



Übungsblatt 5

- Kontrollstrukturen I -

Aufgabe 1: Java Code Conventions

Code Conventions sind (freiwillige) Konventionen zum Schönschreiben von (Java-)Programmen, die man beim Programmieren anwenden sollte. Teilweise werden diese Konventionen automatisch von eclipse (oder anderen IDEs) angewandt (zum Beispiel das Einrücken von Zeilen). Sie finden die Java Code Conventions unter: <https://www.oracle.com/java/technologies/javase/codeconventions-contents.html>

Lesen und verstehen Sie die Code Conventions. Notieren Sie in einer Zusammenfassung (z.B. tabellarisch) die aus ihrer Sicht wesentlichen Punkte dieser Konventionen. An einigen Stellen kommen Java-Konstrukte vor, die Sie noch nicht kennen. Ignorieren Sie diese Stellen im Moment.

Achtung

Ab sofort sind diese Code Conventions verbindlich für **alle** Java Programme, die Sie schreiben.

Info

Der Zeitbedarf für diese Aufgabe beträgt ca. 2-3 Stunden!

Aufgabe 2: Bitquersumme

Geben Sie ein Java-Programm in einer Klasse BitQuersumme an, das die Quersumme über die niederwertigsten 16 Bits einer int-Zahl berechnet und in einer eigenen Zeile auf dem Bildschirm ausgibt. Die Zahl wird über die Kommandozeile eingegeben. Nutzen Sie eine geeignete Schleifenform in Ihrem Programm.

Beispiel

```
> java BitQuersumme 3
Ausgabe:
2
weil 310 = 0...0112 und davon ist die Bitquersumme 2.
```

Aufgabe 3: Leibniz-Reihe

Die Reihe $\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$ konvergiert gegen $\frac{\pi}{4}$. Schreiben Sie drei Programme, die daraus eine Näherung für π bestimmen:

1. Das erste Programm soll solange Reihenglieder berechnen und aufaddieren, bis insgesamt $n = 100000$ Reihenglieder aufaddiert wurden.
2. Das zweite Programm soll solange Reihenglieder berechnen und aufaddieren, bis der Betrag des neuen Summanden $\leq \varepsilon = 0.0000001$ ist.
3. Das dritte Programm soll solange Reihenglieder berechnen und aufaddieren, bis der berechnete Wert vom Wert `Math.PI` weniger als 0.0000001 abweicht.

Info

- Der Ausdruck $(-1)^i$ bewirkt nur einen Wechsel des Vorzeichens in jeder Iteration. Dazu brauchen (sollen!) Sie keine Potenz berechnen, sondern es genügt, den Faktor 1 in jeder Iteration mit -1 zu multiplizieren.
- Beachten Sie, dass die Formel oben gegen $\frac{\pi}{4}$ konvergiert, Sie aber π annäherungsweise berechnen sollen.

