

**Datum -Zeit: 18.01.2017 – Beginn 9:00 Uhr**

**Nettobearbeitungszeit TEIL A: 60 Min**

**Punkteverteilung**

	Punkte	Erreicht
<b>Teil A</b>		
a1.1	5	
a1.2	5	
a1.3	10	
a1.4	10	
a2.Kiste	7	
a2.Container	13	
a2.Schiff	10	
<b>Teil A Gesamt</b>	<b>60</b>	

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

**Viel Erfolg! ☺**

## Teil A: Verstehen und Programmieren

### A1 Einzelaufgaben

1. **Iteration:** Gegeben die Näherungsformel für  $\frac{13}{18}$  (**5Pkt**)

$$f(n) = \sum_{k=3}^n \frac{8 * k}{(k^2 - 1)^2};$$

- Berechnen Sie die Formel iterativ (Methode **sum(n)**)
- Prüfen Sie die Gültigkeit der Eingabewerte und generieren Sie eine **IllegalArgumentException**, wenn der übergebene Wert nicht im geforderten Bereich liegt.
- Implementieren Sie die statische Methode **sum(int n)** der Klasse **A1**.

2. Implementieren Sie die statische Methode **fehlerSumKleinerEps(double eps)** der Klasse **A1**. (**5Pkt**)

- Die Methode bestimmt das kleinste n für das  $\frac{13}{18} - sum(n) < eps$  ist und gibt diese n als Ergebnis zurück.

3. **Arrays:** Gegeben eine unregelmäßiges 3-dimensionales Array. In diesem Array steht an zufällig bestimmten Positionen (das macht der Test) die Zeichenkette "**easter egg**".

Implementieren Sie die statische Methode **sucheOsterEier(String[][][] osterWelt)**, die alle Vorkommen der Zeichenketten "**easter egg**" findet und diese in einer Liste von 3-elementigen Listen protokolliert. In einer 3-elementigen Liste steht die x,y,z Position des Vorkommens der Zeichenkette "**easter egg**". Die Methode muss auch für leere Arrays, ein Arrays mit leeren Arrays, ein Array von Arrays mit leeren Arrays sowie für Arrays, die keine "**easter egg**" Zeichenkette enthalten, korrekte Antworten liefern (eine leere Liste). (**10Pkt**)

Beispiel:

**Eingabe:** Hier stehen die eckigen Klammern für Arrays

```
[[[null, null, null, null], [null], [null, null]],  
 [[null]],  
 [[easter egg, null, null, null, null, null], [null, null, null],  
  [null, null, null, null], [null], [easter egg, null]],  
 [[easter egg, null, null], [null]],  
 [[null, null, null, null, null, easter egg]],  
 [[easter egg, null, easter egg, null]]  
 ]
```

**Ergebnis:** Hier stehen die eckigen Klammern für Listen

```
[[2, 0, 0], [2, 4, 0], [3, 0, 0], [4, 0, 5], [5, 0, 0], [5, 0, 2]]
```

4. **Rekursion:** Implementieren Sie die statische Methode **maxDepth(Collection<?> col)** der Klasse A1. Die Methode bestimmt die maximale Verschachtelungstiefe einer Collection von Collections und gibt diese als Ergebnis zurück (**10Pkt**)

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

## A2 Collections & Streams & Sortieren

Gegeben ein einfaches Modell für ein Containerschiff, das aus den Klassen **Kiste**, **Palette**, **Container**, **Schiff** sowie den **Enums WarenTyp** und **GewichtsKlasse** besteht.

1. Eine **Kiste** fasst Waren eines bestimmten Typs (**Enum WarenTyp**) zusammen. Eine Kiste hat einen **wert** für den Gesamtwarenwert und ein **gewicht** für das Gesamtwarengewicht. Die Klasse und alle Methoden sind bereits implementiert.
2. Eine **Palette** kann mit Kisten bepackt (Methode **packen(Kiste)**) werden, dabei ist sicher gestellt, dass alle Kisten den gleichen Warentyp haben. Im Konstruktor wird das Nettogewicht und der Warentyp übergeben. Bei der Erzeugung wird für die Palette eine für einen Programmlauf eindeutige Id erzeugt. Der Konstruktor, die Methode **packen** und einige weitere Methoden sind bereits implementiert.

