

## Übungsblatt 2

### 3 Klasse Stdin

Damit nicht jede Klasse ihr eigenes `Scanner` zum Lesen von `System.in` macht, und da diese Funktionalität eigentlich keiner geometrischen Klasse gehören sollte, wird eine Klasse speziell für dieses Objekt gemacht. Wegen Einfachheit wird diese Klasse dem Paket `geometrie` gehören.

```
package geometrie;

import java.util.*;

public class Stdin {
    public static Scanner sc = new Scanner(System.in);
}
```

### 4 Klasse Punkt

Im Paket `geometry` sollen Sie eine neue Klasse `Punkt` schreiben. Diese Klasse enthält die folgende Elementen:

1. Zwei `protected double` Variablen `x` und `y`.
2. Default-Konstruktor, der diese zwei Werte auf 0 einstellt.
3. Konstruktor `Punkt(double, double)`, der zwei reellen Werte bekommt, auf die `x` und `y` einzustellen sind.
4. So genannten Copy-Konstruktor `Punkt(Punkt p)`, der `x` und `y` gleich wie bei dem bekommenen Punkt `p` einstellt.
5. Methode `public void eingabe()`, die diese zwei Werte von der Tastatur liest.
6. Methode `public String toString()`, die diese zwei Werte als String im Format „(x,y)“ ohne LF<sup>7</sup> am Ende zurückgibt.
7. Methode `public double distanz(Punkt)`, die Distanz eines Punktes vom gegebenen Punkt zurückliefert.
8. Methoden `public double getX()` und `public double getY()`, die die Werte von `x` bzw. `y` zurückliefern.
9. Machen Sie die `main`-Methode, indem Sie vier Objekten der Klasse `Punkt` erzeugen. Erste drei sollen als Demonstration der drei Konstruktoren erzeugt werden. Der vierte Punkt soll mittels Default-Konstruktors erzeugt und danach mit der `eingabe`-Methode initialisiert werden. Danach sollen die Koordinaten aller vier Punkte auf `stdout` ausgedrückt werden.

---

<sup>7</sup>LF = Line Feed, neue Zeile.

Da all dies im Prinzip für Sie wieder neu ist, wird für diese Klasse eine Musterlösung gegeben. Weiterhin werden meistens nur Neuigkeiten erklärt. Von Ihnen wird es erwartet, den Code selbst zu schreiben und die entstandene Probleme mit dem Tutor per E-Mail oder in Übungen zu besprechen.

```
package geometrie;

public class Punkt {
    protected double x, y;

    public Punkt() {
        x = 0;
        y = 0;
    }

    public Punkt(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public Punkt(Punkt p) {
        x = p.x;
        y = p.y;
    }

    public void eingabe() {
        x = Stdin.sc.nextDouble();
        y = Stdin.sc.nextDouble();
    }

    public double distanz(Punkt p) {
        return Math.sqrt(Math.pow(p.x - x,2) + Math.pow(p.y - y,2));
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public String toString() {
        return "(" + x + "," + y + ")";
    }

    public static void main(String[] args) {
        Punkt a = new Punkt(),
            b = new Punkt(1, 2),
            c = new Punkt(b),
            d = new Punkt();
        System.out.println("Geben Sie Koordinaten eines Punktes ein (eine je Zeile):");
        d.eingabe();
        System.out.println("a: " + a + "\nb: " + b + "\nc: " + c + "\nd: " + d);
    }
}
```

Autor: M. A.

## 4.1 toString() Methode

Falls man ein Objekt mit einem String verknüpft, wird diese Methode automatisch gerufen, und ihr Resultat wird mit dem String verknüpft anstatt des Objekts. Diese Methode kann (nach dem `override`<sup>8</sup>) als Resultat von uns formatierte Daten über einem Objekt zurückliefern. Das ist sehr bequem beim Druck von diesen Daten durch `println` oder `printf` — wie im Beispiel oben.

