Probeklausur:
Programmieren I
INF (1. Semester)

Vorname	Nachname	Matrikelnummer

Allgemeine Hinweise

- *Diese Probeklausur ist deutlich länger als normal (etwa doppelt bis dreifach). Sie umfasst eine Zusammenstellung alter Klausuraufgaben. Die Aufgaben in dieser Probeklausur werden Jahr fur Jahr fortgeschrieben und folgen keiner thematischen Gliederung oder einem steigenden Schwierigkeitsgrad.*
- *Nehmen Sie sich Zeit, die Fragestellungen zu verstehen. Bevor Sie mit der Bearbeitung der Fragen beginnen, lesen Sie bitte alle Fragen vorher durch.*
- *Halten Sie sich nicht zu lange an einer Aufgabe auf, wenn Sie Probleme mit der Lösung haben. Versuchen Sie möglichst viele Aufgaben zu bearbeiten. "Verbeißen" Sie sich nicht an einer Aufgabe. Stellen Sie Aufgaben notfalls zurück und bearbeiten Sie diese am Ende.*
- *Bevor Sie eine Aufgabe bearbeiten, stellen Sie sicher, dass Sie die Fragestellung verstanden haben. Punkte erhalten Sie für inhaltliche Antwort(teile) zur Fragestellung. Es gibt keine Punkte für Antwort(teile), die sich nicht auf die Fragestellung beziehen.*
- *Diese Klausur ist **Open Book**. Sie können zur Beantwortung der Fragen Ihre Unterlagen, Mitschriften und nicht elektronische Literatur heranziehen. Die Zeit ist jedoch so bemessen, dass Sie sich die Antworten zu den Fragestellungen nicht erst in der Klausur anlesen können. Sie müssen wissen, wo Sie nachschlagen und sich daher den Stoff im Vorfeld erarbeitet haben.*
- *Nicht erlaubt sind elektronische und/oder kommunikationstechnische Geräte aller Art (Laptop, Taschenrechner, Handy, etc.).*
- **Kleiner Tipp: Versuchen sie diese Aufgaben alle erst auf Papier zu lösen (auch in der Vorbereitung auf die Klausur). Auch in der Klausur müssen Sie Aufgaben auf Papier lösen.**

1 Aufgabe: Zeilen filtern

Entwickeln Sie nun bitte eine Methode `filterLongLines()`, die aus einer Zeichenkette, Zeilen selektiert, die länger als eine vorgegebene Länge n sind. Zeilen sind dabei durch `'\n'` getrennt. Die Methode soll ein Mapping von Zeilen auf zu lange (d.h. Zeilen der Länge $> n$) Zeilen zurückliefern. Die Zeilen werden dabei ab 1 gezählt.

Folgende Zeilen

```
String document =
    ‘‘Lorem Ipsum\n’’ +
    ‘‘1234567891011\n’’ +
    ‘‘abcdegfh\n’’ +
    ‘‘tralala und Filibu\n’’ +
    ‘‘Filubu\n’’ +
    ‘‘tralala\n’’;
Map<Integer, String> toLong = filterLongLines(document, 10);
System.out.println(toLong);
```

sollen bspw. folgende Ausgabe (Standardausgabe einer Map gem. Java) produzieren:

```
{1=Lorem Ipsum, 2=1234567891011, 4=tralala und Filibu}
```

Für `filterLongLines(‘‘’’,n)` soll das leere Mapping `{}` zurück gegeben werden. Die Methode ist für `filterLongLines(null, n)` nicht definiert.

Hinweis: *Sie müssen die Methode nicht mittels JavaDoc kommentieren.*

Lösung:

Erreichte Punkte	Erzielbare Punkte
	18

2 Aufgabe: Mehrdimensionale Arrays und Schleifen

Gegeben sind bspw. folgende mehrdimensionale Arrays

```
1   int[][] ls1 = {
2     { 1, 2, 3 },
3     { 4, 5, 6 },
4     { 7, 8, 9 }
5   };
6
7   int[][] ls2 = {
8     { 1 },
9     { 2, 3 },
10    { 4, 5, 6 },
11    { 7, 8 },
12    { 9 }
13  };
```

Der Aufruf folgender Codezeilen

```
1   String out1 = toTable(ls1, "-", "\n");
2   String out2 = toTable(ls2, "", ", ");
```

erzeugt folgende Konsolenausgabe.

```
1-2-3
4-5-6
7-8-9
1, 23, 456, 78, 9
```

Aufgabe: Geben Sie eine Implementierung für die Methode `toTable()` an, die oben stehendes Verhalten realisiert. Die Methode `toTable()` muss nur zweidimensionale Arrays verarbeiten können, die nicht leer sind (weder auf der ersten noch der zweiten Dimension) oder `null` sind.

