

Modul Datenstrukturen, Algorithmen und Programmierung 1

Zusatzaufgaben Doppelt verkettete Liste

- Die Aufgaben stammen aus den Klausuren der vergangenen Jahre.
- Die Aufgaben sind unterschiedlich schwer und unterschiedlich umfangreich. Dieses wurde in den Klausuren durch die Zahl der zugeordneten Punkte berücksichtigt.
- Alle Aufgaben lassen sich mit den Kenntnissen der Vorlesungsinhalte bis einschließlich Kapitel 12 bearbeiten.
- Die Aufgaben dienen in verschiedener Art dem Training für die Klausur:
 - In Programmierung bereits geübte Studierende können sich an die Form der Klausuraufgaben gewöhnen.
 - Bei allen Studierenden wird das algorithmische Denken trainiert, das zum Lösen der Aufgaben zu fortgeschrittenen Themen benötigt wird.
- Zu diesen Aufgaben werden keine Beispiellösungen veröffentlicht.
- Die Zusammenstellung ist ausgehend von den vorliegenden Klausuren erfolgt. Möglicherweise sind einzelne Aufgaben – oder Varianten davon – bereits in die Übungs- oder Praktikumsaufgaben übernommen worden und tauchen jetzt doppelt auf.
- Die Aufgaben der hier präsentierten Form haben in den Klausuren etwa 15 bis 25 Prozent des Gesamtumfangs ausgemacht.
- In allen Aufgaben soll die aus der Vorlesung bekannte Klasse `DoublyLinkedList<T>` ergänzt werden, die im Anhang noch einmal aufgeführt ist.
- Bei der Implementierung der geforderten Methoden dürfen aber **nur** die **im Anhang** aufgeführten Methoden genutzt werden.

Aufgabe 1

Vervollständigen Sie die Methode `void exchange()`.

Die Methode `exchange` soll die ersten beiden vom Listenanfang aus erreichbaren Inhalte, die nicht `null` sind, miteinander vertauschen. Gibt es keine zwei Inhalte, die ungleich `null` sind, soll nichts geschehen.

```
public void exchange()
{
    Element<T> current =  ;

    Element<T> firstHit =  ;

    while (  )
    {
        
    }
}
```

Aufgabe 2

Vervollständigen Sie die Methode `int splitBehind(T c)`.

Die Methode `splitBehind` soll alle Elemente aus der Liste entfernen, die auf das erste Element mit dem Inhalt `c` folgen. Die Anzahl der entfernten Elemente soll zurückgegeben werden. Kommt der Inhalt `c` nicht vor, soll kein Element entfernt und `0` zurückgegeben werden.

Der Vergleich soll mit der Methode `equals` vorgenommen werden.

```
public int splitBehind( T c )
{
    Element<T> current =  ;

    int newSize =  ;

    while (  )
    {
        

    }

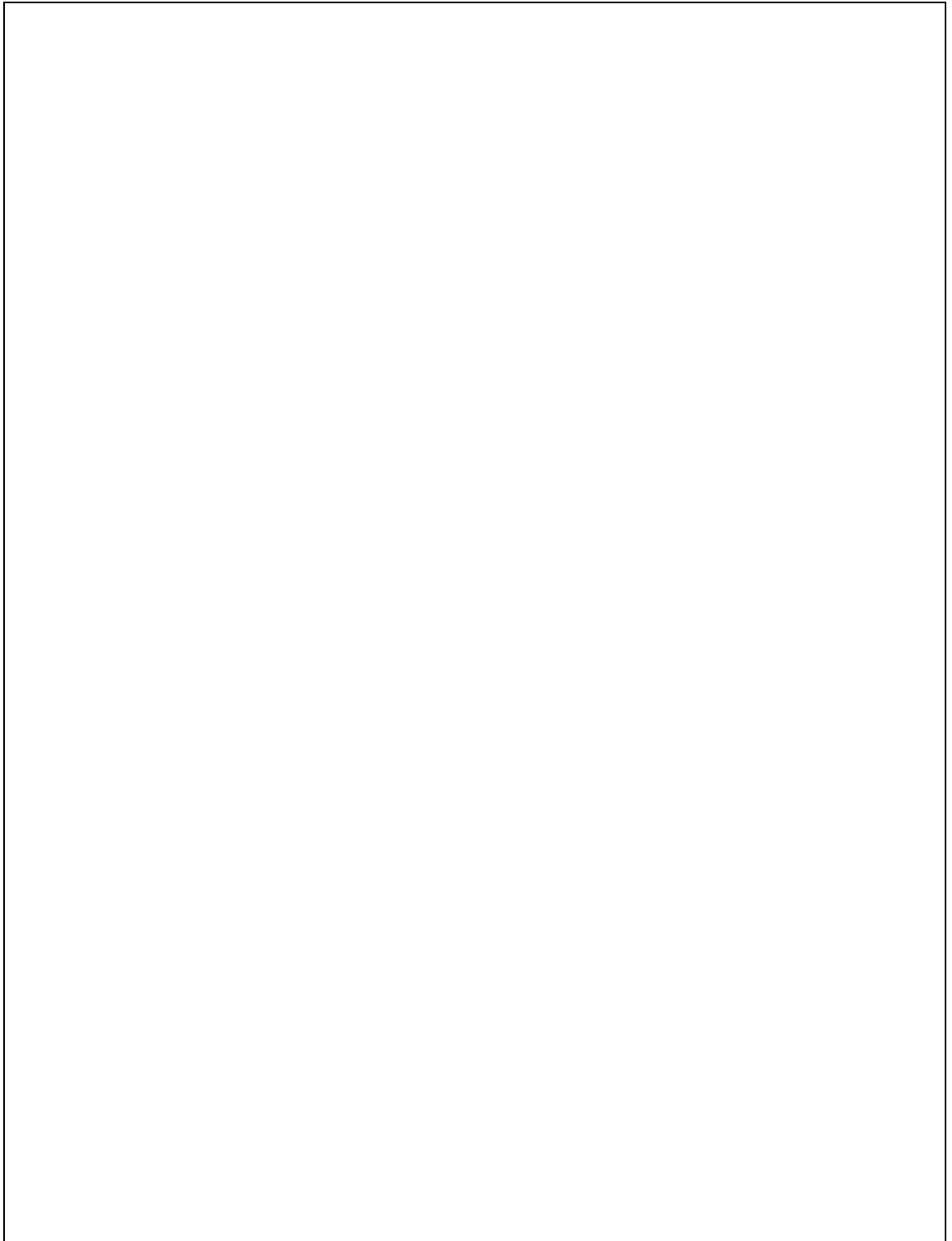
    return 0;
}
```

Aufgabe 3

Vervollständigen Sie die Methode `int[] positions()`.

Die Methode `positions` soll ein Feld zurückgeben, in dem genau nur die Positionen derjenigen Elemente der Liste stehen, die `null` als Inhalt besitzen. Das erste Element der Liste hat die Position `0`. Kommt `null` nicht als Inhalt vor, soll ein Feld der Länge `0` zurückgegeben werden.

```
public int[] positions()  
{
```



```
    return result;  
}
```

Aufgabe 4

Vervollständigen Sie die Methode `void appendFirst()`.

Die Methode `appendFirst` soll das erste Element der Liste an das Ende der Liste verschieben. Besitzt die Liste nicht mindestens zwei Elemente, soll nichts geschehen.

```
public void appendFirst()
{
    if (  )
    {
        
    }
}
```

Aufgabe 5

Vervollständigen Sie die Methode `void inject(DoublyLinkedList<T> into, int p)`.

Die Methode `inject` soll unmittelbar hinter dem Element an der Position `p` der ausführenden Liste alle Elemente der Liste `into` einfügen. Diese Elemente sollen aus der Liste `into` entfernt werden. Das erste Element der Liste hat die Position `0`. Hat die ausführende Liste keine `p+1` Elemente, soll nichts geschehen.

```
public void inject( DoublyLinkedList<T> into, int p )
```

```
{
```

```
    Element current =
```

```
    ;
```

```
    if (
```

```
    )
```

```
    {
```

```
    }
```

Aufgabe 6

Vervollständigen Sie die Methode `boolean allEqual(DoublyLinkedList<T> d)`.

Die Methode `allEqual` soll `true` zurückgeben, wenn die ausführende Liste und die als Argument übergebene Liste die gleiche Zahl von Elementen aufweisen und an jeder Position den gleichen Inhalt besitzen. Sonst soll `false` zurückgegeben werden. Der Vergleich soll mit der Methode `equals` erfolgen. Gehen Sie davon aus, dass der Parameter `d` immer ungleich `null` ist.

```
public boolean allEqual( DoublyLinkedList<T> d )
{
    if (  )
    {
        Element c1 =  ;
        Element c2 =  ;
        while (  )
        {
            
        }
         ;
    }
    else
    {
        return false;
    }
}
```

Aufgabe 3

Vervollständigen Sie die Methode `void moveToHead(int n)`.