Mögliche Lösung:

```
public class piReiheA {  
    public static void main(String[] args) {  
        int i; //Schleifenzaele  
        int n; //Anzahl Reihenglieder  
        double vorz; // Vorzeichen  
        double sum; // Summand  
        double pi = 0; // Approximation von Pi  
  
        // Initialisierungen  
        n = 100000;  
        vorz = 1.0;  
        pi = 0;  
  
        // nun wird die Reihe bis zum n-ten Glied berechnet  
        for (i = 0; i < n; i++) {  
            // Summand durch laufende Folge: 1/1, 1/3, 1/5, usw.  
            sum = 1.0 / (2.0 * i + 1);  
            // Aufaddieren  
            pi += vorz * sum;  
            // Vorzeichen wechseln  
            vorz *= -1;  
        }  
  
        // wir wollen aber pi, nicht pi/4  
        pi *= 4.0;  
  
        System.out.println("Naeherung nach " + n + " Reihengliedern: " + pi);  
        System.out.println("Absoluter Abstand von " + Math.PI + ": " + Math.abs(Math.PI -  
pi));  
    }  
}  
  
public class piReiheB {  
    public static void main(String[] args) {  
        int i;  
        double smin; // Grenzwert für Summanden  
        double vorz; // Vorzeichen  
        double sum; // aktuelle Summe  
        double pi = 0; // Approximation von Pi  
  
        // Initialisierungen  
        smin = 1.0e-7;  
        vorz = 1.0;  
        pi = 0;  
  
        // noch keine Reihenglieder bestimmt  
        i = 0;  
  
        // nun wird die Reihe berechnet, solange neuer Summand groß genug  
        do {
```

```

    // Summand durch laufende Folge: 1/1, 1/3, 1/5, usw.
    sum = 1.0 / (2.0 * i + 1);
    // Aufaddieren
    pi += vorz * sum;
    // Vorzeichen wechseln
    vorz *= -1;
    // Zähler um eins erhöhen
    i++;
} while (sum >= smin);

// wir wollen aber pi, nicht pi/4
pi *= 4.0;

System.out.println("Näherung nach " + i + " Reihengliedern: " + pi);
System.out.println("Absoluter Abstand von " + Math.PI + ": " + Math.abs(Math.PI - pi));
}

public class piReiheC {
    public static void main(String[] args) {
        int i;
        double eps; // Genauigkeit, die wir erreichen wollen
        double vorz; // Vorzeichen
        double sum; // aktuelle Summe
        double pi = 0; // Approximation von Pi

        // Initialisierungen
        eps = 1.0e-8;
        vorz = 1.0;
        pi = 0;

        // noch keine Reihenglieder bestimmt
        i = 0;

        // nun wird die Reihe berechnet, bis die Näherung gut genug ist
        while (Math.abs(Math.PI - 4 * pi) >= eps) {
            // Summand durch laufende Folge: 1/1, 1/3, 1/5, usw.
            sum = 1.0 / (2.0 * i + 1);
            // Aufaddieren
            pi += vorz * sum;
            // Vorzeichen wechseln
            vorz *= -1;
            // Zähler um eins erhöhen
            i++;
        }

        // wir wollen aber pi, nicht pi/4
        pi *= 4.0;

        System.out.println("Näherung nach " + i + " Reihengliedern: " + pi);
        System.out.println("Abweichung von " + Math.PI + ": " + (Math.PI - pi));
    }
}

```

Die drei Programme liefern folgende Näherungen:

```

> java piReiheA
Näherung nach 100000 Reihengliedern: 3.1415826535897198
Abstand von 3.141592653589793: 1.0000000073340232E-5

```

```

> java piReiheB
Naeherung nach 50000001 Reihengliedern: 3.1415926735902504
Abstand von 3.141592653589793: 2.0000457290336726E-8
> java piReiheC
Naeherung nach 99995329 Reihengliedern: 3.141592663589793
Abweichung von 3.141592653589793: -9.99999993922529E-9

```

Aufgabe 4: Wurfparabel

Eine Wurfparabel beschreibt die Flugbahn eines Körpers während eines Wurfs. Der allgemeine Fall ist der sog. schiefe Wurf. Schreiben Sie ein Java-Programm, das die Flugbahn eines Körpers bei einem schiefen Wurf annäherungsweise in einem zweidimensionalen System berechnet.

Der Körper wird zu einem Zeitpunkt t_0 von der Position $\vec{r}_0 = (x_0, y_0)$ mit einer Geschwindigkeitsbetrag von $|v_0|$ und einem Winkel θ geworfen. Simulieren Sie das Flugverhalten bis der Körper wieder auf dem Boden auftritt, d.h. $y = 0$ gilt. Gehen Sie bei der Simulation wie folgt vor. Wenn man zu einem Zeitpunkt t die Position $\vec{r}_t = (x_t, y_t)$ und den Geschwindigkeitsvektor $\vec{v}_t = (v_x, v_y)$ kennt und möchte die Angaben für den Zeitpunkt $t + \Delta t$ ermitteln, so kann man sich folgender Zusammenhänge bedienen.

