ist Institut für Softwaretechnik

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

JAVA BLOCKKURS

Programmieraufgaben

Dr. Volker Riediger
Institut für Softwaretechnik
Universität Koblenz-Landau

riediger@uni-koblenz.de

INHALTSVERZEICHNIS

JAVA BLOCKKURS

Programmieraufgaben

Dr. Volker Riediger

Generelles

Bearbeiten Sie die Aufgaben möglichst der Reihe nach. Wenn Ihnen eine Aufgabe zu schwierig erscheint, überspringen Sie diese.

Erstellen Sie pro Aufgabe eine eigene Java-Klasse. Manche Aufgaben sind als Erweiterung der vorangehenden Aufgabe gedacht. Für diese brauchen Sie natürlich keine neue Klasse schreiben. Damit Sie die Übersicht nicht verlieren, können Sie die Klassen nach Aufgabennummern benennen. Verwenden Sie z.B. den Klassennamen `A02Summe` für die zweite Aufgabe.

Die Aufgaben sind mit Schwierigkeitsgraden markiert:

* einfach
** schon etwas schwieriger
*** sehr schwierig

Die *** Aufgaben sind für fortgeschrittene Programmierer gedacht, die sich bei den anderen Aufgaben zu sehr "langweilen". Das soll aber auch die Anfänger nicht davon abhalten, die Lösung zu versuchen, oder zumindest darüber nachzudenken!

Versuchen Sie in allen Programmen, die Benutzereingaben entgegen nehmen, auf ungültige Eingabewerte mit sinnvollen Fehlermeldungen zu reagieren.

Testen Sie Ihre Programme mit geeigneten, selbst gewählten Eingabewerten.

Verwenden Sie Methoden, um Ihre Lösungen zu strukturieren. Wenn Sie ein Stück Programmtext schreiben, das länger als ca. 20-25 Anweisungen (bzw. Zeilen) ist, sollten Sie anfangen zu überlegen, ob Sie das Programmstück nicht sinnvoll in mehrere Methoden unterteilen können.

JConsole

In Java ist es nicht besonders einfach, Zahlenwerte und Zeichenketten von der Tastatur einzulesen. Um die Eingabe von Werten zu erleichtern, soll die `JConsole` zur Ein- und Ausgabe verwendet werden. Die Programme, die mit Eingaben arbeiten, werden Sie mit der Eclipse-IDE entwickeln. Diese Arbeitsumgebung erzeugt ein Verzeichnis `workspace` in Ihrem Heimatverzeichnis.

JConsole besteht aus einem Java-Archiv (jar-Datei), die Sie auf der Webseite zum Praktikum finden. Speichern Sie die Datei im Eclipse-Workspace (`/home/username/workspace` unter Unix/Linux, Laufwerk `Z:\nt\workspace` unter Windows, `/Users/username/workspace` unter MacOS).

Die Betreuer Ihres Praktikums werden Ihnen helfen, `JConsole` in den "Java Build Path" Ihres Eclipse-Projekts aufzunehmen.

A. Einfache lineare Programme

Schreiben Sie für die folgenden Aufgaben jeweils ein Java-Programm! Benutzen Sie in allen Programmen die `JConsole`!

A.1. Texte ausgeben *

Geben Sie Ihren Namen, Vornamen und die E-Mail-Adresse in drei einzelnen Zeilen aus.

A.2. Ein wenig Rechnen *

Berechnen Sie die Summe von 17 und 4 und geben Sie das Ergebnis aus.

A.3. Währungsumrechnung *

Berechnen Sie, wie viel Euro der DM-Betrag 9876,54 entspricht (Kurs: 1 EUR = 1,95583 DM). Geben Sie das Ergebnis aus. Sie brauchen keine Rücksicht auf die Formatierung der Ausgabe nehmen - es kann sein, dass mehr oder weniger als zwei Nachkommastellen angezeigt werden.

A.4. Eingabe von Zeichenketten *

Lesen Sie Ihren Vor- und Nachnamen in `String`-Variablen ein und geben Sie beides in der Form 'Nachname, Vorname' wieder aus.

