

Informatik I

Processing Numbers

Wolfram Burgard

Motivation

- Computer bzw. Rechenmaschinen wurden ursprünglich gebaut, um schnell und zuverlässig mit Zahlen zu rechnen.
- Erste Anwendungen von Rechenmaschinen und Computern waren die Berechnung von Zahlentabellen, Codes, Buchhaltungen, ...
- Auch heute spielen numerische Berechnungen immer noch eine bedeutende Rolle.
- Nicht nur in Buchhaltungen, graphischen Oberflächen (Kreise, Ellipsen, ...) sondern auch bei der Interpretation von Daten (z.B. Bildverarbeitung, Robotik, ...) wird üblicherweise mit Zahlen gerechnet.
- Auch Java bietet Möglichkeiten, Zahlen zu repräsentieren und arithmetische Berechnungen durchzuführen.

Primitive Datentypen

- Einer der grundlegende Datentypen von Computern sind Zahlen.
- Anstatt Klassen für die Manipulation von Zahlen zur Verfügung zu stellen, bietet Java einen **direkten Zugriff auf Zahlen**.
- Verschiedene Typen von Zahlen (ganze Zahlen, ...) werden in Java auch direkt, d.h. unter direkter Verwendung der zugrunde liegenden Hardware realisiert ohne den Umweg über Klassendefinitionen zu gehen.
- Dies hat den Vorteil, dass numerische Berechnungen besonders effizient ausgeführt werden können.

Operatoren versus Methoden

- Allerdings führt der **Verzicht auf Klassen für Zahlen** dazu, dass **Berechnungen nicht mithilfe von Nachrichten** ausgeführt werden, die an Objekte gesendet werden, sondern mithilfe so genannter **Operatoren**.
- Darüber hinaus ist

`x / (y + 1)`

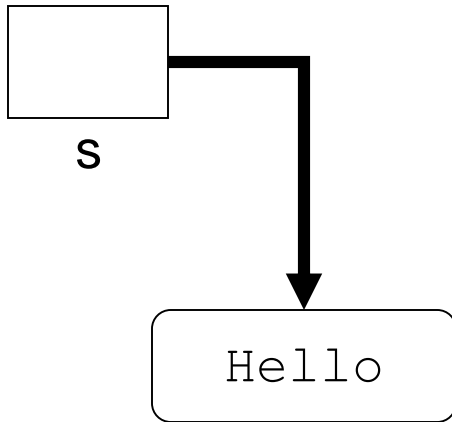
leichter lesbar als

`x.divide(y.add(1))`

Variablen versus Referenzvariablen

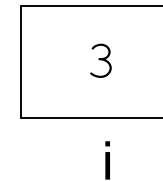
- **Referenzvariablen haben als Wert Referenzen** bzw. Bezüge auf Objekte.
- **Variablen hingegen enthalten Werte einfacher Datentypen** und werden nicht mit Objekten „assoziiert“.
- einer dieser Datentypen ist **int**, der ganze Zahlen repräsentiert.

```
String s = new String("Hello");
```



Objekt, das "Hello" repräsentiert

```
int i = 3;
```



Unterschiede zwischen Variablen und Referenzvariablen

| | Referenzvariable | Einfache Variable |
|------------------------------|--------------------------|--------------------------|
| Definiert durch | Klassendefinition | Sprache |
| Wert erzeugt durch | new | System |
| Wert initialisiert durch | Konstruktor | System |
| Variable initialisiert durch | Zuweisung einer Referenz | Zuweisung eines Wertes |
| Variable enthält | Referenz auf ein Objekt | primitiven Wert |
| Verwendet zusammen mit | Methoden | Operatoren |
| Nachrichtenempfänger | ja | nein |

Grundlegende Arithmetische Operatoren

Einige der Operatoren, die in Java im Zusammenhang mit ganzen Zahlen (`int`) benutzt werden können, sind.

+ Addition

Ergebnis ist die Summe der beiden Operanden: $5 + 3 = 8$

- Subtraktion

Ergebnis ist die Differenz der beiden Operanden $5 - 2 = 3$

* Multiplikation

Ergebnis ist das Produkt der beiden Operanden $5 * 3 = 15$

/ Division

Ganzzahlige Division ohne Rest: $5 / 3 = 1$

% Rest

Ergebnis ist der Rest bei ganzzahliger Division: $5 \% 3 = 2$

Operatoren, Operanden und Ausdrücke

- **Operatoren** korrespondieren zu Aktionen, die Werte berechnen.
- Die Objekte, auf die Operatoren angewandt werden, heißen **Operanden**.
- Operatoren zusammen mit ihren Operanden werden **Ausdrücke** genannt.
- In dem Ausdruck x / y sind x und y die Operanden und $/$ der Operator.
- Da **Ausdrücke** wiederum **Werte repräsentieren**, können Ausdrücke auch als Operanden verwendet werden.

