Java PM2 WS 2017/2018
Prof. Dr. B. Wendholt

Datum -Zeit: 24.01.2018 – Beginn 9:00 Uhr

Nettobearbeitungszeit TEIL A: 60 Min

Punkteverteilung

	Punkte	Erreicht
Teil A		
a1.sumNIter	5	
a1.sumNRek	5	
a1.toRegular	8	
a1.collectLeafs	10	
a1.reverse	6	
a2.StudieDataReader	20	
a2.StudieDataStatistics		
punkteProStudie	3	
cleversterStudie	5	
anzahlKurse	5	
belegteKurse	7	
Teil A Gesamt	**74**	

Hinweise

- Die maximal gewerteten Punkte für Teil A sind **60** Punkte. Alles was > 60 Punkte ist, wird nicht gewertet!
- Verwenden Sie unbedingt die in den Projekten vorgegebenen Methoden.

Schreiben Sie in jede Datei Ihre Matrikelnummer und wenn Sie mögen Ihren Namen.

Viel Erfolg! ☺

Java PM2 WS 2017/2018
Prof. Dr. B. Wendholt

Teil A: Verstehen und Programmieren

A1 Einzelaufgaben

1. **Iteration und Rekursion**: Gegeben die Summenformel für $\frac{n*(n+1)}{2}$ **(10 Pkt, pro Methode 5Pkt)**

$$sum(n) = \sum_{k=1}^{n} k\,;$$

   a. Berechnen Sie die Formel iterativ (statische Methode ***sumNIter(int n)***)
   b. Berechnen Sie die Formel rekursiv (statische Methode ***sumNRek(int n)***)
   c. Prüfen Sie in beiden Methoden die Gültigkeit der Eingabewerte und generieren Sie eine ***IllegalArgumentException***, wenn der übergebene Wert nicht im geforderten Werte-Bereich liegt.

2. **Arrays:** Gegeben ein unregelmäßiges 2-dimensionales ***double[][]*** Array. Schreiben Sie die statische Methode ***toRegular(double[][] in)***, die das unregelmäßige Array in ein regelmäßiges ***double[][]*** Array umwandelt (**8Pkt**)

   Beispiel: Hier stehen die geschweiften Klammern für Arrays

   Eingabe:

```
{ { 0.0 }, {}, { 1.0, 0.0 }, { 1.0 }, {},
  { 0.0 }, {} };
```

   Ergebnis:

```
{ { 0.0, 0.0 }, { 0.0, 0.0 }, { 1.0, 0.0 }, { 1.0, 0.0 }, { 0.0, 0.0 },
  { 0.0, 0.0 }, { 0.0, 0.0 } };
```

3. **Rekursion**: Gegeben eine Klasse ***Node***, die zur Konstruktion eines Baums verwendet wird. Ein Node kann beliebig viele direkte Kind-Knoten haben. Die Kind-Knoten bekommen Sie mit der Methode ***getChildren()***. Die Methode ***isLeaf()*** liefert ***true***, wenn der Knoten ein Blatt des Baumes ist. Die Klasse ***Node*** finden Sie im Package ***a1.donottouch***.

   Schreiben Sie die statische rekursive Methode ***collectLeafs(Node n)***, die alle Blätter eines Baums in einer Liste zurückliefert. (**10Pkt**)

4. **Arrays**: Schreiben Sie die statische generische Methode ***static <T> reverse(T[] ary)***, die die Reihenfolge der Elemente in ***ary*** umdreht und als Ergebnis zurück liefert. ***ary*** darf dabei nicht verändert werden. (**6Pkt**)

Tests zum ersten Aufgabenteil finden Sie in A1Test. Sie dürfen die Tests an keiner Stelle modifizieren!!!!!

Java PM2 WS 2017/2018
Prof. Dr. B. Wendholt

A2 Dateien einlesen / Auswertungen mit Hilfe von Streams

Gegeben ein einfaches Modell für Studierende, das aus der Klasse ***Studie*** und der Klasse ***Kurs*** besteht. Studies haben eine Matrikelnummer und eine gewisse Anzahl von Kursen, die sie bereits belegt haben. Jedem Kurs ist eine Punktzahl, das Prüfungsergebnis zugeordnet.

1. Ein Studie wird mit der Matrikelnummer erzeugt.
2. Ein Kurs wird mit einer Kursnummer erzeugt.
3. Kurse werden mit der Methode ***put(Kurs k, Integer punkte)*** eingetragen.
4. Die Methoden zum Auslesen der Kurse sind
    a. ***Set<Kurs> getKurse()***
    b. ***Set<Entry<Kurs,Integer>> getKurseUndPunkte()***
    c. ***Collection<Integer> getPunkte()***

TODO's:

1. Implementieren Sie in der Klasse ***StudieDataReader*** die Methode ***List<Studie> readStudieList()***, die den Inhalt einer Datei parst und diesen in eine Liste von Studies überführt. (**20 Pkt**)
    a. Das Öffnen der Datei ist bereist vorgegeben. Der Dateiname und die Separatoren werden bei der Erzeugung eines ***StudieDataReader*** übergeben.
    b. Der Inhalt der Datei sieht wie folgt aus:

```
M1#1::1#2::5#3::12#4::12#5::3#6::7#7::3#9::13
M2#1::5#2::6#3::15#4::10#5::13#0::11
M3#2::13#4::4#6::7#7::9#8::10#9::11#0::9
M4#4::6#6::4#7::11#9::0#0::12
M5#4::6#5::7#6::2#7::8#8::14#9::13
M6#2::9#4::12#5::4#7::9#9::6
M7#2::4#3::9#4::10#7::4#8::14#9::2
M8#2::3#3::3#4::0#5::10#6::12#7::0#8::5#9::6#0::4
M9#3::12#4::11#5::13#6::10#9::0#0::0
M10#1::15#4::10#6::2#8::5#9::2#0::1
M11#3::4#4::11#5::15#7::7#8::9
M12#1::10#4::1#5::0#6::15#7::9#9::4#0::13
M13#1::2#2::11#3::13#4::5#5::15#6::9#9::11#0::4
M14#1::10#2::1#3::5#6::13#7::10#9::13
M15#1::9#2::6#6::14#7::13#8::12#9::7#0::13
M16#1::9#2::15#3::4#4::12#5::8#6::13#7::15#8::0
M17#1::2#4::7#5::13#6::12#7::7#9::14#0::8
M18#1::3#2::5#4::11#5::0#7::1#8::11#0::2
M19#1::14#2::4#3::6#5::7#6::0#9::2#0::1
M20#1::3#3::13#6::12#7::15#8::10#9::1
```

    Interpretation am Beispiel: ***M1*** steht für die Matrikelnummer. Das Paar ***2::5*** steht für den Kurs und die erreichten Punkte. ***2*** ist die Kursnummer und ***5*** die erreichte Punktzahl.

2. Implementieren Sie in der Klasse ***StudieDataStatistics*** die folgenden statischen Methoden unter Verwendung des Java-Streaming API's für Collections:
   a. ***static int punkteProStudie(Studie studie)***: Die Methode summiert alle Punkte der Kurse und gibt den akkumulierten Wert zurück. (**3Pkt**)
   b. ***static Studie clevesterStudie(List<Studie> lstd)***: Die Methode berechnet den Studie, der die höchste Punktzahl über alle Kurse erreicht hat. Verwenden Sie für die Lösung die Methode aus A2.2.a. (**5Pkt**)
   c. ***static int anzahlKurseGesamt(List<Studie> lstd)***: Die Methode berechnet die Summe aller Kurse aller Studies. Doppelte Kurse (das tritt auf, wenn mehrere Studies den gleichen Kurs belegt haben) werden hier selbstverständlich mitgezählt. (**5Pkt**)
   d. ***static Set<Kurs> belegteKurse(List<Studie> lstd)***: Die Methode sammelt alle Kurse aller Studies in einer Menge ein. (**7Pkt**)

Alle Tests zu A2 finden sich in der Klasse *A2Test*. Sie dürfen die Tests auf keinen Fall modifizieren!!!!

A2.1 und A2.2 bauen *NICHT* aufeinander auf und können unabhängig voneinander bearbeitet werden.

Java PM2 WS 2017/2018 MatNr:
Prof. Dr. B. Wendholt Name (optional):

Datum -Zeit: 24.01.2018 – Beginn 9:00/10:00 Uhr

Nettobearbeitungszeit TEIL B: 60 Min

Punkteverteilung

	Punkte	Erreicht
Teil B		
1	6	
2	6	
3	3	
4	3	
5	2	
6	4	
7	6	
8	10	
9	4	
10	5	
11	6	
12	3	
13	4	
14	2	
15	3	
Teil B Gesamt	**67**	

Schreiben Sie auf jedes Blatt Ihre Matrikelnummer und wenn Sie mögen Ihren Namen.

Maximale erreichbare Punktzahl für Teil B sind 60 Punkte.

Viel Erfolg! ☺

Teil B: Wissen und Verstehen

1. Welche Bestandteile der Sprache Java sind nicht konform zu objekt-orientierten Prinzipien? Erläutern Sie warum. (**6Pkt**)

2. Gegeben die 2 Arrays iAry1 und iAry2:

```
Integer[][][] iAry1 =
    new Integer[][][] {
            new Integer[][] { new Integer[] { 1, 2, 5, 7, 89 }, new Integer[] { 22 } },
            new Integer[][] { new Integer[] { 7 } } };
Integer[][][] iAry2 =
    new Integer[][][] {
            new Integer[][] { new Integer[] { 1, 2, 5, 7, 89 }, new Integer[] { 22 } },
            new Integer[][] { new Integer[] { 7 } } };
```

   Welche drei Methoden kennen Sie um die beiden Arrays auf Gleichheit zu prüfen? Welches Ergebnis liefert der jeweilige Vergleich? (**6Pkt**)