Die Methode `moveToHead` soll die letzten `n` Elemente vom Ende der Liste entfernen und in unveränderter Reihenfolge an den Anfang der Liste stellen. Ist `n` nicht positiv oder ist die Länge der Liste kleiner oder gleich `n`, soll die Liste unverändert bleiben.

```
public void moveToHead( int n )
```

```
{
```

```
    if (
```

```
)
```

```
{
```

```
}
```

```
}
```


Aufgabe 3

Vervollständigen Sie die Methode `int strip()`.

Die Methode `strip` soll alle Elemente aus der Liste entfernen, deren Inhalt auf `null` verweist. Die Methode `strip` soll zusätzlich einen `int`-Wert zurückgeben, der die Anzahl der entfernten Elemente angibt. Kommt `null` nicht als Inhalt vor, soll die Liste unverändert bleiben.

```
public int strip()
```

```
{
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `boolean allIn(T[] test)`.

Die Methode `allIn` soll `true` zurückgeben, wenn alle Inhalte von `test` mindestens einmal in der ausführenden Liste vorkommen. Gehen Sie davon aus, dass `test` nicht auf `null` verweist und auch nur Werte enthält, die ungleich `null` sind. Die Vergleiche sollen mit der Methode `equals` vorgenommen werden.

```
public boolean allIn( T[] test )
```

```
{
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `void exchangeHalfs()`.

Die Methode `exchangeHalfs` soll die hinteren `size/2` Elemente vom Ende der Liste entfernen und in unveränderter Reihenfolge an den Anfang der Liste stellen.

```
public void exchangeHalfs()
```

```
{
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `void delete(T p1, T p2)`.

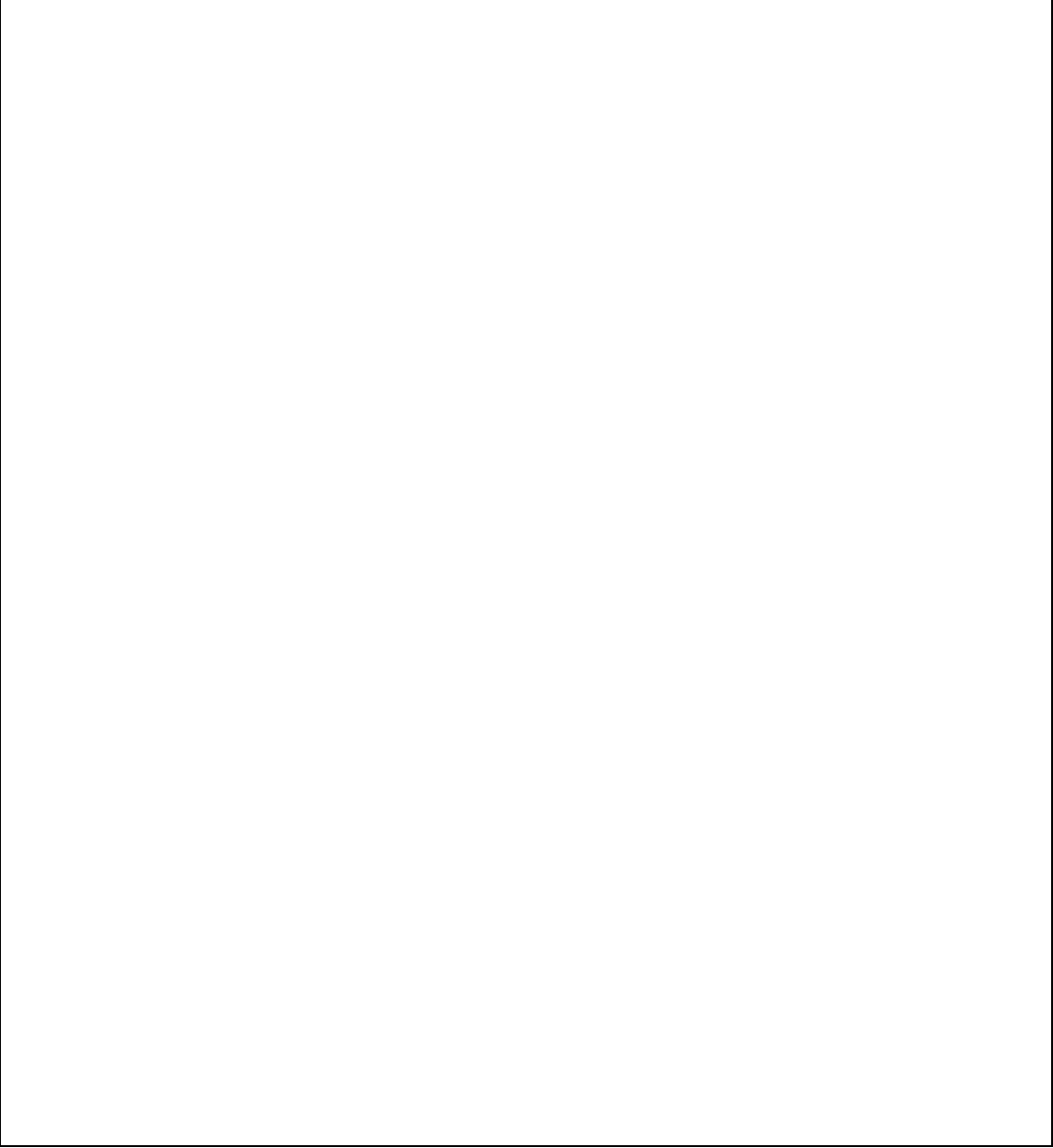
Falls `p1` und `p2` als Inhalte in der Liste vorkommen, soll die Methode `delete` alle Elemente aus der Liste entfernen, die zwischen dem ersten Vorkommen von `p1` und dem letzten Vorkommen von `p2` liegen.

Kommen `p1` oder `p2` nicht als Inhalt vor, soll die Liste unverändert bleiben. Gehen Sie davon aus, dass `p1` und `p2` nicht auf `null` verweisen. Die Vergleiche sollen mit der Methode `equals` vorgenommen werden.

```
public void delete( T p1, T p2 )  
{
```

Aufgabe 6

- a) Vervollständigen Sie die Methode `boolean noOneIn(T[] test)`. Die Methode `noOneIn` gibt `true` zurück, wenn kein Inhalt von `test` in der ausführenden Liste vorkommt. Sonst wird `false` zurückgegeben. Sind das Feld `test` oder die ausführende Liste leer, wird `true` zurückgegeben. Gehen Sie davon aus, dass `test` nur Werte enthält, die ungleich `null` sind. Die Vergleiche sollen mit der Methode `equals` vorgenommen werden.

```
public boolean noOneIn( T[] test )
{
    if ( size() > 0 && test.length > 0 )
    {
        
