Programmierstarthilfe SS 2008

Fakultät für Ingenieurwissenschaften und Informatik

5. Blatt

Für den 26. und 27.5.2008

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.

1. Typecasts

In manchen der letzten Aufgaben habt Ihr mit Buchstaben, also dem Datentyp `char` gearbeitet. Dabei ist euch aufgefallen, dass man in Java zu Buchstaben (Typ `char`) eine Zahl (Typ `int`) hinzuaddieren kann. Dass das möglich ist, hängt mit Typecasts, d.h. Typanpassung, zusammen.

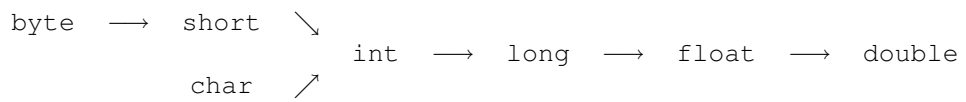
In Java werden ganz automatisch Typecasts ausgeführt, sogenannte "implizite Typcasts", wenn unterschiedliche Datentypen miteinander verrechnet werden. Im Folgenden zähle ich beispielsweise auf das Zeichen `c` 10 Zeichen hinzu, also mache ein `m` daraus. Das Ergebnis wird wieder als Buchstabe gespeichert.

```
1 char zeichen = 'c'; int ganzzahl = 10;
2 zeichen += ganzzahl;
```

Außerdem werden implizite Typecasts verwendet, wenn eine Zahl in ihrem Wertebereich erweitert wird, wenn ich zum Beispiel eine ganze Zahl (Typ `int`) als Kommazahl (Typ `double`) abspeichere:

```
1 double kommazahl = ganzzahl;
```

Solche impliziten Typecasts nimmt Java nur vor, wenn dadurch der Wertebereich erweitert wird, nicht aber in Gegenrichtung. Das Schaubild unten zeigt mögliche implizite Typecasts in Pfeilrichtung an. Es ist natürlich auch möglich, einige Stufen zu überspringen und beispielweise `char` direkt auf `double` zu casten.



Auf den letzten Blättern und auch oben haben wir uns den Trick erlaubt, `zeichen += ganzzahl` zu verwenden, um mit `char` zu rechnen. Eigentlich müsste man ja stattdessen auch schreiben dürfen:

```
1 char neues_zeichen = zeichen + ganzzahl;
```

Das geht aber nicht ohne weiteres, wie man am Schaubild sieht, denn Java rechnet zuerst `zeichen + ganzzahl` aus und castet dabei auf `int`. Nun aber zurückzucasten auf `char` ist nicht mehr in Pfeilrichtung möglich.

Manchmal möchte man trotzdem in Gegenpfeilrichtung casten. Dafür gibt es die Möglichkeit den Zieldatentyp explizit anzugeben, daher auch der Name "expliziter Typcast". Man trägt diesen einfach in Klammer ein:

```
1 char neues_zeichen = (char) (zeichen + ganzzahl);
2 int neue_ganzzahl = (int) kommazahl;
```

Beim expliziten Casten ist Vorsicht geboten, denn der Wertebereich wird unter Umständen verkleinert. Wenn ich beispielsweise eine Kommazahl nach `int` caste, gehen die Nachkommastellen verloren (kein Runden, es wird abgeschnitten).

1.1. Dividieren

a) Ich möchte "0.0", "0.1", ..., "1.0" ausgeben. Warum reicht dafür Folgendes nicht aus?

```
1 for(int i=0; i<=10; i++)
2     System.out.println(i/10);
```

Wie geht es richtig?

b) Um die Summe aller Zahlen von 1 bis n zu berechnen, kann man die Gaußsche Summenformel verwenden:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Schreibe eine entsprechende Methode, die diese Formel benutzt und teste sie. Welche Division muss hier genutzt werden?

Hinweis: Java kennt zwei Divisionen: Teilt man `int`s durcheinander, so kommt auch wieder eine ganze Zahl heraus, der Rest wird ignoriert. Teilt man Kommazahlen durcheinander, so ist das Ergebnis ebenfalls eine Kommazahl.

