

Aufgabe 3: Dreiecke zählen

Team: „Was ist JAVA?“

Team-ID: 00107

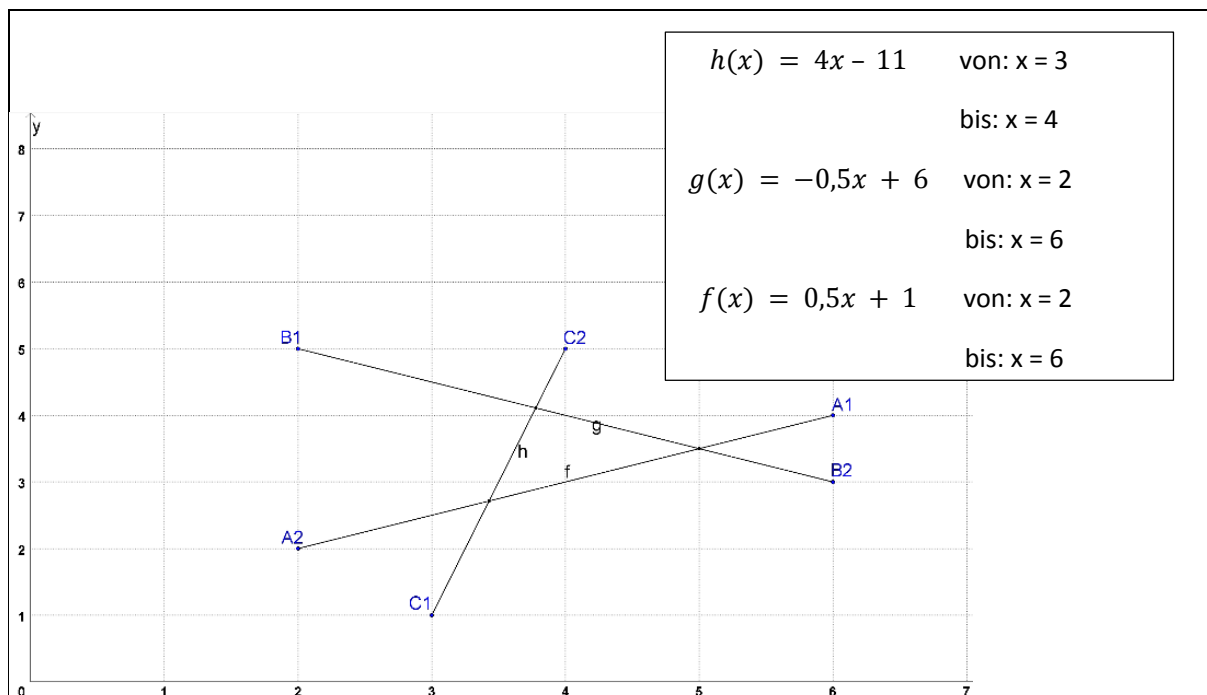
24. November 2017

Inhaltsverzeichnis

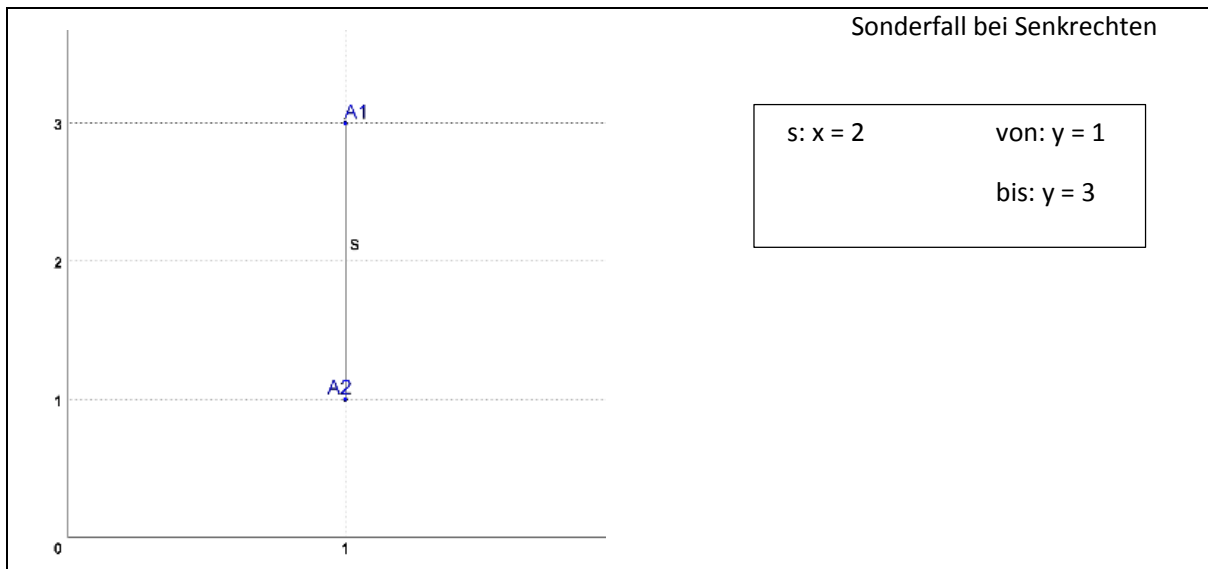
1.	Lösungsidee	S. 1
2.	Umsetzung	S. 3
3.	Beispiele	S. 4
4.	Quellcode	S. 4
5.	Lösung der Aufgabe	S. 8