$$\text{Betrag der Geschwindigkeit } [\frac{m}{s}]: |v| = \sqrt{v_x^2 + v_y^2}$$

$$\text{Beschleunigung in x-Richtung } [\frac{m}{s^2}]: a_x = -C \cdot |v| \cdot v_x$$

$$\text{Beschleunigung in y-Richtung } [\frac{m}{s^2}]: a_y = -g - C \cdot |v| \cdot v_y$$

$$\text{Neue Geschwindigkeit in x-Richtung } [\frac{m}{s}]: v_x = v_x + a_x \cdot \Delta t$$

$$\text{Neue Geschwindigkeit in y-Richtung } [\frac{m}{s}]: v_y = v_y + a_y \cdot \Delta t$$

$$\text{Neue x-Position } [m]: x_{t+\Delta t} = x + v_x \cdot \Delta t$$

$$\text{Neue y-Position } [m]: y_{t+\Delta t} = y + v_y \cdot \Delta t$$

Die Konstante $g = 9,81 \frac{m}{s^2}$ ist die Erdbeschleunigung. C ist der Strömungswiderstandskeoeffizient, den wir hier beispielhaft mit $C = 0,002$ annehmen. Als Startwerte nehmen sie für Ihr Programm $|v_0| = 80 \frac{m}{s}$ und $\theta = 60$ Grad. Geben sie für jeden Stritt die Position $\vec{r}_t = (x_t, y_t)$ aus.

Info

- Für den initialen Geschwindigkeitsvektor brauchen Sie folgende trigonometrischen Gleichungen: $v_x = |v| \cdot \cos(\theta)$, $v_y = |v| \cdot \sin(\theta)$.
- Denken Sie daran, dass in Java Winkel nicht in Grad, sondern in Radiant angegeben werden. Überlegen Sie sich einen sinnvollen Wert für Δt .
- In der Programmierung arbeitet man häufig in interdisziplinären Teams. Es ist hier erstmal nicht Schlimm, wenn Sie die Physik dahinter nicht komplett verstehen. Sie können die Formeln trotzdem einfach in Code überführen.

Mögliche Lösung:

```

/**
 * Flugbahn eines Körpers
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Flugbahn {
    public static void main(String[] args) {

        //Konstanten
        double G = 9.8; // Gravitationskonstante
        double C = 0.002; // Luftwiderstandskeoeffizient
    }
}

```

```

// Startposition
double x = 0.0;
double y = 0.0;

// Startgeschwindigkeit
double v = Double.parseDouble(args[0]);

// Winkel
double theta = Double.parseDouble(args[1]);

// Winkel umwandeln in Radianen
theta *= Math.PI / 180.0;

// Geschwindigkeitsvektor
double vx = v * Math.cos(theta);
double vy = v * Math.sin(theta);

// Beschleunigung
double ax = 0.0;
double ay = -G - C * v * vy;

// Startzeit
double t = 0.0;

// Zeitsprünge
double dt = 0.01;

// Simulationsschleife, bis Körper wieder Boden erreicht (y=0)
while (y >= 0) {
    //aktuelle Position ausgeben
    System.out.println("(" + x + ", " + y + ")");

    // Betrag Geschwindigkeit
    v = Math.sqrt(vx * vx + vy * vy);

    // Beschleunigungsvektor a
    ax -= C * v * vx;
    ay -= G - C * v * vy;

    // Geschwindigkeitsvektor Veraenderung
    vx += ax * dt;
    vy += ay * dt;

    // Positionsvektor veraendern
    x += vx * dt;
    y += vy * dt;
}

}
}

