die Länge $3.5m$ haben. Die Breite eines Stoffs wird *nicht* gespeichert, da fast alle Stoffe ca. $1.35m$ breit sind. Die Einheit wird ebenfalls nicht gespeichert. Ein Stoff kann mit der Methode `void sew()` vernäht werden. (Der Stoff wird mit sich selbst vernäht, nicht mit einem anderen Stoff.)
- Jeder Stoff gehört zu einer bestimmten Unterart.
- Canvas ist eine Art gewebter Stoff. Das Erscheinungsbild dieser Stoffe hängt hauptsächlich von der Dicke der Fäden ab, aus denen der Stoff gewebt ist. Diese wird in *Denier* gemessen. Da übliche Fäden mehrere hundert Denier stark sind, müssen keine Nachkommastellen gespeichert werden.
- Canvas ist nur ein Oberbegriff für verschiedene Stoffarten mit ähnlichen Eigenschaften, keine tatsächliche Stoffart.
- Leinencanvas, Baumwollcanvas und Cordura sind Canvas Stoffe.
- Leinencanvas läuft beim Waschen ein, daher ist es wichtig zu wissen, ob der Stoff bereits gewaschen wurde oder noch nicht.
- Baumwolle lässt sich sehr gut färben und bedrucken. Bei Baumwollcanvas werden daher immer die Farbe und das Muster als `String`s gespeichert.
- Cordura wird manchmal beschichtet. Das Material, mit dem der Stoff beschichtet wurde, wird als `String` gespeichert.
- Gabardine ist ein Stoff, der auch für Anzüge verwendet wird. Bei Gabardine ist daher der Preis interessant.
- Jersey ist ein Stoff, der in einer Richtung elastisch ist, da Jersey gestrickt und nicht gewebt wird. Bei Jersey wird daher angegeben, um wie viel Prozent sich der Stoff in eine Richtung dehnen kann.
- Naturgewebe sind Stoffe, die aus Naturfasern gewebt sind. Alle Naturgewebe lassen sich in Form bügeln. Dieser Vorgang nennt sich *dressieren*. Leinencanvas, Baumwollcanvas und Gabardine sind Naturgewebe und bieten daher eine Methode `void dressieren()` an. Ansonsten haben Naturgewebe nichts gemeinsam.

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

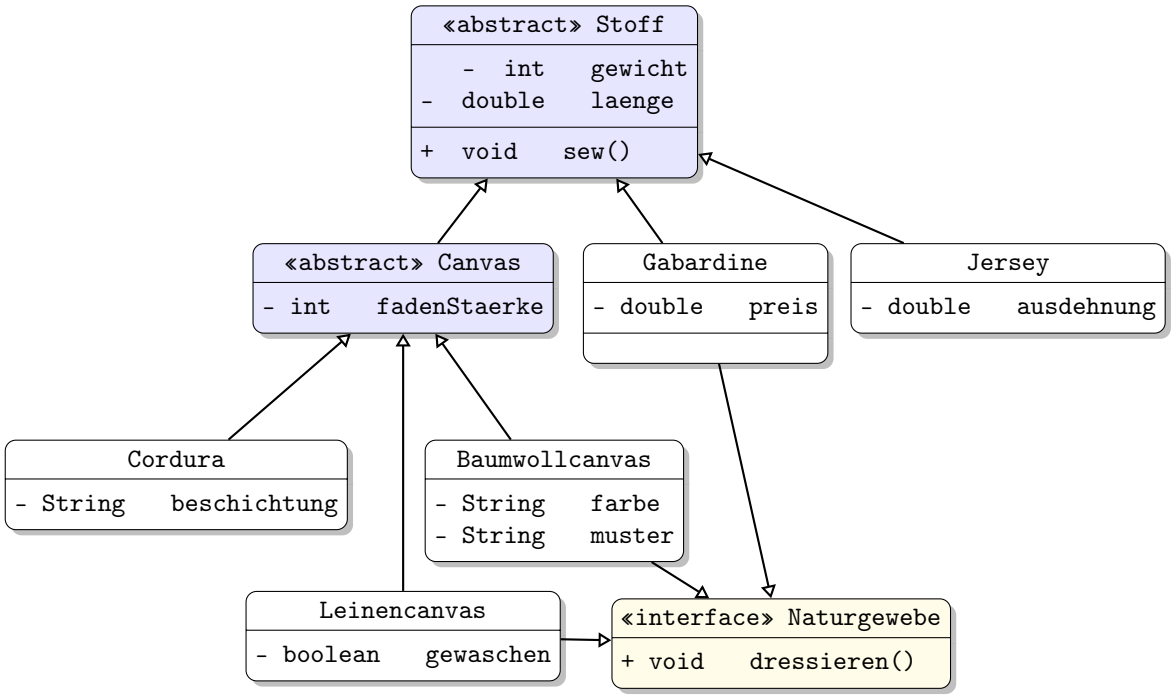
b) Schreiben Sie eine Java-Methode mit der folgenden Signatur:
`public static void verarbeiten(Stoff[] stoffe)`
Diese Methode soll alle Stoffe vernähen, außer nicht gewaschenes Leinencanvas, da das Produkt sonst beim Waschen Falten bekommen würde. Stoffe, bei denen dies möglich ist, müssen vor dem Vernähen noch dressiert werden. Nehmen Sie dazu an, dass das übergebene Array `stoffe` nicht `null` ist.