**TODOS:**

- a. Implementieren Sie unter Verwendung des Streaming API's die Methode **gewicht()**, die das Gewicht der Palette berechnet. Das Gewicht der Palette setzt sich zusammen aus dem Nettogewicht der Palette und der Summe der Gewichte der gepackten Kisten.
  - b. Implementieren Sie unter Verwendung des Streaming API's die Methode **wert()**, die den Wert der Palette berechnet. Der Wert der Palette ist die Summe der Wert der gepackten Kisten. (a. und b. **7 Pkt**)
3. Ein **Container** kann mit Paletten beladen werden (Methode **beladen(Palette)**). Ein Container wird mit einem Nettogewicht erzeugt. Bei der Erzeugung wird für den Container eine für einen Programmlauf eindeutige Id erzeugt. Der Konstruktor, die Methode **beladen** und einige weitere Methoden sind bereits implementiert. (Gesamt **13Pkt**)

**TODOS:**

- a. Implementieren Sie unter Verwendung des Streaming API'S die Methode **gewicht**. Verwenden Sie eine geeignete Methode von Palette und berücksichtigen Sie das Nettogewicht eines Containers.
  - b. Implementieren Sie unter Verwendung des Streaming API's die Methode **wert**. Verwenden Sie eine geeignete Methode von Palette. (a. und b. **7 Pkt**)
  - c. Implementieren Sie unter Verwendung des Streaming API's die Methode **gewichtsklasse**, die das Gewicht des Containers in eine der 3 Gewichtsklassen (siehe Enum **Gewichtsklasse**) einordnet und diese Gewichtsklasse als Ergebnis zurückgibt. Das Enum **Gewichtsklasse** bietet dazu die Methode **enthaltenIn(double gewicht)** an. Erzeugen Sie einen Strom über die Enum-Werte und filtern Sie die gesuchte **Gewichtsklasse**. (6 Pkt)
4. Das **Schiff** nimmt nur Container auf. Mit der Methode **beladen(Container)** werden die Container auf das Schiff geladen. Ein Schiff wird mit einem Namen erzeugt. Das Eigengewicht interessiert für diese Aufgabenstellung nicht. Die Methode **beladen**, der Konstruktor und Methoden, um aus einem Schiff Container über Ids oder Paletten über Ids zu lesen, sowie einige weitere Methoden sind bereits implementiert. (Gesamt **10 Pkt**)

**TODOS:**

- a. Implementieren Sie die Methode **ladungNachWert**, die die enthaltenen Container nach ihrem Wert unter Verwendung des Streaming-API's sortiert. Die Methode gibt eine nach Wert sortierte Liste von Container-Objekten zurück.
- b. Implementieren Sie die Methode **ladungNachGewicht**, die enthaltenen Container nach ihrem Gewicht unter Verwendung des Streaming-API's sortiert. Die Methode gibt eine nach Gewicht sortierte Liste von Container-Objekten zurück. (a. und b. **5 Pkt**)

- c. Implementieren Sie die Methode ***ladungNachGewichtsKlasse***, die die enthaltenen Container nach ihrer ***GewichtsKlasse*** sortiert. Lösen Sie dies mittels einer anonymen inneren Klasse, die die Ordnung der Objekte definiert. Die Methode gibt eine nach ***GewichtsKlasse*** sortierte Liste von Container-Objekten zurück.

**Hinweis:** Enums sind in Java ***Comparable***.

Die Methode darf die ursprüngliche Reihenfolge der Container-Objekte des Schiffes (Instanzvariable ***ladung***) **nicht verändern**. (5 Pkt)

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

**Die Testdaten werden in der Klasse *ContainerSchiffDatenGenerator* erzeugt.  
Auch diese Klasse dürfen Sie auf keinen Fall modifizieren!!!!**

**Datum -Zeit: 18.01.2017 – Beginn 10:00 Uhr**

**Nettobearbeitungszeit TEIL B: 60 Min**

**Punkteverteilung**

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

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

Maximale erreichbare Punktzahl für Teil B sind 60 Punkte. Sie haben 7 Punkte Puffer.

**Viel Erfolg! ☺**

## Teil B: Wissen und Verstehen

1. Erklären Sie, warum Konstruktoren, Operatoren und Basisdatentypen den objekt-orientierten Prinzipien widersprechen. (5Pkt)
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 } } };
```

Wie müssen Sie die beiden Arrays vergleichen, damit der Vergleich true ergibt? Erklären Sie kurz warum? (5Pkt)

3. Die nachfolgenden 2 Zeilen sind gültiger Java Code. Warum stellen Sie dennoch ein Problem dar? Wie nennt sich das Phänomen, das mit den 2 Zeilen dargestellt wird? (3Pkt)

```
Number[] nAry = new Integer[]{1,2,3};  
nAry[0] = 5.5;
```

4. Geben Sie je ein Beispiel für Coercion und das Liskovsche Substitutionsprinzip. (3 Pkt)
5. Ändern Sie die nachfolgende Methodensignatur und Methodendefinition so, dass (1) die erste Liste mit beliebigen Collections „merged“ werden kann und (2) die Methode merge die erste Liste l1 nicht modifiziert. (3Pkt)

```
public static <T> List<T> merge(List<T> l1,  
                           List<? extends T> l2) {  
    List<T> tResult = l1;  
    tResult.addAll(l2);  
    return tResult;  
}
```

6. Wie heißen die beiden Mechanismen, die nachfolgende Zuweisung in Java möglich macht? (2Pkt)

