Kapitel 05

Datentypen

Fachgebiet Knowledge Engineering
Prof. Dr. Johannes Fürnkranz

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhalt des 5. Kapitels
Datentypen

5.1 Einleitung

5.2 Eingebaute Datentypen

- Übersicht
- Die Datentypen char, float und double
- Standardwerte
- Operatoren
- Konversion / Type-Cast
- Datentyp von Literaten
- Ausdrücke

5.1 Einleitung

Grundsätzliche Unterscheidung zwischen zwei Arten von Datentypen:

- Eingebauten Typen
  - grundlegende (primitive) Typen - wie **`int`**, **`double`**, etc.
  - In diesen Datentypen finden alle konkreten, tatsächlichen Datenmanipulationen statt.
- Klassen (Bausteintypen)
  - Zusammenfassen von mehreren grundlegenden Datentypen möglich
  - Wieder verwendbare Bausteine zur Entwicklung größerer Programme
  - Wiederverwendung geht vor Neuentwicklung
    - Komponenten, Biblotheken
  - Einkapselung der technischen Details
  - Verwendung von Schnittstellen

5.2 Eingebauten Datentypen - Übersicht

Typ	Länge (in byte)	Wertebereich	Standardwert
boolean	1	true, false	false
char	2	Alle Unicode-Zeichen	U+0000
byte	1	-128...127	0
short	2	-32768...32767	0
int	4	$-2^{31}...2^{31}-1$	0
long	8	$-2^{63}...2^{63}-1$	0
float	4	$+/-3.40282347 * 10^{38}$	0
double	8	$+/-1.7976931348623157 * 10^{308}$	0

Der Datentyp „`char`“

`char`-Literale werden zwischen einfache Hochkommas gesetzt.

- Beispiele: 'A', 'a', '+', '\n', etc.

Dient zur Abspeicherung eines einzelnen Zeichens

- die meisten Programmiersprachen verwenden ASCII-Codes
- in Java: Unicode-Zeichen, daher 2 Bytes!

Unicode Zeichen

- Form: **U+xxxx** (jedes x entspricht einer hexadezimalen Ziffern).
- Auflistung der Zeichenbereiche
  - http://de.selfhtml.org/inter/unicode.htm
  - Beispiel: U+0020 für das Leerzeichen

Die Datentypen `double` und `float`

Historisch:

- **`float`** (= floating–point-numbers)
  war in anderen Programmiersprachen (z.B. C) gedacht als *der* Datentyp für reelle Zahlen schlechthin.

  32 bits (23 Mantisse + 8 Exponent + 1 Vorzeichen)

- **`double`** (= double precision)
  verwendet mehr Bits als float zum Abspeichern reeller Zahlen und war nur für Berechnungen mit besonders kniffligen numerischen Fehlerproblemen vorgesehen.

  64 bits (53 Mantisse + 10 Exponent + 1 Vorzeichen)

→ Heute gibt es kein Speicherplatzproblem mehr, **`double`** ist heute der Standard für die Speicherung reeller Zahlen

Der Standardwert

Wenn ein Objekt eines eingebauten Datentyps bei seiner Einrichtung nicht explizit initialisiert wird, dann wird das Objekt mit dem zugehörigen Standardwert initialisiert.

```
int i;

int i = 0;
```

Beide Möglichkeiten sind äquivalent

Standardwerte für eingebaute Typen:

- Numerische Typen: Wert 0.
- Zeichentyp char: Das ”Nichtzeichen“ mit Unicode–Wert 0.
- Logiktyp boolean: Wert ”false“.

  → Da die Werte alle 0 sind (false wird hier auch als 0 angesehen), nennt man diese Standardwerte auch **Nullwerte**.