A.5. Summenberechnung *

Lesen Sie zwei Zahlen ein und berechnen Sie die Summe. Die Ausgabe soll folgende Form haben:

```
Die Summe von 123 und 234 ist 357.
```

A.6. Vertauschen zweier Werte *

Lesen Sie zwei Zahlen ein und vertauschen Sie die Werte der beiden Variablen.

Beispiel: Vor dem Vertauschen hätten die Variablen `a` und `b` die Werte `a=5` und `b=3`, nach dem Vertauschen soll `a=3` und `b=5` gelten.

A.7. Vertauschen ohne Hilfsvariable **

Haben Sie für die Lösung der vorigen Aufgabe eine Hilfsvariable benötigt? Versuchen Sie es auch einmal ohne!

B. Programme mit Verzweigungen

B.1. Sortieren von zwei Zahlen *

Bringen Sie zwei eingegebene Zahlen a und b in die richtige Reihenfolge bringt (a ≤ b) und geben Sie die Zahlen wieder aus. Auch hier gilt, dass der Inhalt der Variablen geändert werden soll, nicht nur die Ausgabe der Werte.

B.2. Schaltjahre **

Schreiben Sie ein Programm, das zu einer eingegebenen Jahreszahl ermittelt, ob es sich um ein Schaltjahr handelt. Zur Erinnerung: Ein Schaltjahr gibt es dann, wenn die Jahreszahl durch 4 teilbar ist, aber nicht durch 100, oder durch 400 teilbar ist.

B.3. Sortieren von drei Zahlen **

Bringen Sie drei eingegebene Zahlen `a`, `b` und `c` in die richtige Reihenfolge bringt (`a` ≤ `b` ≤ `c`) und geben Sie die Zahlen wieder aus.

B.4. Zufallszahl *

Berechnen Sie eine ganzzahlige Zufallszahl im Intervall [1..1000]. Nutzen Sie dazu die in Java vordefinierte Methode `Math.random()`, die einen `double`-Wert aus dem Intervall [0.0, 1.0[liefert.

B.5. Methode verwenden *

Das Programm soll einen Euro-Betrag einlesen, um die Mehrwertsteuer erhöhen und den Gesamtbetrag in Euro und DM ausgeben. Benutzen Sie Unterprogramme (Methoden) zur Berechnung der Mehrwertsteuer und zur Umrechnung der Währung!

C. Programme mit Schleifen

C.1. Summe mit while-Schleife *

Berechnen Sie die Summe der Zahlen von m bis n (m und n werden eingelesen) und geben Sie diese aus. Benutzen Sie eine Methode mit `while`-Schleife!

C.2. Andere Schleifenarten **

Lösen Sie das vorige Problem mit einer `for`- bzw. einer `do...while`-Schleife! Schreiben Sie dazu zwei neue Methoden und überprüfen Sie im Programm, ob alle drei Methoden für gleiche Eingaben den gleichen Wert berechnen!

C.3. Komfortablerer Währungsrechner **

Realisieren Sie einen einfachen Währungsrechner. Geben Sie dazu zunächst die beiden Währungsnamen, dann den Umrechnungskurs ein. Danach sollen mehrere Beträge umgerechnet werden, bis der Benutzer 0 (null) eingibt.

C.4. Weihnachtsbäume **

Geben Sie stilisierte "Weihnachtsbäume" mit "Stamm" aus. Lassen Sie vom Benutzer die gewünschte die Höhe `h` eingeben und den "Baum" über eine Methode mit einem Parameter (der Höhe) ausgeben. Das Ergebnis, z.B. für z.B. `h=5`, soll so aussehen:

```
    *
   ***
  *****
 *******
*********
    #
```

C.5. Zahlworte ***

Erzeugen Sie Zahlworte für die Zahlen 0...999.999.999. Beispiel:

1234567 → einemillionzweihundertvierundreißigtausendfünfhundertsiebenundsechzig.

Zerlegen Sie die Aufgabe in Teile, z.B. Zahlen bis 9, Zahlen bis 99, Zahlen bis 999 etc. ausgeben, und verwenden Sie diese Teile, um möglichst wenig doppelt zu programmieren. Zahlen außerhalb des Bereichs sollen zurückgewiesen werden.