    }
    else
    {
        return true;
    }
}
```

Aufgabe 6

Vervollständigen Sie die Methode `void prepend(T[] toPrep)`.

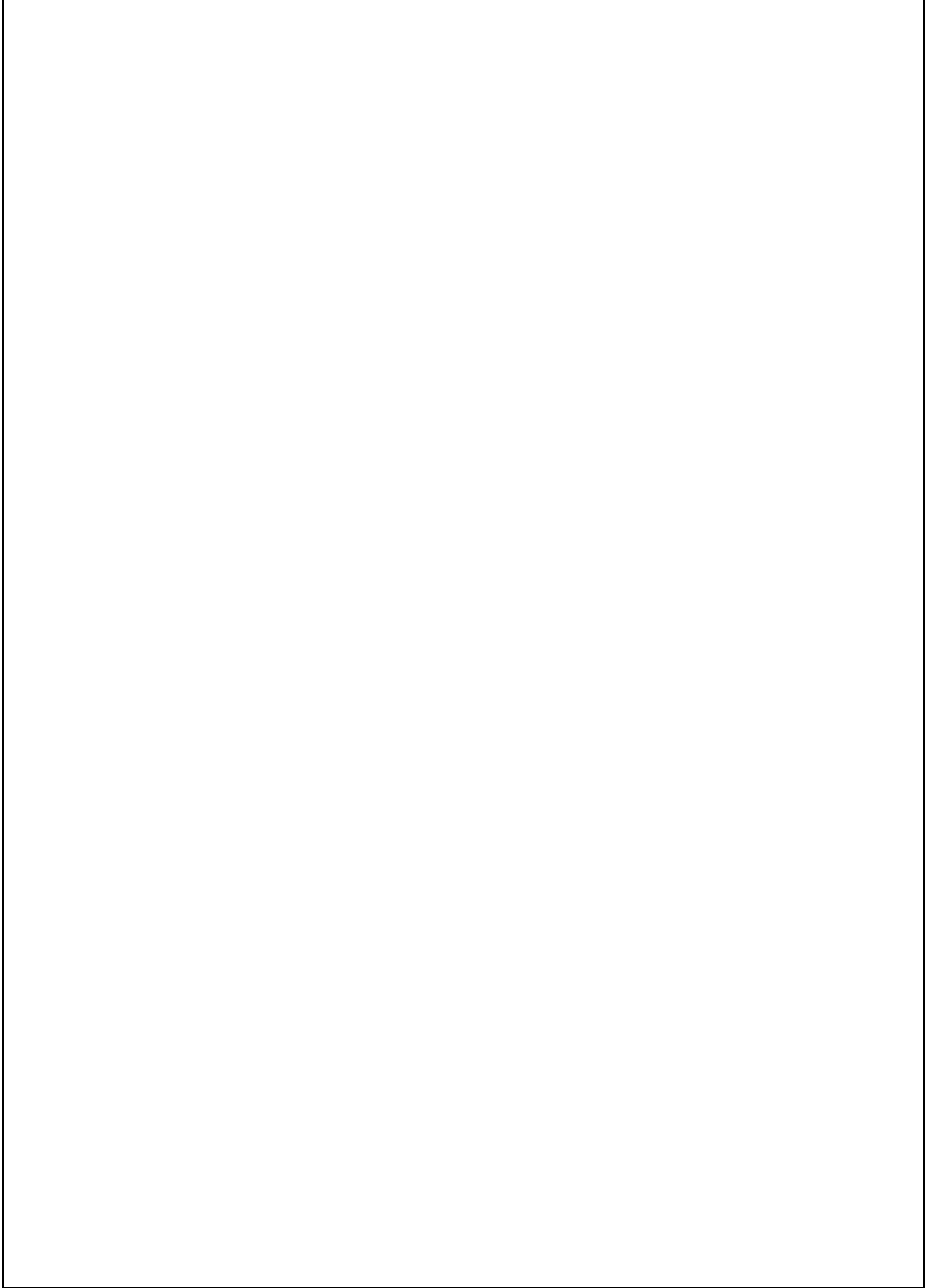
Die Methode `prepend` fügt die Elemente des Feldes `toPrep` in unveränderter Reihenfolge am Anfang der ausführenden Liste hinzu.

```
public void prepend( T[] toPrep )
```

```
{
```

```
    if ( toPrep.length > 0 )
```

```
    {
```



```
    }
```

```
}
```

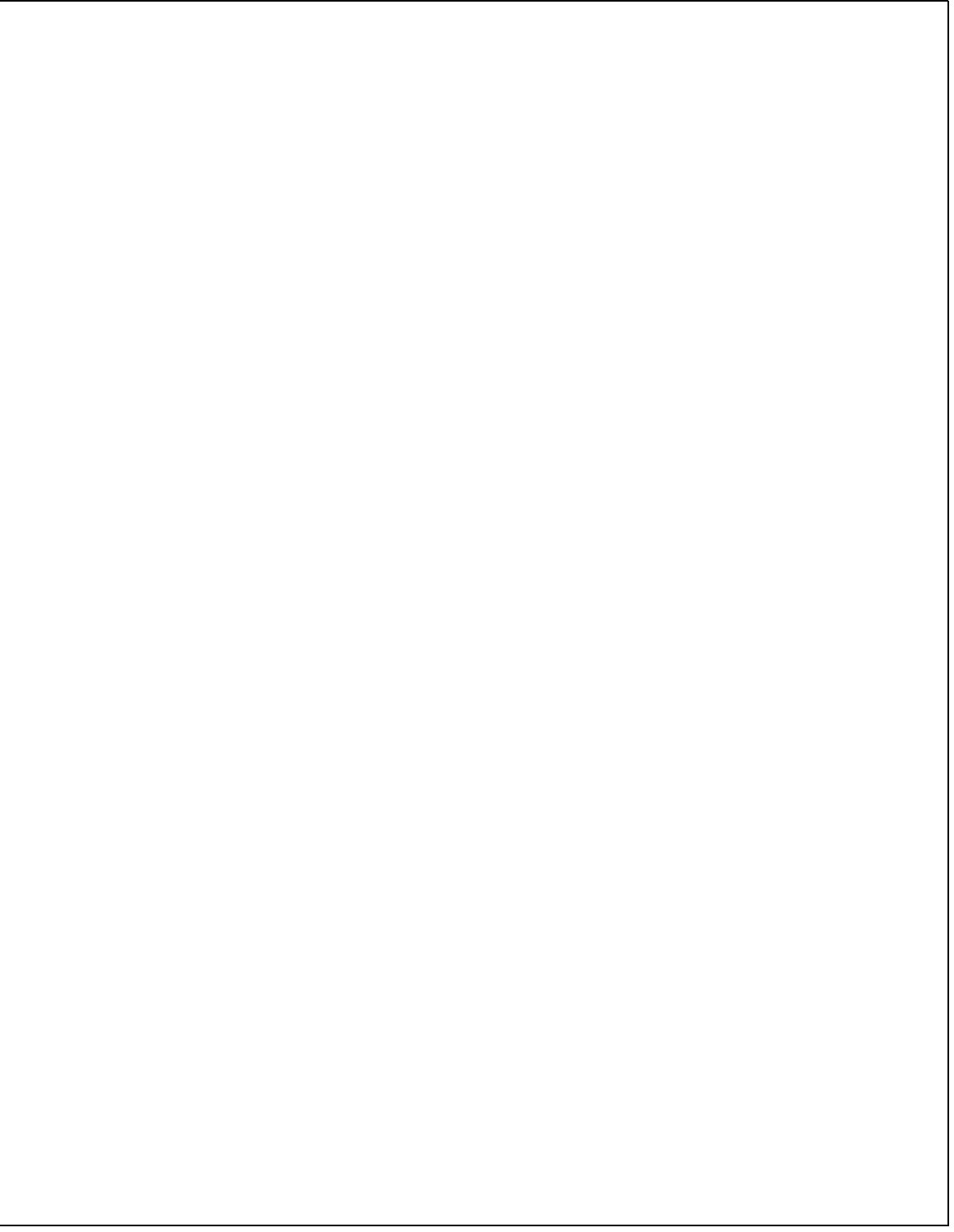
Aufgabe 6

Vervollständigen Sie die Methode `DoublyLinkedList<T> sub(T p)`.

Falls `p` als Inhalt in der Liste vorkommt, entfernt die Methode `sub` alle Elemente aus der ausführenden Liste, die *zwischen* dem ersten Element und dem letzten Vorkommen von `p` liegen. Kommt `p` nicht oder nur als Inhalt des ersten oder zweiten Elements vor, bleibt die Liste unverändert. Die Methode `sub` gibt die entfernten Elemente in unveränderter Reihenfolge als Liste zurück. Werden keine Elemente entfernt, wird eine leere Liste zurückgegeben.

Gehen Sie davon aus, dass `p` nicht auf `null` verweist. Vergleiche sollen mit der Methode `equals` vorgenommen werden.

```
public DoublyLinkedList<T> sub( T p )  
{
```



```
}
```

Aufgabe 7

Vervollständigen Sie die Methode `boolean unequals(DoublyLinkedList<T> param)`.
Die Methode `unequal` soll `true` zurückgeben, falls es keine Position gibt, an der der Inhalt der aufrufenden Liste und der Inhalt der als Argument übergebenen Liste gleich sind.
Sonst soll `false` zurückgegeben werden.
Der Vergleich soll mit der Methode `equals` vorgenommen werden.

```
public boolean unequals( DoublyLinkedList<T> param )
{
    if (  )
    {
        Element<T> ref1 =  ;
        Element<T> ref2 =  ;
        while (  )
        {

        }
    }
    return true;
}
```


Aufgabe 6

Vervollständigen Sie die Methode **void pack()**.

Die Methode **pack** soll Elemente in der Liste so löschen, dass von jeder Teilfolge von unmittelbar aufeinander folgenden Elementen mit gleichen Inhalten jeweils nur genau ein Element erhalten bleibt. Der Vergleich soll mit der Methode **equals** vorgenommen werden.