Operatoren

- Definition nach Wikipedia
  - Ein Operator ist allgemein ein *Betreiber*, *Bewirker* oder *Macher*.
  - In der Computerwelt zur Steuerung von Befehlsfolgen da.
- Arten von Operatoren
  - Relationale Operatoren
  - Arithmetische Operatoren
  - Logische Operatoren
  - Short Circuit Operatoren
  - Zuweisungsoperatoren

Der Sprachaufbau von Java – Abstraktionsebenen

Spezifikatorische Ebene	Übereinstimmung der Ergebnisse von Programm-aufrufen mit den ursprünglich gesetzten Zielen
Logische Ebene	Ergebnisse von Programmaufrufen ohne Fehler auf darunter liegenden Ebenen
Semantische Ebene	Bedeutung von Texten, die lexikalisch und syntaktisch korrekt formuliert sind
Syntaktische Ebene	Gruppierung von lexikalischen Einheiten und trennenden Elementen zu Ausdrücken
Lexikalische Ebene	Unterste Ebene, zusammengesetzt aus lexikalischen Einheiten und trennenden Elementen

Relationale Operatoren

Relationale Operatoren geben einen **`boolean`** Wert zurück:

→ **`true`** oder **`false`**

Sie arbeiten auf allen numerischen Datentypen.

Operator	Bezeichnung	Bedeutung
==	**Gleich**	**a == b** ergibt true, wenn **a** gleich **b** ist.
!=	**Ungleich**	**a != b** ergibt true, wenn **a** ungleich **b** ist.
<	**Kleiner**	**a < b** ergibt true, wenn **a** kleiner **b** ist.
<=	**Kleiner gleich**	**a <= b** ergibt true, wenn **a** kleiner oder gleich **b** ist.
>	**Größer**	**a > b** ergibt true, wenn **a** größer **b** ist.
>=	**Größer gleich**	**a >= b** ergibt true, wenn **a** größer oder gleich **b** ist.

Arithmetische Operatoren

Operator	Bezeichnung	Bedeutung
+	**Pos. Vorzeichen**	**+n** ist gleichbedeutend mit **n**
-	**Neg. Vorzeichen**	**-n** kehrt das Vorzeichen von **n** um
+	**Summe**	**a + b** ergibt die Summe von **a** und **b**
-	**Differenz**	**a - b** ergibt die Differenz von **a** und **b**
*	**Produkt**	**a * b** ergibt das Produkt aus **a** und **b**
/	**Quotient**	**a / b** ergibt den Quotienten von **a** und **b**
%	**Restwert**	**a % b** ergibt den Rest der ganzzahligen Division von **a** durch **b**. In Java läßt sich dieser Operator auch auf Fließkommazahlen anwenden.
++	**Präinkrement**	**++a** ergibt **a+1** und erhöht **a** um 1
++	**Postinkrement**	**a++** ergibt **a** und erhöht **a** um 1
--	**Prädekrement**	**--a** ergibt **a-1** und verringert **a** um 1
--	**Postinkrement**	**a--** ergibt **a** und verringert **a** um 1

Arithmetische Operatoren

Beispiel zu Pre- und Postinkrement

```
int a = 5;

int b = a++

// b hat den Wert 5
```

```
int a = 5;

int c = ++a

// c hat den Wert 6
```

Arithmetische Operatoren

- Arithmetische Operatoren erwarten numerische Operanden und liefern einen numerischen Rückgabewert.
  - Haben die Operanden unterschiedliche Typen, beispielsweise int und float, so entspricht der Ergebnistyp des Teilausdrucks dem größeren der beiden Operanden.
  - Zuvor wird der kleinere der beiden Operanden mit Hilfe einer erweiternden Konvertierung in den Typ des größeren konvertiert (siehe Type-Cast).
- Neben den Operatoren gibt es noch sehr viele mathematische Funktionen (java.lang.Math.sqrt(…)). java.lang wird automatisch geladen und kann daher auch weggelassen werden.
- Sonderfall **`Strings`**:
  - **`Strings`** können addiert werden (entspricht der Konkatenation)
    - Beispiel:

```
String Name = „David " + „Thomas";
```

Arithmetische Operatoren

- Sonderfall **`chars`**:
  - Zeichen vom Typ char können addiert bzw. subtrahiert werden (entspricht der Addition/Subtraktion der Zeichen-Codes)
    - Beispiel:

```
char c = 'c';
c = (char)(c-'a'+'A');
```

<< RICHTIG

FALSCH >>

```
char c = 'c';
c = c-'a'+'A';
```

    - von 'c' wird der Code des Kleinbuchstaben 'a' abgezogen und der Code von 'A' addiert → man erhält den Code von 'C',
    - Arithmetische Operationen sind für **`char`** eigentlich gar nicht definiert. Im obigen Ausdruck werden die drei **`char`**–Werte (eine Variable, zwei Literale) daher implizit zu int konvertiert (siehe Type-Cast).
    - Das Ergebnis solcher arithmetischen Operationen auf **`int`**–Werten ist generell wieder vom Typ **`int`**.
    - Um das Ergebnis in einer **`char`**–Variablen abzuspeichern, muss es daher explizit nach **`char`** konvertiert werden.

