

| | |
|-------------------------|--------------------------------|
| Begonnen am | Dienstag, 9. April 2024, 12:25 |
| Status | Beendet |
| Beendet am | Dienstag, 9. April 2024, 15:25 |
| Verbrauchte Zeit | 3 Stunden |

Information




Willkommen bei der FOP Prüfung des Wise2023/2024 mit Safe Exam Browser am eigenen Computer

Ihnen steht STR+C und STRG+V zur Verfügung

Als Wörterbuch stehen zur Verfügung:

- [deepL](#) 
- [Pons](#) 

Als Taschenrechner steht der

- [Desmos Taschenrechner](#) 
- [web2.0 Rechner](#) 
- [Uni Tübingen TR](#) 

zur Verfügung.

Whiteboards für Schmierpapier:

- Mathematisches Whiteboard: <https://webdemo.myscript.com/views/math/index.html> 
- Text Whiteboard: <https://webdemo.myscript.com/views/text/index.html> 
- Diagramme: <https://webdemo.myscript.com/views/diagram/index.html#/> 
- Diagramme: <https://app.diagrams.net/> 

Information

Aufgabe 1 (16 Punkte)

Diese Aufgabe besteht aus drei Teilaufgaben.

Frage 1

Vollständig

Erreichbare Punkte: 7.00

1(a):

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie ein generisches Functional **public**-Interface **A** mit dem generischen Typparameter **C**. **C** ist beschränkt auf **Container** und alle Subtypen von **Container**.

Default-Methode **fnc**: ist generisch mit einem Typparameter **T**, der auf **Number** und die Subtypen von **Number** beschränkt ist. **fnc** verfügt über eine Rückgabe vom Typ **boolean**. **fnc** verfügt über zwei Parameter. (1) **ason** vom generischen Typ **HashSet**, welcher auf **Number** und alle Supertypen von **Number** beschränkt ist. (2) **t** vom formalen Typ **T**. Falls **t** nicht in **ason** erhalten ist, soll **t** dem **HashSet** hinzugefügt werden. Andernfalls soll **t** aus dem **HashSet** entfernt werden. Die Methode gibt die Information zurück, ob **t** in **ason** enthalten war.

Funktionale Methode **apply**: hat einen Parameter **f** vom formalen Typ **DoubleFunction** mit dem generischen Typen **Integer** und einen Parameter **d** vom Typ **Double**. **apply** hat einen Rückgabewert vom Typ **Integer**.

Hinweis: Sie müssen hier Restricted Genericity von Wildcards unterscheiden.

Erinnerung: Die Methode **contains** von **HashSet** prüft, ob der gegebene Parameter im spezifischen **HashSet** enthalten ist. Diese Information wird als **boolean** zurückgegeben. Hierfür benötigt **contains** den zu überprüfenden Parameter.

Die Methode **add** von **HashSet** fügt ihren Parameter dem **HashSet** hinzu. **add** benötigt hierfür den einzufügenden Parameter. **add** kann als **void**-Methode verwendet werden.

Die Methode **remove** von **HashSet** entfernt den übergebenen Parameter aus dem **HashSet**. **set** kann als **void**-Methode verwendet werden.

```
public interface A <C extends Container>{
    public default <T extends Number> boolean fnc(HashSet<? super Number> ason, T t){
        if(!(ason.contains(t))){
            ason.add(t);
        }
        else{
            ason.remove(t);
        }
        return ason.contains(t);
    }
    public Integer apply(DoubleFunction<Integer> f, Double d);
}
```

Frage 2

Vollständig

Erreichbare Punkte: 5.00

1(b):

Erinnerung 1: Der Typparameter von **A** aus Teilaufgabe (a) heißt **C** und ist auf **Container** und alle Subtypen von **Container** beschränkt.

Erinnerung 2: Methode **apply** von **A** hat einen Parameter **f** vom formalen Typ **DoubleFunction** mit dem Typen **Integer** und einen Parameter **d** vom Typ **Double**. **apply** hat einen Rückgabewert vom Typ **Integer**. Die Anwendungsfunktion von **DoubleFunction** heißt **apply**.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public**-Klasse **B**, die das Interface **A** und das Interface **ContainerListener** (**java.awt.event.ContainerListener**) implementiert.

Methode **apply** wendet den ersten Parameter auf den zweiten an und liefert das Resultat zurück. Methoden des Interface **ContainerListener** sind in **B** nicht implementiert.

```
public abstract class B<C extends Container> implements A<C>, ContainerListener{
    public Integer apply(DoubleFunction<Integer> f, Double d){
        return f.apply(d);
    }
}
```

Frage 3

Vollständig

Erreichbare Punkte: 4.00

1(c)

Erinnerung 1: Der Typparameter von **A** aus Teilaufgabe (a) heißt **C** und ist auf **Container** und alle Subtypen von **Container** beschränkt.

Erinnerung 2: **B** implementiert **ContainerListener**. Die Methode **componentAdded** von **ContainerListener** hat einen Parameter **e** vom formalen Typ **ContainerEvent** und verfügt über keine Rückgabe. Die parameterlose Methode **getId** von **ContainerEvent** hat den Rückgabetyt **int**.

Hinweis 1: Das Interface **ContainerListener** verfügt neben der Methode **componentAdded** zusätzlich über die Methode **componentRemoved**. **componentRemoved** soll in der implementierten Klasse nicht integriert werden.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public-Klasse D**, die direkt von **B** abgeleitet ist und ein **private**-Objektattribut **id** vom primitiven Datentyp **int** hat. Implementieren Sie die Methode **componentAdded** von **ContainerListener** so, dass sie die Rückgabe der Methode **getId** des **ContainerEvent** nach **id** kopiert.

```
public class D extends B<C extends Container> {  
    private int id;  
    public void componentAdded(ContainerEvent e){  
        id = e.getId();  
    }  
}
```

Information**Aufgabe 2 (22 Punkte)**

Diese Aufgabe besteht aus drei Teilaufgaben.

