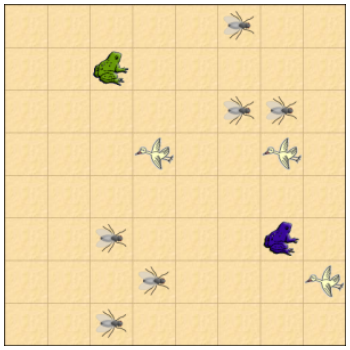
Inoffizielle Lösungen und Erklärungen - inf-schule© (*Lizenz*)

2.3 Objektorientierte Programmierung mit Java

2.3

Frog



Benötigte IDEs:

Greenfoot, BlueJ

Verfasser:

Niko Diamadis

Erstellungs-/ Änderungsdatum

30. Juni 2020

Inhaltsverzeichnis

1 Spielidee

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

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

2 Implementierung

2.1 Vorlage öffnen

Der gewünschte Effekt tritt nicht ein, da zwar abgefragt wird, welches Bild das zu erzeugende Objekt erhalten soll, jedoch wird dem Objekt noch kein Bild zugewiesen, sodass das Objekt das Standardbild (einen Greenfoot-Fuß) erhält.

2.2 Aufbau der Dokumentation

Beim Öffnen der Dokumentation landet man auf der Übersicht aller Klassen inklusive Beschreibung dieser.
Wenn man nun auf eine der Klassen klickt, kommt man auf eine klasseneigene Seite. Dort stehen oben eine Beschreibung, wofür diese Klasse benutzt wird, und weitere Informationen zur Nutzung der Klasse.

Weiter unten findet man anschließend mögliche Konstruktoren, teilweise auch hier mit Beschreibung.

Noch weiter unten sind alle Methoden mit Signatur und Beschreibung aufgelistet. Einige der Methoden werden unter der Tabelle weiter beschrieben und erklärt, wie sie zu benutzen sind.

2.3 Methoden finden

- void **setImage**(*Greenfoot Image*) - Actor
- int **getX()** - Actor
  int **getY**() - Actor
- void **setLocation**(*int x, int y*) - Actor
- void **addObject**(*Actor object, int x, int y*) - World
- int **getWidth**() - World
- static int **getRandomNumber**(int limit) - Greenfoot
- static void **stop**() - Greenfoot

2.4 Basisfunktionalität

➤ Setzen des Bildes:

```
1  public Frosch(String startbild) {
2      kraft = 20;
3      setImage(startbild);
4  }
```

➤ Erzeugen eines grünes und eines blaues Frosch am Anfang (die Größe der Frosch-Welt ist im Konstruktor abzulesen, sie ist 8 x 8 Felder groß):

```
1  public FrogWorld() {
2      super(8, 8, 60);
3
4      int x = Greenfoot.getRandomNumber(getWidth());
5      int y = Greenfoot.getRandomNumber(getHeight());
6
7      addObject(new Frosch("froschgruen.png"), x, y);
8
9      x = Greenfoot.getRandomNumber(getWidth());
10     y = Greenfoot.getRandomNumber(getHeight());
11
12     addObject(new Frosch("froschblau.png"), x, y);
13 }
```

Damit die Zeilen wieder nicht zu lang werden, habe ich hier wieder lokale Variablen benutzt.

2.5 Springen

Zum Lösen benutze ich den schon gegebenen Lösungsansatz.

```
1  public void act() {
2      ...
3      kraft--;
4  }
5
6  public void springen() {
7      int xDiff = Greenfoot.getRandomNumber(3) - 1;
8      int yDiff = Greenfoot.getRandomNumber(3) - 1;
9
10     setLocation(getX() + xDiff, getY() + yDiff);
11 }
```

2.6 Fliegen würfeln

➤
- **1. Zeile**

  Bedingung, welche erfüllt ist, wenn eine bei jeder Wiederholung neu erstellte Zufallszahl kleiner 5 ist. Solange sie erfüllt ist, wird der Code in den geschweiften Klammern immer wieder wiederholt.