```
public void pack()
{
    Element<T> current = first;
    while (  )
    {
        
    }
}
```

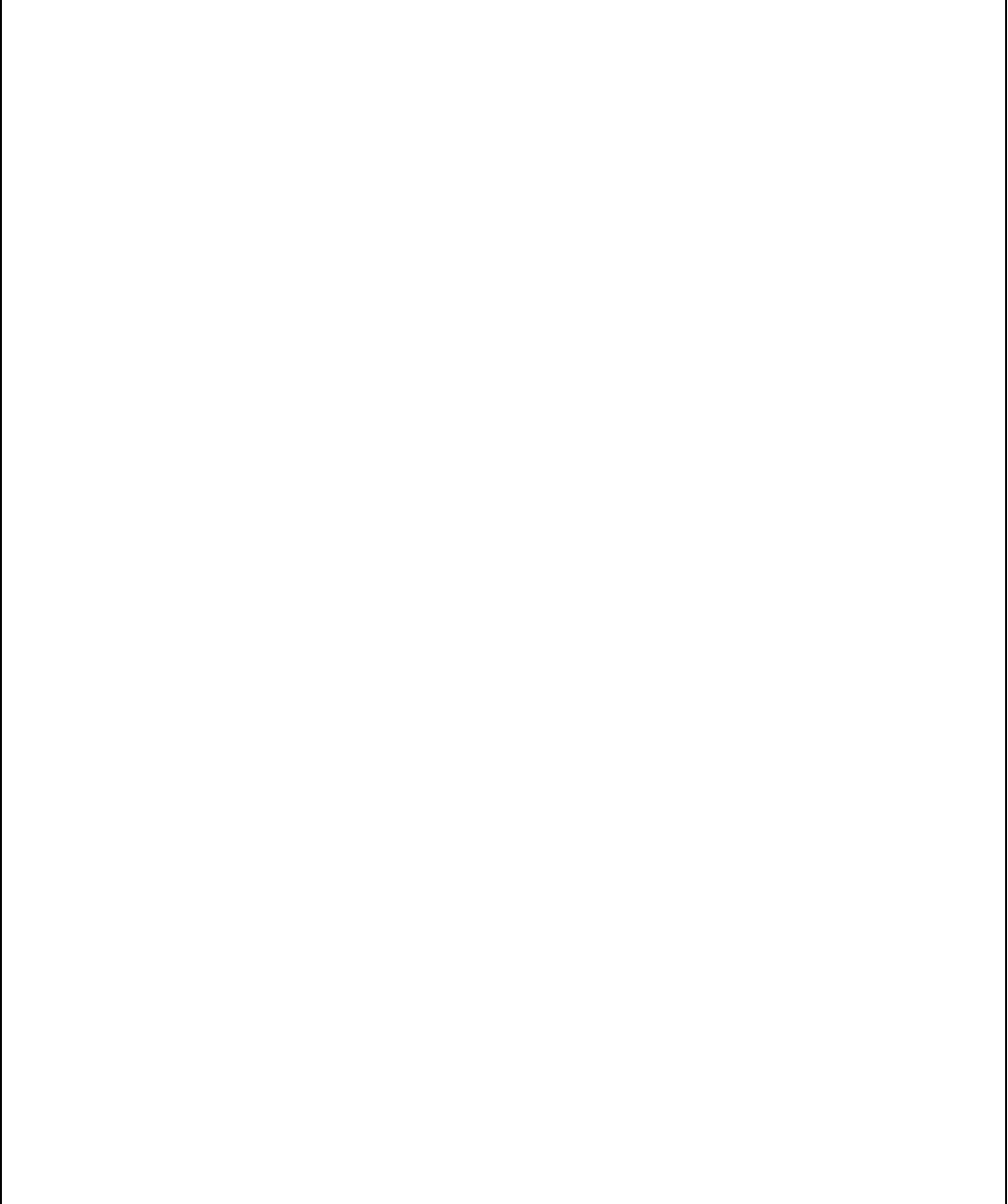
Aufgabe 6

Vervollständigen Sie den Konstruktor

`DoublyLinkedList(DoublyLinkedList<T>[] lists)`.

Der Konstruktor soll eine neue Liste erzeugen, die nacheinander die Elemente aller dem Parameter `lists` übergebenen Listen miteinander verknüpft.

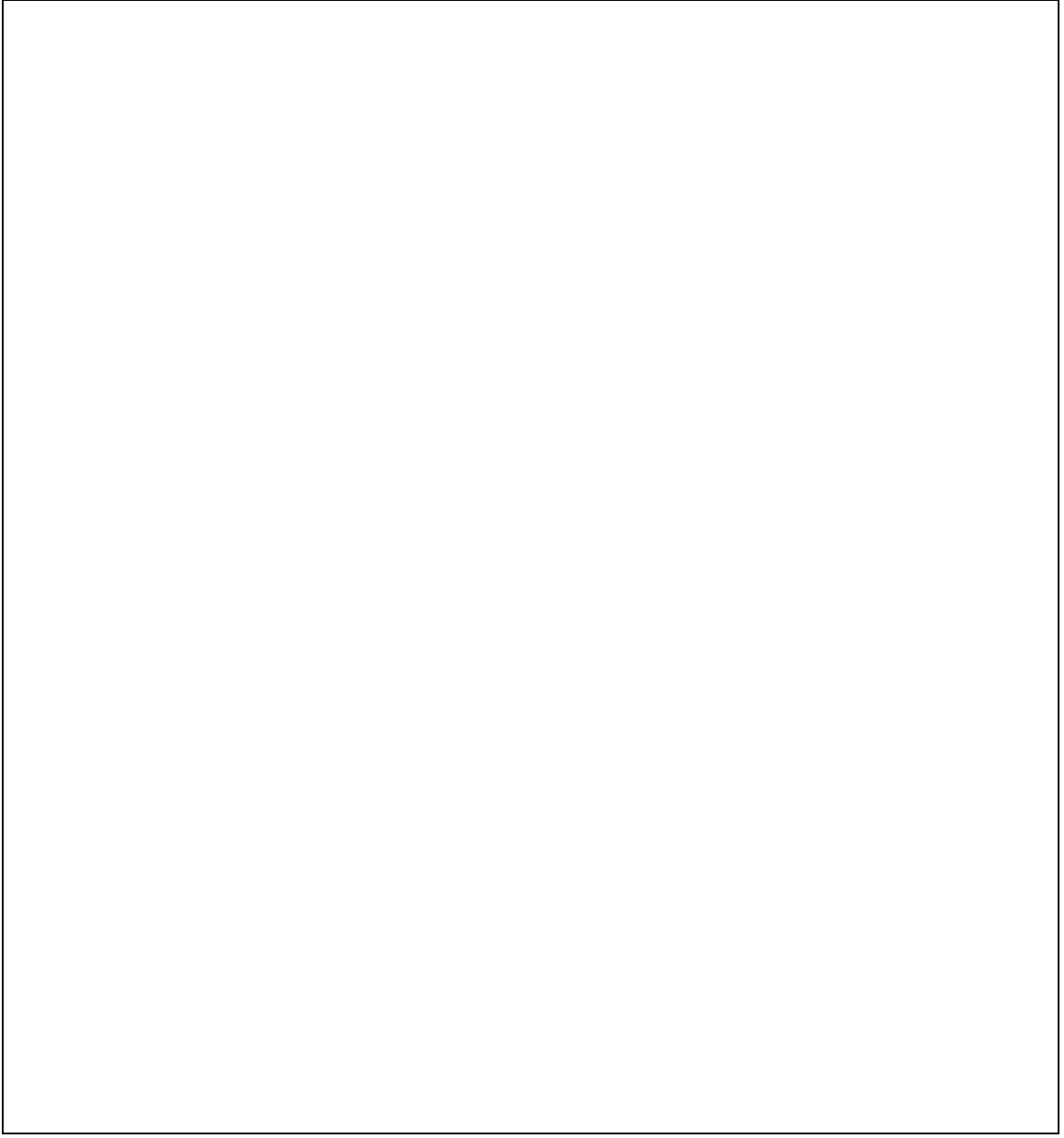
- Es sollen keine neuen Elemente erzeugt werden.
- Die als Argument übergebenen Listen sollen anschließend leer sein.
- Enthält das Feld keine Inhalte, so ist die erzeugte Liste leer.

```
public DoublyLinkedList( DoublyLinkedList<T>[] lists )
{
    this();
    if ( lists != null )
    {
        for ( DoublyLinkedList<T> cand : lists )
        {
            
        }
    }
}
```

Aufgabe 6

Vervollständigen Sie einen weiteren Konstruktor für die Klasse `DoublyLinkedList<T>`. Der Konstruktor soll eine Liste mit zwei Elementen anlegen, deren Inhalte auf die als Parameter übergebenen Objekte verweisen.

