Funktionale und objektorientierte Programmierkonzepte

Kim Berninger, Jan Bormet, Dustin Glaser,
Julius Hardt und Alexander Mainka
(Gesamtleitung: Prof. Dr. Karsten Weihe)

Wintersemester 20/21
v1.2

Übungsblatt 3

Themen: FopBot, Klassen, Interfaces, Schleifen, Arrays, Methoden

Relevante Foliensätze: 01d-g,03a-03b

Abgabe der Hausübung: 04.12.2020 bis 23:50 Uhr

Wichtig und verpflichtend! Dokumentation in Java

Mittels JavaDoc und den Tags `@param` und `@return` wollen wir nun im Java-Teil eine geeignete Dokumentation unserer Methoden vornehmen. Das JavaDoc können Sie automatisch vor jeder Methode mittels `/**` gefolgt von der Enter-Taste generieren. Ein Beispiel könnte wie folgt aussehen:

```java
/**
 * This method accepts two real numbers belonging to a
 * vector and calculates the euclidean norm of said
 * vector.
 *
 * @param x first component of two-dimensional vector  (x, y)
 * @param y second component of two-dimensional vector (x, y)
 * @return Euclidean norm of the vector (x,y)
 */
double euclid2(float x, float y) {
    return Math.sqrt(x*x + y*y);
}
```

Die Dokumentation mit Vertrag ist Pflicht für alle Java-Methoden! Dies gilt insbesondere für die Hausübungen. Orientieren Sie sich dabei an obigem Beispiel. **Für jede fehlende Dokumentation einer Methode Ihrer abgegeben Hausübung erhalten Sie einen Punkt Abzug auf Ihre erreichten Punkte. Sie können auf diese Art und Weise bis zu 20 % der Punkte einer Hausübung abgezogen bekommen.**

V Vorbereitende Übungen

V1 Die Würfel sind gefallen! ★ ☆ ☆

Mit der Funktion `Math.random()` können Sie eine Zufallszahl im Bereich 0 (inklusive) und 1 (exklusive) erzeugen. Schreiben Sie nun eine Methode `void diceRoll()`.

Diese soll einen Würfelwurf simulieren und die gewürfelte Augenzahl auf der Konsole zurückgeben. Dabei soll der Würfel fair sein, das heißt alle Augenzahlen sollen mit identischer Wahrscheinlichkeit auftreten.

Hinweise: Überlegen Sie sich, wie Sie die erzeugten Zahlen aus dem Intervall $[0, 1)$ auf die diskrete Menge $\{1, 2, 3, 4, 5, 6\}$ abbilden können. Mit der Funktion `Math.ceil()` können Sie zum nächstgrößeren ganzzahligen Wert aufrunden.

V2 Schleifen ★ ☆ ☆

```
1 public int foo(int x) {
2     x = x++;
3     while (x*x < 1000) {
4         x++;
5     }
6     return x;
7 }
```

Welche Funktion erfüllt die oben stehende Methode `foo`? Ersetzen Sie die `while`-Schleife einmal durch eine `for`- und einmal durch eine `do-while`-Schleife, und zwar so, dass weiterhin die gleichen Schritte ausgeführt werden.

V3 Interfaces ★ ☆ ☆

Schreiben Sie ein Interface `I1` mit einer parameterlosen `void`-Methode `m1`. Nun schreiben Sie ein Interface `I2`, das von `I1` erbt. `I2` hat eine zusätzliche `void`-Methode `m2` mit einem `int`-Parameter `i`. Abschließend schreiben Sie eine Klasse `C1`, die das Interface `I2` implementiert. Die Klasse hat ein `int`-Attribut `number`. Beim Aufruf der Methode `m1` soll `number` auf `-1` gesetzt werden. Wird Methode `m2` aufgerufen, so soll `number` auf den übergebenen Wert gesetzt werden.

