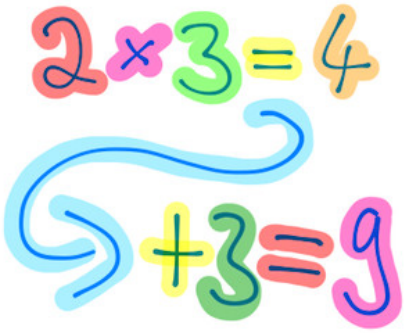
Inoffizielle Lösungen und Erklärungen - inf-schule (*Lizenz*)

7.1 Objektorientierte Programmierung mit Java

2.1

Superbrain



Benötigte IDEs:

BlueJ

Verfasser:

Niko Diamadis (Cyb3rKo)

Erstellungs-/ Änderungsdatum

11. September 2021

Inhaltsverzeichnis

1 Basisversion

1.1 Grundgerüst

Die Methode `rateEinmal` bewirkt, dass die Konsole mit der Ausgabe „Wie viel ist x + y?“ erscheint. Anschließend wird auf eine Eingabe in der Konsole gewartet. Sobald diese geschehen ist, wird „Das Ergebnis war z“ ausgegeben.
Es fällt jedoch auf, dass `z` oft nicht mit der Summe aus `x` und `y` übereinstimmt. Das liegt daran, dass bei der zweiten Ausgabe neue Zufallszahlen erzeugt werden und auf die vorherigen Zahlen nicht mehr eingegangen wird.

Um noch kurz auf die Syntax einzugehen:
Ein „+“ ein Anführungszeichen wird als String in der Konsole ausgegeben, ohne Anführungszeichen ist es ein mathematisches Rechenzeichen und dient nur der Berechnung, es wird also nicht für den Nutzer sichtbar.

1.2 lokale Variablen

Es gibt mehrere Möglichkeiten der Zeilenanordnung, jedoch nur eine der Schnipselanordnung.
Hier zeige ich die richtige Schnipselanordnung und eine der Zeilenanordnungen:

```
int zahl1 =    zufallszahl(10);
int zahl2 =    zufallszahl(10);
int loesung =  zahl1 + zahl2;

System.out.println( "Wie viel ist "  + zahl1 + " + "  + zahl2 + "?");

int antwort = leseZahl();

System.out.println( "Das Ergebnis war "  + loesung );
```

1.3 Fallunterscheidungen in Java

```
if( antwort == loesung ) {
    System.out.println("Super! Richtig gerechnet.");
}
else {
   System.out.println("Leider falsch. Richtig war:" +
   loesung);
}
```

Auch hier ist es wichtig, die Syntax kennenzulernen.

Ein einfaches „=“ bedeutet, dass der Variable links vom „=“-Zeichen der Wert auf der rechten Seite zugewiesen wird.

Ein zweifaches „=“ bewirkt, dass überprüft wird, ob der Wert auf der linken Seite dem rechten entspricht.

1.4 rateEinmal mit Rückgabe

```
1  boolean rateEinmal() {
2      ...
3
4      if (antwort == loesung) {
5          System.out.println("Super! Richtig gerechnet.");
6          return true;
7      } else {
8          System.out.println("Leider falsch. Richtig war:" + loesung);
9          return false;
10     }
11 }
```

Damit die Methode zurückgeben kann, ob richtig oder falsch beantwortet wurde, ändert man den Rückgabetypen von `void` zu `boolean`.
Zuletzt gibt man „wahr“ als Rückgabe an (`return true`), wenn die Antwort richtig war, bzw. „falsch“, falls die Antwort nicht richtig war (`return false`).

1.5 eineRundeSpielen

Ich gebe natürlich wieder die Lösung der Hilfe an, jedoch ist darunter eine (meiner Meinung nach) etwas einfachere und kürzere Version dieses Codes, Erklärungen folgen danach.