```
public DoublyLinkedList( T p1, T p2 )  
{
```



```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `void deleteHead(int n)`.

Die Methode `deleteHead` soll die ersten `n` Elemente aus der Liste löschen.

- Hat die Liste weniger als `n` oder genau `n` Elemente, so sollen alle Elemente gelöscht werden.
- Hat `n` einen negativen Wert, so soll nichts geschehen.

```
public void deleteHead( int n )
```

```
{
```

```
    if ( n > 0 )
```

```
    {
```

```
        if (
```

```
             )
```

```
        {
```

```
        }
```

```
    else
```

```
    {
```

```
    }
```

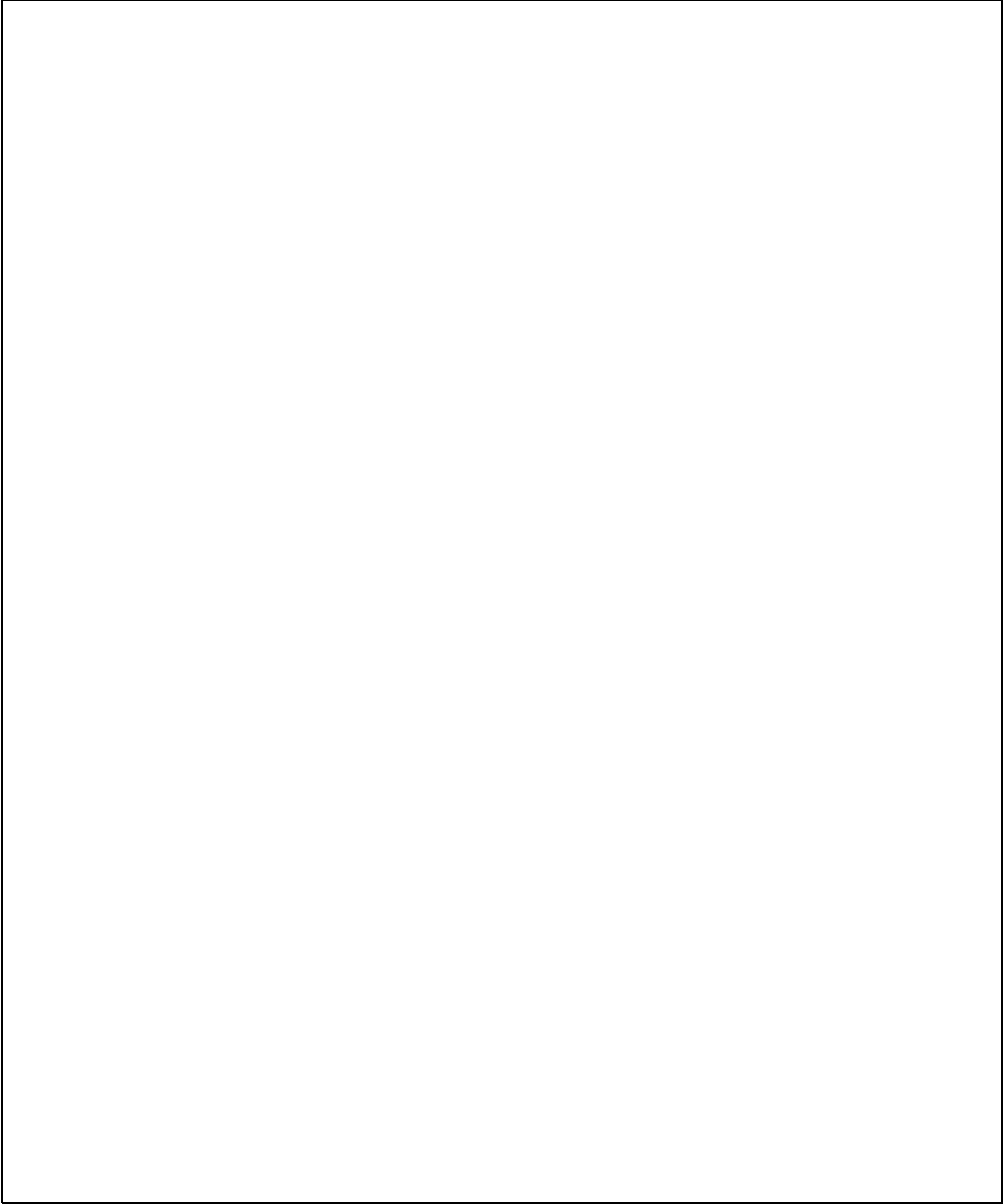
```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `boolean isDiff(DoublyLinkedList<T> list)`.

Die Methode soll `true` zurückgeben, falls kein Inhalt der aufrufenden Liste in der als Parameter `list` übergebenen Liste vorkommt. Sonst soll `false` zurückgegeben werden.

Die Gleichheit soll mit der Methode `equals` festgestellt werden.

```
public boolean isDiff( DoublyLinkedList<T> list )
{
    if ( list != null )
    {
        
    }
    return true;
}
```

Aufgabe 1

Vervollständigen Sie die Methode `reverse()`.

Die Methode `reverse` soll die Inhalte einer Liste umdrehen, indem diejenigen Elemente der Liste vertauscht werden, die vom Anfang und vom Ende der Liste aus den gleichen Abstand besitzen.

Es sollen also jeweils getauscht werden:

Die Inhalte des ersten und des letzten Elements, des zweiten und des vorletzten Elements, des dritten und des drittletzten Elements, usw.

```
public boolean reverse()
{
    Element<T> up = first;
    Element<T> down = last;

    for (  )
    {

    }
}
```

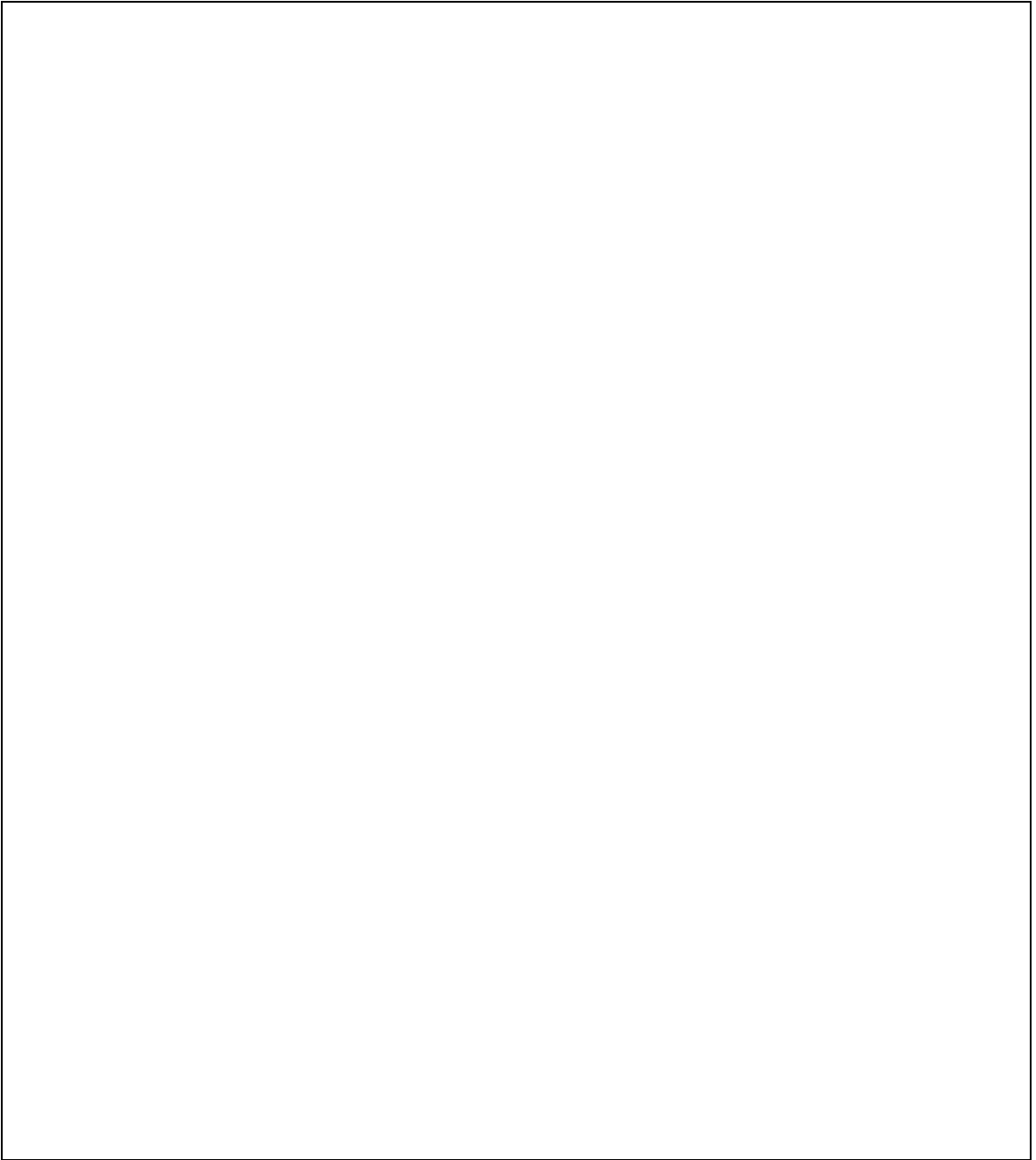
Aufgabe 6

Vervollständigen Sie die Methode `deleteNext(T content)`.

Die Methode `deleteNext` soll das **erste** Element in der Liste suchen, dessen Inhalt gleich dem übergebenen Argument ist. Das auf dieses Element folgende Element soll aus der Liste entfernt werden.

- Gibt es kein folgendes Element, so soll nichts geschehen.
- Die Gleichheit soll mit der Methode `equals` bestimmt werden.