Für die Integer-Variablen x , y und z lassen sich folgende **Ausdrücke** bilden:

$$x + y$$

$$z / x$$

$$(x - y) * (x + y)$$

Literale

- Innerhalb von **Ausdrücken** dürfen auch konkrete (Zahlen)-Werte verwendet werden.
- **Zahlenwerte, die von der Programmiersprache vorgegeben werden**, wie z.B. -1 oder 2 , heißen **Literale**.

Damit sind auch folgende Ausdrücke zulässig:

$$2 * x + y$$
$$75 / x$$
$$33 / 5 + y$$

Präzedenzregeln

- Sofern in einem Ausdruck mehr als ein Operator vorkommt, gibt es **Mehrdeutigkeiten**.
- Je nachdem, wie der Ausdruck ausgewertet wird, erhält man unterschiedliche Ergebnisse.
- Beispielsweise kann $4 * 3 + 2$ als 14 interpretiert werden, wenn man zunächst 4 mit 3 multipliziert und anschließend 2 addiert, oder als 20, wenn man zuerst 3 und 2 addiert und das Ergebnis mit 4 multipliziert.
- Java verwendet so genannte „**Präzedenzregeln**“, um solche Mehrdeutigkeiten aufzulösen.
- Dabei haben $*$, $/$ und $\%$ eine **höhere Präzedenz als** die **zweistelligen Operatoren** $+$ und $-$.
- Die **einstelligen Operatoren** $+$ und $-$ (**Vorzeichen**) wiederum haben höhere Präzedenz als $*$, $/$ und $\%$.

Präzedenzregeln und Klammern

Der Ausdruck

$$4 * 3 * -2 + 2 * -4$$

ist somit äquivalent zu

$$((4 * 3) * (-2)) + (2 * (-4))$$

Ebenso wie in der Mathematik kann man **runde Klammern** verwenden, um **Präzedenzregeln zu überschreiben**:

$$(4 * 3) + 2$$
$$4 * (3 + 2)$$

Wertzuweisungen und zusammengesetzte Wertzuweisungen

- Ausdrücke (wie die oben verwendeten) können auf der rechten Seite von Wertzuweisungen verwendet werden:

$x = y + 4;$ $y = 2 * x + 5;$

- Verschiedene Wertzuweisungen tauchen jedoch sehr häufig auf, wie z.B.

$x = x + y;$ $y = 2 * y;$

Für diese Form der Wertzuweisungen stellt Java **zusammengesetzte Wertzuweisungen** zur Verfügung:

$x += y;$ entspricht $x = x + y;$
 $y *= 2;$ entspricht $y = y * 2;$

Inkrement und Dekrement

- Die häufigsten arithmetischen Operationen sind das Addieren und das Subtrahieren von 1.
- Auch hierfür stellt Java spezielle Operatoren zur Verfügung:

`x++; ++x;`

`y--; --y;`

- Die oberen zwei Operatoren heißt **Inkrement-Operatoren**. Die unteren zwei werden **Dekrement-Operatoren** genannt.
- Diese Statements stehen für eine Wertzuweisung, durch welche der Wert der entsprechenden Variable um eins erhöht bzw. erniedrigt wird.
- Dementsprechend dürfen die Argumente dieser Operatoren weder Literale noch zusammengesetzte Ausdrücke sein.

Methoden für Integers

- Die Menge der Operatoren ist auf die Grundrechenarten eingeschränkt.
- Häufig benötigt man jedoch **weitere Funktionen**.
- Da die eingebauten Datentypen wie `int` eingeführt wurden, um bei Berechnungen den Zusatzaufwand einer Objektorientierung zu vermeiden, stellt sich nun die **Frage, wie solche Funktionen realisiert werden können**.
- Da wir **keine Objekte** mehr haben, **denen wir eine Nachricht schicken können**, müssen wir uns entsprechende Alternativen suchen.

Methoden für Integers

- In Java besteht die Lösung darin, dass solche **Methoden in den jeweiligen Klassen realisiert werden**.
- Die entsprechenden **Nachrichten werden dann nicht an ein Objekt sondern an die entsprechende Klasse gesendet**.
- Beispielsweise werden auch für Integer-Objekte einige Methoden in den vordefinierten Klasse `Integer` und `Math` zur Verfügung gestellt.
- Eine dieser Methoden ist z.B. `Math.abs`:

```
int i = -2;
```

```
int j = Math.abs(i);
```

Das Schlüsselwort `static`

- **Methoden und Variablen**, die **nicht an Instanzen einer Klasse gebunden** sind, sollten das Schlüsselwort `static` tragen.
- **Statische Methoden/Variablen** „gehören“ somit **zur Klasse** und nicht zu Instanzen einer Klasse (den Objekten).
- **Statische Methoden** haben daher **keinen Zugriff** auf die **Instanzvariablen** der Klasse.
- Statische Methoden haben **keinen Zugriff** auf **nicht-statische Methoden** der Klasse.