Oben im Quelltext der Klasse müssen drei Attribute eingefügt werden mit `int punkte1;`, `int punkte2;` und `int runden;`. Die Methode `eineRundeSpielen` besteht aus den (sortierbaren) Zeilen:

```
void eineRundeSpielen()
{
runden = runden + 1;
boolean richtigGeraten1 = rateEinmal();
boolean richtigGeraten2 = rateEinmal();
if(richtigGeraten1)
{
punkte1 = punkte1 + 1;
}
if(richtigGeraten2)
{
punkte2 = punkte2 + 1;
}
System.out.println("Spieler 1 hat " + punkte1 + " Punkt(e)");
System.out.println("Spieler 2 hat " + punkte2 + " Punkt(e)");
System.out.println(); // Für etwas Abstand zur nächsten Runde
}
```

```
1  void eineRundeSpielen() {
2      runden++;
3
4      if (rateEinmal()) {
5          punkte1++;
6      }
7
8      if (rateEinmal()) {
9          punkte2++;
10     }
11
12     System.out.println("Spieler 1 hat " + punkte1 + "Punkt(e)");
13     System.out.println("Spieler 2 hat " + punkte2 + "Punkt(e)");
14
15     System.out.println(); //Fuer etwas Abstand zur naechsten Runde
16 }
```

Um eventuell Verwirrung vorzubeugen, hier eine Erklärung zu dem Ausdruck `if(rateEinmal())`: Normalerweise erwartet man ein `==true` oder `==false`, wenn man diese Ausdrücke jedoch weglässt, wird automatisch von `==true` ausgegangen.
`if(rateEinmal())` bedeutet also soviel wie `if(rateEinmal() == true)`.

Nun zu meiner Variante:
Ich habe die Variablen `richtigGeraten` ausgelassen, da man direkt in der if-Bedingung die Methode aufrufen und so den boolean-Wert abfragen kann ohne extra dafür Variablen erstellen zu müssen.

Dadurch, dass ich zweimal eine if-Schleife mit der Methode `rateEinmal` als Bedingung aufgestellt habe, werden meist auch verschiedene Werte für die beiden Spieler erzeugt, da die Methode dementsprechend auch zweimal aufgerufen wird.
Außerdem habe ich in meiner Version die Addition von 1 wieder etwas vereinfacht (durch das `++`).

1.6 spielSteuern

Die Schleife zum Erfüllen der Aufgabe, dass nämlich `eineRundeSpielen` aufgerufen werden soll, solange `runden` < 5 erfüllt ist, ist vorgegeben.

Der Rest ist mithilfe des angegebenen Struktogramms einfach umzusetzen, sodass ich nicht davon ausgehe, dass Fragen zu dem Code entstehen:

```
1  void spielSteuern() {
2      while (runden < 5) {
3          eineRundeSpielen();
4      }
5
6      if (punkte1 == punkte2) {
7          System.out.println("Unentschieden");
8      } else {
9          if (punkte1 > punkte2) {
10             System.out.println("Spieler 1 hat gewonnen");
11         } else {
12             System.out.println("Spieler 2 hat gewonnen");
13         }
14     }
15 }
```

1.7 Spielende

Die grün markierten Ausdrücke sind geeignet, um die gewünschte Gewinnregel zu erfüllen:

runden $<$ 10 && !(punkte1 $>=$ 5 && punkte2 $>=$ 5)

(runden $<$ 10 && !(punkte1 $>=$ 5)) && (runden $<$ 10 && !(punkte2 $>=$ 5))

(runden $<$ 10 && !(punkte1 $>=$ 5)) || (runden $<$ 10 && !(punkte2 $>=$ 5))

runden $<$ 10 && !(punkte1 $>=$ 5 || punkte2 $>=$ 5)

runden $<$ 10 && punkte1 $<$ 5 && punkte2 $<$ 5

Einen der Ausdrücke setzt du jetzt nur noch in die while-Bedingung ein, dann kannst du es auch schon testen.

