Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE16-Rekursion
(Stand 09.12.2011)

Aufgabe 1:

Implementieren Sie in Java ein Programm, das solange einzelne Zeichen vom Terminal einliest, bis ein `#`-Zeichen eingegeben wird, und anschließend die eingegebenen Zeichen in umgekehrter Reihenfolge wieder auf den Bildschirm ausgibt (Achtung: keine Arrays oder Strings verwenden!).

Beispiel:

```
Eingabe: a
         b
         c
         #
Ausgabe: cba
```

Aufgabe 2

Schreiben Sie ein Programm, das zunächst eine positive Zahl vom Terminal einliest. Anschließend soll die duale Repräsentation der Zahl auf dem Bildschirm ausgegeben werden. Entwickeln Sie eine rekursive Lösung, d.h. Sie dürfen keine Schleifen-Anweisungen (`while`, `for`, `do`, ...) benutzen.

Beispiel:

```
Eingabe: 23
Ausgabe: 10111
```

Aufgabe 3:

Schreiben Sie ein Java-Programm, dass solange Zahlen von der Tastatur einliest, bis der Nutzer eine 0 eingibt, und das diese Zahlen in umgekehrter Reihenfolge wieder auf den Bildschirm ausgibt. Eine Speicherung der Zahlen in einem Array oder einer ähnlichen Datenstruktur ist nicht erlaubt!

Aufgabe 4:

Implementieren Sie in Java eine **rekursive** Funktion

```
static double potenz(double zahl, int pot)
```

die die `pot`-te Potenz von `zahl` berechnet. Der Wert von `pot` kann auch negativ sein!. Schreiben Sie ein kleines Programm, das die Funktion testet.

Beispiele.:

potenz(2.0, 3) == 8

potenz(2.0, -3) == 0.125

Aufgabe 5:

Folgendes Programm berechnet iterativ die Quersumme einer eingegebenen positiven Zahl:

```
public class Quersumme {

  public static int quersumme(int zahl) {
    int ergebnis = 0;
    while (zahl != 0) {
      ergebnis = ergebnis + zahl % 10;
      zahl /= 10;
    }
    return ergebnis;
  }

  public static void main(String[] args) {
    int eingabe = IO.readInt("Zahl (>=0): ");
    IO.println(quersumme(eingabe));
  }

}
```

Formen Sie das Programm um in ein gleichwertiges rekursives Programm (`while`, `for`, `do` sind nicht erlaubt!)!

Aufgabe 6:

Folgendes Programm berechnet iterativ die Spiegelzahl einer eingegebenen Zahl:

```
public class Spiegelzahl {

  public static int reverse(int zahl) {
    int ergebnis = 0;
    while (zahl != 0) {
      ergebnis = ergebnis * 10 + zahl % 10;
      zahl /= 10;
    }
    return ergebnis;
  }

  public static void main(String[] args) {
    int eingabe = IO.readInt("Zahl (>=0): ");
    IO.println(reverse(eingabe));
  }
```

```
}
```

Formen Sie das Programm um in ein gleichwertiges rekursives Programm (`while`, `for`, `do` sind nicht erlaubt!)!

Aufgabe 7:

Beim folgenden Java-Programm handelt es sich um einen rekursiven Primzahltest:

```
public class PrimzahltestRekursiv {
  public static boolean test(int teiler, int zahl) {
    if (zahl % teiler == 0) {
      return false;
    } else if (teiler > zahl/teiler) {
      return true;
    } else {
      return test(teiler + 2, zahl);
    }
  }

  public static void main(String[] args) {
    int eingabe = IO.readInt("Zahl (> 3): ");

    if ((eingabe % 2 != 0) && test(3, eingabe)) {
      IO.println(eingabe + " ist Primzahl");
    } else {
      IO.println(eingabe + " ist keine Primzahl");
    }
  }
}
```

Formen Sie das Programm um in ein gleichwertiges iteratives Java-Programm!