```
double d = new Integer(78);
```

7. Gegeben die nachfolgende sehr einfache Klassendefinition: (Gesamt: **6Pkt**)

```
public class Simple {  
    private int[] num;  
    public Simple(int[] num){  
        this.num = num;  
    }  
    public int[] getNum(){  
        return num;  
    }  
    @Override  
    public String toString() {  
        return String.format("Simple(%s)", Arrays.toString(num));  
    }  
}
```

- a. Erläutern Sie, dass die Klasse einen Datentyp definiert, deren Objekte **nicht immutable** (**2Pkt**) sind.
  - b. Ändern Sie die Definition derart, dass die Objekte immutable werden. (**2Pkt**).
  - c. Sind die Objekte nach der Änderung dann auch unique? Begründen Sie Ihre Antwort kurz. (**2Pkt**)
8. Für welche Referenztypen in Java gilt, dass sie immutable und unique sind? (**2Pkt**)
  9. Warum sollten Sie bei Problemen, die häufiges, wiederholtes Konkatenieren von Zeichenketten erfordern, nicht die Klasse String sondern die Klasse StringBuilder/StringBuffer verwenden? (**4Pkt**)
  10. 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. (**8Pkt**)

( )	Lists, Sets und Maps sind Collections.
( )	Jede Collection hat einen ListIterator.
( )	Kopier-Konstruktoren ermöglichen das Konvertieren zwischen den Collection-Klassen.
( )	Jede Collection ist iterierbar.
( )	Zu jedem Collection Interface gibt es eine Standardimplementierung für die Basisfunktionen als abstrakte Basisklasse.
( )	Objekte der abstrakten Basisklasse AbstractList sind nicht modifizierbare Listen.
( )	Die Methode keySet() ist die effizienteste Lösung, um über die key-value Paare einer Map zu iterieren.
( )	Iterieren mittels Iterator und Löschen von Elementen einer Liste ist nur erlaubt, wenn nur genau ein Iterator zur Zeit die Liste verarbeitet.

11. Nennen / Zeigen Sie für **einen** Wildcard-Typen (**3Pkt**)

- a. die Kompatibilitätsregel
- b. die Regeln zum Lesen von Elementen
- c. die Regeln zum Schreiben von Elementen

12. Ergänzen Sie die nachfolgenden Signaturen, so dass diese minimal gültige Einschränkungen für die Elemente in I2 definieren (**4 Pkt**).

```
public static <T> List<    > copyFrom(
    List<T> l,
    List<        > l2) {
    l.addAll(l2);
    return l;
}

public static <T> List<        > copyTo(
    List<T> l,
    List<        > l2) {
    l2.addAll(l);
    return l2;
}
```

13. 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. (**8Pkt**)

( )	Klassen können nur von einer Klasse ableiten aber dürfen mehrere Interfaces implementieren.
( )	Ein Objekt einer Klasse hat im allgemeinen mehreren Typen.
( )	Abstrakte Klassen, die nur abstrakte Methoden enthalten, sollten durch ein Interface ersetzt werden.
( )	Abstrakte Klassen dürfen keine konkreten Methoden enthalten.
( )	Template-Methoden kann es nur in abstrakten Klassen geben.
( )	Abstrakte Klassen dürfen von konkreten Klassen ableiten.
( )	Es macht keinen Sinn abstrakte Klassen final zu markieren.
( )	Abstrakte Klassen dürfen keinen Konstruktor und keine Instanz-Variablen enthalten.

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

15. Geben Sie ein einfaches Beispiel, das den Effekt des dynamischen Bindens deutlich macht.  
**(6Pkt)**

Datum -Zeit: 18.01.2017 – Beginn 9:00 Uhr

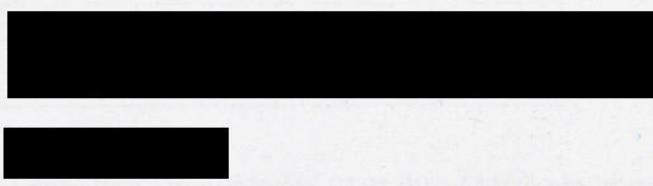
Nettobearbeitungszeit TEIL A: 60 Min

**Punkteverteilung**

	Punkte	Erreicht
<b>Teil A</b>		
a1.1	5	5
a1.2	5	4,5
a1.3	10	10
a1.4	10	9
a2.Kiste	7	7
a2.Container	13	11
a2.Schiff	10	10
<b>Teil A Gesamt</b>	<b>60</b>	<b>56,5</b>

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

Viel Erfolg! ☺



A: 56,5

B: 58

$\Sigma$ : 114,5      N.F. 15      9.2.2017      Wendholt

Datum -Zeit: 18.01.2017 – Beginn 10:00 Uhr

Nettobearbeitungszeit TEIL B: 60 Min

Punkteverteilung

	Punkte	Erreicht
<b>Teil B</b>		
1	5	4,5
2	5	4,5
3	3	3
4	3	1,5
5	3	3
6	2	2
7	6	1,3
8	2	2
9	4	3,5
10	8	6
11	3	3
12	4	4
13	8	7
14	5	5
15	6	6
<b>Teil B Gesamt</b>	<b>67</b>	<b>58</b>

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

Maximale erreichbare Punktzahl für Teil B sind 60 Punkte. Sie haben 7 Punkte Puffer.

Viel Erfolg! ☺

## Teil B: Wissen und Verstehen

1. Erklären Sie, warum Konstruktoren, Operatoren und Basisdatentypen den objekt-orientierten Prinzipien widersprechen. (5Pkt) **4.5 / 5 Pkt**

Konstruktoren sind nicht polymorph und werden nicht vererbt. ✓  
Operatoren sind keine Methoden für Referenztypen. (Kommune + bei  
Referatypen String) ✓  
Bandsatztypen sind keine Objekte, es können keine Methoden und  
Ihre aufgerufen werden. ✓

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 } } };
```