1.8 Erste Anpassungen

Ich nehme als Anpassungen, die ich machen möchte, die Beispiele, die aufgeführt wurden:

- Ich möchte die Ausgabe insofern anpassen, dass statt `Spieler 1` bzw. `2` der Name benutzt wird, den man am Anfang angibt. Dies löse ich mithilfe des Konstruktors:

```
String spieler1;
String spieler2;

Spiel(String spieler1, String spieler2) {
    this.spieler1 = spieler1;
    this.spieler2 = spieler2;
}
```

  Der Ausdruck `this.x = x` weist einer klassenweiten Variable, dem Attribut `x`, den Wert x aus der Methode (also in dem Fall den im Konstruktor anzugebenen Wert) zu.
  Zu beachten ist, dass die Namen im Konstruktor aufgrund des Datentypes in Anführungszeichen angegeben werden muss.

  Nun kann man in den Konsolenausgaben jeweils das `Spieler 1` bzw. `2` durch `spieler1` bzw. `2` ersetzen.

- Den Zahlenbereich kann man einfach durch Anpassen des Parameters ändern, mit welchem man die Methode `zufallszahl` aufruft, z.B. `zufallszahl(1000)`.

- Abschließend füge ich noch Zugriffsmodifikatoren bei Methoden und Attributen hinzu:

```
    private int runden;
    private int punkte1;
    private int punkte2;
    private String spieler1;
    private String spieler2;

```

```
7     public Spiel(...) {
8         ...
9     }
10
11    public void spielSteuern() {
12        ...
13    }
14
15    private void eineRundeSpielen() {
16        ...
17    }
18
19    private boolean rateEinmal() {
20        ...
21    }
22
23    private int zufallszahl(...) {
24        ...
25    }
26
27    private int leseZahl() {
28        ...
29    }
```

Das einzige, worauf der Nutzer zugreifen soll ist der Konstruktor und die Methode `spielSteuern`, da diese das Aufrufen der anderen Methoden übernimmt.

2 Fachkonzept - lokale Variablen

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

Wenn doch Fragen aufkommen, schreib' einfach an **niko@cyb3rko.de**.

3 Fachkonzept - Fallunterscheidung

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

Wenn doch Fragen aufkommen, schreib' einfach an **niko@cyb3rko.de**.

4 Fachkonzept - Bedingungen

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

Wenn doch Fragen aufkommen, schreib' einfach an **niko@cyb3rko.de**.

5 Übungen

5.1 Zufälliger Operator

```
int zahl1;
int zahl2;
int loesung;
String operator;
int operatorNummer = zufallszahl(4);
if(operatorNummer == 0) {
operator = "+";
zahl1 = zufallszahl(100) + 100;
zahl2 = zufallszahl(100) + 100;
loesung = zahl1 + zahl2;
}
else if(operatorNummer == 1) {
operator = "-";
zahl1 = zufallszahl(100) + 100;
zahl2 = zufallszahl(100) + 100;
loesung = zahl1 - zahl2;
}
else if(operatorNummer == 2) {
operator = "*";
zahl1 = zufallszahl(15) + 1;
zahl2 = zufallszahl(15) + 1;
loesung = zahl1 * zahl2;
}
else {
operator = "/";
loesung = zufallszahl(10);
zahl2 = zufallszahl(9)+1; // 0 darf hier nicht vorkommen.
zahl1 = zahl2 * loesung;
}
System.out.println("Wie viel ist " + zahl1 + operator + zahl2 + "?");
```

5.2 Zeitmessung

Bei dieser Aufgabe muss ziemlich viel angepasst werden, um die Zeitmessung zu implementieren.

Da die while-Bedingung nun nicht mehr die Runden und die Punktzahl beinhaltet, soll sie jetzt aus Punktzahl und der Restzeit pro Spieler bestehen. Daher kann das Runden-Attribut nun entfernt werden. Stattdessen habe ich für beide Spieler analog zu der Punktzahl ein Restzeit-Attribut hinzugefügt, diese habe ich dann in die while-Bedingung eingebaut.