- **2. Zeile**

  Die lokale Variable `x` wird erstellt und zufällig mit einem Wert maximal so groß wie die Breite der Welt belegt.

- **3. Zeile**

  Die lokale Variable `y` wird erstellt und zufällig mit einem Wert maximal so groß wie die Höhe der Welt belegt.

- **4. Zeile**

  Ein neues Objekt wird in der Welt an die Stelle (`x`—`y`) gesetzt.

➤ Damit es weniger Störche gibt als Fliegen, habe ich die Warscheinlichkeit, dass die Bedingung

in der zweiten Schleife erfüllt ist, herabgesetzt.

Außerdem benutze ich `x` und `y` wieder, da man die Koordinaten der beiden Frösche nicht mehr braucht.

```
while (Greenfoot.getRandomNumber(6) < 5) {
    x = Greenfoot.getRandomNumber(getWidth());
    y = Greenfoot.getRandomNumber(getHeight());
    addObject(new Fliege(), x, y);
}

while (Greenfoot.getRandomNumber(6) < 3) {
    x = Greenfoot.getRandomNumber(getWidth());
    y = Greenfoot.getRandomNumber(getHeight());
    addObject(new Storch(), x, y);
}
```

2.7 Fliegen fester Anzahl

Auch hier habe ich das Beispiel verwirklicht.

```
int i = 0;

while (i < 5) {
    x = Greenfoot.getRandomNumber(getWidth());
    y = Greenfoot.getRandomNumber(getHeight());
    addObject(new Fliege(), x, y);

    i++;
}

i = 0;
```

```
15 while (i < 3) {
16     x = Greenfoot.getRandomNumber(getWidth());
17     y = Greenfoot.getRandomNumber(getHeight());
18     addObject(new Storch(), x, y);
19
20     i++;
21 }
```

2.8 Fliegen zufälliger Anzahl

Auch hier habe ich das Beispiel verwirklicht.

```
1  while (i < Greenfoot.getRandomNumber(5) + 4) {
2      //Fliegen
3      ...
4  }
5
6  ...
7
8  while (i < Greenfoot.getRandomNumber(4) + 2) {
9      //Stoerche
10     ...
11 }
```

2.9 Transparenz

```
1  public void act() {
2      if (kraft <= 0) {
3          getImage().setTransparency(128);
4          Greenfoot.stop();
5      }
6
7      ...
8  }
```

Um etwas halbtransparent darzustellen, habe ich als Transparenz-Wert gerundet die Hälfte des maximalen Wertes (welcher in der Dokumentation zu finden ist) benutzt, nämlich 128.

2.10 Essen

```
1 public void essen() {
2     if (isTouching(Fliege.class)) {
3         kraft += 10;
4         removeTouching(Fliege.class);
5     }
6 }
```

2.11 Treffe Storch

```
1 public void treffeStorch() {
2     if (isTouching(Storch.class)) {
3         Greenfoot.stop();
4         getImage().setTransparency(128);
5     }
6 }
```

2.12 for-Schleifen

Der in **2.7** und **2.8** aufgezeigte Code kann durch folgenden Code mit for-Schleifen ersetzt werden:

```
1  for (int i = 0; i < Greenfoot.getRandomNumber(5) + 4; i++) {
2      x = Greenfoot.getRandomNumber(getWidth());
3      y = Greenfoot.getRandomNumber(getHeight());
4      addObject(new Fliege(), x, y);
5  }
6
7  for (int i = 0; i < Greenfoot.getRandomNumber(4) + 2; i++) {
8      x = Greenfoot.getRandomNumber(getWidth());
9      y = Greenfoot.getRandomNumber(getHeight());
10     addObject(new Storch(), x, y);
11 }
```