Hinweis: *Sie müssen die Methode `toTable()` **nicht** mittels JavaDoc kommentieren.*

Lösung:

Erreichte Punkte	Erzielbare Punkte
	18

3 Aufgabe: Rekursives Bestimmen durchschnittlicher Wortlängen

Entwickeln sie nun bitte eine **rekursive** Methode `avgStringLength()`, die auf einer Liste von Zeichenkette die durchschnittliche Länge aller Zeichenketten bestimmt. Die Methode soll rekursiv sein, darf aber andere (ebenfalls rekursive Methoden aufrufen).

Sie dürfen zur Lösung dieser Aufgabe keine Schleifen einsetzen.

Folgende Codezeilen

```
List<String> strings = Arrays.asList("abc", "abcd", "a", "abcddef");
double avg = avgStringLength(strings);
System.out.println(avg);
```

sollen

```
3.75
```

auf der Konsole ausgeben. Ihre Lösung soll aber natürlich mit Listen beliebiger Länge > 0 umgehen können. Die Methode `avgStringLength()` ist für leere Listen und null Listen nicht definiert.

Lösung:

Erreichte Punkte	Erzielbare Punkte
	16

4 Aufgabe: Funktionstypen

Definieren sie bitte die folgenden Lambdafunktionen und weisen sie diese einem geeigneten Funktionstyp zu. Geben Sie anschließend zur Erläuterung jeweils einen beispielhaften und zu `true` ausgewertetem booleschen Ausdruck mit dieser Lambdafunktion an.

Beispiel:

Prüfen Sie, ob eine ganzzahlige Zahl gerade ist.

Lambdafunktion:.

```
Predicate<Integer> even = x -> x % 2 == 0;
```

Beispielhafter Aufruf inkl. Ergebnis

```
even.test(4) == true
```

Teilaufgabe 1 (8 Punkte):

Prüfen Sie, ob zwei Strings ungleich `null` gleiche Länge haben.

Lambdafunktion:

Beispielhafter boolescher Ausdruck:

Teilaufgabe 2 (8 Punkte):

Bestimmen Sie die Länge einer Zeichenkette ungleich `null`.

Lambdafunktion:

Beispielhafter boolescher Ausdruck:

Teilaufgabe 3 (8 Punkte):

Quadrieren einer Fließkommazahl.

Lambdafunktion:

Beispielhafter boolescher Ausdruck:

Erreichte Punkte	Erzielbare Punkte
	16

5 Funktionale Programmierung

Gegeben seien nun Zeichenketten beliebiger Länge. Innerhalb von Zeichenketten sollen sie nun nur mittels Streams und Lambdafunktionen die kürzesten Wörter bestimmen. Doppelt vorkommende Wörter sollen dabei nur einmal bestimmt werden. Wörter sind alle Zeichenketten, die durch mehr als ein Leerzeichen voneinander getrennt sind.

Für die Zeichenkette

```
String document = ‘‘Hallo ich bin ein Beispiel. Ein sehr kurzes Beispiel bin ich.’’;
```

sind die kürzesten Wörter “ich”, “bin”, “ein”, “Ein”, “bin”, “ich”. Die doppelten “bin” und “ich” sind dabei zu streichen.

Sie dürfen zur Lösung des Problems nur die Methoden `stream()`, `map()`, `reduce()` sowie die `split()` Methode der Klasse `String` verwenden, sowie selbst definierte Lambdafunktionen. Sie dürfen natürlich mehrere Stream-basierte Ausdrücke definieren, um ihr Problem zu lösen.