```

Aufgabe 5: Spezielle Relativitätstheorie

Nach der speziellen Relativitätstheorie nimmt die dynamische Masse eines Körpers zu, je schneller er sich (in einem Inertialsystem) bewegt. Die Formel dazu ist:

$$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Wobei m_0 die Ruhemasse (das Gewicht) in kg des Körpers ist, v in $\frac{m}{s}$ die Geschwindigkeit und $c = 3 \cdot 10^8 \frac{m}{s}$ die Lichtgeschwindigkeit ist. Schreiben Sie ein Java-Programm, das einen Körper ihrer Wahl (zum Beispiel ihren Körper) mit dem Gewicht m_0 von $v = 0 \frac{m}{s}$ Richtung Lichtgeschwindigkeit $v = 3 \cdot 10^8 \frac{m}{s}$ „beschleunigt“ (im Sinne von: zu mehreren zunehmenden Geschwindigkeiten die dynamische Masse ausrechnen). Da es erst nahe der Lichtgeschwindigkeit interessant wird, erhöhen Sie ihre Geschwindigkeit so, dass sie zu Beginn sehr schnell zunimmt, zum Ende hin aber langsamer, also mehr Messpunkte vorhanden sind (wie kann man das in einer Schleife ausdrücken?). Geben Sie für jeden Berechnungspunkt (Geschwindigkeit) jeweils die Geschwindigkeit und die dynamische Masse aus.

Mögliche Lösung:

```
/*
 * Berechnung der dynamischen Masse eines Körpers abhängig von der Geschwindigkeit
 * @author Sigrid Weil
 * @version 1.0
 */
public class DynamischeMasse {
    public static void main(String[] args) {
        // Ruhemasse (in Kg)
        double m0 = 70;
        double m; // beschleunigte Masse (in Kg)
        double v = 0; // Geschwindigkeit (in Meter/Sekunde)
        double c = 3e8; // Lichtgeschwindigkeit (konstant)
        double c2 = c * c; // Quadrat der Lichtgeschwindigkeit
        double eps = 1.0e-6; // Genauigkeitsgrenze für die Berechnung

        // Simulationsschleife, bis die Geschwindigkeit nahezu gleich der Lichtgeschwindigkeit
        ist
        while (c - v > eps) {
            // Neue Masse berechnen und ausgeben
            m = m0 / Math.sqrt(1.0 - v * v / c2);
            System.out.println("v = " + v + "\t m = " + m);

            // Geschwindigkeit erhöhen (Abstand zu halten)
            v += 0.5 * (c - v);
        }
    }
}
```

Aufgabe 6: Newton Verfahren

Programmieren Sie das Newton-Verfahren zur Berechnung einer Nullstelle einer Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ (siehe dazu auch <https://de.wikipedia.org/wiki/Newtonverfahren>). Gesucht wird zu einer gegebenen Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ also eine Stelle $x \in \mathbb{R}$, für die gilt, dass $f(x) = 0$ ist. Das Newton-Verfahren versucht ausgehend von einem (beliebigen) Startwert x_0 sukzessive weitere x-Werte zu bestimmen, mit denen man näher an eine Nullstelle kommt. Dabei funktioniert das Verfahren wie folgt. Beginnend bei einem (beliebigen) Startwert x_0 wird für einen Wert x_i der Folgewert x_{i+1} mittels der Iterationsvorschrift

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

berechnet, bis $|f(x_i)| < \varepsilon$ für eine vorgegebene Genauigkeit $\varepsilon > 0$ ist. Verwenden Sie als Beispiefunktion $f(x) = x^2 - 2$ mit dem Startwert $x_0 = 1$ und die Genauigkeit $\varepsilon = 10^{-15}$. Zur Erinnerung: Für $f(x) = x^2 -$

2 ist die Ableitung gegeben durch $f'(x) = 2x$. Verwenden Sie je eine Methode zur Berechnung von $f(x)$, von $f'(x)$ und eine für die Berechnung des Absolutbetrags. Nennen Sie diese Methoden in Ihrem Java-Programm f, fstrich und abs und nutzen Sie diese Methodennamen in der obigen Berechnungsvorschrift.

- Wieviele Schleifendurchläufe benötigt Ihr Programm, bis Sie bei der vorgegebenen Funktion und Genauigkeit zu einer Lösung kommen? Wie oft wird dabei die Funktion f und die Funktion f' aufgerufen?
- Auch die Ableitung einer Funktion an einer Stelle x kann numerisch berechnet werden, indem die Näherung $f'(x) = \frac{f(x+d)-f(x)}{d}$ für $d \rightarrow 0$ bestimmt wird.

