

Aufgabe 1 (Modellieren und Schreiben von Klassen)

Sie sollen in dieser Aufgabe Klassen entwickeln, die Informationen über Impfstoffe repräsentieren.

- a) Entwickeln Sie eine Klasse **IS**. Ein Objekt dieser Klasse repräsentiert Informationen über einen Impfstoff. Sie soll vier Eigenschaften haben: einen Namen des Impfstoffs, der durch eine Zeichenkette dargestellt ist, ebenso den Namen des Herstellers, sowie einen Aufzählungswert, der anzeigt, um was für eine Impfstoffkategorie es sich handelt. Hier gibt es derzeit zwei Werte: Impfstoffe vom Typ **mRNA** und Impfstoffe vom Typ **vector**.

Ein viertes Feld **dosen** soll die Anzahl der verfügbaren Impfdosen des Impfstoffs angeben.

Schreiben Sie einen Konstruktor, der alle Felder initialisiert.

Überschreiben Sie die Methode **equals** auf adäquate Weise.

Überschreiben Sie die Methode **toString**.

```
enum Art { mRNA, vector; }
```

```
class IS {
```

```
    String name;
```

```
    String herst;
```

```
    Art art;
```

```
    int dosen;
```

```
    IS(String n, String h, Art a, int d) {
```

```
        name = n; herst = h; art = a; dosen = d;
```

```
    }
```

@Override

```
public boolean equals(Object o)?
```

```
if (o == null) return false;
```

```
if (o instanceof IS that)?
```

```
return this.name.equals(that.name)
```

```
    if this.host.equals(that.host)
```

```
    if this.art == that.art
```

```
    if this.dose == that.dose;
```

```
}
```

```
return false;
```

```
}
```

@Override

```
public String toString()?
```

```
return name + " von " + host;
```

```
}
```

```
}
```

- b) Schreiben Sie eine Unterklasse `RIS` der Klasse `IS`. Impfstoffe der Klasse `RIS` benötigen für einen umfassenden Impfschutz zwei Impfungen. Objekte dieser Klasse haben ein zusätzliches Feld, das den Abstand zwischen den beiden Impfungen in Tagen angibt.

```
class RIS extends IS {  
    int abstand;  
    RIS(String n, String h, Art a, int d, int ab){  
        super(n, h, a, d);  
        abstand = ab;  
    }  
}
```

- c) Gegeben sei die Klasse `Impfdosen`, deren Objekte eine Liste von Impfstoffen enthält:

```

class Impfdosen {
    java.util.List<IS> pos = new java.util.ArrayList<>();
}

```

Listing 1: `Impfdosen.java`

Schreiben Sie für diese Klasse eine Methode

```
int summeDerImmunisierbarenPersonen()
```

die die Summe aller Personen, die mit den vorhandenen Impfstoffen immunisiert werden können, berechnet. Berücksichtigen Sie dabei, dass man bei RIS Impfstoffen zwei Dosen zur Immunisierung einer Person braucht. **/*

```

    int r = 0;
    for (var is : pos) {
        if (is instanceof RIS ris) {
            r = r + ris.dosen / 2;
        } else {
            r = r + is.dosen;
        }
    }
    return r;
}

```

- d) Was brauchen Sie für ein Konstrukt, um die letzte Teilaufgabe zu lösen? Wo könnte man die Lösung mehr in objektorientierter Art und Weise realisieren?

Man brauche 'instanceof'.

Das ist nicht sehr objektorientiert.

Besser: es ist eine Methode, die mit den Dosen immunisierte Personen angibt.

Diese Methode dann in RIS zu übersetzen.

Zeile	x	y	i	Ausgabe
3, 4, 7, 8	14	5	5	I: 5 14 5
9, 10		8	4	A: 13 8
7, 14	13		<u>0</u>	I: 8 13 8
7, 1, 9		11	8	I: 3 13 11
10			7	I: 3 13 11
7, 8, 9		14	3	
10			2	
7, 16, 17	12		-2	A: 12 14

b) Betrachten Sie folgende Klasse:

```

1  Aufgabe2b {
2      e(a, b) {
3          (a == 0)    b;
4          (b == 0)    a;
5          a > 0 && a < b ? e(a-1, b) : e(a, b-1);
6      }
7
8      main(String[] args) {
9          System.out.println(e(19, 51));
10     }
11 }

```

Listing 3: Aufgabe2b.java

Berechnen Sie schrittweise das Ergebnis des Ausdrucks $e(19, 51)$.