Aufgabe 8:

Implementieren Sie folgende Funktionen rekursiv, d.h. die Verwendung von Schleifen ist verboten!

- `int getAnzahlZiffern(int zahl)` ; liefert die Anzahl an Ziffern der übergebenen Zahl; Bsp.: `getAnzahlZiffern(2542) -> 4` bzw. `getAnzahlZiffern(-342) -> 3`
- `int getZiffernWert(int zahl, int stelle)` ; liefert den Wert der Ziffer der übergebenen Zahl an der Stelle `stelle`; die Stellen werden dabei von rechts nach links angegeben und beginnen bei 0; Sie können davon ausgehen, dass gilt: 0 <= stelle < getAnzahlZiffern(zahl); Bsp.: `getZiffernWert(27381, 3) -> 7` bzw. `getZiffernWert(-27381, 0) -> 1`
- `int ersetzeZiffer(int zahl, int stelle, int wert)` ; ersetzt die Ziffer der übergebenen Zahl an der Stelle `stelle` durch den übergebenen

`wert` und liefert die neue Zahl; die Stellen werden dabei von rechts nach links angegeben und beginnen bei 0; Sie können davon ausgehen, dass gilt: zahl >= 0 und 0 <= stelle < getAnzahlZiffern(zahl) und 0 <= wert <= 9; Bsp.: `ersetzeZiffer(24135, 3, 7) -> 27135` bzw. `ersetzeZiffer(12345, 0, 6) -> 12346`

Aufgabe 9:

Das aus der Mathematik bekannte Pascalsche Dreieck hat folgende Gestalt

```
0                          1
1                      1       1
2                  1       2       1
3              1       3       3       1
4          1       4       6       4       1
5      1       5      10      10       5       1
6      .   .   .   .   .   .   .   .   .   .
```

a) Ermitteln Sie das **rekursive** Bildungsgesetz für das Pascalsche Dreieck und schreiben Sie eine rekursive Java-Funktion `int pascal(int zeile, int spalte)`, die den Wert des Pascalschen Dreieck in der Zeile „`zeile`“ und der Spalte „`spalte`“ rekursiv berechnet (`while`, `for`, `do` sind nicht erlaubt!)!

b) Schreiben Sie ein Java-Programm, das das Pascalsche Dreieck auf den Bildschirm ausgibt (die ersten 10 Zeilen).

Aufgabe 10:

Entwickeln Sie ein iteratives Java-Programm zur Lösung des Problems "Türme von Hanoi", d.h. der Einsatz von Rekursion ist nicht erlaubt.

Aufgabe 11:

Implementieren Sie eine Funktion mit folgender Signatur

```
static boolean zifferEnthalten(int zahl, int ziffer)
```

Als erster Parameter wird eine Zahl (beliebiger int-Wert) übergeben, als zweiter Parameter eine Ziffer (int-Wert zwischen 0 und 9). Die Funktion soll überprüfen, ob die Ziffer in der Zahl vorkommt. In diesem Fall (und nur dann) soll die Funktion den Wert `true` liefert. Ansonsten soll sie den Wert `false` liefern.

Beispiele:
```
zifferEnthalten(4567, 6) -> true
zifferEnthalten(-3356, 8) -> false
```

Aufgabe (a): Implementieren Sie die Funktion iterativ, d.h. durch Nutzung von Schleifen.

Aufgabe (b): Implementieren Sie die Funktion rekursiv, d.h. ohne Nutzung von Schleifen.

Aufgabe 12:

Schreiben Sie ein Java-Programm, das den Benutzer zunächst auffordert, eine Zahl größer gleich 2 einzugeben und das, sobald der Benutzer eine entsprechende Zahl eingegeben hat, einen entsprechend großen Pfeil der folgenden Form auf den Bildschirm ausgibt:

Zahl = 2:

```
\
 \
 /
/
```

Zahl = 4:

```
\
 \
  \
   \
   /
  /
 /
/
```

Teilaufgabe (a): Sie sollen Wiederholungsanweisungen benutzen!