3 Fachkonzept - Klassendokumentation

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

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

4 Fachkonzept - Klassen und primitive Datentypen

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

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

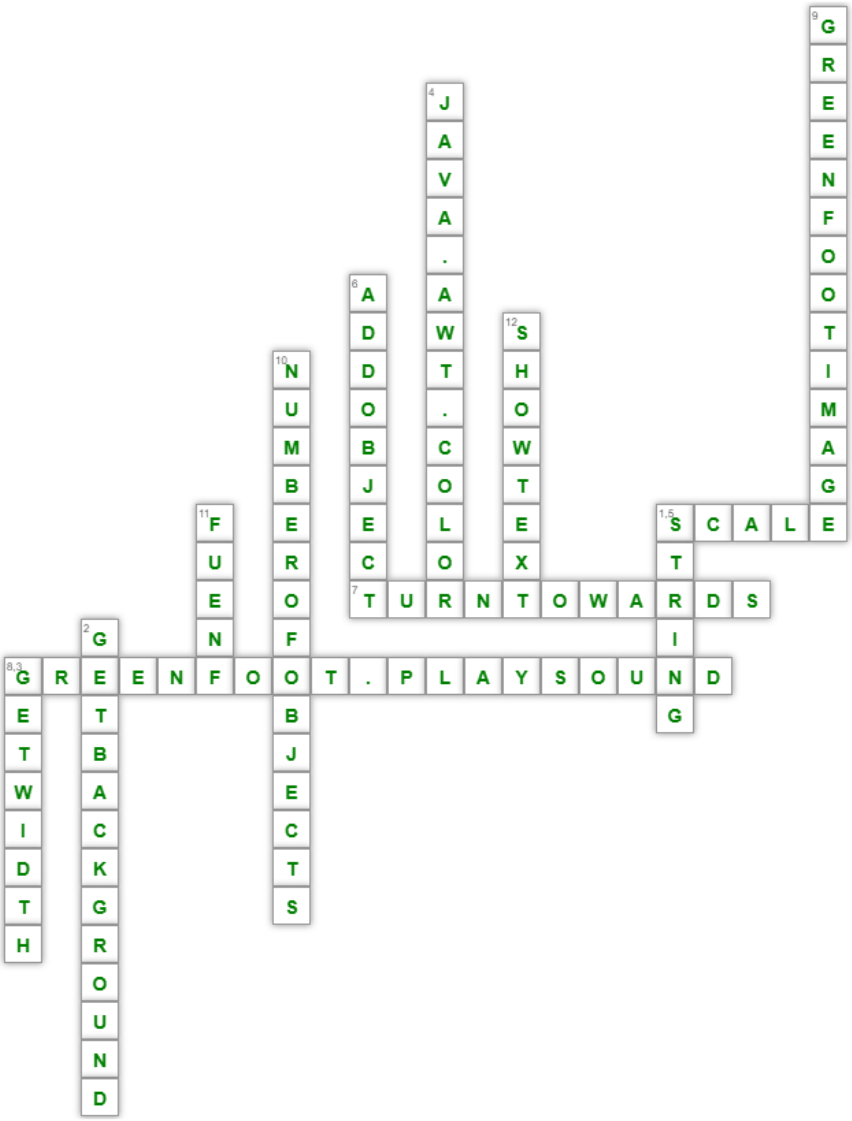
5 Fachkonzept - Wiederholungen

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

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

6 Übungen

6.1 Greenfoot-Klassen



6.2 Frog erweitern

1. Zuerst muss die Datei im Projektordner in den Ordner „sounds“ abgelegt werden.

```
public FrogWorld() {
    ...

    new GreenfootSound("frosch.mp3").play();
}
```

2. Mit den Parametern muss man ein bisschen herumprobieren, irgendwann kommt man auf angemesse Ergebnisse.

```
public void act() {
    if (kraft <= 0) {
        ...
        getImage().scale(35, 35);
        ...
    }

public void treffeStorch() {
    if (isTouching(Storch.class)) {
        ...
        getImage().scale(35, 35);
    }
}
```