V4 Geometrische Formen I ★ ☆ ☆

Gegeben seien folgende zwei Klassen:

```
1 public class Circle {
2     public double radius;
3 }
4
5 public class Rectangle {
6     public double length;
7     public double width;
8 }
```

Schreiben Sie zunächst ein Interface `ComputeArea` mit einer parameterlosen double-Methode `computeArea`. Erweitern Sie nun die zwei oben genannten Klassen, sodass beide das Interface `ComputeArea` implementieren. Die Methode `computeArea` soll den Flächeninhalt des Kreises bzw. des Rechtecks berechnen und zurückliefern.

V5 Geometrische Formen II ★ ☆ ☆

Schreiben Sie eine Methode `double computeTotalArea` mit zwei Parametern `Circle[] circles` und `Rectangle[] rectangles`. Die Methode summiert die Flächeninhalte aller übergebenen geometrischen Formen in den beiden Arrays auf und gibt diese Summe zurück.

V6 Spieglein, Spieglein ... ★ ★ ☆

Wir nennen eine Gruppe von Elementen in einem Array Spiegel, wenn sie irgendwo im Array nochmal auftaucht, nur in umgekehrter Reihenfolge. Beispielsweise ist im Array `[7, 6, 5, 1, 9, 8, 5, 6, 7]` ein Spiegel vorhanden, und zwar `[7, 6, 5]`. Schreiben Sie eine Methode `int maxMirror(int[] arr)`. Diese bekommt ein Array übergeben und gibt die Länge des größten Spiegels im übergebenen Array zurück. Gibt es keinen Spiegel, so wird einfach 0 zurückgeliefert.

Hinweis: Starten Sie mit zwei Zeigern auf dem ersten und letzten Element. Vergleichen Sie nun die Elemente paarweise und überlegen Sie sich, wann Sie die beiden Zeiger weiter in die Mitte bewegen.

V7 Matrix-Multiplikation ★ ★ ☆

Der folgende Code stellt beispielsweise die Matrix $\begin{pmatrix} 5 & 8 \\ 1 & -3 \end{pmatrix}$ dar.

```
int[][] matrix = new int[2][2];
matrix[0][0] = 5;
matrix[0][1] = 8
matrix[1][0] = 1
matrix[1][1] = -3;

// alternativ und kuerzer: int[][] matrix = {{5,8},{1,-3}}
```

Sie sehen also, dass Sie einem Array in Java beliebig viele Dimensionen geben können.

Schreiben Sie eine Methode `int[][] matrixMul(int[][] mat1, int[][] mat2)`.

Die Methode bekommt zwei Matrizen, dargestellt durch zwei zweidimensionale Arrays, übergeben und gibt die resultierende Produktmatrix zurück. Sollte die Multiplikation aufgrund falscher Dimensionen nicht möglich sein, so geben Sie eine entsprechende Nachricht auf dem Bildschirm aus und liefern `null` zurück.

Hinweis: Verwenden Sie drei ineinander geschachtelte `for`-Schleifen. Die erste iteriert über die Reihen von `mat1`, die zweite über die Spalten von `mat1` und die Reihen von `mat2` und die letzte über die Spalten von `mat2`.

Weitere Roboter-Klassen

In vielen Aufgaben reichen uns die eingeschränkten Methoden eines Roboters der FopBot-Werke nicht. Daher definieren wir uns neue Roboter, welche die technischen Anforderungen erfüllen. In den Foliensätzen zu FopBot haben Sie bereits Beispiele wie den `SymmTurner` Roboter dazu gesehen.

V8 CoinPutter ★ ☆ ☆

Implementieren Sie zunächst in der aus der Vorlesung bekannten Roboter-Klasse `SymmTurner` die `public void`-Methode `coinMove(int countOfSteps)`: Diese soll `countOfSteps` Schritte nach vorne gehen und vor jedem Schritt einen Coin ablegen. Sollte die geforderte Anzahl an Schritten größer sein als die Anzahl an Coins soll der Roboter einfach stehen bleiben und sich ausschalten. Verwenden Sie dazu die Ihnen aus der Vorlesung bekannte `public`-Methode `void turnOff()`. Sollten mehr Coins vorhanden sein als Schritte gefordert sind, soll er an seiner finalen Position alle verbleibenden Coins ablegen.