$e(19, 51)$
 $\rightarrow e(19, 51 \% 19)$
 $\rightarrow e(19, 13)$
 $\rightarrow e(19 \% 13, 13)$
 $\rightarrow e(6, 13)$
 $\rightarrow e(6, 13 \% 6)$
 $\rightarrow e(6, 1)$
 $\rightarrow e(0, 1)$
 $\rightarrow 1$

- a) Schreiben Sie in der Klasse AL eine Methode `howMany`:

Es soll angegeben werden, wie viele Elemente in der Liste enthalten sind, die gleich zum Parameter `o` sind.

```
27 howMany(A, o) {  
28     int r = 0;  
29     for (var i = 0; i < size(); i++) {  
30         if (get(i).equals(o)) r++;  
31     }  
32     return r;  
33 }
```

Listing 5: AL.java

b) Gegeben sei zusätzlich folgende Schnittstelle:

```
1 public interface Property<A> {  
2     test(A a);  
3 }
```

Listing 6: Property.java

Sie soll verwendet werden, um zu testen, ob ein Objekt eine bestimmte Eigenschaft hat.

Schreiben Sie jetzt in der Klasse AL eine Methode `int howMany(Property<A> p)`.

Es soll angegeben werden, wie viele Elemente in der Liste enthalten sind, für die die Eigenschaft des Parameter `p` true ergibt.

```
1 int howMany(Property<A> p) {  
2       
3     int r = 0;  
4     for (var i = 0; i < size(); i++) {  
5         if (p.test(get(i))) r++;  
6     }  
7     return r;  
8 }
```

Listing 7: AL.java

- e) Schreiben Sie jetzt die Methode aus Aufgabenteil a) neu, unter Verwendung der Methode von Aufgabenteil b).

```

1  howMany(A a) {
2
3      return howMany(a -> a.equals(d));
4
5  }

```

Listing 8: AL.java

- d) Schreiben Sie folgende Methode `boolean isSymmetric()`:

Es soll getestet werden, ob die Liste vorwärts wie rückwärts gelesen, eine gleiche Elementreihenfolge enthält. Entsprechende Wörter werden auch als Palindrom bezeichnet, wie z.B. das Wort `LAGERREGAL`.

```

1  isSymmetric() {
2
3      for (var i = 0; i < size() / 2; i++) {
4          if (!get(i).equals(get(size() - 1 - i)))
5              return false;
6      }
7      return true;
8  }

```

Listing 9: AL.java

Aufgabe 4 Zusammenhänge

Erklären Sie in kurzen Worten:

a) Was ist beim Überschreiben von Methoden bezüglich Ausnahmen zu beachten?

Sie darf 'gefährlicher' werden.
Es dürfen keine 'neuen' Exceptions in
der throws Klausel sein, die nicht
bereits in der überschriebenen Version
stehen.

b) Welche Arten von Methoden können Bestandteil einer Schnittstelle sein?

- default - Methoden
- abstrakte - Methoden
- statische - Methoden

i) Aufruf des einer Konstruktors
des Oberklasse. Hier folgen dann
die Parameter in runden Klammern.
Die hier nur der erste Befehl ist
einem Konstruktor sein.

ii) Aufruf der speziellen Methode, wenn
die überschrieben wird, z.B.
super.equals (that)

- c) Können Sie eine eigene Unterklasse der Klasse String definieren?

Nein, denn String ist eine als final erklärte Klasse.

- d) Welche zwei Anwendungen hat das Schlüsselwort `super`? Was ist dabei zu beachten?
- e) Betrachten Sie folgende Klasse:

```

1  class A {
2      Object[] xs;
3      xs[1] =
4      {
5          main(String[] args){
6              Integer[] xs = {1,2,3,4};
7              f(xs);
8              System.out.println(xs[1]*2);
9          }
10     }

```

Kompiliert diese Programmfehler? Wenn nein, was für einen Fehler meldet der Compiler? Wenn ja, was ist an diesem Programm trotzdem problematisch?

Ja es kompiliert (wenn das Semitoken erfüllt wird).

Es kommt zu einem Laufzeitfehler, denn in dem Integer-Array versucht wird ein String zu speichern. (Zeile 3).