Lösung:

Erreichte Punkte	Erzielbare Punkte
	20

6 Bäume

Gegeben sei folgende Liste von Werten.

```
5, 8, 14, 27, 56, 32, 23, 67, 0, -1, 3, 17
```

Gegeben sei ferner folgende Datenstruktur `Node`

```
class Node {
    public Node left;
    public Node right;
    public int value;
    public Node(int v, Node l, Node r) {
        this.left = l;
        this.right = r;
        this.value = v;
    }
}
```

Es soll aus dieser Liste ein **sortierter** Binärbaum aufgebaut werden, in dem auf dem Baum `t` folgende `insert` Operationen ausgeführt werden (`insert` funktioniert wie in der Vorlesung definiert):

```
Node t = new Node(5, null, null);
insert(8, t);
insert(14, t);
...
insert(17, t);
```

Aufgabe 1: Geben Sie bitte den resultierenden Binärbaum in grafischer Form an.

Antwort:

Aufgabe 2: Geben Sie bitte die resultierende inorder Ausgabe dieses Baums an.

Antwort:

Aufgabe 3: Entwickeln Sie eine Methode `int sum(Node)`, die die Summe aller Werte in einem Binärbaum (`Node`) bestimmt.

Hinweis: *Wenden Sie die drei Schritte zur Entwicklung von rekursiven Methoden an, die Ihnen in der Vorlesung gezeigt worden sind. Rekursive Methoden sind meistens sehr kurz!*

Antwort:

Erreichte Punkte	Erzielbare Punkte
	34

7 Selbstdefinierte Datentypen

Sie sollen nun einen Datentyp entwickeln. Der Datentyp soll wie folgt angelegt und ausgegeben werden können:

```
1  Raum r1 = new Raum(17, 0, 10);
2  Raum r2 = new Raum(18, 0, 1);
3  Raum r3 = new Raum(2, 0, 1);
4
5  System.out.println(r1);
6  System.out.println(r2);
7  System.out.println(r3);
```

Dies würde folgende Ausgabe auf der Konsole erzeugen.

```
17-0.10
18-0.01
2-0.01
```

Zwei Räume sind als inhaltlich gleich anzusehen, wenn ihre Zeichenkettenrepräsentationen gleich sind. Implementieren sie nun bitte eine geeigente Klasse `Raum`, die es ermöglicht Räume in oben angegebener Art und Weise anzulegen und auszugeben. Objekte der Klasse `Raum` sollen ferner

- miteinander inhaltlich (d.h. nicht ihre Referenzen) gem. der Java Konventionen verglichen
- und gem. der Java Konventionen geklont werden können.

Lösung:

Erreichte Punkte	Erzielbare Punkte
	29

8 Aufgabe: Zeichenpaare zählen

Entwickeln sie nun bitte eine Methode `doubles()` die für beliebige Zeichenketten ungleich `null`, die in der Zeichenkette vorkommenden Paare gleicher Zeichen ('aa', 'bb', 'cc', etc.) in absoluter Häufigkeit zählt und das Ergebnis mittels eines typgebundenen Mappings von Strings auf ganzzahlige Werte zurück gibt. Das Zählen der Zeichenpaare soll case-insensitive erfolgen. D.h. die Vorkommen von 'Aa', 'aA', 'AA' und 'aa' sind bspw. wie 'aa' zu zählen. Das Mapping darf nur doppelte Zeichen umfassen, die auch in der Zeichenkette vorkommen. Überlappen sich Zeichenpaare, so sind alle Zeichenpaare zu zählen. Bspw. enthält die Zeichenkette 'aaa' zweimal das Zeichenpaar 'aa'.

Die Methode soll bspw. wie folgt aufgerufen werden können:

```
Map<String, Integer> result = doubles(“WWWaiting for a terrific summer ...”);
System.out.println(result);
```

und folgende Ausgabe (Standardausgabe einer Map gem. Java) produzieren:

```
{..=2, mm=1, rr=1, ww=2}
```

Für `doubles(“”)` soll das leere Mapping `{}` zurück gegeben werden. Die Methode ist für `doubles(null)` nicht definiert.

Die Zeichenpaare sollen dabei immer alphabetisch aufsteigend sortiert ausgegeben werden.

Hinweis: *Sie müssen die Methode nicht mittels JavaDoc kommentieren.*

Lösung:

Erreichte Punkte	Erzielbare Punkte
	27

9 Aufgabe: Rekursive Verarbeitung von Iteratoren

Entwickeln sie nun bitte eine **rekursive** Methode

```
public static List func(Iterator, Iterator)
```

die keine Seiteneffekte hat (d.h. bspw. keine Konsolenausgabe erzeugt) und mittels eines Iterators zwei Listen nicht verbrauchend verarbeiten kann.