V9 Richtungsdreher ★ ☆ ☆

In dieser Aufgabe soll eine neue Roboterklasse definiert werden, deren Roboter sich mittels eines einzigen Aufrufs in eine beliebige Richtung drehen können. Erstellen Sie dafür die Klasse `DirectionTurner`, die direkt von der Klasse `Robot` erbt und die parameter- und rückgabelosen `public`-Methoden `turnUp`, `turnRight`, `turnDown` und `turnLeft` so implementiert, dass der Roboter nach Aufruf einer dieser Methoden in die entsprechende Richtung blickt.

V10 TeamRobot ★ ★ ★

Erstellen Sie eine neue Klasse `TeamRobot`, die die Klasse `Robot` erweitert, also von ihr erbt. Der Konstruktor der Klasse `TeamRobot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und besitzt zusätzlich die Parameter `int left` und `int right`. Der Parameter `int left` gibt an, wie viele zusätzliche Roboter beim Aufruf des Konstruktors links neben des `TeamRobots` platziert werden. Der Parameter `int right` ist analog, für die Roboter rechts. Der `TeamRobot`, sowie die Roboter links und rechts von ihm bilden ein Team. Die zusätzlichen Roboter werden vom `TeamRobot` im Konstruktor erzeugt. Bekommt der `TeamRobot` einen Befehl, so soll dieser von allen Robotern im Team ausgeführt werden. Die zusätzlichen Roboter selbst sind dabei nicht ansprechbar, dass heißt auf ihnen können keine Methoden aufgerufen werden. Überlegen Sie sich, wie Sie die Roboter des Teams in der `TeamRobot`-Klasse speichern können und wie Sie die Befehle die ein `TeamRobot` erhält, an alle Roboter im Team weiterreichen können. Die Befehle meinen hier die Methoden: `move()`, `turnLeft()`, `pickCoin()` und `putCoin()`.

Beispiel: Beim Erstellen eines `TeamRobots` mit den Parametern `right = 1` und `left = 2` an der Position $(4, 4)$, werden zusätzlich 3 Roboter erstellt, die dem Team angehören, nämlich an den Position $(2, 4)$, $(3, 4)$ (links) und $(5, 4)$ (rechts).

V11 The final Countdown ★ ★ ★

In Unix-basierten Systemen wird die Zeit traditionell als vorzeichenbehaftete 32 Bit Ganzzahl gespeichert, die die seit dem 1. Januar 1970 vergangenen Sekunden repräsentiert.

Schauen Sie folgenden Java-Code an. Beheben Sie sämtliche eingebauten Fehler, um den Code lauffähig zu machen. Was müssen Sie im Code ändern, um die folgende Ausgabe zu erhalten?

```
"Am 19.1.2038 kommt es zu einem Ueberlauf des Unix Zeitstempels."
```

```java
public final class A {
    private int value1 = 0, value2 = 0;
    private final int value3 = 0;

    private int getValue1() {
        return value1;
    }

    private int getValue2() {
        return value2;
    }

    private void setValue1(int newValue1) {
        value1 = newValue1;
    }

    private void setValue2(int newValue2) {
        value2 = newValue2;
    }

    public final void changeValue3(final int newValue3) {
        value3 = 0;
    }
}

class B extends A {
    public void changeValue3(final int newValue3) {
        value3 = newValue3;
    }

    public static void main(String[] args) {
        B obj = new B();
        obj.setValue1(19);
        obj.setValue2(1);
        obj.value3 = 2038;

        String result = "Am " +
        getValue1() + "." +
        getValue2() + "." +
        obj.value3 +
        " kommt es zu einem Ueberlauf des Unix Zeitstempels.";

        System.out.println(result);
    }
}
```

H Dritte Hausübung Gesamt 25 Punkte
Fibonacci Zahlen & FOP-Zoo

Schwerpunkt dieser Hausübung sind grundlegende Konzepte wie verschachtelte Schleifen, abstrakte Klassen und Interfaces. H1 beschäftigt sich mit einem mathematischen Konstrukt, H2 mit einem Modell der Realität mithilfe von Klassen, Interfaces und Vererbung.