Die Zeit, die man nun also benötigt, um eine Aufgabe zu lösen, wird in einem Attribut gespeichert und später von der Restzeit des Spielers abgezogen. Zusätzlich dazu wird die verbrauchte und die verbleibende Zeit ausgegeben.

Da der Code dazu nun stark angepasst wurde, hier mein Quellcode:

```
1  class Spiel {
2      private int punkte1;
3      private int punkte2;
4      private String spieler1;
5      private String spieler2;
6      private int restzeit1;
7      private int restzeit2;
8      private int gebrauchteZeit;
9
10     public Spiel(String spieler1, String spieler2) {
11         restzeit1 = 30;
12         restzeit2 = 30;
13         this.spieler1 = spieler1;
14         this.spieler2 = spieler2;
15     }
16
17     public void spielSteuern() {
18         while (punkte1 < 5 && punkte2 < 5 && restzeit1 > 0 && restzeit2 > 0)
               {
19              eineRundeSpielen();
```

```
20          }
21
22          if (punkte1 == punkte2) {
23              System.out.println("Unentschieden");
24          } else {
25              if (punkte1 > punkte2) {
26                  System.out.println(spieler1 + " hat gewonnen");
27              } else {
28                  System.out.println(spieler2 + " hat gewonnen");
29              }
30          }
31      }
32
33      private void eineRundeSpielen() {
34          if (rateEinmal()) {
35              punkte1++;
36              restzeit1 -= gebrauchteZeit;
37              System.out.println("Restzeit: " + restzeit1 + " Sekunden");
38          } else {
39              restzeit1 -= gebrauchteZeit;
40              System.out.println("Restzeit: " + restzeit1 + " Sekunden");
41          }
42
43          if (rateEinmal()) {
44              punkte2++;
45              restzeit2 -= gebrauchteZeit;
46              System.out.println("Restzeit: " + restzeit2 + " Sekunden");
47          } else {
48              restzeit2 -= gebrauchteZeit;
49              System.out.println("Restzeit: " + restzeit2 + " Sekunden");
50          }
51
52          System.out.println(spieler1 + " hat " + punkte1 + "Punkt(e).");
53          System.out.println(spieler2 + " hat " + punkte2 + "Punkt(e).");
54
55          System.out.println();
```

```
56      }
57
58      private boolean rateEinmal() {
59          int zahl1;
60          int zahl2;
61          int loesung;
62          String operator;
63          int operatorNummer = zufallszahl(4);
64
65          if (operatorNummer == 0) {
66              operator = "+";
67              zahl1 = zufallszahl(100) + 100;
68              zahl2 = zufallszahl(100) + 100;
69              loesung = zahl1 + zahl2;
70          } else if (operatorNummer == 1) {
71              operator = "-";
72              zahl1 = zufallszahl(100) + 100;
73              zahl2 = zufallszahl(100) + 100;
74              loesung = zahl1 - zahl2;
75          } else if (operatorNummer == 2) {
76              operator = "*";
77              zahl1 = zufallszahl(15) + 1;
78              zahl2 = zufallszahl(15) + 1;
79              loesung = zahl1 * zahl2;
80          } else {
81              operator = "/";
82              loesung = zufallszahl(10);
83              zahl2 = zufallszahl(9) + 1;
84              zahl1 = zahl2 * loesung;
85          }
86
87          System.out.println("Wie viel ist " + zahl1 + operator + zahl2 + "?");
88          int start = (int)(System.currentTimeMillis() / 1000);
89          int antwort = leseZahl();
90          int ende = (int)(System.currentTimeMillis() / 1000);
91          gebrauchteZeit = ende - start;
```

```
92
93          if (antwort == loesung) {
94              System.out.println("Super! Richtig gerechnet.");
95              System.out.println("Du hast " + gebrauchteZeit + " Sekunden
                    gebraucht.");
96              return true;
97          } else {
98              System.out.println("Leider falsch. Richtig war:" + loesung);
99              System.out.println("Du hast " + gebrauchteZeit + " Sekunden
                    gebraucht.");
100             return false;
101         }
102     }
103
104     private int zufallszahl(int n) {
105         return new java.util.Random().nextInt(n);
106     }
107
108     private int leseZahl() {
109         return new java.util.Scanner(System.in).nextInt();
110     }
111 }
```