Frage 4

Vollständig

Erreichbare Punkte: 5.00

2(a):

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Gegeben seien eine direkt von Klasse **Exception** abgeleitete Klasse **Exc1** und eine von **Exc1** direkt abgeleitete Klasse **Exc2**. Dabei habe **Exc1** einen parameterlosen **public**-Konstruktor und **Exc2** habe einen **public**-Konstruktor mit einem Parameter vom formalen Typ **int**.

Schreiben Sie eine **public**-Klasse **Exc3**, die direkt von **Exc2** abgeleitet ist. Klasse **Exc3** soll einen **public**-Konstruktor mit einem Parameter **lst** vom formalen Typ **List<Integer>** haben. Falls **lst** gleich **null** ist oder -- im Fall ungleich **null** -- die Liste **lst** leer ist, soll der Konstruktor von **Exc3** den Konstruktor von **Exc2** mit 0 aufrufen, andernfalls mit dem ersten Element in **lst**.

Hinweis: Wie Sie wissen, darf **super** nur einmal aufgerufen werden. Sie müssen also die Fallunterscheidung in den aktuellen Parameter von **super** integrieren.

Erinnerung: Zur Überprüfung, ob eine Liste leer ist, hat **List<E>** die parameterlose boolesche Methode **isEmpty**.

Die Methode **get** von **List<E>** liefert das Element an dem Index zurück, der durch den aktuellen Parameter angegeben ist (wie üblich beginnend mit 0).

```
// Exc1 extends Exception ->Exc1()
// Exc2 extends Exc1 -> Exc2(int i)
public class Exc3 extends Exc2 {
    public Exc3(List<Integer> lst){
        super( lst == null || lst.isEmpty() ? 0 : lst.get(0) );
    }
}
```

Frage 5

Vollständig

Erreichbare Punkte: 8.00

2(b):

Erinnerung: **Exc1** ist direkt von **Exception**, **Exc2** ist direkt von **Exc1** und **Exc3** ist direkt von **Exc2** abgeleitet. Klasse **Exc1** hat einen parameterlosen **public**-Konstruktor, **Exc2** einen mit einem Parameter vom formalen Typ **int** und **Exc3** einen mit einem Parameter vom formalen Typ **List<Integer>**. Klasse **Optional<T>** hat eine parameterlose Methode **isEmpty** sowie eine parameterlose Methode **get**, die bei einem nicht-leeren **Optional**-Objekt einen Verweis auf das enthaltene **T**-Objekt zurückliefert. Die parameterlose Methode **intValue** von **Integer** liefert die eingekapselte Zahl zurück.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public**-Klassenmethode **m** mit Rückgabotyp **int**, einem Parameter **lst** vom formalen Typ **List<Integer>** und einem Parameter **i** vom formalen Typ **Optional<Integer>**. Die Klasse, zu der **m** gehört, müssen Sie nicht schreiben.

Falls **lst** nicht auf ein Objekt verweist, soll **m** eine **Exc1** werfen. Andernfalls: Falls **i** nicht auf ein Objekt verweist oder leer ist, wirft **m** eine **Exc2**, wobei der Konstruktor von **Exc2** mit 1 aufgerufen wird. Im verbliebenen Fall soll Folgendes passieren: Falls die in **i** enthaltene Zahl kleiner als 0 ist, soll **m** eine **Exc3** mit **lst** werfen; andernfalls liefert **m** die in **i** enthaltene Zahl zurück.

Verbindliche Anforderung: Die **throws**-Klausel soll genau eine Exception-Klasse enthalten. Diese darf aber nicht Klasse **Exception** oder Klasse **Throwable** sein (sonst Punktabzug).

```
public static int m(List<Integer> lst, Optional<Integer> i) throws Exc1 {  
    if (lst == null){  
        throw new Exc1();  
    }  
    if(i == null || i.isEmpty()){  
        throw new Exc2(1);  
    }  
    if (i.get().intValue() < 0){  
        throw new Exc3(lst);  
    }  
    return i.get().intValue;  
}
```

Frage 6

Vollständig

Erreichbare Punkte: 8.00

2(c):

Erinnerung: **Exc1** ist direkt von **Exception**, **Exc2** ist direkt von **Exc1** und **Exc3** ist direkt von **Exc2** abgeleitet. Klasse **Exc1** hat einen parameterlosen **public**-Konstruktor, **Exc2** einen mit einem Parameter vom formalen Typ **int** und **Exc3** einen mit einem Parameter vom formalen Typ **List<Integer>**. Die Klassenmethode **m** aus Teilaufgabe (b) hat Rückgabotyp **int**, einen Parameter vom formalen Typ **List<Integer>** und einen zweiten Parameter vom formalen Typ **Optional<Integer>**. Die Klassenmethode **of** von **Optional<T>** erzeugt ein **Optional<T>**-Objekt mit ihrem Parameter vom formalen Typ **T**.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public**-Objektmethode **m2** mit Rückgabotyp **int** und einem Parameter **lst** vom formalen Typ **List<Integer>**. Methode **m2** gehört nicht zur selben Klasse wie Methode **m**. Die Klasse, zu der **m** gehört, heiße **X**. Die Klasse, zu der **m2** gehört, müssen Sie nicht schreiben.