Die Aufgaben H1 und H2 sind unabhängig voneinander zu lösen. Beachten Sie, dass die Dokumentation des Quelltextes gemäß Seite 1 von Ihnen erwartet wird. Dies gilt insbesondere für H1.

Achtung: Achten Sie akribisch auf die Bezeichnungen Ihrer Methoden und Klassen Ihrer Abgabe, sodass diese denen der Aufgabenstellungen gleichen. Abweichende Bezeichner können zu Punktabzug führen.
Die bei der Definition einer Methode genutzten Bezeichner in der Klammer nach dem Methodennamen nennt man **formale Parameter**. Im Falle eines Methodenkopfes `void doSomething(int number)` ist `int number` der formale Parameter. Den Wert, mit dem die Methode im einzelnen Fall tatsächlich aufgerufen wird, nennt man dagegen **aktualer Parameter**. So wäre im Falle von `doSomething(5);` der Wert 5 der aktuale Parameter. Sollte dies noch unklar sein, finden Sie weitere Erläuterungen und Beispiele zu den Begrifflichkeiten in Foliensatz 03c am Anfang.

H1 Fibonacci 12 Punkte

In dieser Aufgabe geht es hauptsächlich um die Fibonacci Zahlen. Sie werden im Laufe der Aufgabe Schleifen, Methoden und Arrays benutzen, um die Fibonacci Zahlen auf verschiedene Weise zu berechnen.
Bei den Fibonacci Zahlen handelt es sich um eine Folge mit interessanten Eigenschaften. Sie weist viele Verbindungen zu Proportionen, Spiralen und Phänomenen der Natur auf. Zählt man zum Beispiel die rechts- und linksdrehenden und die vertikalen Spiralen der Ananasfrucht, so wird man feststellen, dass ihre Anzahlen in der Regel drei aufeinanderfolgende Fibonacci Zahlen sind, nämlich 8, 13 und 21. Die Fibonacci Zahlen sind mathematisch wie folgt definiert:

$$f_n = f_{n-1} + f_{n-2} \qquad \text{für } n \in \mathbb{N} \text{ mit } n \geq 2 \text{ und Startwerten } f_0 = 0,\ f_1 = 1 \qquad (1)$$

Daraus ergeben sich die ersten 10 Werte:

n	0	1	2	3	4	5	6	7	8	9
f_n	0	1	1	2	3	5	8	13	21	34

Alle Methoden der Aufgabe H1 werden in der Klasse `Fibonacci` erstellt.
Erstellen Sie alle Methoden so, dass sie auch außerhalb des Packages aufrufbar sind. Deklarieren Sie außerdem alle Methoden, die Sie in der Klasse `Fibonacci`implementieren, als statische Methoden, indem sie `static` davorschreiben.
Die Methoden werden so zu Klassenmethoden und können z.B. mit dem Klassennamen ohne ein Objekt der Klasse `Fibonacci` zu verwenden aufgerufen werden.
Sie haben eine solche Methode schon mal in Foliensatz 01f gesehen. Dort wurde in der Klasse `SlowMotionRobot` die Methode `Thread.sleep()` aufgerufen, ohne ein Objekt der Klasse Thread zu verwenden.

