Inoffizielle Lösungen und Erklärungen - inf-schule© (*Lizenz*)

2.3 Objektorientierte Programmierung mit Java

1.2

Teddy

Benötigte IDEs:

Greenfoot, BlueJ

Verfasser:

Niko Diamadis

Erstellungs-/ Änderungsdatum

29. Juni 2020

Inhaltsverzeichnis

1 Das Spiel **1**

1.1 Vorbereitungen . . . 1

1.2 Basis-Teddy . . . 1

1.3 Attribute . . . 1

1.4 Syntaxfehler . . . 2

1.5 Konstruktor . . . 2

1.6 act . . . 2

1.7 verringereLeben . . . 3

1.8 Wert . . . 3

1.9 Spiel fertig stellen . . . 3

2 Fachkonzept - OOM **4**

3 Fachkonzept - OOP **5**

4 Übungen **6**

4.1 Struktur von Klassen . . . 6

4.2 Eine Würfel-Klasse . . . 6

4.3 Eine Klasse für Brüche . . . 7

4.4 Ein objektorientiertes Modell herleiten . . . 8

5 Übungsprojekt - Autobewertung **9**

5.1 Ein mathematisches Modell für ein Auto finden . . . 9

5.2 Ein objektorientiertes Modell für ein Auto finden . . . 9

5.3 Ein Automodell objektorientiert implementieren . . . 10

5.4 Objekte initialisieren . . . 10

5.5 Attributwerte verändern . . . 11

5.6 Erweiterung des Klassendiagramms . . . 11

5.7 Erweiterung der Implementierung . . . 11

5.8 Eigene Ideen . . . 12

5.9 Grenzen von Modellen 12

1 Das Spiel

1.1 Vorbereitungen

Zu dieser Aufgabe sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig. Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

1.2 Basis-Teddy

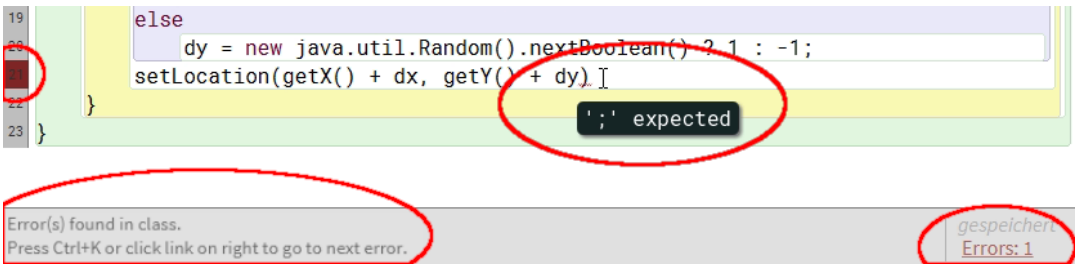
Anhand des Quellcodes ist zu erkennen, dass folgende Attribute und Methoden einem Teddy schon zugeordnet sind:

- Attribute
  - sämtlichte Attribute der Überklasse Actor (`class Teddy extends Actor`)
- Methoden
  - `springen` ohne Rückgabewert und ohne Parameter (`void springen()`)
  - sämtlichte Methoden der Überklasse Actor (`class Teddy extends Actor`)

1.3 Attribute

Im Objektinspektor sind wie erwartet die Attribute `leben` und `alter` wiederzufinden, es hat demnach alles geklappt.

1.4 Syntaxfehler



In Greenfoot sind Syntaxfehler im Quellcode wie im Bild zu sehen an vier Stellen erkennbar:

- rot markierte Zeilenangaben
- rot unterstrichener Teil des Quellcodes und Fehlerbeschreibung
- Fehlermeldung und Angabe, wie man schnell zum Fehler gelangt (Strg + K oder Klick auf Meldung unten rechts)
- Angabe der Fehleranzahl unten rechts in der Ecke

1.5 Konstruktor

Zu dieser Aufgabe sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig. Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

1.6 act

Es gibt mehrere Arten, die Variable um 1 zu erhöhen:

1. (funktioniert auch mit komplexeren Rechnungen)

```
1 public void act() {
2     alter = alter + 1;
3 }
```

2. (funktioniert auch bei Addition mit anderen Zahlen)

```
1 public void act() {
2     alter += 1;
3 }
```

3. (funktioniert nur für Addition mit 1)

```
1 public void act() {
2     alter++;
3 }
```

1.7 verringereLeben

```
1 public void verringereLeben(int betrag) {
2     leben = leben - betrag;
3 }
```

1.8 Wert

- In der Methode `getWert` ist als Rückgabewert `int` festgelegt.

  Der zurückzugebene Wert ergibt sich aus Abziehen des Wertes des Attributs `alter` von 100.

- In der Methode `hatWert` ist als Rückgabewert `boolean` festgelegt.

  Wenn der zurückzugebene Wert der Methode `getWert` größer als 0 ist, gibt diese Methode `true` zurück, andernfalls wird `false` zurückgegeben.

