

Datenstrukturen und Algorithmen

Autor: Dipl.-Math.
E. Engelhardt

Stand: 17. Mai 2009

Inhaltsverzeichnis

- ◆ Elementare Datentypen
- ◆ Strings / StringBuffer
- ◆ große Zahlen
- ◆ Datumformate
- ◆ Arrays

Elementare (einfache) Datentypen (1/4)

◆ ganzzahlige Datentypen

- byte 1 Byte -128 ... +127
- short 2 Byte $-2^{15} .. +2^{15} - 1$
- int 4 Byte $-2^{31} .. +2^{31} - 1$
- long 8 Byte $-2^{63} .. +2^{63} - 1$
- char 2 Byte (Unicode)

Objekt

- Byte
- Short
- Integer
- Long
- Character

◆ Gleitkommazahlen

- float 4 Byte 7 Stellen
- double 8 Byte 15 Stellen

- Float
- Double

◆ Wahrheitswerte

- boolean 1 Bit (true oder false) Boolean

Elementare (einfache) Datentypen (2/4)

- ◆ Elementare Datentypen sind keine Objekte!
- ◆ Die Länge ist plattformunabhängig immer gleich
- ◆ Die ganzen Zahlen sind immer vorzeichenbehaftet
 - byte 8 Bit - 128 ... +127
 - short 16 Bit - 32.768 .. +32.767
 - int 32 Bit - 2.147.483.648 .. +2.147.483.647
 - long 64 Bit - 92.23.372.036.854.775.808 ..
+
92.23.372.036.854.775.807
- ◆ Integer-Literale sind vom Typ „**int**“
 - long-Werte werden durch nachgestelltes „l“ oder „L“ gekennzeichnet (z.B. 1234566L)
 - hexadezimal: 0xbadFace, 0xFF
 - oktal: 0765, 0100

Elementare (einfache) Datentypen (3/4)

- ◆ Der Typ „char“ dient zum Speichern eines einzelnen Unicode-Zeichens
- ◆ Es können ohne Konvertierung ganze Zahlen von 0 bis 65.535 zugewiesen werden (z.B.: `char c1= 97; // `a``)
- ◆ char-Literale werden durch einfache Anführungszeichen begrenzt (`'a'`)!
- ◆ Escape-Sequenzen:
 - `\b` Rückschritt `\"` doppelter Anführungsstrich
 - `\t` Tabulator `'` einfacher Anführungsstrich
 - `\\` Backslash `\n` Zeilenvorschub
 - `\f` Seitenvorschub `\ooo` oktale Konstante
 - `\r` Zeilenrücklauf `\uxxxx` hexadezimale Konstante
- Beispiele: (`'a'`, `'\101'`, `'\u0041'`, `'\n'`)

Elementare (einfache) Datentypen (4/4)

- ◆ float-Werte müssen durch nachgestelltes „f“ bzw. „F“ gekennzeichnet werden (z.B.: `float f1= 123.45f;`)
- ◆ Gleitpunkt-Literale können auch in Exponentenschreibweise notiert werden (z.B.: `0.11e-05`)
- ◆ Der boolean-Datentyp kann nur die Werte „true“ und „false“ annehmen
- ◆ Bei den logischen Operatoren gibt es die Operatoren „UND“ und „ODER“ mit Kurzschluss-Auswertung und mit vollständiger Auswertung
 - `&`, `|` : vollständige Auswertung
 - `&&`, `||` : Kurzschluss-Auswertung
 - `^` : Exklusiv-ODER

Wahrheitstabelle der logischen Operatoren

a	b	a && b	a b	a ^ b
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

- ◆ Elementare logische Operationen
 - Konjunktion (UND, AND, &&, &)
 - Disjunktion (ODER, OR, ||, |)
 - Antivalenz, Exklusiv-ODER (XOR, ^)