1.2. Würfel-Spiel

Die Methode `Math.random()` liefert bei jedem Aufruf eine Zufallszahl zwischen 0 und 1 als `double`.

a) Schreibe eine Methode `wurf`, welche mit Hilfe von `Math.random()` eine gleichverteilte ganzzahlige Zufallszahl von 1 bis 6 zurückliefert, also einen Würfel simuliert.

b) Schreibe ein Programm, welches diese Methode `anzahl` mal ausführt und ausgibt, wie oft welche Zahl aufgetreten ist.

c) Schreibe weiter eine Methode `histogram`, welche vom Programm zur Ausgabe der Ergebnisse in einem Histogramm aufgerufen wird. Kam zum Beispiel die 1 5-mal vor, die 2 7-mal, die 3 4-mal, die 4 7-mal, die 5 6-mal und die 6 5-mal, so sollte die Ausgabe etwa so aussehen:

```
1: =====
2: =======
3: ====
4: =======
5: ======
6: =====
```

2. Ein- und Ausgabe

Dieses Code-Beispiel zeigt dir, wie man Strings und Integer von der Eingabe einlesen kann:

```
1 import java.util.*;
2 public class Test {
3     public static void main(String[] args) {
4         Scanner scanner = new Scanner(System.in);
5         String myString = scanner.nextLine();
6         int myInt = scanner.nextInt();
7     }
8 }
```

Natürlich kannst du dir Methoden bauen, die das Einlesen für dich übernehmen.

2.1. Testen der Einlesefunktion

a) Verändere das “Hello-World”-Programm vom ersten Blatt so, dass du zunächst nach deinem Namen gefragt und dann entsprechend begrüsst wirst.

b) Dein Programm soll nun dein Geburtsdatum abfragen und dann entsprechend dein Alter ausgeben.

*c) Schreibe eine Methode, die nach Einlesen deines Geburtsdatums und des heutigen Datums ausrechnet, wieviele Tage du alt bist. Verwende hierfür deinen Code von Blatt 2. Du kannst die Aufgabe vereinfachen, indem du davon ausgehst, dass jeder Monat 30 Tage hat.

2.2. **Nimm-Spiel

Schreibe ein kleines Spiel: Zunächst soll die Anzahl der verwendeten Stäbchen eingegeben werden, dann nehmen der Spieler und der Computer abwechselnd 1 bis 3 Stäbchen weg (wobei der Computer immer die optimale Strategie verfolgt, schreibe dazu eine Methode `rechnerNimmt()`). Verloren hat derjenige, der das letzte Stäbchen wegnimmt. Der Spieler kann anschließend entscheiden, ob er nochmal spielen will.

Überlege dir zunächst den Programmablauf und skizziere ihn durch Kommentare in deinem Quellcode.

3. Arrays

Ein Array (deutsch "Feld") ist eine Datenstruktur, in der mehrere Elemente eines Datentyps gespeichert werden können. Zum Beispiel kannst du folgendermaßen einen dreidimensionalen Integer-Vektor anlegen und mit Werten füllen:

```
1  int vector1[] = new int[3];
2  vector1[0] = 3;
3  vector1[1] = 2;
4  vector1[2] = 1;
5
6  int[] vector2 = new int[3];
7  vector2[0] = 3;
8  vector2[1] = 2;
9  vector2[2] = 1;
10
11 int[] vector3 = {3, 2, 1};
```

Diese drei Schreibweisen sind äquivalent, d.h. in den Vektoren stehen jetzt dieselben Werte. Die Länge eines (beliebigen) Arrays kann man ähnlich abfragen wie bei Strings:

```
1  int laenge = vector1.length;
```

Es gibt auch mehrdimensionale Arrays. So wird z.B. eine 2×2-Integer-Matrix angelegt und gefüllt:

```
1  int[][] matrix1 = new int[2][2];
2  matrix1[0][0] = 1;
3  matrix1[0][1] = 2;
4  matrix1[1][0] = 3;
5  matrix1[1][1] = 4;
6
7  int matrix2[][] = new int[2][2];
8  matrix2[0][0] = 1;
9  matrix2[0][1] = 2;
10 matrix2[1][0] = 3;
11 matrix2[1][1] = 4;
12
13 int[][] matrix3 = {{1, 2}, {3, 4}};
```

Auch hier sind die Schreibweisen äquivalent und die Matrizen dementsprechend gleich.