1.9 Spiel fertig stellen

Zu dieser Aufgabe sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig. Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

2 Fachkonzept - OOM

Zu dieser Seite sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig.

Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

3 Fachkonzept - OOP

Zu dieser Seite sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig.

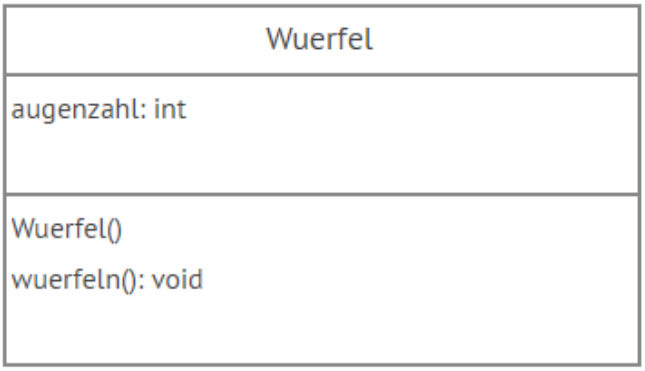
Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

4 Übungen

4.1 Struktur von Klassen

```
class Klassenname {

datentyp attributname ;

Klassenname( startwerte) {
  // Initialisierung
}

rückgabetyp methodenname (formale parameter) {
  // Implementierung
}

}
```

4.2 Eine Würfel-Klasse



Wir erstellen nur eine Methode, welche man selber aufrufen kann und welche im Konstruktor aufgerufen wird. Diese Methode soll dem Würfel eine Zahl von 1 bis 6 zuweisen.

```
1  public class Wuerfel {
2      int augenzahl;
3
4      Wuerfel() {
5          wuerfeln();
6      }
7
8      void wuerfeln() {
9          augenzahl = new java.util.Random().nextInt(6) + 1;
10     }
11 }
```

Der Ausdruck `new java.util.Random().nextInt(6)` bewirkt, dass eine Zufallszahl von 0 bis 5 erstellt wird, da ein Würfel aber die Augenzahlen 1 bis 6 besitzt, addieren wir einfach 1.

4.3 Eine Klasse für Brüche

Im Vorhinein ist noch zu sagen, dass ich (wie vielleicht schon erkannt) Freund von kurzem Code bin (siehe **1.6**). Daher bevorzuge ich die dort auch schon angesprochene Schreibweise (wenn der Summand größer als 1 ist):

```
1  alter += 1;
```

In dieser Aufgabe sind aber Multiplikationen und Divisionen gefordert. Die einfache Schreibweise ähnelt der ersten Schreibweise bei **1.6**.
Wenn wir nun aber die Schreibweise vereinfachen wollen, können wir bei der zweiten Schreibweise von **1.6** das „+“ durch ein „*“ bzw. ein „/“ ersetzen, womit Multiplikationen und Divisionen mit dem dann folgenden Faktor bzw. Divisor ausgedrückt werden.

Nun zurück zur Aufgabe...

```
1  public class Bruch {
2      int zaehler;
3      int nenner;
4
5      Bruch(int z, int n) {
6          zaehler = z;
7          nenner = n;
8      }
9
10     void erweitern(int k) {
11         zaehler *= k;
12         nenner *= k;
13     }
14
15     void kuerzen(int k) {
16         zaehler /= k;
17         nenner /= k;
18     }
19 }
```

Wenn man Brüche zu kürzen versucht, die nicht mehr zu kürzen sind, dann werden Zähler und Nenner jeweils mit dem Faktor dividiert und das Ergebnis jeweils gerundet.
In diesem Falle ist dann das Ergebnis folglich falsch!

4.4 Ein objektorientiertes Modell herleiten

Hier gebe ich keine Lösungsansätze bzw. Lösungen, da es sich jeder unterschiedlich vorstellt, wie ein Autorennen im Computer aussehen soll. Hier kann man seinen Ideen freien Lauf lassen.
Solange man sich an die richtige Syntax hält, kann bei dieser Aufgabe nichts falsch gemacht werden.

5 Übungsprojekt - Autobewertung

5.1 Ein mathematisches Modell für ein Auto finden

Man erkennt hoffentlich schnell, dass es z.B. eine Exponentialfunktion sein kann.
Wenn man dann in die allgemeine Funktionsvorschrift verschiedene Werte unter 1 als Basis eingibt und die resultierenden Graphen mit dem auf inf-schule vergleicht (und entsprechend die Basis anpasst), kommt man ungefähr zu folgender Funktion:

$$f(alter) = neupreis * 0,87^{alter}$$

Die Funktion muss die y-Achse schneiden, da der y-Achsenabschnitt in unserem Falle den Neupreis des Autos darstellt.