In **m2** wird die Methode **m** aus Teilaufgabe (b) aufgerufen. Der erste aktuelle Parameter bei diesem Aufruf ist **lst**; der zweite aktuelle Parameter ist nicht leer und enthält den Wert 1. Falls **m** bei diesem Aufruf eine **Exc1** wirft, soll **m2** eine **Exc3** mit dem Wert **lst** werfen. Falls **m** hingegen eine **Exc3** wirft, soll **m2** den Wert -1 zurückliefern. Im verbliebenen Fall, dass **m** keine Exception wirft, soll **m2** die Rückgabe von **m** zurückliefern.

Achtung: Denken Sie über die Reihenfolge der **catch**-Blöcke nach!

Verbindliche Anforderung: Die **throws**-Klausel soll genau eine Exception-Klasse enthalten. Diese darf aber nicht Klasse **Exception** oder Klasse **Throwable** sein (sonst Punktabzug).

```
public int m2(List<Integer> lst) throws Exc3 {
    try {
        return X.m(lst,Optional<Integer>.of(new Integer(1)));
    }
    catch(Exc3 e){
        return -1;
    }
    catch (Exc1 e){
        throw new Exc3(lst);
    }
}
```

Information

Aufgabe 3 (39 Punkte)

Diese Aufgabe besteht aus drei Teilaufgaben.

Frage 7

Vollständig

Erreichbare Punkte: 15.00

3(a):

Folgende Funktion **foo**, gegeben zwei Listen **l1** und **l2**, erstellt eine neue Liste **l**. Jedes Element an Position **p** der Liste **l** ist das Element aus Liste **l1** an Position **p** mit entweder -1, 1 oder 0 multipliziert; je nachdem, ob die Summe der Elemente von **l2** von Position **p** bis ans Ende von **l2** negativ, positiv oder Null ist. Sollte die Länge der Liste **l2** **p** unterschreiten, wird jedes Element aus **l1** ab Position **p** mit Null multipliziert. Dementsprechend hat die Ergebnisliste **l** die gleiche Länge wie **l1**.

Gegeben sei folgende Funktion in Racket:

```
( define ( foo l1 l2 )
  ( local
    (
      ( define ( bar l2 acc v)
        ( cond
          [ ( and ( empty? l2 ) ( = acc 0 ) ) 0 ]
          [ ( and ( empty? l2 ) ( > acc 0 ) ) v ]
          [ ( and ( empty? l2 ) ( < acc 0 ) ) (- v) ]
          [ else ( bar ( rest l2 ) ( + acc ( first l2 ) ) v ) ]
        )
      )
    )
  ( cond
    [ ( empty? l1 ) empty ]
    [ ( empty? l2 ) ( cons 0 ( foo ( rest l1) l2 ) ) ]
    [ else ( cons ( bar l2 0 ( first l1 ) ) ( foo ( rest l1 ) ( rest l2 ) ) ) ]
  )
)
```

Folgende Klasse kennen Sie aus der Vorlesung und den Übungen:

```
public class ListItem<T> {
    public T key;
    public ListItem<T> next;
}
```

Sie können davon ausgehen, dass **ListItem<T>** zwei Konstruktoren zur Verfügung stellt. Einen parameterlosen Konstruktor, der **key** sowie **next** auf **null** initialisiert, sowie einen Konstruktor mit einem Parameter **T t**, der **key** auf **t** und **next** auf **null** initialisiert. Es ist Ihnen freigestellt, welchen von beiden Konstruktoren Sie zur Lösung der Aufgabe verwenden.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie in Java **public**-Objektmethoden **foo** und **bar** analog zum Racket-Code oben. Diese zwei Methoden gehören zur selben Klasse (diese Klasse müssen Sie nicht selbst schreiben). Die formalen Parametertypen von **foo** sind allesamt **ListItem<Integer>**. Die formalen Parametertypen von **bar** sind vom Typ **ListItem<Integer>** für **l2** und **Integer** für **acc** und **v**. Der Rückgabotyp von **foo** ist **ListItem<Integer>**, der von **bar** ist **Integer**. Im Gegensatz zum Racket-Code oben ist die Java-Methode **bar** nicht in **foo** definiert, sondern die beiden Methoden folgen aufeinander. Sie können davon ausgehen, dass die beiden Parameter von **foo** auf korrekt gebildete lineare Listen verweisen, die zwar leer sein können, aber niemals ein Element mit Wert **null** enthalten.

Verbindliche Anforderungen:

1. Die Methoden **foo** und **bar** sind rein rekursiv, das heißt, Schleifen sind nicht erlaubt.
2. Übersetzungen in andere Datenstrukturen (Arrays, Streams etc.) sind nicht erlaubt.
3. Die Eingabelisten **l1** und **l2** von **foo** dürfen in **foo** und **bar** nicht verändert werden. Das heißt: Sie müssen die **ListItem<Integer>**-Objekte in der Ergebnisliste mit Operator **new** erzeugen (die **key**-Werte dürfen Sie hingegen mit "=" kopieren).

```
public ListItem<Integer> foo( ListItem<Integer> l1, ListItem<Integer> l2){
    if (l1 == null){
        return null;
    }
    if(l2 == null){
        ListItem<Integer> copyL1 = new ListItem<>();
        copyL1.key = l1.key
```

```
copyL1.next = l1.next;
ListItem<Integer> copyL2 = new ListItem<>();
copyL2.key = l2.key;
copyL2.next = l2.next;
copyL2 = l2;

    ListItem<Integer> tmp = new ListItem<>();
    tmp.key = 0;
    tmp.next = foo(copyL1.next, copyL2);
    return tmp;
}

ListItem<Integer> copy1 = new ListItem<>();
copy1.key = l1.key;
copy1.next = l1.next;
l1 = copy;
ListItem<Integer> copy2 = new ListItem<>();
copy2.key = l2.key;
copy2.next = l2.next;
copy2 = l2;

    ListItem<Integer> temp = new ListItem<>();
    temp.key = bar(copy2,0,copy1.key);
    temp.next = foo(copy1.next, copy2.next);
    return temp;
}

public Integer bar (ListItem<Integer> l2,Integer acc, Integer v){
    if(l2 == null && accu == 0){
        return 0;
    }
    if(l2 == null && accu > 0){
        return v;
    }
    if(l2 == null && accu < 0){
        return -v;
    }
    ListItem<Integer> tmpTwo = new ListItem<Integer>();
    tmpTwo.key = l2.key;
    tmpTwo.next = l2.next;
    return bar(tmpTwo.next,accu + tmpTwo.key, v);
}
```