In der Matrix stehen jetzt die Werte

```
1 2
3 4
```

`matrix1.length` gibt in diesem Fall `2` zurück, was der Anzahl der Zeilen entspricht. Mit `matrix1[0].length` erhält man die Anzahl der Elemente der ersten Zeile, mit `matrix1[1].length` entsprechend die Anzahl der Elemente der zweiten Zeile.

3.1. Nicht-rechteckige Arrays

Mehrdimensionale Arrays müssen nicht rechteckig sein. Schreibe ein Programm, das ein dreieckiges Char-Array, das mit '*' gefüllt ist, anlegt und ausgibt. Probiere auch andere Formen aus, z.B. Rauten. Schaffst du es, einen Stern anzulegen und auszugeben?

3.2. Polynome

Entwickle eine Anwendung zur Berechnung des Funktionswerts von Polynomen $p(x) = a_n x + ... + a_1 x + a_0$. Das Programm soll zuerst den Grad n des Polynoms als `int` und dann die Koeffizienten a_i (als `doubles`) in ein entsprechend zu dimensionierendes Array einlesen. Danach sollen so lange Argumente x eingelesen und die Funktionswerte $p(x)$ ausgegeben werden, bis für x die 0 eingegeben wird.

3.3. Matrizen-Addition

Implementiere eine Methode `matrixAdd(int m1[][], int m2[][])` derart, dass sie die beiden Eingabematrizen `m1` und `m2` addiert und die Ergebnismatrix zurückgibt.

Eine Matrix wird mit einer anderen Matrix addiert,indem die Matrixelemente einzeln addiert werden. Das heißt: Seien A, B Matrizen mit I Zeilen und J Spalten und a_{ij}, b_{ij} das jeweilige Element in Zeile i und Spalte j. Dann ist $C = A + B$ gegeben durch $c_{ij} = a_{ij} + b_{ij}$ für alle i, j. Eine Matrixaddition ist nur möglich, wenn beide Operanden die selbe Anzahl von Zeilen und Spalten besitzen.

3.4. **Matrizen-Multiplikation

Schreibe ein Programm, welches zwei Matrizen miteinander multipliziert. Die zu multiplizierenden Matrizen sollen dabei als Eingabe vom Benutzer gelesen werden. Gehe bei der gesamten Aufgabe davon aus, dass nur Ganzzahlen als Matrizenelemente behandelt werden.

a) Schreibe eine Methode, welche eine Matrix von der Standardeingabe einliest und an die aufrufende Funktion als zweidimensionales Array zurückgibt. Prüfe beim Einlesen der Matrix auch, ob die Eingabe wirklich eine gültige Matrix ist und gib andernfalls eine Fehlermeldung aus.

b) Implementiere eine Methode, welche eine Matrix im schönen Format wieder auf dem Bildschirm ausgibt. Teste mit Hilfe dieser Methode, ob deine Eingabemethode korrekt funktioniert.

c) Schreibe eine Methode, die zwei Matrizen als Eingabe bekommt, diese miteinander multipliziert und das Ergebnis zurückgibt. Die Methode sollte auch prüfen ob die beiden übergebenen Matrizen überhaupt miteinander multipliziert werden können.

d) Setze alle bisher erstellten Methoden zu einem kompletten Programm zusammen, welches zuerst zwei Matrizen von der Standardeingabe einliest, diese dann multipliziert und das Ergebnis dann wieder auf dem Bildschirm ausgibt.

Hinweis: Matrizen-Multiplikation wird hier anschaulich erklärt:
`http://de.wikipedia.org/wiki/Falksches_Schema`