Wie müssen Sie die beiden Arrays vergleichen, damit der Vergleich true ergibt? Erklären Sie kurz warum? (5Pkt) **4.5 / 5 Pkt**

warum? (5Pkt) *Ansps. deepEquals(iArg1, iArg2);* ↳ wie arbeitet der Operator?

→ weil: Array.equals(a1,a2) nur "flach" Prüft und (nicht für  
genauere  
Array)  
array1.equals(array2) auf Identität ✓

3. Die nachfolgenden 2 Zeilen sind gültiger Java Code. Warum stellen Sie dennoch ein Problem dar? Wie nennt sich das Phänomen, das mit den 2 Zeilen dargestellt wird? (3Pkt) **3/3 Pkt**

```
Number[] nAry = new Integer[]{1,2,3};  
nAry[0] = 5.5;
```

Es handelt sich um eingeschlossene Kavarien von Areys  
Stetische Typen Nahrer und Hoffler sind zusammen homogenisiert,  
wobei  $\frac{1}{2} S.5$   $\frac{1}{2} S.5$

also kein Teller zu stark komplexität.  $\rightarrow$  dynamischer Typ!  
Teller zu leicht, der S.S. will zu integrität kompatibel ist.

4. Geben Sie je ein Beispiel für Coercion und das Liskovsche Substitutionsprinzip. (3 Pkt)

```
double d;
d = 1 + 3;    // int werden zu double
d = 3.0        // (implizit) konvertiert ✓
```

```
public class A {
    String a;
    return "a";
}
public class B extends A {
    String b;
    return "a" in b
}
Methoden a in B erweitert Methoden a in A
das ist nicht lösbar
```

5. Ändern Sie die nachfolgende Methodensignatur und Methodendefinition so, dass (1) die erste Liste mit beliebigen Collections „gemerged“ werden kann und (2) die Methode merge die erste Liste l1 nicht modifiziert. (3Pkt) 3/3 PLR

```
public static <T> List<T> merge(List<T> l1,
    Collection<List<? extends T>> l2) {
    List<T> tResult = l1; new ArrayList<T>(); tResult.addAll(l2);
    tResult.addAll(12);
    return tResult;
}
```

6. Wie heißen die beiden Mechanismen, die nachfolgende Zuweisung in Java möglich macht?  
(2Pkt) 2/2 PLR

double d = new Integer(78);

Autoboxing und Coercion ✓

7. Gegeben die nachfolgende sehr einfache Klassendefinition: (Gesamt: 6Pkt)

```
public class Simple {
    private int[] num;
    public Simple(int[] num){
        this.num = num;
    }
    public int[] getNum(){
        return num;
    }
    @Override
    public String toString() {
        return String.format("Simple(%s)", Arrays.toString(num));
    }
}
```

```
private List<Integer> num;
public Simple(int[] num) {
    this.num = Arrays.asList(num);
}
public List<Integer> getNum(){
    return num;
}
List<Integer> to String
```

hier steht das Problem  
bestehen?