Frage 8

Vollständig

Erreichbare Punkte: 14.00

3(b):

Folgende Funktion **foo**, gegeben zwei Listen **l1** und **l2**, erstellt eine neue Liste **l**. Jedes Element an Position **p** der Liste **l** ist das Element aus Liste **l1** an Position **p** mit entweder -1, 1 oder 0 multipliziert; je nachdem, ob die Summe der Elemente von **l2** von Position **p** bis ans Ende von **l2** negativ, positiv oder Null ist. Sollte die Länge der Liste **l2** **p** unterschreiten, wird jedes Element aus **l1** ab Position **p** mit Null multipliziert. Dementsprechend hat die Ergebnisliste **l** die gleiche Länge wie **l1**.

Gegeben sei folgende Funktion in Racket:

```
( define ( foo l1 l2 )
  ( local
    (
      ( define ( bar l2 acc v)
        ( cond
          [ ( and ( empty? l2 ) ( = acc 0 ) ) 0 ]
          [ ( and ( empty? l2 ) ( > acc 0 ) ) v ]
          [ ( and ( empty? l2 ) ( < acc 0 ) ) (- v) ]
          [ else ( bar ( rest l2 ) ( + acc ( first l2 ) ) v ) ]
        )
      )
    )
  ( cond
    [ ( empty? l1 ) empty ]
    [ ( empty? l2 ) ( cons 0 ( foo ( rest l1) l2 ) ) ]
    [ else ( cons ( bar l2 0 ( first l1 ) ) ( foo ( rest l1 ) ( rest l2 ) ) ) ]
  )
)
```

Folgende Klasse kennen Sie aus der Vorlesung und den Übungen:

```
public class ListItem<T> {
    public T key;
    public ListItem<T> next;
}
```

Sie können davon ausgehen, dass **ListItem<T>** zwei Konstruktoren zur Verfügung stellt. Einen parameterlosen Konstruktor, der **key** sowie **next** auf **null** initialisiert, sowie einen Konstruktor mit einem Parameter **T t**, der **key** auf **t** und **next** auf **null** initialisiert.

Es ist Ihnen freigestellt, welchen von beiden Konstruktoren Sie zur Lösung der Aufgabe verwenden.

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie in Java **public**-Objektmethoden **foo** und **bar** analog zum Racket-Code oben. Diese zwei Methoden gehören zur selben Klasse (diese Klasse müssen Sie nicht selbst schreiben). Die formalen Parametertypen von **foo** sind beide **ListItem<Integer>**. Gleiches gilt für **l2** von **bar**. Die Parameter **acc** und **v** sind vom Typ **Integer**. Der Rückgabotyp von **foo** ist **ListItem<Integer>**, der von **bar** ist **Integer**. Die Java- Methode **bar** ist nicht in **foo** definiert, sondern die beiden Methoden folgen aufeinander. In **foo** verwalten Sie zweckmäßigerweise zwei Verweise, **head** und **tail**, auf Anfang und Ende der Ergebnisliste, falls diese nicht leer ist. Sie können davon ausgehen, dass die beiden Parameter von **foo** auf korrekt gebildete lineare Listen verweisen, die zwar leer sein können, aber niemals ein Element mit Wert **null** enthalten.

Verbindliche Anforderungen:

1. Die Methoden **foo** und **bar** sind rein iterativ, das heißt, Rekursion ist nicht erlaubt.
2. Übersetzungen in andere Datenstrukturen (Arrays, Streams etc.) sind nicht erlaubt.
3. Die Eingabelisten **l1** und **l2** von **foo** dürfen in **foo** und **bar** nicht verändert werden. Das heißt: Sie müssen die **ListItem<Integer>**-Objekte in der Ergebnisliste mit Operator **new** erzeugen (die **key**-Werte dürfen Sie hingegen mit "=" kopieren).

```
public ListItem<Integer> foo(ListItem<Integer> l1, ListItem<Integer> l2){
    ListItem<Integer> head = new ListItem<>();
    ListItem<Integer> tail = new ListItem<>();
    ListItem<Integer> p = new ListItem<>();
    while(p != null){
        if (head == null){
```

```
ListItem<Integer> tmp = new ListItem<>();
head = tmp;
tail = tmp;
}
head.key = bar(l2, 0, p.key);
head.next = p.next;
head = p;
tail = p.next.next;
p = p.next;
}
}

public Integer bar(ListItem<Integer> l2, Integer accu, Integer v){
int result = 0;
if (l2 == null && accu == 0){
return 0;
}
if (l2 == null && accu > 0){
return v;
}
if(l2 == null && accu < 0){
return -v;
}
ListItem<Integer> p = new ListItem<>();
p = l2;
while(p != null){
result = p.key + acc + v;
p = p.next;
}
return result;
}
```

Frage 9

Vollständig

Erreichbare Punkte: 11.00

3(c):

Folgende Funktion **foo**, gegeben zwei Listen **l1** und **l2**, erstellt eine neue Liste **l**. Jedes Element an Position **p** der Liste **l** ist das Element aus Liste **l1** an Position **p** mit entweder -1, 1 oder 0 multipliziert; je nachdem, ob die Summe der Elemente von **l2** von Position **p** bis ans Ende von **l2** negativ, positiv oder Null ist. Sollte die Länge der Liste **l2** **p** unterschreiten, wird jedes Element aus **l1** ab Position **p** mit Null multipliziert. Dementsprechend hat die Ergebnisliste **l** die gleiche Länge wie **l1**.