Lösungsidee

Unsere Idee zur Lösung der Aufgabe, war es die Strecken zwischen den zwei Punkten, durch Funktionen der Form $f(x) = mx + b$, mit der Angabe von x_1 bis x_2 , darzustellen. Da Funktionen für jeden x -Wert aber nur ein y -Wert haben dürfen, ist es mit dieser Methode nicht möglich senkrechte Strecken darzustellen. Aus diesem Grund musste unser Programm die Punkte auch in senkrechte Strecken (der Form $s: x$; von y_1 bis y_2) umwandeln können:



Aufgabe 3: Dreiecke zählen



Um nun herausfinden zu können, ob und wie viele Dreiecke vorhanden sind müssen die Schnittpunkte errechnet werden. Um herausfinden zu können welche Strecken/Senkrechten sich miteinander schneiden, müssen beide Funktionen gleichgesetzt werden:

$$f(x) \stackrel{!}{=} g(x) \rightarrow m_1 * x + b_1 = m_2 * x + b_2 \rightarrow x = \frac{b_1 - b_2}{m_2 - m_1}$$

Man erhält die x-Koordinate des Schnittpunktes. Wenn eine der Funktionen jedoch eine Senkrechte ist muss anders verfahren werden:

$$f(x) \stackrel{!}{=} s \rightarrow x = s$$

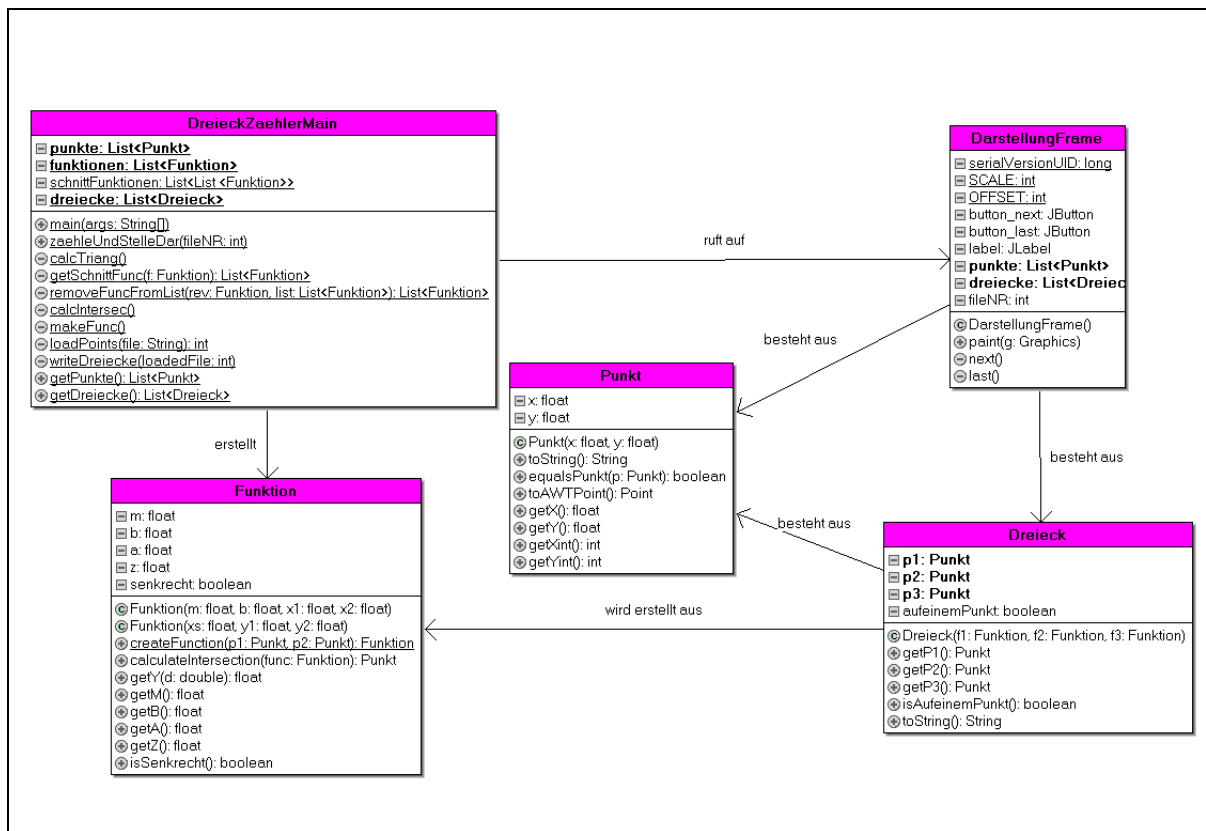
Wenn man nun die x-Stelle des Schnittpunktes gefunden hat, muss nur noch geschaut werden, ob auch der Punkt auf beiden Funktionsabschnitten (Strecken) liegt.