Anwendung des Schlüsselworts `static`

- Beispiel einer statischen Methode:

```
class StaticTest {
    StaticTest() {}
    public static int sum(int a, int b) {
        return a+b;
    }
}
```

- Da eine statische Methode nicht zu einem Objekt gehört, verwendet man den Klassennamen als Empfänger:

```
StaticTest.sum(1,2); // yields 1+2=3
```

- Eine typische Anwendung statischer Methoden und Variablen sind die mathematische Funktionen `Math.cos()`, `Math.sin()` sowie die Konstante `Math.PI`.

Auswertung von Ausdrücken

1. **Ausdrücke werden von links nach rechts unter Berücksichtigung der Präzedenzregeln und der Klammerung ausgewertet.**
2. Bei **Operatoren mit gleicher Präzedenz** wird **von links nach rechts** vorgegangen.
3. Dabei werden die **Variablen und Methodenaufrufe, sobald sie an die Reihe kommen, durch ihre jeweils aktuellen Werte ersetzt.**

Beispiele für die Ausdrucksauswertung

Gegeben:

```
int p = 2, q = 4, r = 4, w = 6, x = 2, y = 1;
```

Dies ergibt:

```
p * r % q + w / x - y  
2 * 4 % 4 + 6 / 2 - 1      ----> 2
```

```
p * x * x + w * x + -q  
2 * 2 * 2 + 6 * 2 + -4     ----> 16
```

```
(p + q * 2) + ((p - 2) * r - w)  
(2 + 4 * 2) + ((2 - 2) * 4 - 6)  ----> 4
```

Zuweisungen und Inkrementoperatoren in Ausdrücken

- Sowohl die Wertzuweisung `=` als auch die Inkrementoperatoren `++` und `--` stellen **Operatoren** dar.
- Sie dürfen daher auch in Wertzuweisungen vorkommen.
- Der Ausdruck `x = y` hat als Wert den Wert von `y`.
- Hat `x` den Wert 3, liefert `++x` als Ergebnis den Wert 4. Dabei wird `x` von 3 auf 4 erhöht.
- `x++` liefert in derselben Situation den Wert 3. Danach wird `x` um 1 erhöht.
- Zulässig sind daher

`x = y = z = 0; x = y = z++; x = z++ + --z;`

Empfehlung: Keine Zuweisungen und Operatoren mit Seiteneffekten in Ausdrücken verwenden!

Einlesen von Zahlen von der Tastatur

- Um Zahlen von der Tastatur einzulesen, benötigen wir entsprechende Methoden.
- In Java wird das durch die **Komposition von zwei Methoden** erreicht.
- Die erste liest ein **String-Objekt aus dem Eingabestrom**.
- Die zweite Methode **wandelt die Zeichen dieses String-Objektes in eine Zahl um**:

```
String s = br.readLine();  
int i = Integer.parseInt(s);
```

- Kompakter geht es mit:

```
int i = Integer.parseInt(br.readLine());
```

Mögliche Fehler

- Damit das Einlesen einer Zahl **erfolgreich** ist, muss sich die eingelesene Zeile tatsächlich auch in **eine Zahl umwandeln lassen**.

- Folgende Eingaben sind zulässig:

2

75

-1

- Bei folgenden Eingaben hingegen wird ein Fehler auftreten:

Hello

75 40

12o

Der Datentyp `long` für große ganze Zahlen

- Der Typ `int` modelliert ganze Zahlen in dem Bereich `[-2147483648, 2147483647]`
- Leider reicht dieser Wertebereich für viele Anwendungen nicht aus: Erdbevölkerung, Staatsschulden, Entfernungen im Weltall etc.
- Java stellt daher auch den Typ `long` mit dem Wertebereich `[-9223372036854775808, 9223372036854775807]` zur Verfügung, der für fast alle Anwendungen im Bereich Administration und Handel ausreicht.
- Für den Datentyp `long` gelten die gleichen Operatoren wie für `int`.
- **Long-Literale** werden durch ein abschließendes `L` gekennzeichnet.

```
long x = 2000L, y = 1000L;  
y *= x;  
x += 1500L;
```

Warum `int` und `long` und nicht nur `long`?

- Variablen vom Typ `int` benötigen nur vier Byte=32 Bit, während solche vom Typ `long` acht Byte = 64 Bit benötigen. Wenn also ein Programm sehr viele ganze Zahlen verwendet, verbraucht man bei Verwendung von `ints` nur die Hälfte an Speicherplatz.
- In der Praxis muss man die Anforderungen an die Genauigkeit sehr genau untersuchen und kann ggf. auf die speicherplatzsparenden `ints` zurückgreifen.