Gegeben sei folgende Funktion in Racket:

```
( define ( foo l1 l2 )
  ( local
    (
      ( define ( bar l2 acc v)
        ( cond
          [ ( and ( empty? l2 ) ( = acc 0 ) ) 0 ]
          [ ( and ( empty? l2 ) ( > acc 0 ) ) v ]
          [ ( and ( empty? l2 ) ( < acc 0 ) ) (- v) ]
          [ else ( bar ( rest l2 ) ( + acc ( first l2 ) ) v ) ]
        )
      )
    )
  ( cond
    [ ( empty? l1 ) empty ]
    [ ( empty? l2 ) ( cons 0 ( foo ( rest l1) l2 ) ) ]
    [ else ( cons ( bar l2 0 ( first l1 ) ) ( foo ( rest l1 ) ( rest l2 ) ) ) ]
  )
)
```

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie in Java **public**-Objektmethoden **foo** und **bar** analog zum Racket-Code oben. Diese zwei Methoden gehören zur selben Klasse (diese Klasse müssen Sie nicht selbst schreiben). Die formalen Parametertypen von **foo** sind beide **List<Integer>** (gemeint ist **java.util.List**). Der formale Parametertyp von **l2** von **bar** ist vom Typ **Iterator<Integer>**. Die formalen Parametertypen von **acc** und **v** von **bar** sind vom Typ **Integer**. Der Rückgabotyp von **foo** ist **List<Integer>**. Der Rückgabotyp von **bar** ist **Integer**. Im Gegensatz zum Racket-Code oben ist die Java-Methode **bar** nicht in

foo definiert, sondern die beiden Methoden folgen aufeinander. Sie können davon ausgehen, dass die beiden Parameter von **foo** auf korrekt gebildete lineare Listen verweisen, auch wenn diese eventuell keine Listenelemente beinhalten.

Sie können ebenfalls davon ausgehen, dass diese Listen nicht **null** sind und niemals ein Element mit Wert **null** enthalten. Falls Sie in Racket eine leere Liste zurückgeben würden, geben Sie in Java eine Liste mit 0 Elementen zurück.

Verbindliche Anforderungen:

1. Auf die Elemente von **l1** und **l2** wird ausschließlich mit Iteratoren zugegriffen.
2. Übersetzungen in andere Datenstrukturen (Arrays, Streams etc.) sind nicht erlaubt.

Erinnerung: Interface **List<T>** hat eine parameterlose Objektmethode **iterator** mit Rückgabotyp **Iterator<T>**, eine Objektmethode **add** mit einem Parameter vom Typ **T**, sowie eine weitere Methode **add** mit einem Parameter vom formalen Typ **int** und einen zweiten Parameter vom formalen Typ **T**. Die Methode **add** fügt ihren Parameter hinten an die Liste an, die zweite am Index. Die boolesche Methode **isEmpty** von **List<T>** ist parameterlos.

Wichtig: Zusätzlich dürfen Sie die Funktion **listIterator(int idx)** von **List<T>** benutzen. Diese Funktion gibt einen Iterator zurück, der sich vor dem Element mit dem übergebenen Index **idx** befindet. Beispielsweise gibt der Aufruf **listIterator(0)** einen Iterator zurück, der sich vor dem ersten Element der Liste befindet, analog zu **iterator()**. Interface **Collection<T>** hat eine parameterlose Methode **size**.

Interface **Iterator<T>** hat zwei parameterlose Methoden **hasNext** und **next**, wobei **hasNext** Rückgabotyp **boolean** und **next** Rückgabotyp **T** hat. Klasse **LinkedList<T>** hat einen parameterlosen Konstruktor.

```
public List<Integer> foo(List<Integer> l1, List<Integer> l2){
    Iterator<Integer> it1 = l1.iterator();
    Iterator<Integer> it2 = l2.iterator();
    if(l1 == null){
        return null;
    }
}
```

```
}  
while(it1.hasNext())  
{  
    public Integer bar(Iterator<Integer> l2, Integer accu, Integer v){  
        int result = 0;  
        if(l2 == null && accu == 0){  
            return 0;  
        }  
        if(l2 == null && accu > 0){  
            return v;  
        }  
        if(l2 == null && accu < 0){  
            return -v;  
        }  
        while(l2.hasNext()){  
            result = accu + l2.next() + v;  
        }  
        return result;  
    }  
}
```

Information

Aufgabe 4 (27 Punkte)

Diese Aufgabe besteht aus drei Teilaufgaben.

Frage 10

Vollständig

Erreichbare Punkte: 7.00

4(a):

Gegeben sei folgende Funktion in Racket:

```
(define (foobar foo bar init)
  (local
    ((define (h acc a b)
      (cond
        [(empty? a) acc]
        [(foo (first a) b) (h (bar (first a) acc) (rest a) b)]
        [else (h acc (rest a) b)]
      )
    ))
    (lambda (a b) (h init a b))
  )
)
```

Wie Sie sehen, sind **foo** und **bar** Funktionen. Gegeben sei zudem der folgende beispielhafte Aufruf von **foobar**:

```
( (foobar (lambda (x b) (> x b)) + 0) '(1 2 3) 1 )
```

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie in Java ein nichtgenerisches funktionales **public**-Interface **Foo** mit funktionaler Methode **apply**, das auf **foo** passt, wobei **Integer** der Zahlentyp ist.