H1.1 Rekursive und iterative Implementation 2 Punkte

Schreiben sie nun die Methode `fibRek()`, die als formalen Parameter ein `int` hat und Rückgabetyp `int` hat. Die Methode soll bei Eingabe des Index `n` f_n nach Rekursionsvorschrift (1) rekursiv berechnen und zurückliefern.
Rekursive Methoden sind Methoden, die sich selbst aufrufen. Sie besitzen typischerweise einen Rekursionsanker, der die Methode abhängig von einem übergebenen Wert terminieren lässt. Bilden Sie die Startwerte der Fibonaccifolge auf den Rekursionsanker ab.
Dazu kommt noch eine Rekursionsvorschrift, die bestimmt, wie die Methode zur Berechnung des korrekten Wertes sich selbst aufruft. Verwenden Sie dazu die oben genannte Rekursionsvorschrift.
`fibRek()` mit formalem Parameter `n` soll also je nach Wert von `n` entweder einen der Startwerte zurückliefern, oder `fibRek()` einmal mit aktualem Parameter `n-1` und einmal mit aktualem Parameter `n-2` aufrufen, diese entsprechend der Rekursionsvorschrift verrechnen und zurückgeben.
Danach schreiben Sie eine Methode `fibIt()`, die als formalen Parameter ein Array von `int` erhält und nichts zurückliefert.
Die Methode soll das Array so füllen, dass jeweils an Index i im Array f_i steht. Nutzen sie hierzu einen iterativen Ansatz.

Verbindliche Anforderung: Es muss ein rekursiver Ansatz für `fibRek()` und ein iterativer Ansatz für `fibIt()` gewählt werden. Insbesondere dürfen sich die Methoden nicht gegenseitig aufrufen.

H1.2 Matrix Implementation **2 Punkte**

Eine weitere Darstellung der Fibonacci Zahlen ist die Matrixdarstellung.

Es gilt

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \tag{2}$$

Schreiben Sie eine Methode `fibMat()` mit einem formalen Parameter vom Typ `int` und Rückgabetyp `int`. Berechnen Sie in dieser Methode die Fibonacci Zahl des übergebenen Index unter Verwendung von (2) und der Methode `Util.matrixMultiplication()` aus der Vorlage und liefern diese zurück.

H1.3 Implementation via Summenformel **2 Punkte**

Eine weitere Darstellungsmöglichkeit der Fibonacci Zahlen ist die folgende Summenformel.

$$\begin{aligned} f_n &= \sum_{i=1}^{\frac{n}{2}} \binom{n-i}{i-1} && \text{für gerade } n \in \mathbb{N} \\ f_n &= \sum_{i=1}^{\frac{n+1}{2}} \binom{n-i}{i-1} && \text{für ungerade } n \in \mathbb{N} \end{aligned} \tag{3}$$

Schreiben Sie nun eine Methode `fibSum()` mit einem formalen Parameter vom Typ `int` und Rückgabetyp `int`. Berechnen Sie in dieser Methode die Fibonacci Zahl des übergebenen Index unter Verwendung von (3) und der Methode `Util.binomialCoefficient()` aus der Vorlage und liefern diese zurück.

H1.4 Collatz Folgen **2 Punkte**

Das sogenannte Collatz Problem ist ein noch ungelöstes Problem der Zahlentheorie. Eine Collatz Folge beginnt mit einem Startwert $n \in \mathbb{N}_{>0}$.
Ist der Wert gerade, so wird er halbiert. Ist der Wert ungerade, so wird er verdreifacht und dann um 1 inkrementiert.
Der Vorgang wird dann immer mit dem erhaltenen Wert wiederholt und so weiter. Es wird vermutet, dass eine so erzeugte Folge für jeden Startwert irgendwann den Zyklus $..., 4, 2, 1, ...$ erreicht und sich ab dann immer wieder wiederholt.
Erstellen Sie eine Methode `collatz()`, die einen formalen Parameter vom Typ `int` und Rückgabewert `int` hat. Ihre Methode soll zurückliefern, wie viele Elemente die Collatz Folge mit übergebenem Startwert hat, bis sie in den Zyklus $..., 4, 2, 1, ...$ gerät. Sie zählen also alle Folgenglieder bis (einschließlich) einmal die Teilfolge $4, 2, 1$ auftaucht.

Am Beispiel des Startwertes 3 ergibt sich die Collatz Folge $3, 10, 5, 16, 8, 4, 2, 1, \ldots$. Das Ergebnis wäre dann 8.

