Programmierstarthilfe SS 2008

Fakultät für Ingenieurwissenschaften und Informatik

2. Blatt

Für den 28. und 29.4.2008

Organisatorisches

Um auf die Mailingliste aufgenommen zu werden schicke einfach eine Mail an `guido.de-melo@uni-ulm.de`.

Die Webseiten zur Veranstaltung sind unter `http://www.uni-ulm.de/in/mi/lehre/ss2008/programmierstarthilfe.html` zu finden.

Die *Programmierstarthilfe* findet während des gesamten Semesters statt.

1. Einfaches Formatieren

Heute gibt es eine kurze Demonstration, warum wir unseren Code immer einrücken. Formatiere heute deinen eigenen Code auch gleich entsprechend!

2. Schleifen

Nachdem das letzte Mal nur ein Teil von euch bis zu den Schleifen gekommen ist wollen wir heute Aufgaben vom letzten Blatt und ein paar neue machen.

2.1. Schleifen 1

a) Schreibe ein Programm, das mit einer `while`-Schleife die ersten 20 Zahlen ausgibt (1, 2, ..., 20).

b) Ändere dein Programm so ab, dass die ersten 20 Zahlen, die ein Vielfaches von 3 sind ausgegeben werden (3, 6, 9, ...).

Da man Schleifen oft etwas zählen lässt, gibt es noch einen anderen Schleifentyp außer `while`. Die `for`-Schleife eignet sich hier ganz gut, da sie in einer Zeile Startwert, Schrittweite und Abbruchbedingung unterbringt. Das ganze sieht dann so aus:

```
int i = 0;
while(i < 20){
        //irgendwas
        i = i + 1;
}
```

```

for(int i = 0; i < 20; i = i + 1){
        //irgendwas

}
```

Die 3 Teile der `for`-Schleife sind durch Semikolons getrennt.

- Der erste Teil wird vor Schleifenbeginn ausgeführt.
- Der mittlere Teil ist die Bedingung, wie oft die Schleife durchlaufen wird. Die Bedingung wird vor jedem Schleifendurchlauf neu überprüft.
- Der letzte Teil wird immer am Ende der Schleife ausgeführt, noch bevor die Bedingung gecheckt wird.

c) Ändere dein Programm so ab, dass es das gleiche mit einer `for`-Schleife macht.

2.2. Schleifen 2

a) Schreibe ein Programm, das mit Hilfe einer Schleife alle `char`-Werte von 0 bis 600 durchzählt (du kannst den `char` hier wie einen `int` verwenden. Initialisiere dazu einen `char c` und erhöhe in in jedem Schleifendurchlauf um 1. Hinweis: Arbeiten mit `char` wird weiter unten erklärt.

Vorschicht: Verwende statt `c=c+1;` die Variante `c++;`

b) Ändere dein Programm so ab, dass mittels `System.out.print(c);` der `char c` ausgegeben wird. Außerdem soll alle 64 Zeichen ein Zeilenumbruch erfolgen.

c) Ändere dein Programm so ab, dass das 10te Zeichen nicht ausgegeben wird. In wiefern ändert sich die Ausgabe? Welches Zeichen ist also das 10te Zeichen?

2.3. * Euler Pi

Schreibe eine Prozedur, die die Kreiszahl π mit Hilfe der folgenden von Euler entwickelten unendlichen Reihe berechnet: $\frac{\pi^2}{6} = \sum_1^n \frac{1}{n^2}$

Für Fortgeschrittene: Die Anzahl der (ansonsten unendlichen) Summenschritte n soll der Prozedur dabei als Parameter übergeben werden.

Schreibe zunächst ein Flussdiagramm, dass den Programmablauf darstellt.

3. Weitere Datentypen

Das letzte Mal haben wir bereits `int` und `boolean` kennengelernt. Für ganze Zahlen kann man auch noch `short`, `byte` und long nehmen. Sie unterscheiden sich alle nur in ihrem

Wertebereich. Ein `byte` geht von -128 bis +127, ein `short` von -32 768 bis 32 767, ein `int` kann 2^{32} und ein `long` 2^{64} Werte speichern.

Für Fließkommazahlen gibt es `float` und `double`. Fließkommazahlen gibt man im Code mit einem Punkt an, dann werden sie als `double` erkannt, möchte man `float` schreibt man z.B. `10.1f`.

Es gibt auch einen besonderen Datentyp für einzelne Buchstaben, der `char` heißt. Er hat 16 Bit, da er Unicode-Zeichen aufnimmt.