Ändern Sie Ihre Methode zur Berechnung der Ableitung so ab, dass d (beginnend mit $d = 1$) solange halbiert wird, bis zwei aufeinanderfolgende Näherungswerte einen Abstand kleiner als ε haben.

Mögliche Lösung:

```
/*
 * Newtons Iterationen
 * @author Sigrid Weil
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class NewtonNS {
    public static void main(String[] args) {
        double x = 1;
        double eps = 1.0e-15;

        // Simulationsschleife, bis die Funktion f(x) kleiner als eps ist und die Ableitung
        fs(x) nicht Null ist
        while ((Math.abs(f(x)) > eps) && (fs(x) != 0)) {
            x = x - (f(x) / fs(x));
            System.out.println("x = " + x + "\nf(x) = " + f(x));
        }
    }

    // Funktion f(x)
    public static double f(double x) {
        return x * x - 2;
    }

    // Ableitung von f(x)
    public static double fs(double x) {
        return 2 * x;
    }

    // Absolutwert-Funktion
    public static double abs(double x) {
        if (x < 0){
            return -x;
        } else{
            return x;
        }
    }
}
```

Um die Anzahl der Schleifendurchläufe zu ermitteln, kann eine Integer-Variable eingebaut werden. Die Funktion f wird in jedem Durchlauf dreimal aufgerufen (einmal zum Auswerten der Schleifen-Abbruch-Bedingung, einmal zu Berechnung des neuen Näherungswertes und einmal für die Ausgabe. Die Funktion f' wird in jedem Durchlauf zweimal aufgerufen. Das sollte durch Einführung von Zwischenvariablen verbessert werden.

Dazu ersetzen wir die Methode fs durch

```

public static double fs(double x) {
    double d = 1.0;
    double fx;
    double f1, f2;

    double eps = 1.0e-15;

    // Numerische Ableitung von f(x) mit dem Differenzenquotienten
    fx = f(x);
    f2 = (f(x + d) - fx) / d;

    do {
        f1 = f2;
        d = 0.5 * d;
        f2 = (f(x + d) - fx) / d;
    } while (Math.abs(f1 - f2) > eps);

    return f2;
}

```

Das Programm liefert die Ausgabe

```

x = 1.5
f(x) = 0.25
x = 1.4166666666666667
x = 1.4142156862745099
x = 1.4142135623746899
x = 1.4142135623730951
f(x) = 0.006944444444444642
f(x) = 6.007304882871267E-6
f(x) = 4.510614104447086E-12
f(x) = 4.440892098500626E-16

```

Aufgabe 7: Zufallszahlen

Schreiben Sie ein Programm, das zu drei Werten x, y, ε aus der Kommandozeile mit $x, y, \varepsilon \in \mathbb{R}, x < y, \varepsilon > 0, |y - x| > \varepsilon$ zwei verschiedene Zufallszahlen aus dem Intervall $[x, y]$ erzeugt und auf dem Bildschirm ausgibt. Zwei Zahlen a, b sollen dann als verschieden gelten, wenn $|a - b| \geq \varepsilon$.

Info

- Verwenden Sie die Methode `double Math.random()`. Diese liefert mit jedem Aufruf eine pseudozufällige Zahl aus dem Intervall $[0, 1)$. Überlegen Sie wie sie mit einer einfachen Berechnung diese Zahl in das angegebene Intervall $[x, y]$ überführen.
- Es ist unwahrscheinlich, aber nicht ausgeschlossen, dass zwei aufeinanderfolgende Aufrufe dieser Methode die gleiche oder fast gleiche Zufallszahl liefern. Sie müssen also solange zwei zulässige Zahlen aus dem gewünschten Intervall erzeugen, bis diese die gewünschte Eigenschaft besitzen.