Mit Startwert 2 erhält man $2, 1, 4, 2, 1, \ldots$ und somit 5.

H1.5 Muster 4 Punkte

In dieser Aufgabe sollen Sie ein immer wiederkehrendes Muster in einem Array erkennen. Schreiben Sie hierzu die Methode `cycle()` mit einem formalen Parameter vom Typ `int[]` und Rückgabewert `int`. Berechnen Sie die Länge des kleinsten, ununterbrochen immer wiederkehrenden Musters im übergebenen Array und liefern diese zurück. Sollte kein solches Muster vorhanden sein, wird -1 zurückgeliefert. Sie können davon ausgehen, dass das übergebene Array mindestens die Länge 2 hat.
Zur Veranschaulichung folgen ein paar Beispiele:

```
{1, 2, 3,  1, 2, 3, 1, 2, 3, 1, 2, 3} -> 3
{3, 9, 3, 9} -> 2
{1, 7, 3, 1, 9, 4, 1, 7, 4} -> -1
{2, 2} -> 1
{1, 2, 5, 9, 1, 2, 5, 1, 2, 5} -> -1
{25, 1, 2, 1, 2, 1, 2} -> -1
{1, 2, 1, 2, 1, 2, 25} -> -1
```

H2 FOP-Zoo 13 Punkte

In dieser Aufgabe werden Sie mithilfe von Java Klassen, Vererbung und Interfaces einen Zoo modellieren. In Package `h2` finden sie bereits eine unvollständige Vorlage unseres Modells.
Unser Zoo soll mehrere Tiere verschiedener Tierarten halten können. Wir ordnen die Tierarten drei Kategorien unter – den Klassen `Vogel`, `Saeugetier` und `Reptil`. Zur Veranschaulichung folgt ein UML-Diagramm unseres Entwurfs.