Für Zeichenketten (Text) verwenden wir `String`, eine Klasse die später noch genauer eingeführt wird. Momentan soll uns genügen, dass wir Variablen anlegen können in die wir Text schreiben können.

```
1 String a = new String("Hallo Welt");
```

Im Beispiel wird in der Variable `a` der Text `"HalloWelt"` abgelegt. Der Unterschied zu den bisherigen Datentypen ist, dass wir bei Strings nun auf Informationen zugreifen können.

```
1 a.charAt(6);   // liefert W
2 a.length();    // ist 10
3 a.equals("Hallo");  // false
```

Mit `charAt(i)` greift man auf ein Zeichen an einer Position zu, dabei hat das erste Zeichen die Position 0. `length()` liefert die Länge und mit `equals()` kann verglichen werden ob zwei Strings gleich sind.

3.1. Datentyp char

Eine Variable vom Typ char (von engl. character=Buchstabe, Symbol) stellt in Java ein einzelnes Zeichen dar. Wird ihr eine Konstante zugewiesen, so muss diese in einfachen Anführungszeichen stehen (z.B. 'A'). Das Zeichen kann nicht nur ein Groß- oder Kleinbuchstabe sein, sondern z.B. auch eine Ziffer ('1'), ein Zeilenumbruch ('\n'), ein Tabulator ('\t') oder ein Sonderzeichen ('?'). Eine String-Variable repräsentiert dagegen eine Zeichenkette, die in doppelten Anführungszeichen geschrieben wird. Hinweis: Zeichen wie `"` oder `'` müssen mit einem vorangestellten Backslash (der sog. Escape-Sequenz) codiert werden, um vom Compiler nicht als Ende eines char oder String interpretiert zu werden, genauso nicht druckbare Zeichen wie newline (`\n`), oder der Backslash selbst `\\`).

```
1 char zeichen1, zeichen2;
2 zeichen1 = 'A';
3 zeichen2 = '\'';
4 String vieleZeichen;
5 vieleZeichen = "Das hier sind ganz viele Zeichen! Und aus-
```

```
6 serdem noch ein paar Anfuehrungszeichen \" und ein Back-
7 slash \\";
```

a) Schreibe eine Klasse UebungZuChar, in der du die Buchstaben des Wortes `belausche` als char speicherst. Gib sie einmal in richtiger, und einmal in anderer Reihenfolge aus (egal welche, aber es ließe sich auch ein anderes Wort daraus bilden...).

b) Was passiert, wenn du sieben char Variablen die Werte 72, 97, 109, 115, 116, 101, 114 zuweist, und in dieser Reihenfolge ausgibst?

c) Der Datentyp char besteht aus 2 Bytes (=16 bit) und kann damit 2hoch16 (=65536 = ziemlich viele) Zeichen darstellen. Die ersten 128 nennt man den ASCII-Zeichensatz, und die Zeichen darin sind durchnummeriert. Sie können in Java auch über ihre Nummer angesprochen werden, deswegen ist es möglich char-Variablen Zahlen zuzuweisen. Suche dir (z.B bei wikipedia) eine ASCII-Tabelle, suche die Buchstaben deines Vornamens, und ändere dein Programm aus (b) so ab, dass dein Vorname ausgeben wird.

3.2. Strings konkatenieren

Strings kann man mit einem Plusoperator hintereinanderhängen, auch konkatenieren genannt (ersterString + zweiterString + ...). Das funktioniert auch mit anderen Datentypen, die daraufhin in einen String konvertiert werden. Wahrscheinlich habt ihr genau das schon mal innerhalb von `System.out.println()` benutzt, denn hier passiert nichts anderes als dass alle durch + getrennte Teile in einen String konvertiert werden, hintereinandergehängt und anschließend ausgegeben.

Schreibe eine Klasse `StringKonkatenation`, definiere Variablen aller Typen die du bisher kennst, und speichere deren Konkatenation in einem weiteren String, den du anschließend ausgibts. (Hinweis: eventuell muss die Konkatenation mindestens einen String enthalten, dies kann aber auch ein leerer (`""`) sein.)

3.3. Palindromtest

Schreibe einen Palindromtester: Er soll feststellen, ob ein String vor- und rückwaerts gelesen gleich lautet. Palindrome sind z.B. Anna, Reliefpfeiler, Rentner, Ein Esel lese nie. Dabei geben wir die Strings dann immer in Kleinbuchstaben an.

4. Arithmetik

Das letzte Mal haben wir schon ganz einfach mit + gerechnet, daneben kann man natürlich noch einiges mehr nutzen. `- * /` ergänzen unsere Grundrechenarten. `%` ist für Modulo und ergibt den ganzzahligen Rest wenn man zwei Zahlen teilt.

Zwei Dinge muss man beim Programmieren häufig tun, eins abziehen oder dazu zählen. Deshalb gibt es dafür Kurzformen: `a++ a--`.

4.1. Arithmetische Operationen

In dieser Aufgabe werden dir grundlegende Rechenoperationen nahe gebracht. Das meiste wird sich wohl so verhalten, wie man es erwarten würde. Trotzdem ist es wichtig die arithmetischen Operationen einmal im Detail durchgesprochen zu haben.