Methoden der Wrapper-Klassen

- ◆ Konstruktoren:
 - `Xxx(String s)` und `Xxx(... wert)`
 - Beispiel: `Double("123.45")`, `Double(123.45)`
- ◆ `int compareTo(Xxx) :` Vergleich mit anderem Wert desselben Objekt-Typs
- ◆ `int compareTo(Xxx) :` Vergleich mit einem Objekt
- ◆ `xxx xxxValue() :` gibt entsprechenden Wert zurück
- ◆ `String xxx.toString() :` gibt String zurück
- ◆ `xxx valueOf(String s) :` gibt entsprechenden Objekt-Wert zurück
- ◆ `xxx parseXxx(String s) :` gibt String zurück
- ◆ `boolean isNaN(Xxx wert) :` gibt String zurück

Klasse „String“

- ◆ String ist kein elementarer Datentyp!
- ◆ String ist Text, also beliebige Folge von Zeichen
- ◆ Sie haben kein spezielles Endezeichen und keine Speicherung der Länge
- ◆ ein Objekt String kann ohne „new“ durch Zuweisung einer konstanten Zeichenkette erzeugt werden Beispiele:

```
str= "Hallo";  
str1= ""; // leere Zeichenkette
```
- ◆ durch den Operator „+“ wird jeder Datentyp in String umgewandelt: `System.println("" + xxx);`
- ◆ StringBuffer sind veränderbare Strings; die Größe wird dynamisch angepasst

Wichtige Methoden der Klasse „String“

- ◆ Konstruktoren:
 - `String()`
 - `String("Hallo")` u.v.a.
- ◆ `char charAt(int index) :` gibt Zeichen an der Position „index“ als „char“ zurück
- ◆ `int compareTo(Xxx) :` Vergleich mit einem Objekt oder anderem String
- ◆ `int indexOf(xxx) :` gibt Position zurück, an der das Zeichen oder die Zeichenkette steht
- ◆ `boolean equals(Xxx) :` Vergleich mit einem Objekt oder anderem String
- ◆ `int length(xxx) :` gibt Länge (Anzahl der Zeichen) zurück

Wichtige Methoden der Klasse „StringBuffer“

- ◆ `append()` : hängt eine Zeichenkette ans Ende an
- ◆ `insert()` : fügt eine Zeichenkette an einer bestimmten Stelle ein
- ◆ `delete()` : löscht einen Teil des Strings
- ◆ `replace()` : ersetzt einen Teil des Strings
- ◆ `length()` : gibt Länge der momentan enthaltenen Zeichenkette
- ◆ `equals()` : es wird geprüft, ob der übergebene Verweis auf das eigene Exemplar verweist (kein Vergleich der Inhalte)

Große Zahlen

- ◆ Im Paket „java.math“ stehen die Klassen „BigInteger“ und „BigDecimal“ für beliebig große ganze Zahlen bzw. Dezimalzahlen zur Verfügung
- ◆ Bei numerischen Berechnungen können damit an Stellen, wo es auf hohe Genauigkeit ankommt diese Zahlen eingesetzt werden
- ◆ Große Ganzzahlen sind z.B. für Verschlüsselungs-verfahren von Interesse (Primzahlen, Faktorisierung)
- ◆ Die Klasse „BigDecimal“ enthält verschiedene Methoden zum Runden

Methoden für große Zahlen

- ◆ Arithmetische Operationen
 - public Bigxxx add(Bigxxx wert)
 - public Bigxxx divide(Bigxxx wert)
 - public Bigxxx multiply(Bigxxx wert)
 - public Bigxxx subtract(Bigxxx wert)
 - public Bigxxx max(Bigxxx wert)
 - public Bigxxx min(Bigxxx wert)
- ◆ Bit-Operationen (nur für BigInteger)
- ◆ Rundungen (nur für BigDezimal)
- ◆ Formatierungen (Nachkommastellen)
- ◆ Konvertierungen
- ◆ Vergleiche