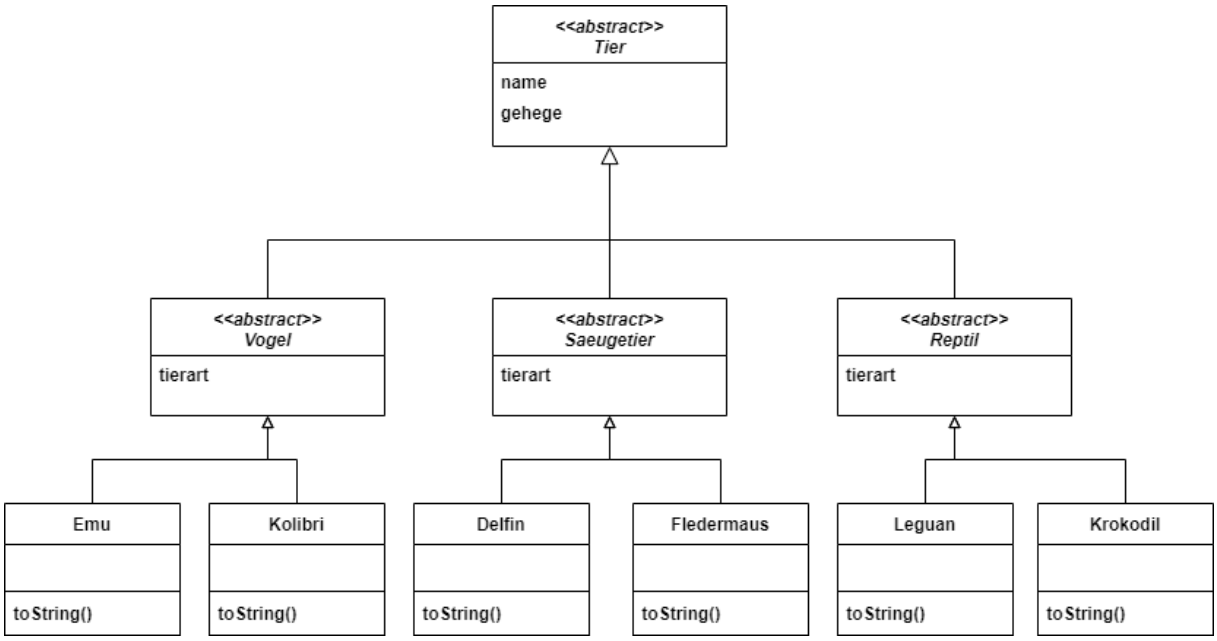

Abbildung 1: Ein vereinfachtes UML-Klassendiagramm unseres Zoo-Modells mit allen für Aufgabe H2.1 relevanten Informationen. Die durchsichtigen Pfeilspitzen bedeuten dabei „erbt von“. Der Rest sollte selbsterklärend sein.

Machen Sie sich zunächst mit der Vorlage vertraut. Ihnen wird auffallen, dass einige

Klassen `abstract` sind, obwohl sie keine unimplementierten Methoden enthalten. Für die Korrektheit des Codes wäre es nicht notwendig, diese Klassen `abstract` zu machen. Allerdings kann man so verhindern, dass Objekte dieser Klasse (und keiner Subklasse davon) erstellt werden. In unserem Modell wollen wir nämlich z.B. keine Reptilien, für die nicht spezifiziert ist, ob sie Leguane oder Krokodile sind.

H2.1 Tierarten — 3 Punkte

In der Vorlage fehlen noch die 6 Tierarten. Implementieren Sie nun die 6 Klassen, wie im obigen UML-Diagramm. Setzen Sie alle Klassen auf `public`.

Implementieren Sie dabei jeweils auch einen Konstruktor, welcher den jeweiligen super-Konstruktor aufruft.

Implementieren Sie in den Klassen jeweils eine Methode `public String toString()` so, dass sie zum Beispiel für ein Objekt der Klasse `Delfin` mit `name = "Flipper"` folgendes zurückliefert: `"Ich bin Delfin Flipper, wohne im Gehege Aqualand und bin ein Saeugetier"`.

Entnehmen Sie dabei den Namen des Geheges dem `name` Attribut des jeweiligen `Gehege` Attributes und die Kategorie dem `tierart` Attribut.

Sollte das `gehege` Attribut `null` sein, wird stattdessen der String `"Ich wurde noch nicht zugeteilt"` zurückgeliefert.

Implementieren Sie außerdem die Methode `vorstellen()` in der Klasse `Zoo`, die für alle Tiere im `tiere` Array in der Reihenfolge im Array die Methode `toString()` aufruft, alle zurückgelieferten Strings zu einem einzigen String mit je einem Zeilenumbruch dazwischen konkateniert (ganz am Ende wird auch ein Zeilenumbruch eingefügt) und diesen zurückliefert.
Zur Illustration hier ein Beispiel:
Sei `d` unser Delfin von oben und `t` ein Tier, dessen Gehegeattribut `null` ist. Sei das Array `tiere = new Tier[]{d, t};`. Dann wird der folgende Ausdruck zu `true` auswerten:

```
vorstellen().equals("Ich bin Delfin Flipper, wohne im Gehege Aqualand und
    bin ein Saeugetier\nIch wurde noch nicht zugeteilt\n")
```

H2.2 Umsiedeln — 2 Punkte

In unserem Modell soll es auch möglich sein, dass die Tiere das Gehege wechseln können. Schreiben Sie dazu die Methode(n) `public void umsiedeln()` mit einem formalen Parameter vom Typ `Gehege`, die das `gehege` Attribut des Tiers auf den übergebenen Parameter setzt. Entscheiden Sie, an welcher/welchen Stelle(n) diese Methode am meisten Sinn ergibt, und implementieren Sie sie dort und nur dort. Die Methode soll mit jeder Referenz von Referenztyp `Tier` oder Subtyp von `Tier` aufgerufen werden können.