So kann man herausfinden welche Strecken sich mit welchen Strecken schneiden. Daraus kann man schließen, ob 3 Strecken sich so schneiden, dass ein Dreieck entsteht und somit die Anzahl der Dreiecke bestimmen.

Aufgabe 3: Dreiecke zählen

Umsetzung

Unser objektorientiertes Programm besteht aus insgesamt fünf Klassen.



Einstiegspunkt des Programms ist die DreieckZaehlerMain-Klasse, die neben der main()-Methode nur weitere statische Methoden enthält. Zuerst ruft die DreieckZaehlerMain-Klasse den Konstruktor der DarstellungFrame-Klasse auf. Diese erstellt ein neues Fenster und ruft, je nachdem welche Dreiecks-Datei gewählt ist, die zaehleUndStelleDar()-Methode in der DreieckZaehlerMain-Klasse mit unterschiedlichen Parametern (die entsprechende File Nummer) aufruft.

Die aufgerufene zaehleUndStelleDar()-Methode lädt zuerst die entsprechende Textdatei in eine Liste von Punkten ein. Diese Punkte werden dann in Funktionen umgewandelt und in einer anderen Liste gespeichert. Danach wird zu jeder Funktion eine Liste mit den Schnittpunkten erstellt, indem für jede einzelne Funktion geprüft wird, ob es einen Schnittpunkt mit einer der anderen Funktionen gibt.

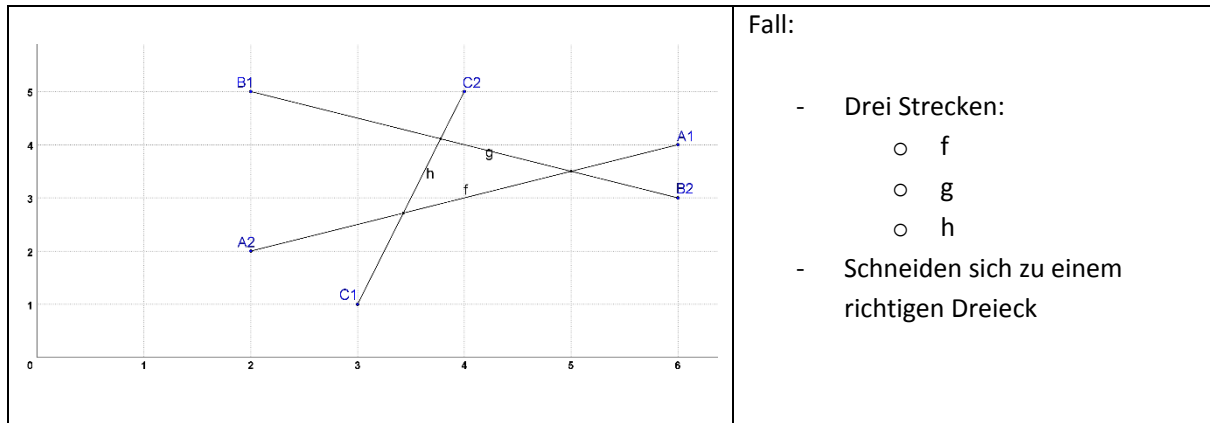
Nun werden alle Listen der Schnittpunkten einzeln durchsucht, für jede Schnittpunkt wird eine neue Liste mit deren Schnittpunkt erstellt und die Ausgangsfunktion entfernt. Die danach erstellte Liste der Schnittpunkten der Schnittpunkten wird nun nach der Ausgangsfunktion abgesucht. Wird diese gefunden, muss noch geprüft werden, ob sich nicht nur drei Funktionen an einem Punkt berühren, so wird das gefundene Dreieck der Liste der Dreiecke hinzugefügt.