5.3 Eigene Erweiterungen

Hier ist wieder deine Kreativität gefragt, viel Spaß beim Implementieren deiner Ideen.

Bei Fragen, einfach an **niko@cyb3rko.de** schreiben.

5.4 Fehlermeldung

➤ Die erste Fehlermeldung erscheint, da nach dem Zurückgeben eines boolean-Wertes die Methode beendet wird und die letzte Zeile in diesem Beispiel gar nicht erst aufgerufen werden kann.

➤ Hier entsteht eine Fehlermeldung, da in der letzten Zeile versucht wird, den String `s` auszugeben, jedoch ist `s` nur innerhalb der beiden Klammern definiert. Außerhalb der Klammern kann nicht mehr darauf zugegriffen werden.

5.5 Typische Fehler

➤ **Fehler 1:**

Wenn es zwei Variablen desselben Namens gibt, wird bei Benutzung die lokale Variable bevorzugt. Auf das Attribut kann mit `this.variable` zugegriffen werden.

Somit wird hier der lokalen Variable `runden` der Wert derselben Variable zugewiesen, das klassenweite Attribut `runden` wird nicht verändert.

➤ **Fehler 2:**

Die Zuweisung ist falsch herum aufgestellt. Mit dem Beispielcode wird der lokalen Variable `r` der Wert von `runden` zugewiesen, bevor sie entfernt wird.

Es müsste dem Attribut `runden` der Wert von `r` zugewiesen werden, ansonsten ändert sich `runden` auch hier nicht.

➤ **Fehler 3:**

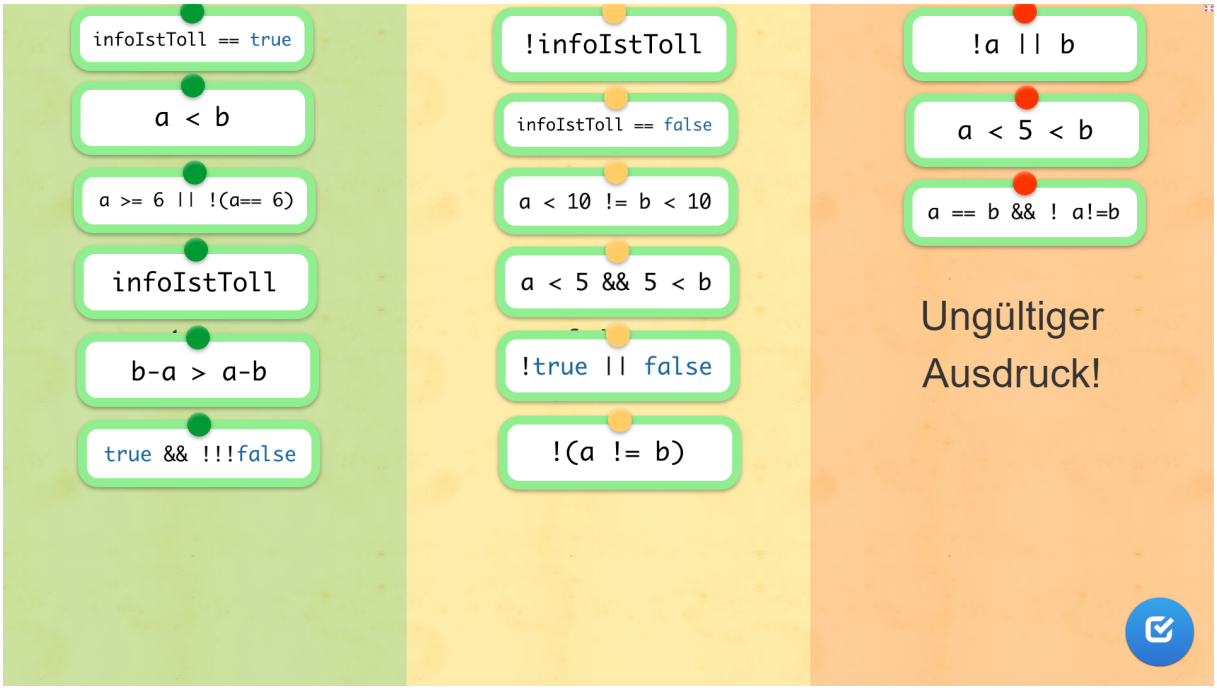
Bei der Zuweisung müsste man das `int` weglassen, da die Variable `runden` im Konstruktor nicht neu implementiert werden darf, da sonst die Variable wieder nur lokal existiert und nach Konstruktor-Aufruf wieder entfernt wird. Auch hier würde sich die klassenweite `runden`-Variable nicht ändern.