Beispiel für große Zahlen (1)

```
import java.math.*;

public class Fibo
{
    public static void main(String args[])
    {
        BigInteger Fibo1 = new BigInteger("1");
        BigInteger Fibo2 = new BigInteger("1");
        BigInteger Hilf = new BigInteger("1");
        BigDecimal Divisor;
        BigDecimal Divident;
        BigDecimal Gold= new BigDecimal("1");

        int i, len;
```

Beispiel für große Zahlen (2)

```
if (args.length > 0)
    len = Integer.parseInt(args[0]);
else
    len = 20;

for (i= 2; i <= len; i++)
{
    Hilf = Fibon2;
    Fibon2 = Fibon2.add(Fibon1); // Fibon2= Fibon2 + Fibon1
    Fibon1 = Hilf;
    Divisor= new BigDecimal(Fibon1);
    Divident= new BigDecimal(Fibon2);
    Gold= Divident.divide(Divisor, 20, 1);
    System.out.println("Ergebnis: " + Fibon2 +
        " Goldener Schnitt: " + Gold);
}
}
```

Arrays versus Objekte

- ◆ Arrays sind in Java Objekte und werden dynamisch angelegt und am Ende ihrer Verwendung automatisch vom Garbage Collector entfernt
- ◆ Array-Typen sind Verweise
- ◆ Array-Elemente werden automatisch initialisiert
- ◆ Arrays haben keine Konstruktoren
- ◆ Es können keine Unterklassen gebildet werden und es können keine Methoden überschrieben werden
- ◆ Arrays sind in Java eindimensional, in denen die Komponenten wieder Arrays sein können
- ◆ Mehrdimensionale Arrays sind in Java geschachtelte Arrays (unterschiedliche Längen pro Komponente)

Deklaration von Arrays

- ◆ Arrays sind in Java semidynamisch, d.h. die Länge der Komponenten kann im Programm ermittelt werden
- ◆ Wenn ein Array angelegt wird, kann die Länge nicht mehr geändert werden
- ◆ Objekte werden dynamisch angelegt und am Ende ihrer Verwendung automatisch vom Garbage Collector entfernt
- ◆ Bei der Deklaration wird der Typ und die Schachtelungstiefe angegeben (durch eckige Klammern)
- ◆ `int field[][];`
- ◆ Anzahl der Elemente wird bei der Initialisierung festgelegt (kann auch direkt bei der Deklaration erfolgen)

Initialisierung von Arrays

- ◆ Bei der Deklaration (ohne direkte Initialisierung) wird noch kein Speicherplatz für das Array belegt!
- ◆ Mit der Methode „**fill**“ kann das Array mit einem Wert gefüllt werden
- ◆ Die eckigen Klammern können nach dem Typ oder auch nach dem Variablen-Namen stehen
- ◆ Beispiel für direkte Initialisierung: `int pascal[][] = {{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}};`
- ◆ Bei der Initialisierung mittels „**new**“ muss die Anzahl der Verweistypen stimmen und es muss die Anzahl der Komponenten/Elemente in der Reihenfolge der eckigen Klammern ohne Lücken angegeben sein. Die Anzahl kann in den letzten eckigen Klammern fehlen

Literaturangaben

- [1] Günter Born: *HTML 4 Kompendium*, Markt&Technik Buch- und Software-Verlag GmbH, ISBN 3-8272-5354-3
- [2] Peter A. Henning: *Taschenbuch Multimedia*, Fachbuchverlag Leipzig im Carl Hanser Verlag, ISBN 3-446-21274-4
- [3] Thomas Kobert: *Das Einsteigerseminar HTML 4*, verlag moderne industrie Buch AG & Co. KG, Landsberg ISBN 3-8266-7150-3
- [4] Stefan Münz: *SelfHTML*, <http://www.teamone.de/selfhtml/>
- [5] *Internet HTML-Seiten*, HERDT-Verlag für Bildungsmedien GmbH, Nackenheim