5.2 Ein objektorientiertes Modell für ein Auto finden

- `neupreis : int`
  ganzzahliger Preis (aufgrund von genereller Ungenauigkeit reicht die Angabe in €)
- `alter : double`
  Alter als Gleitkommazahl, da das Alter in Jahren angegeben wird, man aber auch die Angabe in kleineren Schritten ermöglichen möchte
- Die Methoden haben alle als Rückgabewert `double`, da mit dem Attribut `alter` gerechnet wird, welches vom Datentypen `double` ist, es kommen also Ergebnisse mit Nachkommastellen heraus, welche z.B. mit dem Datentyp `int` nicht darzustellen sind.
- Die Methoden benötigen keine Parameter, da die Attribute in allen Methoden benutzt werden, also die Angabe der Attribute methodenunabhängig getätigt wird, sodass nicht bei jedem Aufruf einer Methode z.B. der Neupreis angegeben werden muss. Die Methoden greifen auf die klassenweiten Attribute zu.

5.3 Ein Automodell objektorientiert implementieren

- Kriegst du hin, oder? Du musst nur den Code aus dem Video kopieren.
- Bisher wird von beiden Methoden nur der Wert `0` ausgegeben, da weder `neupreis` noch `alter` einen Wert von uns zugewiesen bekommen haben.
- Das `return` und der danach folgende Ausdruck gibt an, welchen Wert die Methode zurückgeben soll.
- Der Ausdruck `Math.pow(0.87, alter)` ist die Umschreibung einer Potenz im Java-Code (`pow` für „power“ (englisch für Potenz). Der erste angegebene Wert ist die Basis und der zweite Wert (mit Komma getrennt) stellt den Exponenten dar.
- Mit `return neupreis - getMarktwert()` gibt die entsprechende Methode als Rückgabewert die Differenz von `neupreis` und dem Rückgabewert der `getMartkwert`-Methode zurück.
- Versteht man glaube ich schnell, wenn man es sieht...

```
1 double getHaendlerEk() {
2     return getMarktwert() * 0.8;
3 }
```

5.4 Objekte initialisieren

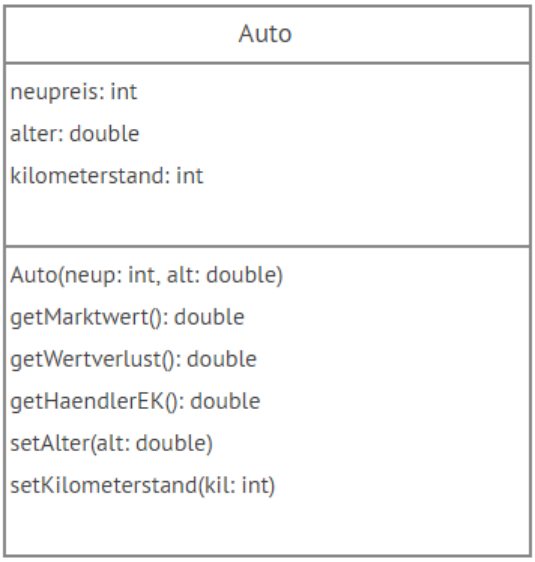
- Keine Anleitungen und/oder Erläuterungen nötig...

Gemeinsamkeiten	Unterschiede
Parameter	Konstruktor kann keinen Wert zurückgeben
	Konstruktorname entspricht immer dem Klassennamen
	Konstruktor nur einmalig vor allen Methoden aufrufbar

5.5 Attributwerte verändern

Zu dieser Aufgabe sind meines Erachtens nach keine Anleitungen und/oder Erläuterungen nötig. Wenn doch Fragen aufkommen, schreib' einfach an **nikodiamond3@gmail.com**.

5.6 Erweiterung des Klassendiagramms



5.7 Erweiterung der Implementierung

Wir geben, wie vorgeschlagen, den Kilometerstand in „Tausend Kilometern" an, sodass ich durch Ausprobieren grob auf folgende Basis kam:

$$f(kilometerstand) = neupreis * 0,995^{kilometerstand}$$

In den Code implementiert könnte es z.B. so aussehen:

```
1 int kilometer;
2
3 Auto(int neup, double alt, int kil) {
4     neupreis = neup;
5     alter = alt;
```

```
6       kilometer = kil;
7   }
8
9   double getMarktwert() {
10      return neupreis * Math.pow(0.87, alter) * Math.pow(0.995, kilometer);
11  }
12  void setKilometerstand(int kil) {
13      kilometer = kil;
14  }
```

5.8 Eigene Ideen

Hier lasse ich deinen Ideen mal freien Lauf, viel Spaß beim Implementieren und Ausprobieren!
Bei Fragen einfach an **nikodiamond3@gmail.com** schreiben.

5.9 Grenzen von Modellen

Das bisher benutzte Modell passt nur zu Autos in der Preisklasse des VW Golfs und im selben Zustand wie dieser (also ohne große Schäden o.Ä.), andere teurere oder billigere Autos würden einen anderen Faktor aufweisen.

Außerdem funktioniert die Modellierung des Werteverlustes nicht unendlich lange, irgendwann werden Autos z.B. wieder wertvoller (wenn der Zustand dementsprechend ist).

Verbessern könnte man auch noch die Rückgabewerte der Methoden, da diese sehr viele Nachkommastellen besitzen können, was im Kontext wenig Sinn ergibt. Ich verschone dich aber lieber mit Funktionen zum Runden.