Wenn das geschehen ist holt sich das Frame die Daten und stellt sie dar.

Aufgabe 3: Dreiecke zählen

Beispiele

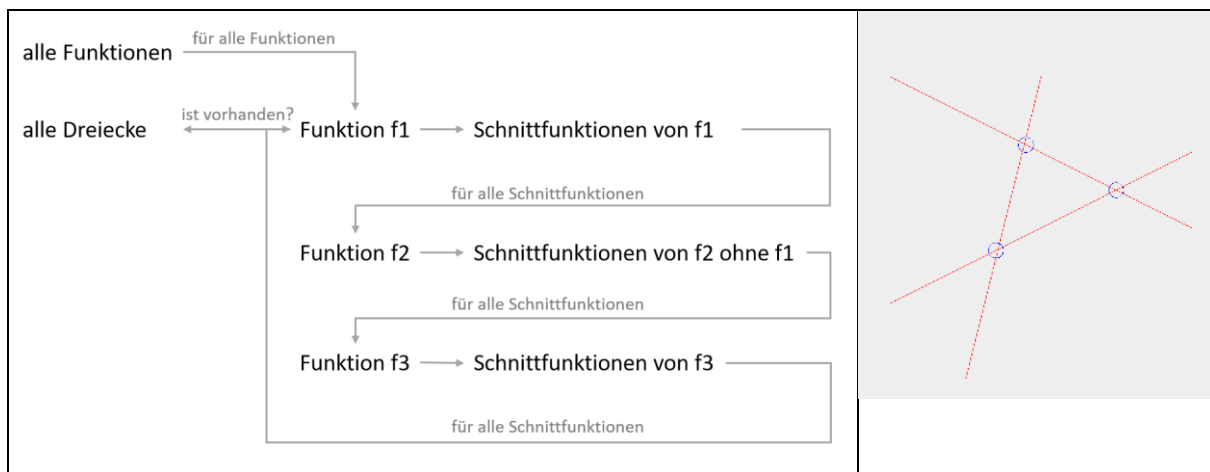
Als Beispiel möchte ich hier eine einfache Situation mit drei sich schneidenden Stecken darstellen:



Zuerst werden aus den Punkten Funktionen erstellt. Die Liste der Funktionen enthält dann 3 Funktionen. Dazu werden die zugehörigen Schnittfunktionen erstellt.



Dann erfolgt das Durchsuchen der Listen.



Man erhält eine Liste von Dreiecken. Die Größe der Liste ist gleichzeitig auch die Anzahl der gefundenen Dreiecke.

Quellcode

```

import java.io.*;
import java.util.*;
public class DreieckZaehlerMain {

    private static List<Punkt> punkte = new ArrayList<Punkt>();

    private static List<Funktion> funktionen = new
        ArrayList<Funktion>();

    private static List<List<Funktion>> schnittFunktionen = new
        ArrayList<List<Funktion>>();
    
```

```

//Speicher für die gelesenen
Punkte
//Speicher für die erzeugten
Funktionen
    
```

```

//Speicher für die
Schnittfunktionen
    
```