4.1.1. Addition, Subtraktion und Multiplikation

Wenig überraschend addiert man Variablen und Zahlen mit einem Plus und subtrahiert sie mit einem Minus. Auch ist das Minus das Vorzeichen für negative Zahlen. Berechnungen in runden Klammern werden zuerst ausgeführt.

```
1  int a = 10; int b = 4; int c;
2
3  c = -42;          // c ist Minus 42
4
5  c = 5 + 3;        // c bekommt als Wert 5 + 3, also 8
6  c = 7 + b;        // c wird 7 + 4, also 11 zugewiesen
7
8  c = a + b;        // c wird nun 10 + 4, also 14 zugewiesen
9  c = a - b;        // c ist nun 10 - 4, also 6
10
11 c = a - b - b;    // c ist nun 10 - 4 - 4. Das ist 2, denn
12                   // der Ausdruck wird von links nach
13                   // rechts abgearbeitet, also (10 - 4) - 4
14
15 c = a - (b - b); // Diesmal ist c gleich 10, da der Ausdruck
16                   // in der Klammer zuerst berechnet wird
```

Die Multiplikation wird mit dem Sternchen durchgeführt. Auch hier ergeben sich keine Überraschungen. Die Regel Punkt vor Strich gilt sinnvollerweise auch bei Java. Das Sternchen kann niemals weggelassen werden, so wie wir das mit dem Multiplikationspunkt beim Rechnen auf Papier tun.

```
1  int a = 2, b = 5, c = 3, d;      // Abkuerzende Schreibweise!
2
3  d = a + b * c;            // d = 2 + 5 * 3 = 2 + 15 = 17
4  d = (a + b) * c;          // d = (2 + 5) * 3 = 7 * 3 = 21
```

Aufgabe: Schreibe ein Programm, dass abhängig von Integern für Sekunden, Minuten, Stunden, Tage und Jahre die du am Anfang deines Programms im Quellcode festlegst (Beispiel: int tage = 11;), ausgibt, wie viele Sekunden das sind. Gehe davon aus, dass alle Jahre genau 365 Tage haben.

4.1.2. Die Division und ihre Tücken

Selbstverständlich kann man in Java auch Zahlen durcheinander teilen. Gemacht wird das mit dem Schrägstrich über der Sieben.

```
1 int a = 10, b = 5;
2 System.out.println(a/b);          // gibt die Zahl 2 aus,
3                                   // denn 10 durch 5 ist 2
```

Aufgabe 1: Führe folgenden Code aus und überlege dir, wie die Ergebnisse zu stande kommen. Hinweis: Selbstverständlich muss nachfolgender Code in der Main-Methode einer Klasse stehen :)

```
1 int a = 3, b = 7, c = 4, d = 12;
2
3 System.out.println(8/4);          // Acht durch vier ist zwei.
4 System.out.println(d/c);          // hier ist alles normal.
5 System.out.println(d/a);          // hier auch.
6 System.out.println(a/c);          // aber was passiert hier?
7 System.out.println(b/c);          // und was ist hiermit?
```

Aufgabe 2: Durch Null teilen ist bekannstlich verboten. Sogar bei Java. Allerdings kann es schnell mal passieren, dass gerade bei sehr komplexen Ausdrücken und irgendwelchen Sonderfällen der Nenner Null werden könnte, weil man nicht aufgepasst hat. Um dir klar zu machen, was dann passiert - auch damit du es dann bei deinem eigenen Code erkennst - provozieren wir es doch einfach. Führe folgenden Code aus (auch wieder in der Main-Methode einer Klasse) und betrachte das Ergebnis.

```
1 System.out.println(42/0);         // sowas tut man nicht...
```

4.1.3. Modulo

Ein ganz besonderer arithmetischer Operator ist das Prozentzeichen. Was es berechnet hat allerdings nichts mit Prozent zu tun, denn es berechnet den Modulo. Kurz gesagt ist A modulo B nichts anderes, als der Rest der die ganzzahlige Division A durch B hinterlässt. Kurz geschrieben: mod. Somit ist 9 mod 2 gleich 1, denn 9 geteilt durch 2 ist 4 Rest 1. Nützlich

ist dies zum Beispiel, wenn man herausfinden möchte, ob eine Zahl gerade ist, denn X mod 2 ist gleich Null genau dann, wenn X gerade ist. Weiterführender Text zum Thema Modulo: http://de.wikipedia.org/wiki/Modulo