3. Es soll also nur bei Berührung mit einer Fliege und nur wenn sie nicht halbtransparent ist die Kraft erhöht werden. Logischerweise wird bei Berührung die Fliege halbtransparent dargestellt (nicht verwirren lassen von den if-Bedingungen, sie sind zu lang für eine Zeile, deshalb wurden Zeilensprünge benutzt; beim Programmieren einfach Zeilensprünge wegdenken).

```
1 public void essen() {
2     if (isTouching(Fliege.class) && getOneIntersectingObject(Fliege.
          class).getImage().getTransparency() != 128) {
3         kraft += 10;
4         getOneIntersectingObject(Fliege.class).getImage().
              setTransparency(128);
5     }
6 }
```

4. Ich habe mich für die Variante entschieden, in der man den Strings des ersten Bildes speichert. Diesen rufe ich jeden `act`-Schritt neu aus, um die vorherige Zahl auf dem Bild nicht einfach mit der neuen zu „überdrucken“.

Außerdem muss man in der if-Schleife am Ende ein `return` ergänzen (so bricht man die Methode vorzeitig ab), damit die Transparenz und das skalierte Bild nicht am Ende wieder durch das Anfangsbild ersetzt werden und eine `-1` statt der erwünschten 0 erscheint (Falls du es nicht verstehst, teste einfach was passiert, wenn man es weglässt).

```
1  String img;
2
3  public void act() {
4      if (kraft <= 0) {
5          ...
6          return;
7      }
8
9      ...
10
11     kraft--;
12
13     setImage(img);
14     getImage().drawString(kraft + "", 5, 10);
15 }
```

Die beiden letzten Parameter der `drawString`-Methode beschreiben die Position der Schrift auf dem Bild, hier habe ich einfach ein bisschen herumprobiert, bis sie ganz gut gepasst hat (und zwar oben links über den Fröschen).

5. Diese Aufgabe ist schon etwas komplizierter. Angefangen habe ich mit dem Implementieren eines neuen Actors zum mehrfachen Zugreifen auf die getroffene Fliege, da ich auch noch darauf zugreifen können muss, wenn der Frosch sie nicht mehr berührt.

Wenn ein Frosch nun also eine noch lebendige Fliege berührt, wird dem neuen Actor diese Fliege zugewiesen, zudem wird sie halbtransparent dargestellt und die `10` erscheint (diesmal oben rechts).

Um nun nach einer bestimmten Anzahl von `act`-Schritten die `10` wieder zu entfernen, brauchen wir eine Zählvariable, welche in jedem `act`-Schritt erhöht wird (bei mir heißt sie `timer`).

Der nächste Schritt ist dann zu erkennen, ob der Timer eine bestimmte Zahl erreicht hat. Da `timer` mit dem Wert `0` initialisiert wird und jeder positive Wert durch das durchgehende Ausführen der `act`-Methode ohne Berührung einer Fliege erreicht werden kann und eine Bedingung, welche z.B. auf den Wert `5` wartet, in dem Fall dann sinnlos wäre, setze ich `timer` beim Essen einer Fliege auf `-6`, damit ich bei Wert `-1` mithilfe einer einfachen if-Schleife die `10` wieder entfernen kann. `-1` kann nämlich nur erreicht werden, wenn `timer` vorher auf unter `0` gesetzt wird (bei uns in der `essen`-Methode).

Dadurch bleibt die `10` 5 `act`-Schritte lang stehen, bis sie dann wieder entfernt wird. Jedoch muss man auch beachten, dass währenddessen auch eine andere Fliege berührt werden kann. Dafür überprüfe ich am Anfang der `essen`-Methode noch, ob dem Actor `tote_fliege` schon eine Fliege zugewiesen ist, wo gerade `10` steht. Wenn der Actor also nicht den Wert `null` hat, wird der Wartevorgang vorzeitig abgebrochen, bei der älteren toten Fliege wird die `10` entfernt und der Vorgang beginnt mit der neuen toten Fliege.

```
1  Actor tote_fliege;
2  int timer;
3
4  public void act() {
5      ...
6
7      timer++;
8  }
9
10 public void essen() {
11     if (isTouching(Fliege.class) && getOneIntersectingObject(Fliege.
           class).getImage().getTransparency() != 128) {
12         if (tote_fliege != null) {
13             tote_fliege.setImage("fly.png");
14         }
15
16         tote_fliege = getOneIntersectingObject(Fliege.class);
17         tote_fliege.getImage().setTransparency(128);
18         tote_fliege.getImage().drawString("10", 30, 10);
19
20         kraft += 10;
21         timer = -6;
22     }
23
24     if (timer == -1) {
25         tote_fliege.setImage("fly.png");
26         tote_fliege.getImage().setTransparency(128);
27         tote_fliege = null;
28     }
29 }
```

