

Lösung Übungsblatt Nr. 2 / ALP 2 zur Abgabe 28.04.2004

Aufgabe 1

```
public static double ln2a(long n) {
    long i = 2;
    double q = 1;
    while(i<=n) {
        if ((i%2)!=1) { q=q-(1.0/(double)i); } else { q=q+(1.0/(double)i); }
        i++;
    }
    return q;
}

public static double ln2b(long n) {
    long i = 2;
    double x = 0, y=0;
    while(i<=n) {
        x--=(1.0/(double)i);
        i+=2;
    }
    i = 1;
    while(i<=n) {
        y+=(1.0/(double)i);
        i+=2;
    }
    return (x+y);
}
```

Doku:

i ist in beiden Varianten eine Zählvariable.

q repräsentiert das Ergebnis der Methode.

n ist die Eingabe des Benutzers wieviele oft die Rechenschleife durchlaufen werden soll.

Die `while`-Schleife läuft solange wie $i \leq n$ ist.

Wenn i modulo 2 ungleich 1 ist, (i -Wert also gerade) wird q um $(1/i)$ herabgesetzt; andernfalls wird q um $(1/i)$ erhöht.

Fast das gleiche passiert in der 2. Variante. Hier werden die beiden `while`-Schleifen solange durchlaufen bis $i \leq n$ ist, wobei aber i bei jedem Durchgang um 2 erhöht wird.

Am Ende werden die beiden Werte addiert und das Ergebnis zurückgegeben.

Die Unterschiede kommen daher, dass die Abstände in der Gleitkomma-Arithmetik des Datentyps `double` unterschiedlich sind. (Je näher der 0, desto genauer.)

Das heisst, der Datentyp `double` kann nicht alle Zahlen darstellen, einige nur gerundet.

Wenn auf einen kleinen Wert (der schon gerundet ist) eine noch kleinere Zahl aufsummiert wird, fällt diese evtl. bei der nächsten Ablegung im Speicher der erneuten Rundung zum Opfer.

Demnach müsste die 2. Variante genauer arbeiten, da sie zuerst 2x parallel sehr kleine Werte summiert und diese dann am Ende addiert.

Eine noch genauere Lösung würde man erhalten, wenn man den Zähler rückwärts laufen liesse, dann würden zuerst extrem kleine Zahlen aufsummiert und danach erst die größeren.

Lösung Übungsblatt Nr. 2 / ALP 2 zur Abgabe 28.04.2004

Aufgabe 2.

- a) Syntaxbaum: nicht kapiert :)
- b) Fallunterscheidung für 0-Werte und negative Werte:

a=0 und b=0 --> es passiert nichts, das programm terminiert nach einem Durchlauf mit Ausgabe 0.
Wobei 0 niemals ein Teiler einer Zahl sein kann, da die Division durch 0 nicht definiert ist.
a=0 und b>0 --> das programm terminiert nicht, da von b in einer endlos-schleife 0 abgezogen wird und sich b somit nicht ändert. Selbiges gilt für a<-->b. (es wird dann von a 0 abgezogen.)
a<0 und b>0 --> das programm terminiert nicht, da von b in einer endlos-schleife a abgezogen wird (durch doppeltes minus also eigentlich addiert) und b somit gegen unendlich wächst, während a gleich bleibt. Für a<-->b analog. (a wächst gegen unendlich)

Aufgabe 3.

```
public static void perfect() {
    long n=1, q=0, i=0;
    do {
        q = 0;
        for (i=1; i<n; ++i) { if (n % i == 0) q+=i; }
        if (q == n) System.out.println(n);
        n++;
    } while(n>0);
}
```

Bei dieser Methode wird geprüft, ob bei einer gegebenen Zahl (n) die Werte von 1 bis n die Zahl n sauber teilen. Jene Werte, die n sauber teilen, werden aufsummiert und das Ergebnis mit n verglichen.

Sind beide Werte identisch wurde eine perfekte Zahl gefunden und ausgegeben.