Schreiben Sie zudem eine nichtgenerische **public**-Klasse **MyFoo**, die **Foo** implementiert und dasselbe Ergebnis zurückliefert wie der Lambda-Ausdruck für **foo** im obigen beispielhaften Aufruf.

Erinnerung: Methode **get** von **List** liefert das Listenelement an der im aktuellen Parameter angegebenen Position zurück, wobei Positionen wie üblich mit 0 beginnen.

```
//foobar: foo bar, init ->
// h : acc(number) , a(list), b(number) ->
// foo : number, Liste -> bool
// bar: number; list number-> h

public interface foo{
    public boolean apply(Integer x, List<Integer> b);
}

public class MyFoo implements foo{
    public boolean apply(Integer x, List<Integer> b){
        return a > b.get(0);
    }
}
```

Frage 11

Vollständig

Erreichbare Punkte: 7.00

4(b):

Gegeben sei folgende Funktion in Racket:

```
(define (foobar foo bar init)
  (local
    ((define (h acc a b)
      (cond
        [(empty? a) acc]
        [(foo (first a) b) (h (bar (first a) acc) (rest a) b)]
        [else (h acc (rest a) b)]
      )
    ))
    (lambda (a b) (h init a b))
  )
)
```

Wie Sie sehen, sind **foo** und **bar** Funktionen. Gegeben sei zudem der folgende beispielhafte Aufruf von **foobar**:

```
( (foobar (lambda (x b) (> x b)) + 0) '(1 2 3) 1 )
```

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.Schreiben Sie in Java ein generisches funktionales **public**-Interface **Bar** mit generischem Typparametern **T** und **U**, das auf **bar** passt, wobei die funktionale Methode **apply** heißen soll.Schreiben Sie in Java eine nichtgenerische **public**-Klasse **MyBar**, die **Bar** mit Instanziierung **Integer** für beide Typparameter implementiert und dasselbe Ergebnis zurückliefert wie der Lambda-Ausdruck für **bar** im obigen beispielhaften Aufruf.

```
public interface Bar<T, U>{
    public T apply(T a, T b);
}

public class MyBar implements Bar<Integer, Integer>{
    public Integer apply(Integer a, Integer b){
        return h(a,b);
    }
}
```


Frage 12

Vollständig

Erreichbare Punkte: 13.00

4(c):

Gegeben sei folgende Funktion in Racket:

```
(define (foobar foo bar init)
  (local
    ((define (h acc a b)
      (cond
        [(empty? a) acc]
        [(foo (first a) b) (h (bar (first a) acc) (rest a) b)]
        [else (h acc (rest a) b)]
      )
    ))
    (lambda (a b) (h init a b))
  )
)
```

Wie Sie sehen, sind **foo** und **bar** Funktionen. Gegeben sei zudem der folgende beispielhafte Aufruf von **foobar**:

```
( (foobar (lambda (x b) (> x b)) + 0) '(1 2 3) 1 )
```

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie in Java eine nicht-generische **public**-Klassenmethode **foobar**, die auf Basis von **Foo** und **Bar** die obige Racket-Funktion **foobar** eins-zu-eins nach Java übersetzt. Rückgabotyp von **foobar** ist **java.util.function.BiFunction** mit passenden Typparametern. Zahlentyp ist **Integer** und Listentyp ist **List<Integer>**.

Sie können in Java eine rekursive Hilfsfunktion **h** definieren, die die Parameter der Racket-Funktion **h** sowie die Parameter der Racket-Funktion **foobar** annimmt. Die Java Funktion **rest** ist gegeben mit der selben Funktionalität wie im Racket-Code:

```
static <T> List<T> rest(List<T> l)
```

Schließlich übersetzen Sie den obigen beispielhaften Racket-Ausdruck nach Java.

Verbindliche Anforderung: Der Lambda-Ausdruck im obigen Racket-Code muss in einen Lambda-Ausdruck in Java übersetzt werden.

Erinnerung: Die Java-Interfaces **Foo** und **Bar** aus Teil (a) und (b) passen auf **foo** und **bar** im obigen beispielhaften Aufruf. Die funktionalen Methoden von **Foo** und **Bar** heißen **apply**. Methode **get** von **List** liefert das Listenelement an der im Parameter angegebenen Position zurück, wobei Positionen wie üblich mit 0 beginnen.

Interface **BiFunction** hat drei generische Typparameter **A**, **B** und **C** (in dieser Reihenfolge) und repräsentiert mathematische Funktionen **A × B → C**. Die funktionale Methode von **BiFunction** heißt **apply**.

```
public static BiFunction<Integer,Integer, Integer> foobar(Foo foo, Bar bar, Integer init){
    return (a, b) -> h(init,a,b);
}

public static Integer h(Integer acc, List<Integer> a, Integer b){
    if ( a== null){
        return acc;
    }
    if(foo(a.get(0),b)){
        return h(bar(a.get(0),acc),List<Integer>.rest(a), b);
    }
    return h(acc, List<Integer>.rest(a), b);
}

// Annahme :foobar gehöre Klasse X
List<Integer> lst = new LinkedList<Integer>();
lst.add(1);
```

```
Ist.add(2);
```

```
X.foobar((x,y -> (x <b), 0, ))
```

Information

Aufgabe 5 (24 Punkte)

Diese Aufgabe besteht aus zwei Teilaufgaben.