Aufgabe 3: Dreiecke zählen

<pre> private static List<Dreieck> dreiecke = new ArrayList<Dreieck>(); public static void main(String[] args) { new DarstellungFrame(); } public static void zaehleUndStelleDar(int fileNR) { punkte.clear(); funktionen.clear(); schnittFunktionen.clear(); dreiecke.clear(); LoadPoints("dreiecke" + fileNR + ".txt"); makeFunc(); calcIntersec(); calcTriang(); System.out.println(dreiecke.size() + " Dreiecke in dreiecke" + fileNR + ".txt"); writeDreiecke(fileNR); } private static void calcTriang() { for (int i = 0; i < funktionen.size(); i++) { Funktion func1 = funktionen.get(i); List<Funktion> list = getSchnittFunc(func1); for (int j = 0; j < list.size(); j++) { Funktion func2 = list.get(j); List<Funktion> list2 = getSchnittFunc(func2); list2 = removeFuncFromList(func1, list2); for (int k = 0; k < list2.size(); k++) { Funktion func3 = list2.get(k); List<Funktion> list3 = getSchnittFunc(func3); for (int l = 0; l < list3.size(); l++) { if (list3.get(l).equals(func1)) { Dreieck dreieck = new Dreieck(func1, func2, func3); if (!dreieck.isAufEinemPunkt()) { dreiecke.add(dreieck); } } } } } } } private static List<Funktion> getSchnittFunc(Funktion f) { for (int i = 0; i < funktionen.size(); i++) { if (funktionen.get(i).equals(f)) { return schnittFunktionen.get(i); } } return null; } private static List<Funktion> removeFuncFromList(Funktion rev, List<Funktion> list) { for (int i = 0; i < list.size(); i++) { if (list.get(i).equals(rev)) { list.remove(i); } } return list; } private static void calcIntersec() { for (Funktion func : funktionen) { List<Funktion> schnitte = new ArrayList<Funktion>(); for (Funktion func2 : funktionen) { if (func.calculateIntersection(func2) != null && !func.equals(func2)) { schnitte.add(func2); } } } } </pre>	<pre> // wird von dem Frame aufgerufen, damit die Listen von Punkten und Dreiecken aktualisiert werden //berechnet mit Hilfe der Listen der Funktionen und Schnittfunktionen die Dreiecke und speichert diese in der Dreiecks Liste. //Hilfsmethode der calcTrinag() Methode: Sucht zu einer bestimmten Funktion die Schnittfunktionsliste heraus. //Hilfsmethode der calcTrinag() Methode: Entfernt eine Funktion aus einer übergebenen Liste. //Erstellt für jede Funktion eine Liste von Schnittfunktionen </pre>
--	--

Aufgabe 3: Dreiecke zählen

<pre> } } schnittFunktionen.add(schnitte); } private static void makeFunc() { for (int i = 0; i < punkte.size() / 2; i++) { funktionen.add(Funktion.createFunction(punkte.get(2 * i), punkte.get(2 * i + 1))); } } private static int loadPoints(String file) { BufferedReader reader = null; try { reader = new BufferedReader(new FileReader(new File(file))); String line; int anzahl = Integer.parseInt(reader.readLine()); while (((line = reader.readLine()) != null) && (!line.isEmpty())) { String[] zahlen; zahlen = line.split(" "); punkte.add(new Punkt(Float.parseFloat(zahlen[0]), Float.parseFloat(zahlen[1]))); punkte.add(new Punkt(Float.parseFloat(zahlen[2]), Float.parseFloat(zahlen[3]))); } reader.close(); return anzahl; } catch (IOException e) { System.err.println("Couldn't load file!"); return 0; } } private static void writeDreiecke(int loadedFile) { BufferedWriter writer = null; try { File out = new File("dreiecke" + loadedFile + "-ergebnis.txt"); out.delete(); writer = new BufferedWriter(new FileWriter(out)); writer.write(dreiecke.size() + " Dreiecke gefunden"); for (Dreieck dreieck : dreiecke) { writer.newLine(); writer.write(dreieck.toString()); } writer.close(); } catch (IOException e) { System.err.println("Error writing File!"); } } public static List<Punkt> getPunkte() { return punkte; } public static List<Dreieck> getDreiecke() { return dreiecke; } } </pre>	<pre> //Erstellt aus der Liste von Punkten eine Liste aus Funktionen //Lädt die Punkte aus der passenden Datei in eine Liste //Scheibt eine Ergebnis zu jeder aufgerufenen Darstellung. //Getter für die Frame-Klasse //Getter für die Frame-Klasse </pre>
<pre> public class Funktion { private float m; private float b; private float a; private float z; } </pre>	<pre> //ist die Steigung bei senkrecht=false //y-Achsenabschnitt bei senkrecht = false , x-Stelle bei senkrecht = true // X Wert "von" bei senkrecht = false Y Wert "von" bei senkrecht = true </pre>