`func` soll bspw. wie nachfolgend gezeigt aufgerufen werden können:

```
1  public static void main(String[] args) {
2    List as = new LinkedList();
3    for (int i=1; i <= 3; i++) as.add(i); // [1, 2, 3]
4
5    List bs = new LinkedList();
6    for (int j=10; j <= 50; j += 10) bs.add(j); // [10, 20, 30, 40, 50]
7
8    System.out.println(func(as.iterator(), bs.iterator()));
9    System.out.println(func(as.iterator(), as.iterator()));
10   System.out.println(func(bs.iterator(), as.iterator()));
11   System.out.println(func(bs.iterator(), bs.iterator()));
12 }
```

Oben gezeigter Aufruf soll folgende Ausgabe produzieren:

```
[1, 10, 2, 20, 3, 30, 40, 50]
[1, 1, 2, 2, 3, 3]
[10, 1, 20, 2, 30, 3, 40, 50]
[10, 10, 20, 20, 30, 30, 40, 40, 50, 50]
```

Hinweis: Ein `Iterator` stellt im wesentlichen zwei Methoden zur Verfügung:

- `hasNext()` liefert einen booleschen Wert ob noch Werte in der zu verarbeitenden Datenstruktur existieren und
- `next()` liefert den nächsten zu verarbeitenden Wert der Datenstruktur und schiebt den Verarbeitungszähler in der Datenstruktur ein Element weiter.
- Einer `List` können sie mit der `addAll()` Methode alle Elemente einer Collection (z.B. einer `List`) auf einmal anhängen.

Lösung:

Erreichte Punkte	Erzielbare Punkte
	18

10 Aufgabe: Rekursive Datenstrukturen

Gegeben sei folgende Datenstruktur `Node`

```
1 public class Node {
2   public Node left;
3   public Node right;
4   public int value;
5
6   public Node(int v, Node l, Node r) {
7     this.left = l; this.right = r; this.value = v;
8   }
9 }
```

Gegeben sei ferner die folgende `insert()` Methode.

```
1   public static void insert(int v, Node tree) {
2     if (tree == null) return;
3
4     if (v <= tree.value) {
5       if (tree.left == null) {
6         tree.left = new Node(v, null, null);
7       } else insert(v, tree.left);
8     }
9
10    if (v > tree.value) {
11      if (tree.right == null) {
12        tree.right = new Node(v, null, null);
13      } else insert(v, tree.right);
14    }
15  }
```

Teilaufgabe 1 (11 Punkte): Entwickeln Sie eine Methode `min()`, die den kleinsten Wert in einem sortierten Binärbaum (vom Datentyp `Node`) bestimmt.

`min()` soll bspw. wie folgt aufgerufen werden können.

```
1    System.out.println("Kleinster Wert des Baums ist: " + min(tree));
```

Für oben stehenden Aufruf soll bspw. folgende Konsolenausgabe erfolgen:

```
Kleinster Wert des Baums ist: 17
```

Antwort:

Teilaufgabe 2 (12 Punkte): Entwickeln Sie für die Methode `downsort()`, die ein Array absteigend mittels eines sortierten Binärbaums sortiert, die fehlende Methode `rkl()`.

```
1  public static List<Integer> downsort(int[] as) {
2    if (as.length == 0) return new LinkedList<Integer>();
3    Node tree = new Node(as[0], null, null);
4    for (int i = 1; i < as.length; i++) {
5      insert(as[i], tree);
6    }
7    List<Integer> ret = rkl(tree);
8    return ret;
9  }
```

`downsort()` kann bspw. wie folgt aufgerufen werden

```
1    int[] xs = { 10, 1, 10, 8, 9, 4, 6, 4, 5, 7, 6, 5, 4 };
2    System.out.println(downsort(xs));
```

und würde folgende Ausgabe auf der Konsole erzeugen:

```
[10, 10, 9, 8, 7, 6, 6, 5, 5, 4, 4, 4, 1]
```

Antwort:

Erreichte Punkte	Erzielbare Punkte
	23

Bewertung:

Summe erreichter Punkte:		von **XYZ** Punkten
Bonuspunkte:		max. 10%
Prozent:		
Endnote:		
Datum:	Lübeck, den	
Unterschrift:	Prof. Dr. Nane Kratzke	