Logische Operatoren

Operator	Bezeichnung	Bedeutung
!	Logisches NICHT	**!a** ergibt false, wenn **a** wahr ist, und true, wenn **a** falsch ist.
&&	UND mit Short-Circuit-Evaluation	**a && b** ergibt true, wenn sowohl **a** als auch **b** wahr sind. Ist a bereits falsch, so wird false zurückgegeben und b nicht mehr ausgewertet.
||	ODER mit Short-Circuit-Evaluation	**a || b** ergibt true, wenn mindestens einer der beiden Ausdrücke **a** oder **b** wahr ist. Ist bereits **a** wahr, so wird true zurückgegeben und **b** nicht mehr ausgewertet.
&	UND ohne Short-Circuit-Evaluation	**a & b** ergibt true, wenn sowohl **a** als auch **b** wahr sind. Beide Teilausdrücke werden ausgewertet.
|	ODER ohne Short-Circuit-Evaluation	**a | b** ergibt true, wenn mindestens einer der beiden Ausdrücke **a** oder **b** wahr ist. Beide Teilausdrücke werden ausgewertet.
^	Exklusiv-Oder	**a ^ b** ergibt true, wenn beide Ausdrücke einen unterschiedlichen Wahrheitswert haben.

Short Circuit Operatoren

- In einigen Fällen ist es oft schon klar, was das Ergebnis des Ausdrucks ist, ohne daß der gesamte Ausdruck ausgewertet werden muß.
  Beispiel:
  - `a || b` Wenn a bereits true ist, dann muß man b nicht mehr betrachten, da der Ausdruck a v b auf jeden Fall wahr sein wird.
- Das kann zur Verbesserung der Laufzeit ausgenutzt werden
  - in vielen Fällen wird das aber nicht gewünscht! z.B. wenn die beiden logischen Ausdrücke Seiteneffekte haben, die für die weitere Durchführung des Programms wichtig sind.
  - unschöne Programmierung!

Zuweisungsoperatoren

- Zuweisung eines Wertes: **„=“**
  - Beispiel: `a = b`
    - `a` erhält den Wert von `b` zugewiesen.
    - Rückgabewert: der zugewiesene Wert
- Veränderung eines Wertes: **„op=“**
  - Wobei op für einen anderen zweistelligen logischen oder arithmetischen Operator steht.
  - Beispiel: `a += b`
    - `a` erhält den Wert von `a + b` zugewiesen
  - Beispiel: `a &= b:`
    - `a` erhält den Wert von `a & b` zugewiesen
  - Rückgabewert: der je zugewiesene Wert

Zuweisungsoperatoren

- Anmerkung:
  - In allen Beispielen auf der vorhergehenden Folie kann b selbstverständlich ein beliebiger Ausdruck sein!
  - Beispiel:

```
int a = 0;
int[] test = {1,2,3,4};

a -= test.length * 2;
```

    - vermindert **`a`** um die doppelte Länge des Arrays **`test`**
    - Ergebnis ist -8

Konversion (Type-Cast) zwischen eingebauten Typen

- Objekte unterschiedlicher numerischer eingebauter Typen können in Zuweisungen, Vergleichen und arithmetischen Operationen direkt miteinander kombiniert werden.
  - Beispiel:

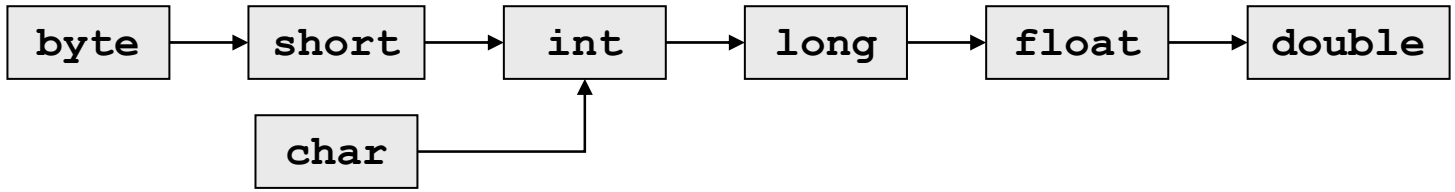
```
int i = 1;
double d = 3.14;

double x = i;       // 1.0
double y = i + d;   // 4.14

if ( i < d ) return trúe;
```