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

- b. Ändern Sie die Definition derart, dass die Objekte immutable werden. (2Pkt). 1/2

- c. Sind die Objekte nach der Änderung dann auch unique? Begründen Sie Ihre Antwort kurz. (2Pkt)

a) Die gegebene Array ist nicht immutable, da mit getNum auf internen Array zugegriffen werden kann und mit zugriff, z.B. num[0]=0, verändert werden kann → nicht immutable  
gilt auch für den Kontrakt 1/2

c) Nein, nicht unique, da Zahlen als Wertespeicher Integers gespeichert werden, diese sind nicht unique. (Begründung)

z Simple Objekte mit gleichen / identischen Werten müssen nicht identisch sein 1/2

8. Für welche Referenztypen in Java gilt, dass sie immutable und unique sind? (2Pkt)

2/2

Wertespeicher enums. ✓

9. Warum sollten Sie bei Problemen, die häufiges, wiederholtes Konatenieren von Zeichenketten erfordern, nicht die Klasse String sondern die Klasse StringBuilder/StringBuffer verwenden? (4Pkt) 3. F / 4 Pkt

String sind immutable, d.h. beim Konatenieren entsteht immer ein neuer String. Bei häufigem Konatenieren wird "entkoppelt" der Garbage Collector ~~verantwortet~~ dieser alte String, aus dem Konatenement wurde. Dazu Progen beeinträchtigt die Performance stark und wird mithilfe des String Pooling nicht benötigt, da ~~wird~~ die Konatenationen unter ohne Verzögerung vollzogen wird ✓

10. 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. (8Pkt) 6/8 Pkt

✓	(A) Lists, Sets und Maps sind Collections.
✓	(F) Jede Collection hat einen ListIterator.
✓	(W) Kopier-Konstruktoren ermöglichen das Konvertieren zwischen den Collection-Klassen.
✓	(W) Jede Collection ist iterierbar.
✓	(F) Zu jedem Collection Interface gibt es eine Standardimplementierung für die Basisfunktionen als abstrakte Basisklasse.
✓	( ) Objekte der abstrakten Basisklasse AbstractList sind nicht modifizierbare Listen.
✓	(F) Die Methode keySet() ist die effizienteste Lösung, um über die key-value Paare einer Map zu iterieren.
✓	(W) Iterieren mittels Iterator und Löschen von Elementen einer Liste ist nur erlaubt, wenn nur genau ein Iterator zur Zeit die Liste verarbeitet.

11. Nennen / Zeigen Sie für **einen** Wildcard-Typen (3Pkt)

3/3 Pkt

- a. die Kompatibilitätsregel
- b. die Regeln zum Lesen von Elementen
- c. die Regeln zum Schreiben von Elementen

Upper Typebound      Extends B

Typ muss zu B kompatibel sein ↘

Lesen erlaubt ↘

Schreiben nicht erlaubt ↘

12. Ergänzen Sie die nachfolgenden Signaturen, so dass diese minimal gültige Einschränkungen für die Elemente in l2 definieren (4 Pkt).

4/4 Pkt

```
public static <T> List<T> copyFrom(      ↘  
    List<T> l,  
    List<? extends T> l2) {           ↘  
    l.addAll(l2);  
    return l;  
}
```

```
public static <T> List<? super T> copyTo(      ↘  
    List<T> l,  
    List<? super T> l2) {           ↘  
    l2.addAll(l);  
    return l2;  
}
```

13. 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. (8Pkt) **7/8 Pkt**

✓	(w) Klassen können nur von einer Klasse ableiten aber dürfen mehrere Interfaces implementieren.
✗	(w) Ein Objekt einer Klasse hat im allgemeinen mehreren Typen.
✗	(w) Abstrakte Klassen, die nur abstrakte Methoden enthalten, sollten durch ein Interface ersetzt werden.
✗	(f) Abstrakte Klassen dürfen keine konkreten Methoden enthalten.
✓	( ) Template-Methoden kann es nur in abstrakten Klassen geben.
✓	(w) Abstrakte Klassen dürfen von konkreten Klassen ableiten.
✗	(w) Es macht keinen Sinn abstrakte Klassen final zu markieren.
✓	(f) Abstrakte Klassen dürfen keinen Konstruktor und keine Instanz-Variablen enthalten.

14. Was wird in Java statisch gebunden? (5Pkt) 5/5 Pkt

- statische Methoden und Attribute ✓
- private Methoden und Attribute ✓
- alle Attribute ✓
- Konstruktor ✓
- alles was mit „final“ markiert ist (Methoden z.B.) ✓  
(Attribute und Methoden)

15. Geben Sie ein einfaches Beispiel, das den Effekt des dynamischen Bindens deutlich macht.  
(6Pkt) 5/6 Pkt

```
public class A {  
    public String a() {  
        return "a";  
    }  
}  
  
public class B extends A {  
    public String b() {  
        return "b";  
    }  
}  
  
{  
    A a = new A();  
    A b = new B();  
    b.b();  
    a.b();  
}
```

das wäre schon die ganze Lösung

// gilt "b" zurück, Methodenaufrufe beginnen im dynamischen Typ B, Methode b wird dort direkt gefunden und dynamisch gebunden

// gilt "a" zurück, Methodenaufrufe beginnen im dynamischen Typ B, dort a nicht gefunden, Suche geht weiter in Superklasse A, dort wird a gefunden und dynamisch gebunden.