Aufgabe 3: Dreiecke zählen

```

private boolean senkrecht = false;

private Funktion(float m, float b, float x1, float x2) {
    this.m = m;
    this.b = b;
    this.a = Math.min(x1, x2);
    this.z = Math.max(x1, x2);
    this.senkrecht = false;
}

private Funktion(float xs, float y1, float y2) {

    this.m = 0;

    this.b = xs;

    this.a = Math.min(y1, y2);

    this.z = Math.max(y1, y2);
    this.senkrecht = true;
}

public static Funktion createFunction(Punkt p1, Punkt p2) {
    if (p1.getX() == p2.getX()) {
        return new Funktion(p1.getX(), p1.getY(),
            p2.getY());

    } else {
        float m = (p2.getY() - p1.getY()) /
            (p2.getX() - p1.getX());
        float b = p1.getY() - m * p1.getX();
        return new Funktion(m, b, p1.getX(),
            p2.getX());
    }
}

public Punkt calculateIntersection(Funktion func) {

    if (!(this.isSenkrecht()) && (!func.isSenkrecht())) {
        if (func.getM() - this.getM() != 0) {
            float schnittpunktX = (float)
                ((this.getB() - func.getB()) /
                    (func.getM() - this.getM()));
            float schnittpunktY = (float)
                this.getY(schnittpunktX);
            if (schnittpunktX >= this.getA() &&
                schnittpunktX <= this.getZ()
                && schnittpunktX >=
                    func.getA() && schnittpunktX
                    <= func.getZ()) {
                return new Punkt(schnittpunktX,
                    schnittpunktY);
            }
        }
    }
    if (this.isSenkrecht() && !func.isSenkrecht()) {
        if (func.getA() <= this.getB() &&
            func.getZ() >= this.getB()) {
            float schnittpunktX = this.getB();
            float schnittpunktY =
                func.getY(schnittpunktX);
            if (func.getY(this.getB()) >=
                this.getA() &&
                func.getY(this.getB()) <=
                this.getZ()) {
                return new Punkt(schnittpunktX,
                    schnittpunktY);
            }
        }
    }
    } else if (!this.isSenkrecht() && func.isSenkrecht()) {
        if (this.getA() <= func.getB() &&
            this.getZ() >= func.getB()) {

```

```

// X Wert "bis" bei senkrecht
= false Y Wert "bis" bei
senkrecht = true

```

```

//Konstruktor-Format: m*x + b
Intervall: a bis z

```

```

//Konstruktor-Format: at (X);
von y1 bis y2
//! wird nicht gebraucht
//! dient als X Stellen
Speicher
//! dient als y Stellen
Speicher
//! dient als y Stellen
Speicher

```

```

// hier gilt: m = unendliche
Steigung; b = X-Wert der
Senkrechten und a und z geben
die Y-Begrenzung an
// Normalfall für nicht
senkrechte Funktionen

```

```

// berechnet den Schnittpunkt
zweier Functions

```

```

// wenn zwei normale
Funktionen

```

```

// liegt der schnittpunkt auf
beiden Func

```

```

// wenn nur diese Func
senkrecht

```

Aufgabe 3: Dreiecke zählen

<pre>float schnittpunktX = func.getB(); float schnittpunktY = this.getY(schnittpunktX); if (this.getY(func.getB()) >= func.getA() && this.getY(func.getB()) <= func.getZ()) { return new Punkt(schnittpunktX, schnittpunktY); } } return null; } public float getY(double d) { return (float) (this.m * d + this.b); } public float getM() {return m;} public float getB() {return b;} public float getA() {return a;} public float getZ() {return z;} public boolean isSenkrecht() {return senkrecht;} }</pre>	<pre>// wenn nur andere Func senkrecht // sonst keine Überschneidung //gibt den Y-Wert einer Funktion an einer gewissen Stelle aus // setters und getters</pre>
--	---

Lösung der Aufgabe

```
6 Dreiecke in dreiecke1.txt
0 Dreiecke in dreiecke2.txt
3 Dreiecke in dreiecke3.txt
5 Dreiecke in dreiecke4.txt
1 Dreiecke in dreiecke5.txt
12 Dreiecke in dreiecke6.txt
```