Hinweise:

- Sie dürfen davon ausgehen, dass es für alle Attribute geeignete Selektoren gibt.

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



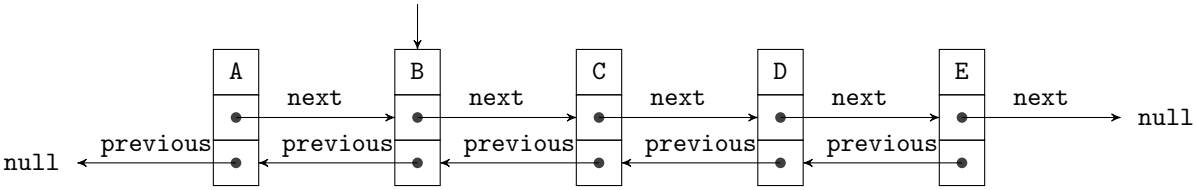
b)
```
public static void verarbeiten(Stoff[] stoffe) {
    for (Stoff s : stoffe) {
        if (s != null && (!s instanceof Leinencanvas || ((Leinencanvas)s).getGewaschen())) {
            if (s instanceof Naturgewebe) {
                ((Naturgewebe)s).dressieren();
            }
            s.sew();
        }
    }
}
```

Aufgabe 4 (Programmierung in Java): (5 + 4 + 3 + 7 + 2 + 10 = 31 Punkte)

In dieser Aufgabe wollen wir die Klasse `DoublyLinkedList<T>` zur Darstellung doppelt verketteter Listen implementieren. Ein `DoublyLinkedList`-Objekt ist ein Listenknoten und enthält einen Wert vom Typ `T`, sowie eine Referenz auf den vorherigen Listenknoten (`previous`) und eine Referenz auf den nachfolgenden Listenknoten (`next`).

```
public class DoublyLinkedList<T> {
    private final T value;
    private DoublyLinkedList<T> next;
    private DoublyLinkedList<T> previous;
}
```

Der erste Listenknoten hat keinen Vorgänger, deswegen zeigt dessen `previous`-Referenz auf `null`. Analog hat der letzte Listenknoten keinen Nachfolger und seine `next`-Referenz zeigt auf `null`.
In der folgenden Grafik ist dargestellt, wie eine Liste mit den Werten `A, B, C, D` und `E` mit der Klasse `DoublyLinkedList` repräsentiert wird.



Wir sehen fünf Listenknoten, dargestellt durch je drei übereinander liegende Quadrate. Das oberste Quadrat enthält den Wert (`value`). Die unteren beiden Quadrate enthalten die Referenzen auf den Vorgänger bzw. Nachfolger. Der linkeste Listenknoten mit `value = A` und `previous = null` ist der erste Listenknoten. Der rechteste Listenknoten mit `value = E` und `next = null` ist der letzte Listenknoten. Der Pfeil auf den Listenknoten mit `value = B` repräsentiert eine Variable, welche diesen Listenknoten referenziert.
Ein Listenknoten `x` ist *wohlgeformt*, falls beide folgende Eigenschaften erfüllt sind:

- Falls `x.next != null`, dann `x.next.previous == x` und `x.next` ist wohlgeformt.
- Falls `x.previous != null`, dann `x.previous.next == x` und `x.previous` ist wohlgeformt.

Hinweis: Sie dürfen in **allen Teilaufgaben** davon ausgehen, dass nur auf **azyklischen Listen** gearbeitet wird (d.h. man erreicht keinen Zyklus, wenn man nur den `next`- oder nur den `previous`-Referenzen folgt).