## A1.java

```

1 package a1;
2
3
4 import java.util.ArrayList;
5
6
7 public class A1 {
8
9     /**
10      *
11      * 5/5 Pkt 27.1.2017 WND
12      *
13      * TODO 5 Pkt
14      *
15      * Berechnet die Formel summe k = 3..n  $8^k / (k^{2-1})^{2-2}$  Wirft eine
16      * IllegalArgumentException, wenn n ausserhalb des gueltigen Wertebereichs,
17      * d.h. n < 3.
18      *
19      * @param n
20      *      Schrittweite der Naerung
21      * @return double berechneter Wert der Naerungsformel
22      */
23
24     public static double sum(int n) {
25         if (n < 3) {
26             throw new IllegalArgumentException();
27         }
28         double erg = 0.0;
29         for (int k = 3; k <= n; k++) {
30             erg += (8 * k) / Math.pow((Math.pow(k, 2) - 1), 2);
31         }
32         return erg;
33     }
34
35 }
36
37 /**
38 *
39 * 4.5/5 Pkt 27.1.2017 WND
40 *
41 * TODO 5 Pkt Berechnet das erste n, fuer die der exakte Wert von der
42 * Naerungsformel um < eps abweicht. Fuer dieses n gilt dann: exakt-sum(n)
43 * < eps. Der Wert von exakt ist  $13.0/18 = 0.7222222222222222$ 
44 *
45 * @param eps
46 *      (double) maximaler Fehler der Naerung < 1
47 * @return int das erste n, fuer das gilt: exakt-sum(n) < eps
48 */
49     public static int fehlerSumKleinerEps(double eps) {
50         int n = 3;
51         while ((13 / 18 - sum(n)) > eps) { // Gleitkommaarithmetik -0.5Pkt
52             n++;
53         }
54         return n;
55     }
56
57 /**
58 *
59 * 10/10 Pkt 27.1.2017 WND
60 *
61 * TODO (10 Pkt)
62 */

```

Online konigstu Exemplar  
Original auf Wund  
erhiltlich WND

### A1.java

```
63 * osterWelt ist ein unregelmaessiges 3-dimensionales Array, das an
64 * zufaelligen Positionen "easter egg" Eintraege enthaelt. Die Methode soll
65 * die Positionen der "easter egg" Eintraege als Liste von 3-elementigen
66 * Listen zurueckgeben. Das erste Element der 3-elem Liste ist die x, das
67 * zweite die y, das dritte die z Position des jeweiligen "easter egg"
68 * Eintrages. Nutzen Sie die Konstante EASTER_EGG, um Tippfehler zu
69 * vermeiden.
70 *
71 * Beispieleingabe:
72 *
73 * [ [[null, null, null, null], [null], [null, null]], [[null]], [[easter
74 * egg, null, null, null, null, null], [null, null, null], [null, null,
75 * null, null], [null], [easter egg, null]], [[easter egg, null, null],
76 * [null]], [[null, null, null, null, null, easter egg]], [[easter egg,
77 * null, easter egg, null]] ]
78 *
79 * Beispielergebnis: [[2, 0, 0], [2, 4, 0], [3, 0, 0], [4, 0, 5], [5, 0, 0],
80 * [5, 0, 2]]
81 *
82 * @param osterWelt
83 *      3-dimensionales unregelmaessiges Arrays
84 * @return Liste mit 3 elementigen Listen, die die x,y,z Position der
85 * "easter egg" Eintraege enthaelt.
86 */
87 private static final String EASTER_EGG = "easter egg";
88
89 public static List<List<Integer>> sucheOsterEier(String[][][] osterWelt) {
90     List<List<Integer>> res = new ArrayList<List<Integer>>();
91     for (int x = 0; x < osterWelt.length; x++) {
92         for (int y = 0; y < osterWelt[x].length; y++) {
93             for (int z = 0; z < osterWelt[x][y].length; z++) {
94                 if (osterWelt[x][y][z] != null && osterWelt[x][y]
95                     [z].equals(EASTER_EGG)) {
96                     List<Integer> templist = new ArrayList<Integer>();
97                     templist.add(x);
98                     templist.add(y);
99                     templist.add(z);
100                    res.add(templist);
101                }
102            }
103        }
104    }
105 }
106
107 /**
108 *
109 * 9/10 Pkt 27.1.2017 WND
110 *
111 * TODO (10Pkt)
112 *
113 * Gegeben eine beliebig geschachtelte Collection von Collections. Bestimmen
114 * Sie die maximale Schachtelungstiefe.
115 *
116 * @param col
117 *      die Collection von Collection
118 * @return int die maximale Schachtelungstiefe. Die aesseere Collection wird
119 *         nicht mitgezaehlt
120 */
```