3. Geben Sie ein Beispiel für ungeschützte Kovarianz von Arrays? (**3Pkt**)

4. Wie heißen die Mechanismen, die die folgenden Zuweisungen in Java erlauben? (**3 Pkt**)

```
double d = 7;

Object o = new Integer(5);
```

5. Wie heißen die Mechanismen, die den folgenden Ausdruck in Java möglich macht? (**2Pkt**)

```
new Integer(2) + new Long(17);
```

6. Ändern Sie die nachfolgende Methodendefinition derart, dass (1) das Argument ***list*** nicht modifiziert wird und (2) beim Aufruf der Methode beliebige Listen übergeben werden können. (**4Pkt**)

```
public static <T extends Comparable<? super T>> List<T>
                                    sortList(ArrayList<T> list) {


        Collections.sort(list);
        return list;

}
```

7. Gegeben die nachfolgende Klassendefinition zur Modellierung von Geschlechtern: (**6Pkt**)

```
public class Gender {

        public static final Gender MALE = new Gender("Male");
        public static final Gender FEMALE = new Gender("Female");
        public static final Gender DIVERSE = new Gender("Diverse");
        private String name;

        public Gender(String name) {
              this.name = name;
        }

        @Override
        public String toString() {
              return name;
        }
}
```

   a. Erläutern Sie, dass die Klasse einen Datentyp definiert, deren Objekte **nicht unique** (**2Pkt**) sind.

   b. Ändern Sie die obige Definition derart, dass die Objekte unique sind. (**1Pkt**).

   c. Müssen Sie nach den Änderungen unter b. für den Inhaltsvergleich zweier Gender-Objekte die Methode ***equals*** von ***Object*** überschreiben? Geben Sie eine kurze Begründung (**2Pkt**)

   d. Mit welchem Java-Typ lässt sich die Klassendefinition ersetzen. (**1Pkt**)

8. Welche der folgenden Aussagen sind für die Sprache Java wahr, welche falsch? Markieren Sie die wahren Aussagen mit einem (**w**), die falschen mit einem (**f**). Nicht korrekte Markierungen geben 1 Punkt Abzug. Korrekte Markierung werden mit 1 Punkt gewertet. Keine Markierungen werden mit 0 Punkten gewertet. Ist die Gesamtpunktzahl negativ, wird auf 0 Punkte aufgerundet. (**10Pkt**)

	Abstrakte Klassen können einen Konstruktor enthalten.
	Abstrakte Klassen müssen die Methoden eines Interfaces implementieren.
	Konkrete Klassen können nur ein Interface implementieren.
	Interfaces können von Interfaces erben.
	Ein generisches Interface ist ein gültiger Typ.
	Eine Klasse, die die Typvariable eines Interfaces konkretisiert, ist ein gültiger Typ.
	Ableitende Klassen, die eine Methode überschreiben, dürfen als Rückgabetyp einen konkreteren Typ verwenden.
	Ableitende Klassen, die eine Methode überschreiben, dürfen die Sichtbarkeit der Methode erweitern.
	Das Überladen einer Methode ersetzt die Methodendefinition in der Superklasse.
	Welche Methodendefinition ausgeführt wird, entscheidet sich **immer** zur Laufzeit.

9. Ergänzen Sie die nachfolgende Signatur, so dass die Methode Listen mit beliebigen Typargumenten verarbeitet (**4 Pkt**).

```
public static List<Number> collectNumbers(List<    > anyList) {
      List<Number> ln = anyList.stream().filter(o -> o instanceof
                  Number).map(e -> (Number)e).collect(Collectors.toList());
      return ln;
}
```

10. Was wird in Java statisch gebunden? (**5Pkt**)

11. Vervollständigen Sie die folgenden Sätze. (**6Pkt**)
    a. Der Compiler verwendet den statischen Typ, um

    b. Die virtuelle Maschine verwendet den dynamischen Typ, um

12. Erläutern Sie den Unterschied zwischen **checked** und **unchecked** Exceptions. (**3Pkt**)

13. Welche Möglichkeiten zum Kopieren von Objekten kennen Sie in Java? (Nur Nennung) (**2Pkt**)
Geben Sie je ein Beispiel mit den Ihnen bekannten Java-Klassen. (**2 Pkt**)

14. Was ist das Standardverhalten beim Kopieren von Objekten? (**2Pkt**)

15. Gegeben ein Ausschnitt aus der Klassendefinition der Klasse ***Node***. Implementieren Sie die noch leeren Konstruktoren mit Hilfe der Technik der Konstruktor-Kaskade. (**3 Pkt**)

```
public class Node {

    private Number content;
    private Node left;
    private Node right;

    public Node(Number content) {



    }

    public Node(Number content, Node left) {



    }

    public Node(Number content, Node left, Node right) {
           this.content = content;
           this.left = left;
           this.right = right;
    }
...
}
```