```
public void deleteNext( T content )  
{
```



```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `singleton(int n)`.

Die Methode `singleton` soll eine neue Liste erzeugen und zurückgeben, die genau nur ein Element mit dem Inhalt des Elements an der Position `n` der aufrufenden Liste enthält. Dieses Element soll aus der aufrufenden Liste **nicht** entfernt werden.

- Der Aufruf `singleton(0)` soll also eine neue Liste mit dem Inhalt des ersten Elements erzeugen, der Aufruf `singleton(1)` mit dem Inhalt des zweiten Elements und so fort.
- Für Argumentwerte, die keine gültige Position kennzeichnen, wird eine `RuntimeException` geworfen.

```
public DoublyLinkedList<T> singleton( int n )
{
    if (  )
    {
        
    }
    else
    {
        throw new RuntimeException();
    }
}
```


Aufgabe 6

Vervollständigen Sie die Methode `boolean deleteTo(T obj)`.

Die Methode `deleteTo` soll alle Elemente der Liste vom Beginn bis vor das erste Auftreten des Inhalts `obj` aus der Liste entfernen. Kommt `obj` in der Liste nicht vor oder hat die Liste weniger als zwei Elemente, so soll sie unverändert bleiben. Die Gleichheit soll mit der Methode `equals` festgestellt werden.

Die Methode soll `true` zurückgeben, falls Elemente entfernt wurden, sonst `false`.

```
public boolean deleteTo( T obj )  
{
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `DoublyLinkedList<T> sel(DoublyLinkedList<T> list)`. Die Methode `sel` soll eine neu erzeugte Liste zurückgeben, die alle Inhalte enthält, die sowohl in der ausführenden als auch in der als Parameter `list` übergebenen Liste an genau der gleichen Position vorkommen. Gibt es keine solchen Inhalte oder besitzen die beiden Listen nicht die gleiche Länge, soll eine leere Liste zurückgegeben werden.

Die Gleichheit soll mit der Methode `equals` festgestellt werden.

```
public DoublyLinkedList<T> sel( DoublyLinkedList<T> list )
{
```

```
    if ( size == list.size )
    {
```

```
    }
```

```
    return
```

```
    ;
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `boolean deleteFrom(T obj)`.

Die Methode `deleteFrom` soll alle Elemente der Liste entfernen, die auf das letzte Auftreten des Inhalts `obj` folgen. Kommt `obj` in der Liste nicht vor, so soll die Liste unverändert bleiben.

Die Gleichheit soll mit der Methode `equals` festgestellt werden.

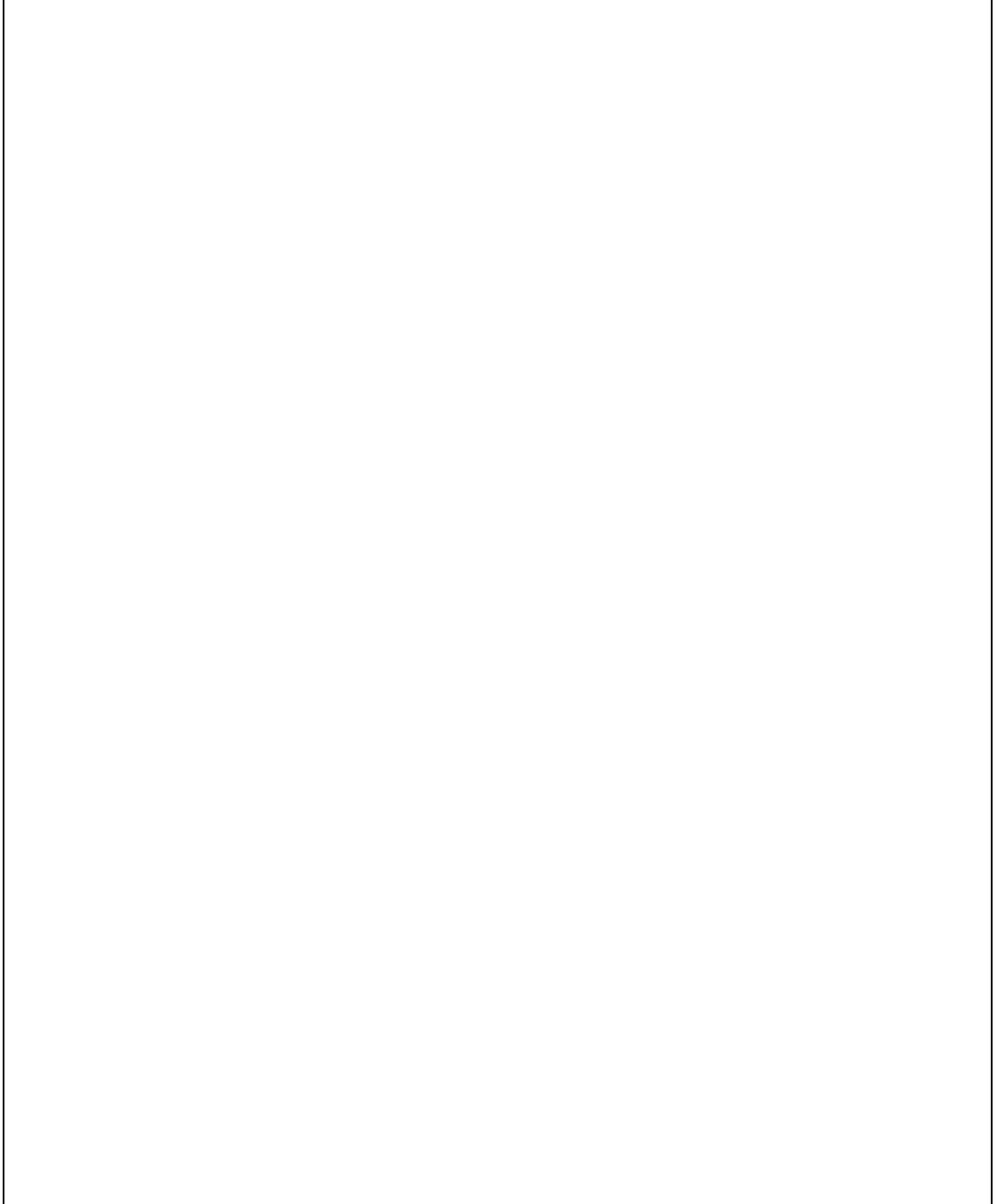
Die Methode soll `true` zurückgeben, falls tatsächlich Elemente entfernt wurden, sonst `false`.

```
public boolean deleteFrom( T obj )
```

```
{
```

```
    if ( obj != null )
```

```
    {
```



```
    }
```

```
    return false;
```

```
}
```

Aufgabe 6

Vervollständigen Sie für die Klasse `DoublyLinkedList<T>` die Methode `int equalHead(DoublyLinkedList<T> other)`.

Die Methode `equalHead` soll die Anzahl der unmittelbar aufeinander folgenden Elemente am Anfang der Liste `other` zurückgeben, die an genau der gleichen Position wie die ausführende Liste die gleichen Inhalte enthält. Ist eine der beiden Listen leer, soll `0` zurückgegeben werden.

Die Gleichheit der Inhalte soll mit der Methode `equals` festgestellt werden.

```
public int equalHead( DoublyLinkedList<T> other )
{
    if (  )
    {

    }
    else
    {
        return 0;
    }
}
```

Aufgabe 1

Vervollständigen Sie die Methode `asymmetric()`.