Mögliche Lösung:

```

/**
 * 2 verschiedene Zufallszahlen zwischen x und y erzeugen
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class ZufallXYZ2 {
    public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);

```

```

double eps = Double.parseDouble(args[2]);

// Zufallszahlengenerator
double r;
double zahl1, zahl2;

do {
    r = Math.random();
    zahl1 = x + r * (y - x);
    r = Math.random();
    zahl2 = x + r * (y - x);
} while (Math.abs(zahl2 - zahl1) < eps);

System.out.println(zahl1 + " " + zahl2);
}
}

```

Aufgabe 8: Längste Kette

Geben Sie ein Java-Programm an, das bestimmt, was die Länge der längsten Kette gleicher natürlicher Zahlen in der Eingabe von der Tastatur ist. Die Beispieleingabe 1 1 2 2 2 3 3 3 3 4 3 3 1 5 . liefert 4 weil vier aufeinanderfolgende Dreien vorhanden sind und keine längere Kette existiert. Die Zahlen der Eingabe werden abgeschlossen durch einen einzelnen Punkt. Sie können folgende Programmteile verwenden, um eine unbekannte Anzahl an Werten nach diesem Schema einzulesen.

```

import java.util.*;
//...
Scanner sc = new Scanner (System.in);
while (sc.hasNextInt()){
    int i = sc.nextInt();
}

```

Die Anzahl der einzulesenden Werte wird größer als 0 sein und nach oben hin ist die Anzahl offen. Überlegen Sie sich, was mögliche Fälle sind und wie Sie diese behandeln. Überprüfen Sie ihr Verfahren / Programm vor der Abgabe erst per Hand / auf dem Papier, dann in eclipse mit verschiedenen Testwerten und geben erst dann ab, wenn dies alles korrekte Ergebnisse liefert.

Achtung

Wie immer: Sie dürfen nur Sprachkonstrukte verwenden, die in der Vorlesung auch bereits vorgestellt wurden!

Aufgabe 9: Lauflängenkodierung

Schreiben Sie ein Java-Programm Lauflängenkodierung, das eine Lauflängenkodierung für beliebig viele Eingabezahlen aus $\{0, 1, 2, \dots, 9\}$ vornimmt und auf dem Bildschirm in entsprechender Form ausgibt. Alle Eingabezahlen werden über die Tastatur übergeben und sind jeweils durch Leerzeichen oder Zeilenendezeichen voneinander getrennt. Die Eingabe wird abgeschlossen durch einen einzelnen Punkt. Siehe dazu die Vorlage zum Einlesen von Werten „Einlesen von Zahlen, deren Anzahl man vorher nicht kennt“, die Sie über unsere Java-Webseite ganz unten über den Link „Beispiele zum Einlesen von Daten“ erreichen. Eine Lauflängenkodierung geschieht dabei folgendermaßen. Für n aufeinanderfolgende gleiche Werte w wird diese Eingabesequenz gleicher Werte in der Ausgabe ersetzt durch die Angabe $nxw.$, also die Anzahl n der Vorkommen, das Zeichen x (kleines x), der eigentliche Wert w (der n -mal vorkam) und als Schlusszeichen ein Punkt. Auch ein einzeln auftretender Wert w wird nach diesem Verfahren kodiert und demzufolge als $1xw.$ erscheinen. Die gesamte Ausgabe wird durch ein Zeilenende abgeschlossen. Zur besseren Lesbarkeit der

Beispiele wurde in den bisherigen Beispielen jeweils ein Leerzeichen zwischen den 4 Komponenten eingefügt, das in ihrer Ausgabe aber weggelassen wird (siehe Referenzausgabe unten).

Beispiel

Eingabe:

1 1 1 7 7 1 1 0 .

Ausgabe: 3x1.2x7.2x1.1x0.

3 Einsen, dann 2 Sieben, dann 2 Einsen,... . Alles hintereinander, ohne Leerzeichen.

Beispiel

Eingabe:

1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 .

Ausgabe: 1x1.2x2.3x3.4x4.5x5.

Info

- Erweitern Sie in geeigneter Weise einen String.
- Dies ist eine alte Klausuraufgabe für ca. 15 Minuten Bearbeitungszeit.