D. Ein wenig Mathe

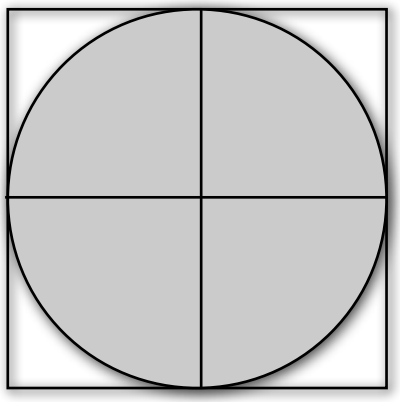
D.1. ggT-Berechung nach Euklid *

Implementieren Sie den Euklid'schen Algorithmus zur Ermittlung des größten gemeinsamen Teilers zweier natürlicher Zahlen p und q!

1. Dividiere `p` durch `q` (ganzzahlig). Dabei ergibt sich ein Rest `r`.
2. Ist `r=0`, so ist `q` der gesuchte ggT
3. Andernfalls setze `p` gleich `q` und `q` gleich `r` und gehe zu Schritt 1.

D.2. Näherungswert für π **

Berechnen Sie einen Näherungswert füt π nach folgender Beschreibung:



Male einen Kreis mit einem Radius von 1 Meter auf den Boden. Zeichne um den Kreis ein Quadrat mit der Seitenlänge 2 Meter und teile den Kreis in 4 Viertelkreise, indem die Mittelpunkte gegenüberliegender Seiten des Quadrats verbunden werden. Lasse es eine Weile lang auf Deine Zeichnung regnen. Sieh Dir ein beliebiges Viertel des Quadrats an: Wenn Du die Anzahl der Regentropfen innerhalb des Viertelkreises durch die Gesamtzahl der Regentropfen in diesem Viertel teilst und mit 4 multiplizierst, erhältst Du eine Näherung für π. Je länger es regnet, desto genauer ist der Näherungswert.

Die "Regentropfen" simulieren Sie am besten mit Zufallszahlen. Betrachten Sie nur das obere rechte Viertel. Lassen Sie die Anzahl der "Tropfen" vom Benutzer eingeben und geben Sie den erzielten Näherungswert aus. Berechnen Sie die Abweichung von der in Java definierten Konstante `Math.PI`.

E. Array-Aufgaben

E.1. Array erzeugen *

Schreiben Sie ein Java-Programm, das ein Array mit `n` ganzen Zahlen erzeugt. Die Anzahl der Elemente `n` soll vom Benutzer eingegeben werden. Das Array soll folgenden Inhalt haben:

n	n-1	...	4	3	2	1

Programmieren Sie auch eine Funktion `void printArray(int[] array)`, die das Feld auf der Konsole ausgibt. Diese Funktion soll Integer-Arrays beliebiger Länge drucken können.

E.2. Suche im Array *

Erweitern Sie das Programm um eine Suchfunktion in einer Methode

```
int searchIndex(int[] a, int value)
```

Diese Methode soll das Array und den zu suchenden Wert als Parameter bekommen, und den Index des Elements liefern, das den Suchwert enthält. Denken Sie auch an den Fall, dass der Wert nicht im Array enthalten ist. Welches Ergebnis liefert die Suche dann?

Testen Sie das Programm durch evtl. wiederholte Eingabe von Suchschlüsseln.

E.3. Vertauschen von Werten *

Schreiben Sie eine Methode

```
void swap(int[] a, int i, int j)
```

die zwei Werte in einem Integer-Array vertauscht. Die Funktion wird mit einem Array und den Indizes der beiden Werte als Parameter aufgerufen. Denken Sie auch an Fehlerfälle bei ungültigen Parametern!

Beispiel (das Array sei über die Variable a deklariert):

Vorher:

4	2	10	3	-5	0	17

```
swap(a, 3, 5)
```

4	2	10	0	-5	3	17

E.4. Umkehren eines Arrays *

Implementieren Sie eine Methode

`void reverseArray(int[] a)`

die die Reihenfolge der Werte umkehrt. Benutzen Sie die Methode `swap(...)` aus der vorigen Aufgabe!