6. Bei dieser Aufgabe habe ich folgenden Lösungsansatz:

Ich erstelle durch die schon bestehenden for-Schleifen die Fliegen und Störche, lasse jedoch jedes dieser Objekte aus ihrer eigenen Klasse heraus überprüfen, ob sie noch ein anderes

Objekt berühren (mithilfe einer Methode, die ich in `World` aufrufe).

In der Fliegen-for-Schleife erstelle ich also eine neue Fliege, welche ich an eine zufällige Position setze und dann die Methode `check` an der Fliege aufrufe. Diese Methode setzt die Fliege solange an eine zufällige Position, bis sie keine andere mehr berührt (Störche müssen noch nicht beachtet werden, es gibt nämlich noch keine).

Anschließend passiert dasselbe mit den Störchen, nur mit dem Unterschied, dass die while-Schleife solange ausgeführt wird, bis weder ein Storch noch eine Fliege berührt wird.

```
1  public FrogWorld() {
2      ...
3
4      for (int i = 0; i < Greenfoot.getRandomNumber(5) + 4; i++) {
5          Fliege fliege = new Fliege();
6          int x = Greenfoot.getRandomNumber(getWidth());
7          int y = Greenfoot.getRandomNumber(getHeight());
8          addObject(fliege, x, y);
9          fliege.check();
10     }
11
12     for (int i = 0; i < Greenfoot.getRandomNumber(4) + 2; i++) {
13         Storch storch = new Storch();
14         int x = Greenfoot.getRandomNumber(getWidth());
15         int y = Greenfoot.getRandomNumber(getHeight());
16         addObject(storch, x, y);
17         storch.check();
18     }
19
20     ...
21 }
```

```
1 //Fliege
2 void check() {
3     while (isTouching(Fliege.class)) {
4         int x = Greenfoot.getRandomNumber(getWorld().getWidth());
5         int y = Greenfoot.getRandomNumber(getWorld().getHeight());
6
7         setLocation(x, y);
8     }
9 }
```

```
1 //Storch
2 void check() {
3     while (isTouching(Storch.class) || isTouching(Fliege.class)) {
4         int x = Greenfoot.getRandomNumber(getWorld().getWidth());
5         int y = Greenfoot.getRandomNumber(getWorld().getHeight());
6
7         setLocation(x, y);
8     }
9 }
```

6.3 Anzeige auf der Welt

Nur die zweite Variante funktioniert, da nur ein Objekt aus der Unterklasse zum Objekt aus der oberen Klasse konvertiert werden, andersherum funktioniert es nicht („Ein Schüler ist ein Mensch, aber ein Mensch ist nicht unbedingt ein Schüler.“)

6.4 Java Klassenbibliothek

Der hier gezeigte Code muss in keine Klasse geschrieben werden, sondern kann ins Codepad (die Direkteingabe) eingegeben werden. Deshalb muss man sich nochmal ins Gedächtnis rufen, dass bei Rückgabe eines Wertes kein Semikolon gesetzt wird, ansonsten wird kein Wert ausgegeben.

Teste es am besten auch mal selber.