A1.java

### Container.java

```
1 package a2;
2
3 import java.util.ArrayList;
4
5
6 /**
7 * Klasse für die Repräsentation von Containern, die mit Paletten beladen
8 * werden. Gesamtpunktzahl: 13 Pkt
9 *
10 *
11 * @author Birgit Wendholt
12 *
13 */
14 public class Container {
15
16     private int id;
17     private int nettoGewicht;
18     private List<Palette> inhalt = new ArrayList<>();
19     private static int counter;
20
21     /**
22      * Konstruktor ist gegeben
23      *
24      * @param id
25      *          Identifikator des Containers.
26      * @param nettoGewicht
27      *          Nettogewicht des Containers in kg (ganzzahlig).
28      */
29     public Container(int nettoGewicht) {
30         this.id = ++counter;
31         this.nettoGewicht = nettoGewicht;
32     }
33
34     public Palette paletteMitId(String id){
35         for(Palette pal:inhalt ) {
36             if (pal.getId().equals(id)) return pal;
37         }
38         return null;
39     }
40     public String getId() {
41         return "C"+id;
42     }
43
44     /**
45      * Aufbereitung als Zeichenkette ist gegeben.
46      */
47     @Override
48     public String toString() {
49         return String.format("%s %2.2f€/%4.2fkg:%s\n", getId(), wert(), gewicht(),
50         inhalt);
51     }
52     /*
53      * Lädt eine Palette in den Container
54      */
55     public void beladen(Palette pal){
56         inhalt.add(pal);
57     }
58
59     //////////////////////////////////////////////////TODOS///////////////////////////////////////////////////
60     /**
```

## Container.java

```
61  *
62  * 7/7 Pkt 29.1.2017 WND
63  *
64  * a. und b. zusammen 7 Pkt
65  * TODO Bestimmt das Gewicht des Containers aus dem Nettogewicht und
66  * der Summe der Gewichte aller Paletten.
67  *
68  * @return Gewicht (in kg) als double
69  */
70 public double gewicht() {
71     return
72         inhalt.stream().map(p->p.gewicht()).reduce(Double.valueOf(nettoGewicht),(akk,g)->akk+
73             =g);
74     }
75 /**
76  * TODO Bestimmt den Wert des Containers aus der Summe der Werte
77  * aller Paletten.
78  *
79  * @return Wert (in €) als double
80  */
81 public double wert() {
82     return inhalt.stream().map(p->p.wert()).reduce(0.0,(w1,w2)->w1+=w2);
83 }
84 /**
85  *
86  * 4/6 Pkt 29.1.2017 WND
87  *
88  * TODO (6Pkt) Bestimmt die GewichtKlasse des Containers. Testet fuer jede
89  * Gewichtsklasse, ob das Gewicht des Containers in der Klasse enthalten
90  * ist. Lösung mit Gewichtsklasse#enthaltenIn(double), einem Strom über die
91  * Gewichtsklassen und einer geeignete Filtermethode.
92  *
93  * @return Gewichtsklasse des Containers
94  */
95 public Gewichtsklasse gewichtsklasse() {
96     List<Gewichtsklasse> gkl=new ArrayList<Gewichtsklasse>();
97     gkl.add(Gewichtsklasse.LEICHT);
98     gkl.add(Gewichtsklasse.MITTEL);
99     gkl.add(Gewichtsklasse.SCHWER);
100    // Stream.of(Gewichtsklasse.values()) -2 Pkt
101    return gkl.stream().filter(c->
102        c.enthaltenIn(this.gewicht())).findAny().get();
103    }
104
105
106 }
107
```

## Palette.java

```
1 package a2;
2
3 import java.util.ArrayList;
4
5 /**
6 * Klasse zur Repräsentation von Paletten. Eine Palette kann nur Waren desselben Typs
7 * aufnehmen. Der Typ wird bei der Erzeugung der Palette mit übergeben.
8 * Gesamtpunktzahl: 7 Pkt
9 * @author Birgit Wendholt
10 */
11
12 public class Palette {
13
14     private int nettoGewicht;
15     private List<Kiste> waren = new ArrayList<>();
16     private WarenTyp warenTyp;
17     private int id;
18     private static int counter = 0;
19
20     /**
21      * Konstruktor
22      * @param nettoGewicht Nettogewicht der Palette in kg (ganzzahlig).
23      * @param warenTyp Warentyp, der auf diese Palette gepackt werden kann.
24      */
25     public Palette(int nettoGewicht, WarenTyp warenTyp) {
26         this.nettoGewicht = nettoGewicht;
27         this.warenTyp = warenTyp;
28         this.id = ++counter;
29     }
30
31
32     public String getId() {
33         return "P"+id;
34     }
35     /**
36      * Anzahl der Kisten der Palette
37      */
38
39     public int size() {
40         return waren.size();
41     }
42     /**
43      * Beladen der Palette
44      * @param ware: eine Ware, die auf die Palette gepackt wird. Der Warentyp der Ware
45      * muss mit dem der Palette übereinstimmen.
46      */
47     public void packen(Kiste ware) {
48         if (ware.getWarenTyp() == warenTyp)
49             waren.add(ware);
50     }
51     /**
52      * Getter für den Warentyp
53      * @return WarenTyp der Waren der Palette
54      */
55     public WarenTyp getWarenTyp() {
56         return warenTyp;
57     }
58 }
```