Beispiel:

4	2	10	3	-5	0	17

`reverse(a)`

17	0	-5	3	10	2	4

E.5. Kopieren eines Arrays *

Wie Sie wissen, sind Arrays durch Zuweisung von Referenzen nicht wirklich kopierbar., sondern Sie erhalten nur ein Alias. Schreiben Sie daher eine Methode

`int[] cloneArray(int[] a)`

die eine 1:1-Kopie des Arrays herstellt und als Funktionsergebnis liefert.

Überlegen Sie sich, wie man testen könnte, ob die Methode korrekt arbeitet.

E.6. Maximum/Minimum bestimmen *

Schreiben Sie zwei Methoden

`int maximum(int[] a)`

`int minimum(int[] a)`

die den größten bzw. kleinsten Wert eines Arrays ganzer Zahlen liefert. Implementieren Sie zusätzlich zwei Methoden

`int maxIndex(int[] a)`

`int minIndex(int[] a)`

die an Stelle der Werte jeweils den Index des größten bzw. kleinsten Elements liefern:

4	2	10	3	-5	0	17

`maximum(a) → 17`

`minimum(a) → -5`

`maxIndex(a) → 6`

`minIndex(a) → 4`

E.7. Selection-Sort **

Das Sortierverfahren Selection-Sort arbeitet nach folgendem Prinzip:

1. Der Index des kleinsten Wertes wird bestimmt.
2. Dieses kleinste Element wird mit dem ersten Element vertauscht (d.h. das erste Element ist nun bereits das kleinste).
3. Danach wird, beginnend mit dem 2, Element, wiederum das Minimum bestimmt und dieses Element mit dem 2. vertauscht (jetzt sind bereits zwei Elemente in der richtigen Reihenfolge)
4. Die Minimum-Suche und das Vertauschen wird so lange wiederholt, bis das vorletzte Element erreicht wurde (dieses kann dann ggfs. mit dem letzten Element vertauscht werden)
5. Danach ist das Array vollständig sortiert

Programmieren Sie das Verfahren Selection-Sort so, dass das Array aufsteigend sortiert wird.

Beispiel:

4	2	10	3	-5	0	17
-5	2	10	3	4	0	17
-5	0	10	3	4	2	17
-5	0	2	3	4	10	17
-5	0	2	3	4	10	17

...ab jetzt keine Änderung mehr, aber das Verfahren läuft noch weiter bis zum vorletzten Element. Das Ergebnis ist ein sortiertes Array:

-5	0	2	3	4	10	17

E.8. Binäre Suche **

Wenn die Elemente eines Arrays sortiert sind, kann zur Suche eines Elements die binäre Suche verwendet werden. Das Verfahren beginnt mit der Suche in der Mitte des Arrays. Ist dort der gesuchte Wert, so kann der Index als Ergebnis geliefert werden. Ist der gesuchte Wert kleiner als das mittlere Element, so wird die Suche in der Mitte des linken Teils fortgesetzt. Ist der gesuchte Wert größer als das mittlere Element, geht die Suche im rechten Teil weiter.

In jedem Schritt wird so die Größe des zu durchsuchenden Bereichs halbiert. Daher ist das Verfahren im Gegensatz zur linearen Suche, die für n Elemente maximal n Schritte benötigt, bereits nach höchstens $\log_2(n)$ Schritten fertig.

Implementieren Sie das Verfahren in einer Methode

```
int binarySearchIndex(int[] a, int value)
```

die den Index des gesuchten Wertes `key` im Array `a` liefert. Falls `value` nicht in `a` vorkommt, soll als Fehleranzeige der ungültige Index -1 ausgegeben werden.

E.9. Lottozahlen **

Schreiben Sie ein Java-Programm, das 6 Lottozahlen aus dem Bereich [1, 49] ausgibt. Die Zahlen sollen natürlich zufällig gewählt werden und keine der 6 Zahlen darf doppelt vorkommen.

Tipp: Benutzen Sie ein Array mit 49 Einträgen und "mischen" Sie die Zahlen!

E.10. Matrizen *

Schreiben Sie eine Java-Methode

```
double[][] createMatrix(int lines, int columns)
```

die eine Matrix mit entsprechend vielen Zeilen und Spalten erzeugt. Die Matrix soll Zufallswerte aus dem Intervall [-100, 100] enthalten. Zur Ausgabe der Daten wird eine Methode

```
void printMatrix(double[][] m)
```

benötigt. Diese Methode soll die Daten zeilenweise ausgeben.