## 4.2 Klasse GMath

Der Klasse `Math` fehlen die getrennte Methoden für Berechnung von zweiten und dritten Grad eines Werts. Somit muss man entweder `Math.pow` benutzen oder diese Methoden selbst schreiben. Etwas anderes wäre nicht angemessen, speziell wenn man braucht, größere Ausdrücke zu quadrieren oder zu kubieren. Hierbei kann man eine neue Klasse `GMath` in demselben Paket machen und da zum Anfang eine Methode `public static double quad(double val)` schreiben:

```
package geometrie;

public class GMath {
    public static double qua(double val) {
        return val * val;
    }
}
```

Merken Sie sich, dass diese Methode statisch ist. Das heißt, dass man sie rufen kann, ohne ein Objekt der Klasse `GMath` zu erzeugen. Sie ist auch `public` d.h. jede äußere Klasse kann sie rufen. Nun könnte die Methode `distanz` aus der Klasse `Punkt` so lauten:

```
return Math.sqrt(GMath.qua(p.x - x) + GMath.qua(p.y - y));
```

Dieser Klasse können Sie auch eine Konstante `EPSILON` (sagen wir  $10^{-4}$ ) einfügen, die zum Vergleich der annähernd gleichen Werte dienen wird. Nun können Sie auch die Methode `equals(double, double)` schreiben, um die reelle Werte in Zukunft bequemer zu vergleichen. Eine weitere Methode, die vom Nutz sein wird, ist `Signum`. Die beiden sollen auch statisch sein:

```
public static final double EPSILON = 1e-4;

public static boolean equals(double a, double b) {
    return Math.abs(a - b) < EPSILON;
}

public static int sgn(double x) {
    return equals(x, 0) ? 0 : x < 0 ? -1 : 1;
}
```

---

<sup>8</sup>überschreiben

## 5 Strecke

Schreiben Sie nun allein die Klasse Strecke. Sie enthält folgendes:

1. Zwei Objekte der Klasse Punkt, a und b, als `protected` definiert.
2. Default-Konstruktor, der diese Strecke auf  $(0,0) - (1,0)$  initialisiert.
3. Konstruktor, der zwei Punkte als Argumenten bekommt und a und b entsprechend initialisiert.
4. Copy-Konstruktor.
5. Methode `public double laenge()` zur Berechnung der Länge der Strecke.
6. Methode `public void eingabe()` die zur Eingabe von Punkten a und b dient. Falls diese zwei Punkte auf dem Abstand größer als EPSILON liegen, dürfen a und b ihre Werte behalten. Sonst muss diese Eingabe wiederholt werden. Realisieren Sie dies durch eine `do-while` Schleife.
7. Methode `public String toString()`, die eine Strecke im Format „ $(x_1, y_1) - (x_2, y_2)$ “ repräsentiert.
8. Die `main`-Methode, die alle vier Weise der Objekterzeugung und -Initialisierung demonstriert und die Daten über den Strecken drückt.

Hierbei könnte man Hilfe bei der Methode Eingabe brauchen. Schreiben wir zuerst eine Basis für diese Methode:

```
public void eingabe() {
    do {
        a.eingabe();
        b.eingabe();
    } while(a.distanz(b) <= GMath.EPSILON);
}
```

Diese Methode kann zur Eingabe der zwei Punkte dienen, sodass sie eine Strecke der Länge größer EPSILON definieren. Bei der Wiederholung der Eingabe wird aber immer noch keine Nachricht über dem möglichen Fehler ausgedrückt und somit könnte der Benutzer sich wundern wieso kann er dieser Eingabe nicht raus. Um diese Nachricht zu drucken, kann man eine zusätzliche `boolean` Variable benutzen, die **nur beim ersten Eintritt zur Schleife** `false` sein wird und sonst `true`. Dann kann man sie zum Druck der Fehlermeldung benutzen.

```
public void eingabe() {
    boolean passedOnce = false;
    do {
        if(passedOnce) { // Fehlermeldung drucken
            System.out.println(
                "! Fehler: die Punkte sind zu nah zueinander. "
                + "Bitte wiederholen Sie die Eingabe."
            );
        }
        a.eingabe();
        b.eingabe();
        passedOnce = true;
    } while(a.distanz(b) <= GMath.EPSILON);
}
```