Die Methode `asymmetric` soll **true** zurückgeben, falls alle Paare von Elementen der Liste jeweils ungleiche Inhalte besitzen, die vom Anfang und vom Ende der Liste aus den gleichen Abstand haben. Der Vergleich soll mit der Methode `equals` vorgenommen werden.

Es sollen also paarweise verglichen werden:

Die Inhalte des ersten und des letzten Elements, des zweiten und des vorletzten Elements, des dritten und des drittletzten Elements, usw.

```
public boolean asymmetric()
{
    Element<T> up = first;
    Element<T> down = last;

    for (  )
    {

    }

    return true;
}
```

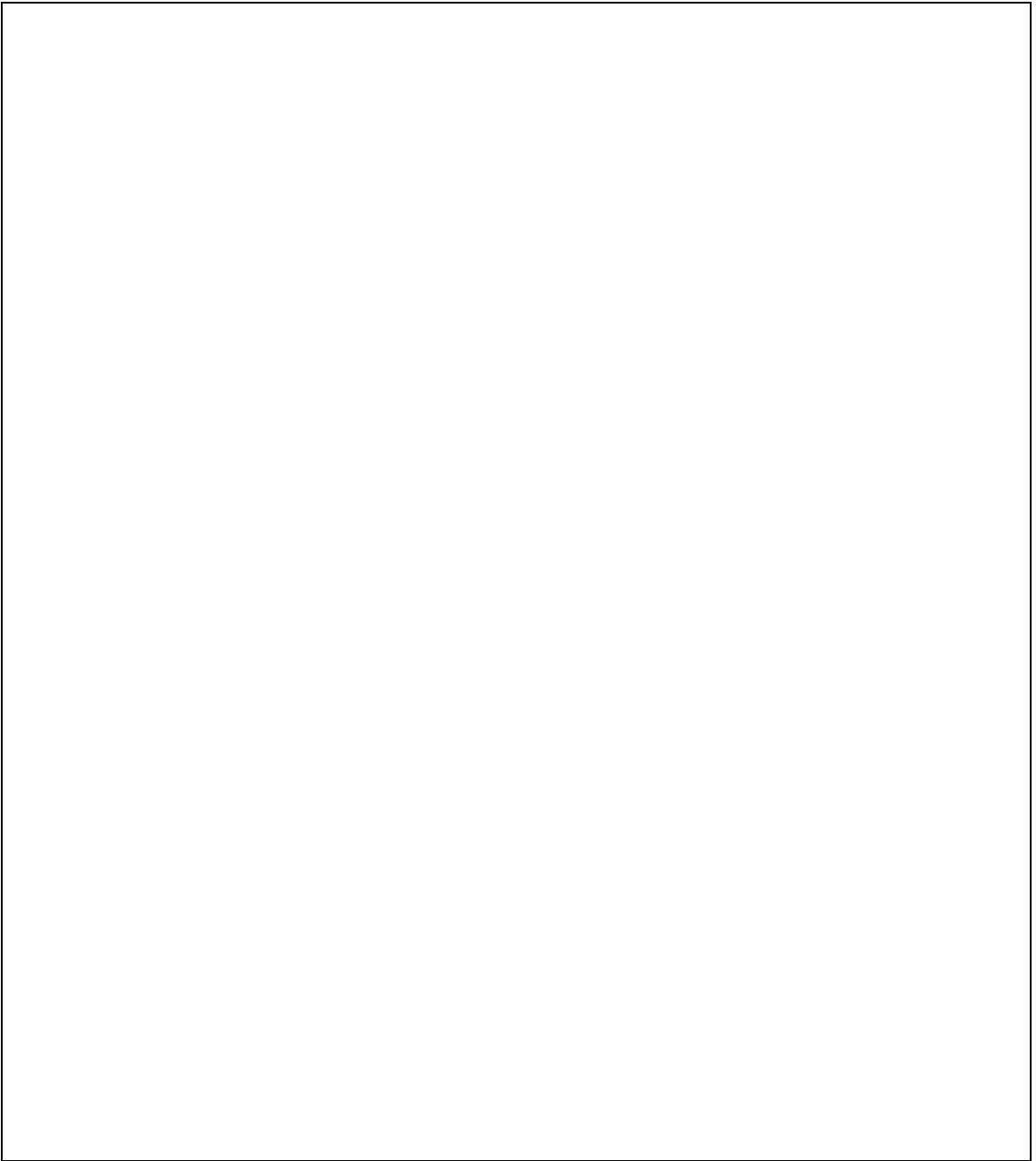
Aufgabe 6

Vervollständigen Sie die Methode `deleteInner(T content)`.

Die Methode `deleteInner` soll das am weitesten hinten stehende Element in der Liste suchen, dessen Inhalt gleich dem übergebenen Argument ist und das **nicht** das erste oder letzte Element der Liste ist. Dieses Element soll aus der Liste entfernt werden.

- Gibt es kein solches Element, so soll nichts geschehen.
- Die Gleichheit soll mit der Methode `equals` bestimmt werden.

```
public void deleteInner( T content )  
{
```



```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `lastTwo()`.

Die Methode `lastTwo` soll eine neue Liste erzeugen und zurückgeben, die genau nur die letzten beiden Elemente der aufrufenden Liste enthält. Diese Elemente sollen aus der aufrufenden Liste entfernt werden. Besitzt die aufrufende Liste keine zwei Elemente, so wird eine `RuntimeException` geworfen.

```
public DoublyLinkedList<T> lastTwo()
{
    if (  )
    {
        
    }
    else
    {
        throw new RuntimeException();
    }
}
```

Aufgabe 6

Vervollständigen Sie für die Klasse `DoublyLinkedList<T>` die Methode `int equalContent(DoublyLinkedList<T> other)`.

Die Methode `equalContent` soll die Anzahl der Elemente in der Liste `other` zurückgeben, die an genau der gleichen Position wie die ausführende Liste die gleichen Inhalte enthalten. Ist eine der beiden Listen leer, soll `0` zurückgegeben werden.

Die Gleichheit der Inhalte soll mit der Methode `equals` festgestellt werden.

```
public int equalContent( DoublyLinkedList<T> other )
{
    if (  )
    {

    }
    else
    {
        return 0;
    }
}
```


Aufgabe 6

Vervollständigen Sie die Methode `int substitute(T p1, T p2)`.

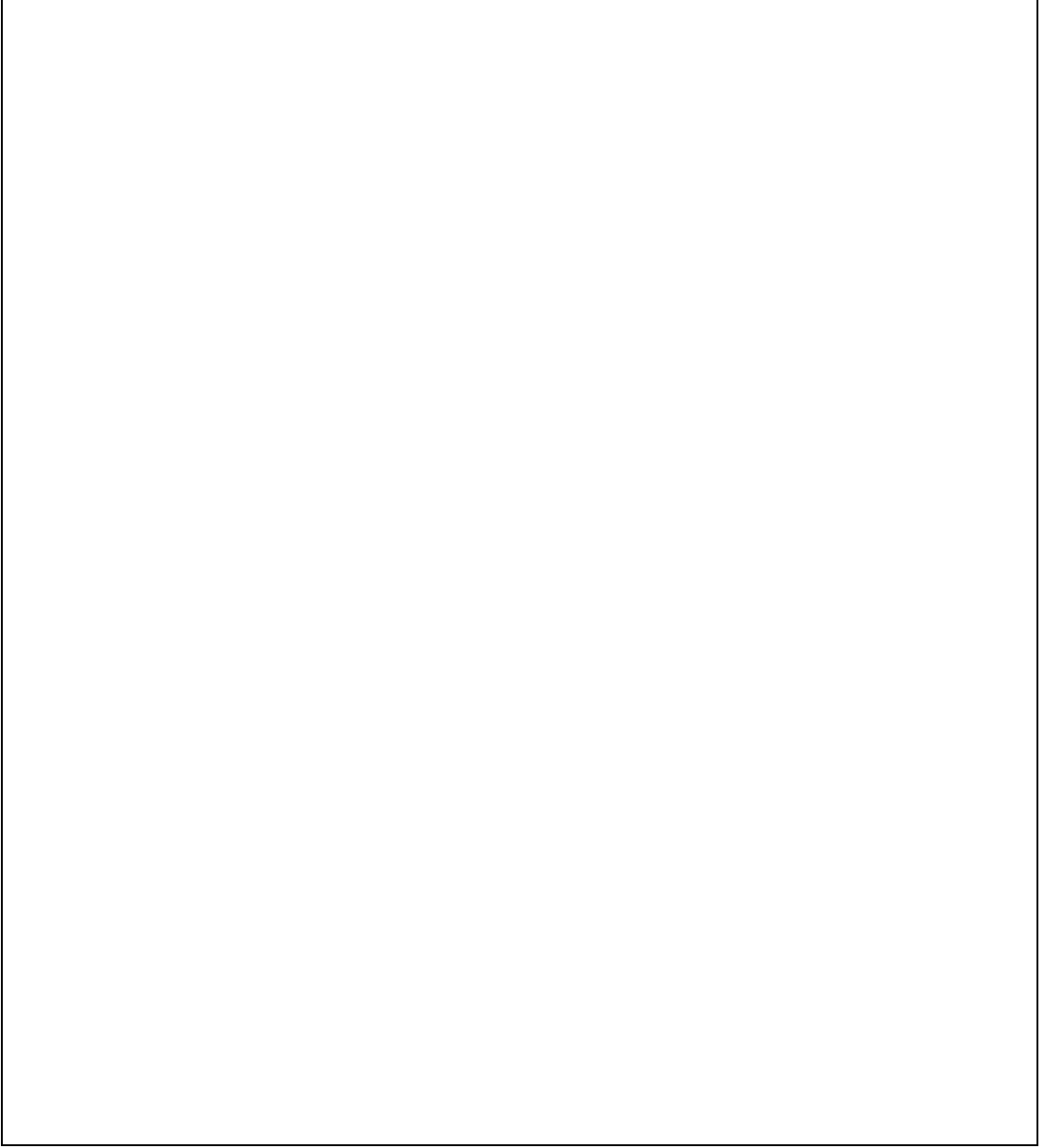
Die Methode `substitute` soll in der Liste jeden Inhalt `p1`, auf den nicht unmittelbar `p2` folgt, durch `p2` ersetzen. Kommt `p1` in der Liste nicht vor oder folgt auf jedes `p1` unmittelbar ein `p2`, so soll die Liste unverändert bleiben.

Die Methode soll die Anzahl der vorgenommenen Ersetzungen zurückgeben.