E.11. Matrizen-Operationen **

Ergänzen Sie in der Klasse aus der vorigen Aufgabe folgende Matrizen-Operationen. Die Operationen sollen die Einaben nicht verändern, sondern immer eine neue Matrix mit dem Ergebnis liefern. Prüfen Sie bei der Addition und der Multiplikation, ob die Dimensionen der Matrizen zueinander passen. Geben Sie anderenfalls eine Fehlermeldung aus!

Multiplikation einer Matrix mit Skalar *

```
double[][] multMatrix(double[][] m, double x)
```

Addition zweier Matrizen *

```
double[][] addMatrix(double[][] a, double[][] b)
```

Bedingung: Beide Matrizen müssen die gleiche Zeilen- und Spaltenzahl haben.

Multiplikation zweier Matrizen **

```
double[][] multMatrix(double[][] a, double[][] b)
```

Bedingung: Wenn `a` die Dimension `m × n` hat, muss `b` die Dimension `n × p` haben. Das Ergebnis `c` hat die Dimension `m × p`. Die einzelnen Komponenten von c bilden sich gemäß der Formel

$$c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \ldots\ldots + a_{in} \cdot b_{mj} \text{ mit } 1 \leq i \leq m \text{ und } 1 \leq j \leq p$$

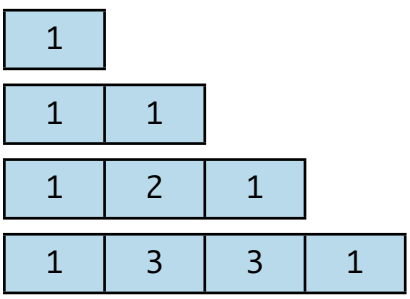
E.12. Pascal'sches Dreieck **

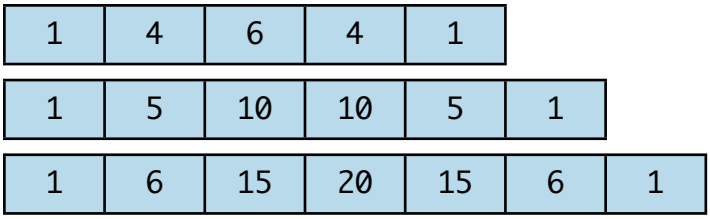
Das so genannte "Pascal'sche Dreieck" ist rekursiv definiert: Jede Zeile `z≥1` enthält genau `z` ganze Zahlen. Die erste und die letzte Zahl einer Zeile ist immer 1. Die weiteren Zahlen (ab Zeile 3) ergeben sich aus der Summe der beiden Zahlen, die in der Zeile `z-1` über dem gesuchten Wert stehen:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
...
```

Beispiel: Die 3. Zahl der 6. Zeile (10) ergibt sich aus der Summe der 2. (4) und 3. Zahl (6) der 5. Zeile.

Schreiben Sie ein Programm, das die ersten `n` Zeilen (`n` vom Benutzer wählbar) des Pascal'schen Dreiecks in einem 2-dimensionalen Integer-Array berechnet und ausgibt. Die einzelnen Zeilen sind dabei unterschiedlich lang und wie folgt anzuordnen:





E.13. Sieb des Eratosthenes **

Das Sieb des Eratosthenes ist ein Verfahren zur Bestimmung von Primzahlen. Das Verfahren berechnet die Primzahlen bis zu einer wahlfreien Obergrenze `n`:

1. Schreibe die Zahlen von `1` bis `n` in einer Liste auf
2. Streiche die `1`
3. Für jede Zahl `x` von `2` bis `n`: Streiche alle Vielfachen von `x` aus der Liste. Beginne mit dem Streichen bei `2x`
4. Alle Zahlen, die nach dem Streichen übrig bleiben, sind Primzahlen.

Lassen Sie die Obergrenze `n` vom Benutzer eingeben. Führen Sie danach das Verfahren durch und geben Sie alle gefundenen Primzahlen aus!

Leichte Beschleunigung des Verfahrens: Sie können mit dem Streichen aufhören, wenn `x` größer als die Quadratwurzel von `n` ist.