Frage 13

Vollständig

Erreichbare Punkte: 9.00

5(a):

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public**-Klassenmethode **foobar**, die zwei Parameter **a** und **b** vom Typ "Array von **int**" und Rückgabotyp "Array von **int**" hat (die Klasse, zu der **foobar** gehört, müssen Sie nicht schreiben). Ihre Methode **foobar** kann ohne Prüfung davon ausgehen, dass **a** und **b** auf **int**-Arrays mit Länge größer 0 verweisen und dass jede Komponente tatsächlich ein gültiger **int**-Wert ist.

Für jeden Index **i** in der Schnittmenge der Indexbereiche von **a** und **b** (also für jedes $i \in \{0, \dots, \min\{a.length, b.length\}\}$), an dem ein Element an dieser Stelle der beiden Arrays den gleichen Betrag hat, enthält das zurückgelieferte Array eine Komponente mit diesem Element zuerst als Negation des Betrags und dann den Betrag selbst als Inhalt. Demnach ist dieses Element zweimal im zurückgelieferten Array enthalten. Die Reihenfolge dieser Komponenten ist dieselbe wie in **a** und **b**. Darüber hinaus enthält das zurückgelieferte Array keine Komponenten.

Beispiel : $a = [1, 5, -7, 2, 0, -8]$, $b = [2, -5, -7, -1, 0, 9, 1] \rightarrow [-5, 5, -7, 7, -0, 0]$

Verbindliche Anforderungen:

1. Lösen Sie die oben genannte Aufgabe rein iterativ, das heißt, Rekursion ist nicht erlaubt.
2. Sie müssen auf Arrays arbeiten. Streams, Listen usw. sind nicht erlaubt.

Erinnerung: Arraytypen haben eine Klassenkonstante `length`. Klasse `Math` hat eine Klassenmethode `abs`, die einen Wert von Typ `int` entgegennimmt und den Betrag des Werts als `int` zurückgibt. Klasse `Math` hat zusätzlich eine Klassenmethode `min`, die zwei Zahlen entgegennimmt und die kleinere der beiden Zahlen zurückgibt. Die akzeptierten Typen von `min` sind `int`, `double`, `float`, und `long`.

```
public static int[] foobar(int[] a, int[] b) {
    int condition = a.length < b.length ? a : b;
    int amount = 0;
    for (int i = 0; i < condition; i++){
        if (Math.abs(a[i]) == Math.abs(b[i])){
            amount += 2;
        }
    }
    int[] result = new int[amount];
    int resultIndex = 0;
    for (int i = 0; i < condition; i++){
        if (Math.abs(a[i]) == Math.abs(b[i])){
            result[resultIndex] = -Math.abs(a[i]);
            resultIndex++;
            result[resultIndex] = Math.abs(a[i]);
            resultIndex++;
        }
    }
    return result;
}
```


Frage 14

Vollständig

Erreichbare Punkte: 15.00

5(b):

Sie brauchen keine **import**- oder **package**-Anweisungen zu schreiben.

Schreiben Sie eine **public**-Klassenmethode **foobar**, die zwei Parameter **a** und **b** vom Typ "Array von **int**" und Rückgabotyp "Array von **int**" hat (die Klasse, zu der **foobar** gehört, müssen Sie nicht schreiben). Ihre Methode **foobar** kann ohne Prüfung davon ausgehen, dass **a** und **b** auf **int**-Arrays mit Länge größer 0 verweisen und dass jede Komponente tatsächlich ein gültiger **int**-Wert ist.

Für jeden Index **i** in der Schnittmenge der Indexbereiche von **a** und **b** (also für jedes $i \in \{0, \dots, \min\{a.length, b.length\}\}$), an dem ein Element an dieser Stelle der beiden Arrays den gleichen Betrag hat, enthält das zurückgelieferte Array eine Komponente mit diesem Element zuerst als Negation des Betrags und dann den Betrag selbst als Inhalt. Demnach ist dieses Element zweimal im zurückgelieferten Array enthalten. Die Reihenfolge dieser Komponenten ist dieselbe wie in **a** und **b**. Darüber hinaus enthält das zurückgelieferte Array keine Komponenten.

Für die Länge des zurückzuliefernden Arrays schreiben Sie eine rekursive Hilfsmethode **foo**, die **a** und **b** sowie ein **int** namens **index** als Parameter hat und die Anzahl der Indizes größer/gleich **index** in **a** und **b** zurückliefert, von denen das Element in das Ergebnisarray gemäß obiger Spezifikation zu übernehmen ist.

Für die Befüllung des zurückzuliefernden Arrays mit Zeichen schreiben Sie eine rückgabelose rekursive Hilfsmethode **bar** mit **a** und **b**, einem **int**-Array **result** sowie zwei **int** namens **indexInAAndB** und **indexInResult** als Parameter. Dabei soll **indexInResult** immer der jeweils nächste Index in **result** sein, an den ein Zeichen zu kopieren ist.

Beispiel : $a = [1, 5, -7, 2, 0, -8]$, $b = [2, -5, -7, -1, 0, 9, 1] \rightarrow [-5, 5, -7, 7, -0, 0]$

Verbindliche Anforderungen:

1. Lösen Sie die oben genannte Aufgabe rein rekursiv, das heißt, Schleifen sind nicht erlaubt.
2. Sie müssen auf Arrays arbeiten. Streams, Listen usw. sind nicht erlaubt.