```
public int substitute( T p1, T p2 )  
{
```

```
    if ( obj != null )
```

```
{
```



```
}
```

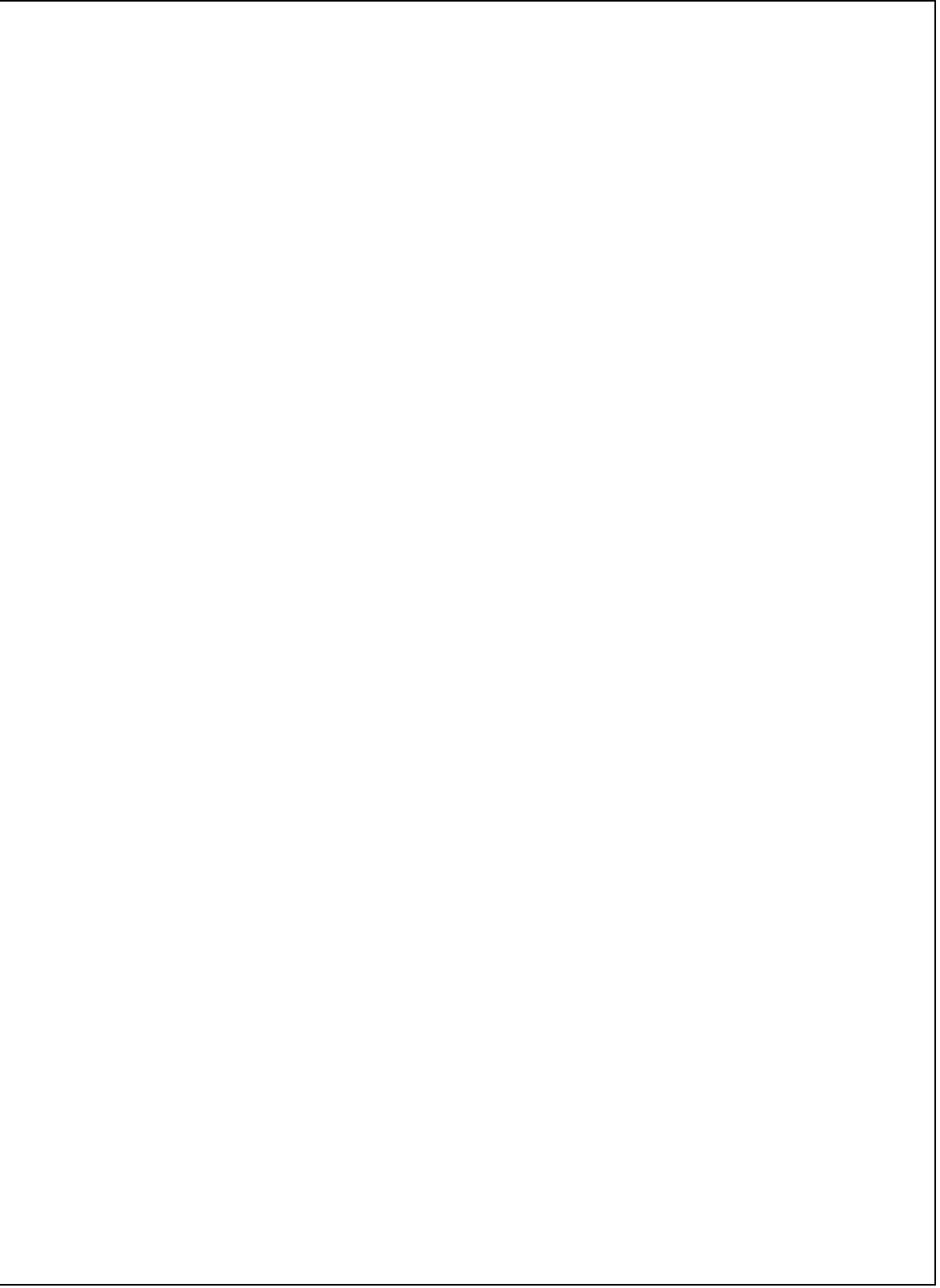
```
}
```

Aufgabe 6

Vervollständigen Sie für die Klasse `DoublyLinkedList<T>` die Methode `void removeMiddle()`.

Hat die Liste eine ungerade Anzahl von Elementen, so soll die Methode `removeMiddle` das mittlere Element löschen. Hat die Liste eine gerade Anzahl von Elementen, so soll die Methode `removeMiddle` die beiden mittleren Elemente löschen. Ist die Liste leer, soll sie unverändert bleiben.

```
public void removeMiddle()  
{
```



```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `void removeUnequalToFirst()`.

Die Methode `removeUnequalToFirst` soll alle Elemente aus der Liste löschen, deren Inhalt ungleich dem Inhalt des ersten Elements ist, sofern dieser nicht `null` ist. Der Vergleich soll mit der Methode `equals` erfolgen.

Ist die Liste leer, soll sie unverändert bleiben.

```
public void removeUnequalToFirst()
{
    if ( !isEmpty() && first.getContent() != null )
    {
        Element current = first.getSucc();
        while (  )
        {
            if ( ! first.getContent().equals( current.getContent() ) )
            {

            }
            else
            {

            }
        }
    }
}
```

Anhang – Programmcode der Klasse BinarySearchTree<T extends Comparable<T>>

```
public class BinarySearchTree<T extends Comparable<T>> {
    private T content;
    private BinarySearchTree<T> leftChild, rightChild;
    public BinarySearchTree() { ... }
    public T getContent() { ... }
    public boolean isEmpty() { ... }
    public boolean isLeaf() { ... }
}
```

Anhang – Programmcode der Klasse DoublyLinkedList<T>

```
public class DoublyLinkedList<T> {
    private Element first, last;
    private int size;
    public DoublyLinkedList() { ... }
    public int size() { ... }
    public boolean isEmpty() { ... }
    // Element
    private static class Element {
        private T content;
        private Element pred, succ;
        public Element( T c ) { ... }
        public T getContent() { ... }
        public void setContent( T c ) { ... }
        public boolean hasSucc() { ... }
        public Element getSucc() { ... }
        public void connectAsSucc( Element e ) { ... }
        public void disconnectSucc() { ... }
        public boolean hasPred() { ... }
        public Element getPred() { ... }
        public void connectAsPred( Element e ) { ... }
        public void disconnectPred() { ... }
    }
}
```

Anhang – Programmcodes der Interfaces

```
public interface Iterator<T> {
    public abstract boolean hasNext();
    public abstract T next();
}
```

```
public interface Iterable<T> {
    public abstract Iterator<T> iterator();
}
```

```
public interface Comparable<T> {
    public abstract int compareTo( T t );
}
```

// Der Rückgabewert ist positiv, falls das
// ausführende Objekt größer als t ist.