F. Rekursive Programme

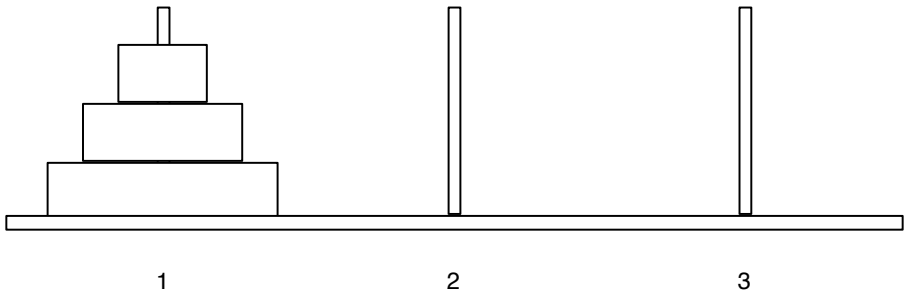
F.1. Binomialkoeffizienten berechnen **

Aus der Aufgabe mit dem Pascal'schen Dreieck ergibt sich die rekursive Berechnungsvorschrift für Binomialkoeffizienten (n>=0, 0<=k<=n):

$$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{cases} 1 & wenn\, n = 0 \\ 1 & wenn\, k = 0 \vee k = n \\ \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} + \begin{pmatrix} n-1 \\ k \end{pmatrix} & sonst \end{cases}$$

Schreiben Sie eine rekursive Java-Methode `int binomial(int n, int k)`, die Binomialkoeffizienten berechnet.

F.2. Türme von Hanoi ***



Schreiben Sie ein Java-Programm, das das Türme-von-Hanoi-Problem mit einer rekursiven Methode löst. Das Problem besteht darin, einen Stapel mit `n` Scheiben von Turm 1 nach Turm 3 zu bewegen. Dabei darf immer nur die oberste Scheibe eines Stapels bewegt werden, und es darf niemals eine größere Scheibe auf eine kleinere gelegt werden. Das Programm soll nun die kürzeste Folge von Zügen ausgeben, die das Problem für eine beliebige Scheibenzahl `n` löst.

Das Beispiel zeigt die Ausgangsposition für `n=3`. Die optimale Zugfolge zur Lösung wäre:

`1→3, 1→2, 3→2, 1→3, 2→1, 2→3, 1→3.`

Dabei bedeutet z.B. `1→3`: Nimm die oberste Scheibe vom Turm 1 und lege sie auf Turm 3.

G. Objektbasierte Aufgaben

G.1. Wortliste ***

Erweitern Sie die Klasse `WordList` aus der Vorlesung! Neue Anforderungen:

Die Liste soll alphabetisch sortiert werden können. Verwenden Sie das Verfahren Selection-Sort. Achten Sie dabei darauf, dass auch "Lücken" in der Liste vorkommen können. Nach dem Sortieren sollen alle Wort am Anfang des Arrays stehen und die freien Einträge am Ende.

Ändern Sie die Einfüge-Methode so, dass keine doppelten Einträge mehr vorkommen können.

Fügen Sie weitere Methoden hinzu, die ermitteln...

- ... wie groß die Kapazität der Wortliste ist
- ... wie viele Worte bereits in der Liste stehen
- ... wie viele Worte noch hinzu gefügt werden können

G.2. Wörterbuch ***

Schreiben Sie eine Klasse `Dictionary`, die die Übersetzung von Worten ermöglicht, z.B. von Deutsch nach Englisch. Benutzen Sie für die einzelnen Einträge im Wörterbuch eine eigene Klasse `DictionaryEntry`, die die Wortpaare speichert. Überlegen Sie, welche öffentlichen Methoden sinnvoll sind und implementieren Sie mindestens folgende Funktionalität:

- Hinzufügen eines Wortpaares
- Übersetzen Deutsch - Englisch
- Übersetzen Englisch - Deutsch
- Ausgabe des Wörterbuchs
- Sortieren nach den Deutschen Worten
- Sortieren nach den Englischen Worten