- Der eingebaute Zeichentyp **`char`** wird auf Basis des Unicodes in Java dargestellt. Jeder Unicode repräsentiert einen Zahlenwert, mit dem direkt gerechnet werden kann.
  - Beispiel:

```
char c = 'a';
int i = c + 1;
```

Automatisches Type-Cast

Automatisch konvertiert werden:



- alle aus dieser unmittelbaren Beziehung sich ergebenden transitiven Konversionen (z.B. **`char`** → **`double`**)

Anmerkung zur Konversion int → float:

- Die Konversion **`int`** → **`float`** ist nicht 100% sicher.
- Bei sehr großen **`int`**–Zahlen (bzw. long-Zahlen) ändert sich der Wert ein wenig bei der Konversion nach **`float`**.
- **`int`**-Werte haben 4 Bytes = 32 bits zur Repräsentation während **`float`**-Werte nur eine Mantissen-Länge (Ziffernspeicher) von 23 bits haben.
- Trotz dieser kleinen Unsicherheit wird diese Konversion als sicher erachtet.

Explizites Type-Cast

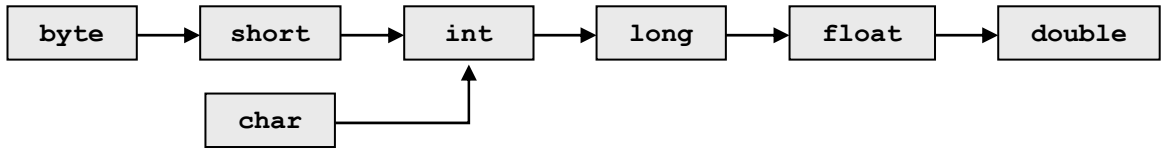
- Wenn bei einer solchen Zuweisung der Typ auf der linken Seite nicht jeden Wert darstellen kann, den der Typ auf der rechten Seite darstellen kann, ist man zur eigenen Sicherheit gezwungen, explizit hinzuschreiben, dass man die Konversion wirklich will.
  → Sonst Fehlermeldung beim Kompilieren.
- Generelle Syntax*:*
  - Der Zieltyp muss in Klammern vor den zu konvertierenden Ausdruck in Klammern hingeschrieben werden.
  - Der zu konvertierende Ausdruck muss ebenfalls in Klammern gesetzt werden, falls er nicht einfach aus einem einzelnen Literal oder Identifier besteht.

```
int i = 'a';
char c1 = (char) i;
char c2 = (char) (i+1);
```

Datentyp von Literalen

- Zeichenliterale sind vom Datentyp ”**`char`**“.
- Ganzzahlige Literale sind vom Typ ”**`int`**“.
- Nicht-ganzzahlige Literale sind vom Typ ”**`double`**“.

Zur Erinnerung:



Die folgende Zeile ergibt daher eine Fehlermeldung beim Kompilieren

```
float f = 3.14;
```

→ Man muss daher schreiben:

```
float f = (float) 3.14;
```

Ausdrücke

Ein Ausdruck besteht aus

- einem Rückgabetyp
- einen Rückgabewert und
- optionalen Seiteneffekten.

Beispiel:

```
double x = (3+6) / 2;
```

Ausdruck

Erläuterungen:

- Rückgabetyp: **`int`** (nicht **`double`**!)
- Rückgabewert: 4
- Seiteneffekt: Bei der Zuweisung an die double–Variable wird dieser int–Wert in einen double–Wert konvertiert (impliziter Cast).

Kontrollfragen zu diesem Kapitel

1. Welches Zeichen steht hinter dem Unicode U+003F ?
2. Was sind Nullwerte?
3. Warum kann es zu Ungenauigkeiten bei der Konversion von `int` nach `float` kommen?
4. Was ist eine Short-Circuit-Evaluation? Ist sie immer sinnvoll? Wenn nein, warum nicht?