- Zuerst muss man in der Dokumentation in der String-Klasse die richtige Methode finden. Danach erstellt man irgendeinen String und ruft an diesem die Methode `length` ohne Parameter auf, als Rückgabe bekommt man die Länge des Strings.

```
String s = "Hallo Welt!";
s.length()
    11   (int)
```

- Auch hier muss man einfach nur die richtige Methode finden und gucken, was in der Signatur bezüglich des Parameters steht. Wir rufen die Methode `startsWith` mit dem Parameter `"Hallo"` am String auf und erhalten `true` oder `false`.

```
s.startsWith("Hallo")
    true   (boolean)
```

- String wird automatisch importiert, Random hingegen muss man manuell importieren. Anschließend erstellen wir ein neues Random-Objekt und rufen die Methode `nextInt` mit der Anzahl an Zahlen als Parameter an.


```
import java.util.Random;
new Random().nextInt(10)
    1   (int)
```

➤ Dies funktioniert genauso wie gerade, nur mit der Methode `nextBoolean` und ohne Parameter.

```
import java.util.Random;
new Random().nextBoolean()
    true   (boolean)
```

➤ Der sogenannte „seed“ ist ein Startwert, von dem aus der Zufallsalgorithmus startet. Demnach sind die Zufallszahlen, welche wir benutzen, nicht wirklich zufällig. Wenn man immer mit demselben Seed starten würde, würden auch immer dieselben Zahlen herauskommen.

Daher versucht man, einen Seed möglichst kompliziert und lang zu machen, jedoch soll es auch nicht zu viel Rechenleistung verbrauchen.

Machen wir einen kleinen Test und erstellen zweimal mit demselben Seed Zufallszahlen und prüfen, ob dasselbe herauskommt.

```
import java.util.Random;
Random r1 = new Random();
Random r2 = new Random();
r1.nextInt(1000000)
    305014   (int)
r2.nextInt(1000000)
    619720   (int)
```

```
r1.setSeed(10);
r2.setSeed(10);
r1.nextInt(1000000)
    587113   (int)
r2.nextInt(1000000)
    587113   (int)
```

Wir sehen, dass zwei Random-Objekte mit automatischen Seeds verschiedene „Zufallszahlen“ erzeugen. Wenn man dann jedoch beiden Objekten denselben Seed zuweist, erzeugen beide Objekte unabhängig voneinander dieselbe „Zufallszahl“. Und nein, das ist kein Zufall :), du kannst es ja auch mal ausprobieren.

Aus diesem Grund kann der Seed bei Java als niedrigsten Wert $-9.223.372.036.854.775.808$ und $9.223.372.036.854.775.807$ als höchsten Wert annehmen. Wie er genau errechnet wird, ist natürlich nicht öffentlich bekannt, sonst wäre es nicht mehr sicher.

6.5 Schleifen

1.

```
1  void whileSchleife() {
2      int zahl = 20;
3
4      while (zahl != 0) {
5          System.out.println(zahl);
6          zahl--;
7      }
8  }
9
10 void forSchleife() {
11     for (int i = 20; i != 0; i--)   {
12         System.out.println(i);
13     }
14 }
```

2. Ich gehe davon aus, dass gemeint ist, dass alle geraden Zahlen über 0 ausgegeben werden sollen, denn sonst würde die Methode niemals enden.

```
1  void geradeZahlen_while(int n) {
2      while (n >= 0) {
3          if (n % 2 == 0) {
4              System.out.println(n);
5          }
6
7          n--;
8      }
9  }
10
11
12
13
14
```

```
15 void geradeZahlen_for(int n) {
16     for (n = n; n >= 0; n--) {
17         if (n % 2 == 0) {
18             System.out.println(n);
19         }
20     }
21 }
```

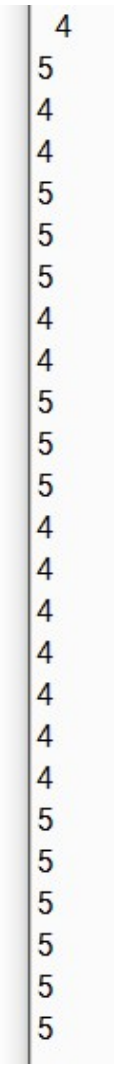
Bei der for-Schleife hätte man auch eine extra Zählvariable ergänzen können, in diesem Fall habe ich aber `n` als Zählvariable benutzt.