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

Der Konstruktor soll ein neues `DoublyLinkedList`-Objekt erzeugen, dessen `value`-Attribut auf den Wert des ersten Arguments gesetzt wird und dessen direkter Vorgänger die im zweiten Argument übergebene `DoublyLinkedList` ist. Schreiben Sie den Konstruktor so, dass auch der Nachfolger des zweiten Arguments 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 `getLast` in der Klasse `DoublyLinkedList<T>`, die keine Argumente bekommt. Die Methode läuft zum letzten Listenknoten und gibt diesen zurück.

Verwenden Sie dazu keine Schleifen, sondern **ausschließlich Rekursion**. Verändern Sie die `DoublyLinkedList` nicht durch Schreibzugriffe.

c) Implementieren Sie die nicht-statische Methode `add` in der Klasse `DoublyLinkedList<T>`, die ein Argument vom Typ `T` übergeben bekommt. Die Methode hängt an den letzten Listenknoten einen neuen Listenknoten an, dessen Wert auf das erste Argument der Methode gesetzt ist. Achten Sie darauf, dass die Liste anschließend weiterhin wohlgeformt ist. Die Methode gibt nichts zurück.

Hinweise:

- Verwenden Sie die Methode `getLast` aus Aufgabenteil b).
- Verwenden Sie den Konstruktor aus Aufgabenteil a).

- Sie dürfen davon ausgehen, dass die Liste bislang wohlgeformt ist.

d) Implementieren Sie die nicht-statische Methode `toLinkedList` in der Klasse `DoublyLinkedList<T>`, die keine Argumente bekommt. Die Methode wandelt die **komplette Liste** in eine `LinkedList<T>` aus dem Java-Collections Framework um und gibt diese zurück.

Hinweise:

- Für die Liste mit dem `value` `B` aus der Grafik würde also die `LinkedList` mit den Werten `A`, `B`, `C`, `D` und `E` entstehen.
- Verwenden Sie die Methode `getLast` aus Aufgabenteil b).
- Verwenden Sie die Methode `void addFirst(T t)` der Klasse `LinkedList<T>`.
- Sie dürfen davon ausgehen, dass die Liste wohlgeformt ist.

e) Erstellen Sie eine neue `Exception`-Klasse `BrokenStructureException`. Der Konstruktor dieser Klasse nimmt keine Parameter entgegen.

f) Implementieren Sie die nicht-statische Methode `ensureValidStructure` in der Klasse `DoublyLinkedList<T>`, die keine Argumente bekommt. Die Methode wirft genau dann eine `BrokenStructureException`, wenn die Liste nicht wohlgeformt ist. Die Methode gibt nichts zurück.

Hinweise:

- Die Liste mit dem `value` `B` aus der Grafik ist beispielsweise wohlgeformt.
- Verwenden Sie die Methode `getLast` aus Aufgabenteil b).
- Achten Sie darauf, dass man das aktuelle Element (`this`) auch vom letzten Listenknoten aus erreichen können muss, wenn man von dort aus den `previous`-Referenzen folgt.

Lösung:

a)
```
public DoublyLinkedList(T value, DoublyLinkedList<T> previous) {
    this.value = value;
    this.previous = previous;
    if (previous != null) {
        previous.next = this;
    }
}
```

b)
```
public DoublyLinkedList<T> getLast() {
    if (this.next == null) {
        return this;
    } else {
        return this.next.getLast();
    }
}
```

c)
```
public void add(T value) {
    new DoublyLinkedList<>(value, getLast());
}
```

d)
```
public LinkedList<T> toLinkedList() {
    LinkedList<T> result = new LinkedList<>();
    DoublyLinkedList<T> current = getLast();
    while (current != null) {
        result.addFirst(current.value);
        current = current.previous;
    }
    return result;
}
```

e)
```
public class BrokenStructureException extends Exception {
}
```

f)
```
public void ensureValidStructure() throws BrokenStructureException {
    boolean visitedThis = false;
    DoublyLinkedList<T> current = getLast();
    while (current != null) {
        if (current.previous != null
                && current.previous.next != current) {
            throw new BrokenStructureException();
        }
        if (current == this) {
            visitedThis = true;
        }
        current = current.previous;
    }
    if (!visitedThis) {
        throw new BrokenStructureException();
    }
}
```