Teilaufgabe (b): Sie dürfen keine Wiederholungsanweisungen benutzen, sondern sollen stattdessen Rekursion einsetzen.

Aufgabe 13:

Kehrt man bei einer Natürlichen Zahl die Reihenfolge der Ziffern um, so erhält man ihre Spiegelzahl. Eine Zahl heißt *Palindrom*, wenn sie mit ihrer Spiegelzahl übereinstimmt (Bsp.: 15851 oder 146641).

Schreiben Sie eine **rekursive** Funktion

```
static boolean istPalindrom(int zahl)
```

die eine Zahl als Parameter übergeben bekommt und abhängig davon, ob es sich bei der Zahl um ein Palindrom handelt, true oder false liefern soll. Sie können davon ausgehen, dass der übergebene Parameterwert > 0 ist und die Ziffer 0 im Parameterwert nicht auftritt. Wenn Sie Hilfsfunktionen definieren, dürfen auch diese keine Schleifen besitzen.

Aufgabe 14:

Die Zahl „Vier“ ist eine bedeutsame Zahl (4 Himmelsrichtungen, 4 Jahreszeiten, …). Und es ist möglich, ausgehend von der Zahl vier jede andere natürliche Zahl durch geeignete Operationen zu erzeugen. Folgende Operationen sind dafür geeignet:

- Man fügt am Ende die Ziffer 4 hinzu
- Man fügt am Ende die Ziffer 0 hinzu
- Man teilt durch 2 (wenn die Zahl gerade ist).

Schreiben Sie ein Java-Programm, bei dem der Benutzer eine natürliche Zahl eingibt und das dann die entsprechenden Zwischenzahlen auf dem Weg zu Erzeugung der Zahl gemäß den obigen Regeln ausgibt.

Beispiel: Eingabe = 2524

```
4
2
1
10
5
50
504
252
2524
```

Tipp: Gehen Sie den umgekehrten Weg ausgehend von der zu erzeugenden Zahl hin zur Zahl vier.

Aufgabe 15:

Schreiben Sie ein Programm, bei dem der Nutzer zunächst aufgefordert wird, eine positive Zahl einzugeben. Das Programm soll anschließend entsprechend viele * auf den Bildschirm ausgeben. Achtung: Sie dürfen keine Schleifen verwenden!

Beispiel:

Eingabe = 8

Ausgabe: `********`

Aufgabe 16:

Schreiben Sie ein Java-Programm, das die Anzahl der Überträge beim Addieren zweier Zahlen berechnet.

Algorithmus: Zunächst sollen vom Benutzer zwei nicht negative Zahlen eingelesen werden. Anschließend soll die Anzahl an Überträgen beim ziffernweisen Addieren der beiden Zahlen von rechts nach links berechnet werden. Zum Schluss soll die Anzahl an Überträgen auf den Bildschirm ausgegeben werden. Achtung: Sie dürfen keine Schleifen verwenden!

Beispiel:

Zahl: 123

Zahl: 594

Anzahl an Uebertraegen: 1

Aufgabe 17:

Denken Sie sich eine Zahl zwischen 1 und 100 aus. Der Computer soll sie in möglichst wenigen Schritten erraten. Implementieren Sie dazu das Halbierungsverfahren. D.h. der Computer fragt zunächst ab, ob sich die Zahl zwischen 1 und 50 befindet. Ist das der Fall, fragt er als nächstes, ob sie sich zwischen 1 und 25 befindet. Ist dies nicht der Fall, muss sie sich ja zwischen 26 und 50 befinden und der Computer fragt daher, ob sie sich zwischen 26 und 38 befindet, usw.. Achtung: Sie dürfen keine Schleifen verwenden!