Erinnerung: Arraytypen haben eine Klassenkonstante **length**. Klasse **Math** hat eine Klassenmethode **abs**, die einen Wert von Typ **int** entgegennimmt und den Betrag des Werts als **int** zurückgibt. Klasse **Math** hat zusätzlich eine Klassenmethode **min**, die zwei Zahlen entgegennimmt und die kleinere der beiden Zahlen zurückgibt. Die akzeptierten Typen von **min** sind **int**, **double**, **float**, und **long**.

```
public static int[] foobar(int[] a, int[] b){
    int amount = foo(a,b,0);
    int[] result = new int[amount];
    bar(a,b,result, 0,0);
    return result;
}

public static int foo(int[] a, int[] b, int index){
    int condition = a.length < b.length? a : b;
    if (!(index < condition)){
        return 0;
    }
    return foo(a,b, index + 1) + Math.abs(a[index]) == Math.abs(b[index])? 1 : 0;
}

public static void bar (int[] a, int[] b ,int[] result, int indexInAAndB, int indexInResult){
    int condition = a.length < b.length? a : b;
    if (!(indexInAAndB < condition)){
        return;
    }
    if(Math.abs(a[indexInAAndB]) == Math.abs(b[indexInAAndB])){
        result[indexInResult] = - Math.abs(a[i]);
        indexInResult++;
        result[indexInResult] = Math.abs(a[i]);
        indexInResult++;
    }
}
```

```
}  
bar(a,b,indexInAAndB +1, indexInResult);  
}
```

Information

Aufgabe 6 (16 Punkte)

Diese Aufgabe besteht aus vier Teilaufgaben.

Information

6(a) Methoden

Aufgaben zu Methoden

Frage 15

Nicht beantwortet

Erreichbare Punkte: 1.00

In welchem Fällen ist das Überladen von Methoden in Java überhaupt relevant?

Answer

Frage 16

Vollständig

Erreichbare Punkte: 1.00

Wie unterscheidet sich das Überladen von dem Überschreiben von Methoden?

Answer

Beim Überladen bleibt die Funktionalität erhalten, die Parameterliste ändert sich, während beim Überschreiben die Methode gleich bleibt

Frage 17

Vollständig

Erreichbare Punkte: 1.00

Wovon hängt die Sichtbarkeit einer Methode ab?

Answer

Ob die Methode private oder public definiert wird

Frage 18

Vollständig

Erreichbare Punkte: 1.00

Wie können Subklassen in Java die Sichtbarkeit von Methoden ihrer Basisklassen verändern und welche Einschränkungen gelten dabei?

Answer

Subklassen in Java können durch protected die Sichtbarkeit von methoden ihrer Basisklasse verändern

Information

6 (b) Abstrake Klassen und Interfaces

Aufgaben zu abstrake Klassen und Interfaces

Frage 19

Nicht beantwortet

Nicht bewertet

Ist eine Klasse abstrakt, genau dann, wenn sie mindestens eine abstrakte Methode hat? Begründen Sie Ihre Antwort!

Answer

Frage 20

Vollständig

Erreichbare Punkte: 2.00

Ist eine Klasse abstrakt, genau dann, wenn sie mindestens eine abstrakte Methode hat? Begründen Sie Ihre Antwort!

Answer

nein, eine abstrakte Klasse ist abstrakt, wenn sie mindestens eine methode eines Interfaces nicht implementiert hat, unabhängig davon, ob sie eine abstrakte Klasse enthält oder nicht

A Nein.

B Abstrakte Klassen müssen keine (abstrakten) Methoden enthalten.

Frage 21

Vollständig

Erreichbare Punkte: 1.00

Nennen Sie einen Vorteil, den Interfaces gegenüber abstrakte Klassen haben.

Answer

Interfaces können Mehrfachvererbung

Information**6(c) Arrays, Listen, Streams**

Aufgaben zu Arrays, Listen, Streams

Frage 22

Vollständig

Erreichbare Punkte: 3.00

Nennen Sie ein Alleinstellungsmerkmal, dass die folgenden Konzepte jeweils gegenüber den anderen beiden Konzepten haben:

Arrays**Listen****Streams**

Beispielhaft bedeutet das: Nennen Sie eine Eigenschaft, die **Arrays** haben, aber **Listen** und **Streams** nicht haben.



Answer

Bei Arrays kann man auf einen bestimmten Index zugreifen
Listen
Streams

**Frage 23**

Vollständig

Erreichbare Punkte: 2.00

Entscheiden Sie für jede der folgenden Zeilen, ob der Java-Compiler eine Fehlermeldung werfen würde, oder nicht. (Gehen Sie davon aus, dass der Code Teil eines Methodenkörpers ist):

```
Number[] n = new Integer[42];  
List<Number> lstNum = new LinkedList<Double>();
```

Answer

1. Fehler
2. kein Fehler, da LinkedList ein Subtyp von List ist und Double Subtyp von Number

**Information**

6(d) IntPredicate

Aufgaben zu IntPredicate

Frage 24

Nicht beantwortet

Erreichbare Punkte: 2.00

6(d):(i) Wie unterscheidet sich das Interface **IntPredicate** von Interfaces im Generellen?

Answer

Frage 25

Vollständig

Erreichbare Punkte: 1.00

Wozu kann **IntPredicate** genutzt werden?

Answer

Um ein Boolean Ausdruck zu verwenden -> Prädikat

Frage 26

Nicht beantwortet

Erreichbare Punkte: 1.00

Nennen Sie mögliche Ein- und Ausgabetypen von **IntPredicate**.

Answer

Information

Warten Sie bis zur Ansage des Klausurendes unter Klausurbedingungen.

Die Aufsicht kann nicht unterscheiden, ob Sie noch die Klausur schreiben oder schon fertig sind.

Bleiben Sie ruhig sitzen oder gehen Sie nochmals durch die Klausur.

Sie erhalten per Ansage zum Klausurende das Quitpasswort für den Safe Exam Browser!