3. Ab sofort werde ich das Paket `java.util.Random`, falls `Random` benötigt ist, importieren, damit nicht bei jeder Zufallsgenerierung der „etwas längliche “ Ausdruck benutzt werden muss. Stattdessen kann man ein Objekt erstellen und dies immer wieder verwenden. So auch hier:

```
1  import java.util.Random;
2
3  class Steuerung {
4      int durchschnitt() {
5          Random r = new Random();
6          int summe = 0;
7
8          for (int i = 0; i < 1000; i++) {
9              summe += r.nextInt(11);
10         }
11
12         return summe / 1000;
13     }
14
15     void durchschnitte() {
16         for (int i = 0; i < 25; i++) {
17             System.out.println(durchschnitt());
18         }
19     }
20
```

```
21      ...
22  }
```

Die zweite Methode zeigt 25 Ergebnisse der ersten Methode, um die Auswertung zu vereinfachen. Die Ausgabe zeigt ein ziemlich eindeutiges Ergebnis:



Anscheinend kommt jedes Mal 4 oder 5 heraus. Dieses Ergebnis beweist erneut, wie „unzufällig“ die Zahlen des Zufallsalgorithmus sind.

4.

```
int quadratzahlen(int n) {
    int summe = 0;

    for (int i = 1; Math.pow(i, 2) < n; i++) {
        System.out.println(Math.pow(i, 2));
        summe += Math.pow(i, 2);
    }

    return summe;
}
```

5. Es wird schwieriger...

Angefangen habe ich damit, herauszufinden, wieviele Würfe man braucht, einmal einen Zweierpasch zu würfeln.

```
void zweierpasch() {
    Random r = new Random();
    int wuerfe = 0;

    int wuerfel1 = 0;
    int wuerfel2 = 0;

    while (wuerfel1 != 2 || wuerfel2 != 2) {
        wuerfel1 = r.nextInt(6) + 1;
        wuerfel2 = r.nextInt(6) + 1;

        wuerfe++;
    }

    System.out.println("Wuerfe: " + wuerfe);
}
```

Als nächstes wollen wir sicherstellen, dass der Wert aussagekräftig ist, also lassen wir den Prozess z.B. 1000 durchlaufen.

```
1  void zweierpasch() {
2      Random r = new Random();
3      int wuerfe = 0;
4
5      for (int i = 0; i < 1000; i++) {
6          int wuerfel1 = 0;
7          int wuerfel2 = 0;
8
9          while (wuerfel1 != 2 || wuerfel2 != 2) {
10             wuerfel1 = r.nextInt(6) + 1;
11             wuerfel2 = r.nextInt(6) + 1;
12
13             wuerfe++;
14         }
15     }
16
17     System.out.println("Wuerfe im Durchschnitt: " + wuerfe / 1000);
18 }
```

Der Wert bewegt sich immer circa um 36. Wenn wir diesen Wert mit dem Wert einer Einzelrunde vergleicht, fällt auf, dass die Einzelwerte starke Unterschiede aufweisen. Je öfter man es jedoch durchführt, desto genauer kann man eine Tendenz erkennen.

Um zu überprüfen, ob das Programm richtig funktioniert, könnte man die 36 mathematisch als richtiges Ergebnis beweisen:

Die Warscheinlichkeit eine 2 auf einem Würfel zu würfeln, beträgt logischerweise $\frac{1}{6}$. Also liegt die Warscheinlichkeit mit zwei Würfeln eine 2 zu würfeln bei $\frac{1}{6} * \frac{1}{6} = \frac{1}{36}$.

Dieser Wert stimmt mit unserem überein. Mit einem von 36 Würfen wirft man im Durchschnitt einen Zweierpasch.

6. Ich gebe zwei Beispiele für Endlosschleifen, denn es gibt viele Beispiele. Viel Spaß beim Kreieren weiterer Endlosschleifen.

```
1 while (true) {
2     ...
3 }
```

```
1 for (int i = 0; i >= 0; i++) {
2     testMethode();
3 }
```