H2.3 Fliegen und Schwimmen 4 Punkte

Zuletzt wollen wir noch modellieren, dass einige Tierarten fliegen bzw. schwimmen können. Diese Fähigkeiten stellen Sie mit den Interfaces `Fliegen` mit der nicht-implementierten Methode `fliegen()` und `Schwimmen` mit der nicht-implementierten Methode `schwimmen()` dar. Beide Methoden erhalten keine formalen Parameter und haben den Rückgabewert `String`.
Die Tierarten `Kolibri` und `Fledermaus` sollen fliegen können. Die Tierarten `Krokodil` und `Delfin` sollen schwimmen können.
Die Methoden `fliegen()` bzw. `schwimmen()` der genannten Tierarten sollen dabei folgende Strings zurückliefern: `"schnelles flattern"`, `"kreuz und quer"`, `"paddeln"` und `"elegantes schwimmen"` (In Reihenfolge Kolibri, Fledermaus, Krokodil, Delfin).
Implementieren Sie außerdem die Methoden `flugshow()` und `schwimmtraining()` der Klasse `Zoo`. Die Methode `flugshow()` soll die Methoden `fliegen()` aller Tiere im `tiere` Array, die fliegen können, aufrufen und die zurückgelieferten Strings wie in Methode `vorstellen()` zu einem String zusammenfassen. Dieser String wird dann zurückgegeben. Dasselbe gilt für die Methode `schwimmtraining()` - natürlich für die Methode `schwimmen()` aller Tiere, die schwimmen können.

Hinweis: Verwenden sie den `instanceof`-Operator.

H2.4 Gehege 4 Punkte

In dieser Aufgabe beschäftigen wir uns mit den Regeln für die Zuweisung der Tiere in ein Gehege. Dabei gibt es folgendes zu beachten:

a) Ein Gehege hat immer nur beschränkt viel Platz. In unserem Fall entspricht die maximale Zahl der Bewohner der Länge des `tiere` Arrays des jeweiligen Geheges. Ein Gehege kann kein Tier aufnehmen, wenn es voll ist.

b) Tiere, die nicht fliegen können und selbst keine Krokodile sind, dürfen nicht zusammen mit Krokodilen in ein Gehege.

c) Emus dürfen nicht mit Tieren in ein Gehege, die fliegen können. Sie werden sonst eifersüchtig.

d) Leguane und Fledermäuse vertragen sich nicht. Sie dürfen nicht zusammen in ein Gehege.

Implementieren Sie die Methode `public boolean add(Tier t)` der Klasse Gehege. Wenn es laut den oben beschriebenen Kriterien keinen Grund gibt, warum das übergebene Tier nicht zu den Tieren im `tier` Array aufgenommen werden sollte, soll sich nach Aufruf der Methode `add(Tier t)` das übergebene Tier im `tiere` Array des Geheges befinden und `true` zurückgeliefert werden. Außerdem soll mittels der bereits von Ihnen implementierten Methode `umsiedeln(Gehege g)` das `gehege` Attribut des Tiers entsprechend auf das `Gehege` Objekt gesetzt werden.
Sollte wenigstens eines der beschriebenen Kriterien verletzt sein, wird das `tiere` Array nicht verändert, die Methode `umsiedeln(Gehege g)` nicht aufgerufen und `false` zurückgeliefert.

In der Klasse `Tier` ist eine Methode `boolean kompatibel(Tier t)` implementiert, die standardmäßig `true` zurückliefert. Sie dürfen diese Methode in den Klassen der Tierarten überschreiben und in `boolean add(Tier t)` aufrufen, um die Kriterien b) - d) zu überprüfen.
Es gibt viele Möglichkeiten und viele Implementationen, um das Problem zu lösen. Wichtig ist, dass die Methode `boolean add(Tier t)` exakt wie beschrieben funktioniert.
Sie können davon ausgehen, dass auch in den Tests Tiere nur über die
`boolean add(Tier t)` Methode zu einem Gehege hinzugefügt werden und außerdem keine Tiere aus einem Gehege entfernt werden.

Verbindliche Anforderung: Der Operator `instanceof` darf in dieser Aufgabe nur in den Methoden `kompatibel(Tier t)` der Klasse `Tier` und deren Unterklassen verwendet werden.