Palette.java

```
59  /**
60   * Aufbereitung als Zeichenkette.
61   */
62  public String toString(){
63      return String.format("%s-%s %4.2f€/%4.2fkg--%s\n"
64      ",getId(),warenTyp,wert(),gewicht(),waren);
65  }
66
67  ////////////////////////////////////////////////////TODOS///////////////////////////////////////////////////
68  /**
69   *
70   * 7/7 Pkt 29.1.2017 WND
71   *
72   * TODO
73   * Bestimmt das Gewicht der Palette als Summe des Nettogewichtes und der Gewichte
74   * aller Waren mit Hilfe des Streaming API's
75   * @return Gewicht in (kg) als double.
76   */
77  public double gewicht() {
78      return
79      waren.stream().map(w->w.gewicht()).reduce(Double.valueOf(nettoGewicht),(akk,g)->akk+=g
80      );
81  }
82  /**
83   * TODO
84   * Bestimmt den Wert der Palette als Summe der Werte aller Waren.
85   * @return Gewicht in (kg) ganzzahlig.
86   */
87  public double wert() {
88      return waren.stream().map(w->w.wert()).reduce(0.0,(w1,w2)->w1+=w2);
89  }
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
```

## Schiff.java

```
1 package a2;
2
3 import java.util.ArrayList;
4
5 /**
6  * Klasse für Schiffe
7  * Gesamtpunktzahl: 10 Pkt
8  *     a.) und b.) 5 Pkt
9  *     c.) 5 Pkt
10 * @author Birgit Wendholt
11 *
12 * @param <T> ein zu Container kompatibler Typ
13 */
14
15 public class Schiff<T extends Container> {
16
17     private List<T> ladung = new ArrayList<>();
18     private String name;
19
20     public Schiff(String string){
21         this.name = string;
22     }
23
24     @Override
25     public String toString() {
26         return String.format("%s:%s", name, ladung );
27     }
28
29     /**
30      * Lädt einen Container auf das Schiff
31      * @param c der zu ladende Container
32      */
33     public void beladen(T c) {
34         ladung.add(c);
35     }
36
37     public Container containerMitId(String id){
38         for(Container c: ladung) {
39             if (c.getId().equals(id)) return c;
40         }
41         return null;
42     }
43
44     public int indexOf(String containerId){
45         int index = 0;
46         for(Container c: ladung) {
47             if (c.getId().equals(containerId)) return index;
48             index++;
49         }
50         return -1;
51     }
52
53     public Palette paletteMitId(String id){
54         for(Container c : ladung){
55             if (c.paletteMitId(id) != null) {
56                 return c.paletteMitId(id);
57             }
58         }
59         return null;
60     }
61
62 }
```

## Schiff.java

```
64    }
65
66    ////////////////////////////////////////////////////TODOS!!!!!!!!!!!!!!////
67    /**
68     *
69     * 5/5 Pkt 29.1.2017 WND
70     *
71     * Liefert eine Liste von Containern, die nach Wert sortiert sind. Lösung mittels
72     * Streaming API.
73     * @return Liste von Containern
74     */
75    public List<T> ladungNachWert(){
76        return ladung.stream().sorted((o1,o2)->Double.compare(o1.wert(),
77            o2.wert())).collect(Collectors.toList());
78    }
79
80    /**
81     * TODO
82     * Liefert eine Liste von Containern, die nach Gewicht sortiert sind. Lösung
83     * mittels Streaming API.
84     * @return Liste von Containern
85     */
86    public List<T> ladungNachGewicht() {
87        return ladung.stream().sorted((o1,o2)->Double.compare(o1.gewicht(),
88            o2.gewicht())).collect(Collectors.toList());
89    }
90
91    /**
92     * 5/5 Pkt 29.1.2017 WND
93     *
94     * TODO
95     * Liefert eine Liste von Containern, die nach Gewichtsklasse sortiert sind.
96     * Lösung mittels einer anonymen inneren Klasse.
97     * Instanzvariable ladung darf dabei dabei nicht modifiziert werden.
98     *
99     * Hinweis: Enums (Gewichtsklasse ist ein Enum) sind Comparable
100    * @return Liste von Containern
101    */
102   public List<T> ladungNachGewichtsklasse() {
103       List<T> templist= new ArrayList<T>();
104       templist.addAll(ladung);
105       Collections.sort(templist, new Comparator<T>(){
106           @Override
107           public int compare(T arg0, T arg1) {
108               return arg0.gewichtsklasse().compareTo(arg1.gewichtsklasse());
109           }
110       });
111       return templist;
112   }
113 }
```