5.6 Lokal vs. Attribut

Beispiel 1 funktioniert, da bei Implementierung eines Attributs als Integer automatisch der Wert 0 zugewiesen wird.

Im Beispiel 2 funktioniert es nicht, da bei Implementierung einer lokalen Variable keine automatische Zuweisung erfolgt und sie gar keinen Wert besitzt.

5.7 Logische Ausdrücke



5.8 Maximum-Funktion

➤ **max1:**

Diese Methode funktioniert wie erwartet. Wenn `a` größer ist als `b`, wird `a` zurückgegeben, ansonsten (also sowohl wenn `b` größer ist als auch wenn `b` gleich groß ist wie `a`) wird **b** zurückgegeben (weil wenn sie gleich groß sind, ist es egal, ob `a` oder `b` zurückgegeben wird).

➤ **max2:**

Diese Methode funktioniert auch wie erwartet, denn sie gleicht der ersten Methode stark. Es wurde einfach das `else` weggelassen, denn wenn die if-Bedingung erfüllt ist, wird `a` zurückgegeben und die Methode beendet, sodass die letzte Zeile nicht mehr durchgeführt wird.

➤ **max3:**

Diese Methode funktioniert nicht aufgrund von einem fehlenden return-Statements. Wenn keine der beiden if-Bedingungen erfüllt ist (was in diesem Beispiel nicht funktionieren kann, aber generell zu beachten ist), gibt es kein return-Statement, welches man ausführen kann. Ich sehe nur eine Lösungsmöglichkeit, wenn man die beiden Bedingungen beibehalten möchte:

`return 0` am Ende einfügen

5.9 Bruch-Klasse

Ob eine Zahl durch eine andere teilbar ist, lässt sich mit dem modulo-Operator überprüfen. Der Modulo gibt den Rest einer Division an.

10 mod 3 (sprich „zehn Modulo drei“)[1] z.B. ergibt also 1.

Um also zu testen, ob der Zähler bzw. Nenner durch `k` teilbar ist, geben wir in der Methode `kuerzbarMit` true zurück, falls der Modulo 0 ist.

```
1 boolean kuerzbarMit(int k) {
2     if ((zaehler % k == 0) && (nenner % k == 0)) {
3         return true;
4     } else {
5         return false;
6     }
7 }
```

[1] Modulo (mod) berechnen, mathe24, http://www.mathe24.net/modulo.html, (Abgerufen: 11.02.20, 17:58)

Um diese Methode nun benutzen zu können, baut man diese dann in die Methode `kuerzen` ein.

```
1 void kuerzen(int k) {
2     if (kuerzbarMit(k)) {
3         zaehler = zaehler / k;
4         nenner = nenner / k;
5     }
6 }
```