Aufgabe: Überlege dir wie die Ausgaben des folgenden Codes zu stande kommen. Teste deine Vermutung durch ausführen des Codes.

```
1  System.out.println(10 % 5);       // Rest von "10 durch 5"
2  System.out.println(11 % 5);
3  System.out.println(5 % 5);
4  System.out.println(-5 % 5);
5  System.out.println(14 % 5);
6  System.out.println(83462434 % 5);
7  System.out.println(3 % -2);
8  System.out.println(-3 % 2);
9  System.out.println(0 % 2);
10 System.out.println(-5 % 4);       // Hier wird man
11 System.out.println(-7 % 4);       // moeglicherweise ueberrascht
```

Hinweis zu den letzten beiden:
Mathematisch gilt meist: $-5 \bmod 4 = 3$, weil $-5 = -8 + 3 = (-2 * 4) + 3$
Bei Java gilt hingegen: $-5 \bmod 4 = -1$, weil $-5 = -4 + (-1) = (-1 * 4) + (-1)$

Wie bei der Division ist auch Modulo 0 nicht sinnvoll und daher verboten (probiers mal aus). Modulo ist in Java im Übrigen nicht nur für Ganzzahlen definiert, aber in den aller meisten Fällen beschränkt man sich auf natürliche Zahlen.

4.1.4. Abkürzende Schreibweisen

Es kommt oft vor, dass man einer Variablen nicht nur das Rechenergebnis zweier anderen Variablen zuweisen möchte, sondern den Wert der Variablen relativ zum aktuellen Wert verändern möchte. Dafür existieren einige abkürzende Schreibweisen:

```
1  int a = 10, b = 20;
2
3  a += b; // ist das selbe wie "a = a + b"
4  a -= b; // ist das selbe wie "a = a - b"
5  a *= b; // ist das selbe wie "a = a * b"
6  a /= b; // ist das selbe wie "a = a / b"
7  a %= b; // ist das selbe wie "a = a % b"
```

Hier wird auch klar, dass sich Variablen in einer Programmiersprache von mathematischen Variablen unterscheiden. $a = a + b$ ist keine Gleichung, sondern eine Zuweisung der Form: a wird als neuer Wert der bisherige Wert von a plus dem Wert von b zugeordnet.

Aufgabe: Welchen Wert hat a jeweils bei der Ausgabe? Teste deine Lösung.

```
int a = 10;
a += 2; a *= 2; a /= 6;
System.out.println(a);

a *= a*a-1;
System.out.println(a);

a %= (a/10)+1; a *= a+2; a -= 3;
System.out.println(a);

a *= (a%(a-1))*2;
System.out.println(a);
```

Tipp: Das End-Ergebnis ist 42.

4.1.5. Inkrementieren, Dekrementieren

Häufig will man den Wert einer Variable gerade um 1 erhöhen oder verringern. Genau hierfür gibt es mit Plus-Plus und Minus-Minus zwei spezielle Abkürzungen, die allerdings sehr tückisch sein können. Steht Plus-Plus oder Minus-Minus vor der Variablen, so wird der Wert zunächst erhöht/verringert und dann verwendet, also zum Beispiel ausgegeben. Steht es statt dessen aber dahinter, wird der alte Wert verwendet und nach dieser Verwendung erst verändert.

```
int a = 10;                 // Vorher: a = 10
System.out.println(a++);// Wert ausgeben (10) und erhoehen
System.out.println(a);   // Jetzt: a = 11
System.out.println(++a);// Wert erhoehen und ausgeben (12)
System.out.println(a);   // Jetzt: a = 12
System.out.println(a--);// Wert ausgeben (12) und verringern
System.out.println(a);   // Jetzt: a = 11
System.out.println(--a);// Wert verringern und ausgeben (10)
System.out.println(a);   // Jetzt: a = 10
```

Hinweise: Plus-Plus und Minus-Minus sind offenbar sehr eigenwillige Operatoren und machen einem das Leben unnötig schwer, wenn man sie in Rechnungen oder Ausgaben einsetzt. Deshalb sollte man Plus-Plus und Minus-Minus nur ganz alleine auf eine Variable in einer Anweisung anwenden.

Bonusaufgabe für besonders Interessierte: Um dir klar zu machen, dass Plus-Plus und Minus-Minus wirklich unangenehm sein können, insbesondere wenn die inkrementierte/dekrementierte Variable im selben Term nochmals auftaucht, ist es deine Aufgabe folgenden Code so

umzuschreiben, dass Plus-Plus nicht mehr vorkommt, aber noch das selbe auf die selbe Weise berechnet wird:

```
1  /* (Anfang Kommentarblock)
2   * Beispiel:
3   *       System.out.println(++a+a);
4   * wird zu
5   *       System.out.println((a+1)+(a+1));
6   * (Ende Kommentarblock)
7   */
8
9  int a = 10;
10 a += ++a+a+++*++a+a++*++a*++a+a;
11 System.out.println(a); // Tipp: Die Ausgabe ist 3300
```