Gleichzeitige Verwendung mehrerer Typen: Casting

- Java erlaubt die Zuweisung eines Wertes vom Typ `int` an eine Variable vom Typ `long`.
- Dabei geht offensichtlich keine Information verloren.
- Umgekehrt ist das jedoch nicht der Fall, weil der zugewiesene Wert außerhalb des Bereichs von `int` liegen kann.
- Wenn man einer Variable vom Typ `int` einen Ausdruck vom Typ `long` zuordnen will und man sicher ist, dass keine **Bereichsüberschreitung** stattfinden kann, muss man eine spezielle Notation verwenden, die **Casting** genannt wird.
- Dabei stellt man dem Ausdruck den Typ, in den sein Wert **konvertiert** werden soll, in Klammern voraus.

Die folgenden Wertzuweisungen sind daher zulässig:

```
long x = 98;  
int i = (int) x;           // casting
```

Modellieren von Messdaten

- **Integer** sind Zahlen, die üblicherweise zum **Zählen** verwendet werden.
- **Integer** sollten daher immer dann verwendet werden, wenn der **Wertebereich einer Variablen in den ganzen Zahlen** liegt (Anzahl Studenten, die immatrikuliert sind, Anzahl der Kinder, ...).
- Insbesondere bei der Verarbeitung von **Messdaten** erhält man jedoch oft Werte, die **keine ganzen Zahlen** sind (540.3 Meter, 10.97 Sekunden, ...).
- Deshalb benötigt man in einer Programmiersprache auch **Werte mit Nachkommastellen**:

12.34

3.1415926

1.414

Fließkommazahlen

- In der Welt der Computer werden Messwerte üblicherweise durch **Fließkommazahlen** repräsentiert.
- Hierbei handelt es sich um Zahlen der Form

$$3.1479 \times 10^{15}$$

- Diese Zahl würde in Java repräsentiert durch

3.1479E15f

- Dabei sind sowohl der **Vorkommaanteil**, der **Nachkommaanteil** und der **Exponent** in der **Anzahl der Stellen begrenzt**.
- **Fließkommazahlen repräsentieren eine endliche Teilmenge der rationalen Zahlen.**

Die Datentypen `float` und `double`

- Java stellt mit `float` und `double` zwei **elementare Datentypen** mit unterschiedlicher Genauigkeit für die Repräsentation von **Fließkommazahlen** zur Verfügung.
- Der Typ `float` modelliert Fließkommazahlen mit **ungefähr siebenstelliger Genauigkeit**. Der **Absolutbetrag** der Werte kann entweder `0` sein oder im Bereich `[1.4E-45f, 3.4028235E38f]` liegen.
- Demgegenüber hat der Typ `double` eine **ungefähr fünfzehnstellige Genauigkeit**. Der **Absolutbetrag** der Werte kann entweder `0` sein oder im Bereich `[4.9E-324, 1.7976931348623157E308]` liegen.

Vergleich der Typen `float` und `int`

- Variablen vom Typ `float` benötigen ebenso wie Variablen vom Typ `int` lediglich 4 Byte = 32 Bit.
- Variablen vom Typ `float` können **größere Werte** repräsentieren als Variablen vom Typ `int`.
- Allerdings haben `floats` nur eine **beschränkte Genauigkeit**.

Beispiel:

```
float f1 = 1234567089f;  
System.out.println(f1);
```

liefert als Ausgabe

```
1.23456704E9.
```

Fließkommazahlen und Rundungsfehler

- **Fließkommazahlen** stellen nur eine **begrenzte Genauigkeit** zur Verfügung.
- Ein typisches Beispiel für mögliche **Rechenfehler** ist:

```
float x = 0.0644456f;  
float y = 0.032754f;  
float z = x * y;  
System.out.println(z);
```

- Ausgabe dieses Programmstücks ist **0.0021108512**.
- **Korrekt wäre** **0.0021108511824**.

Verwendung von `float` oder `double`

- Variablen vom Typ `float` und `double` werden ähnlich verwendet wie Variablen vom Typ `int`.
- Mit folgendem Programmstück wird die Fläche eines Kreises berechnen mit Radius `12.0` berechnet:

```
double area, radius;  
radius = 12.0;  
area = 3.14159*radius*radius;
```

Einlesen von Werten vom Typ `float` und `double`

- Das Einlesen von Werten für `double/float` ist so einfach wie für `int`.
- Java stellt ein **Klasse Double/Float** zur Verfügung, die es erlaubt, einen `double` Wert aus einem **String-Objekt** zu berechnen.

```
double d = Double.parseDouble(br.readLine());
```


Wann soll man `float` oder `double` verwenden?

- **Fließkommazahlen** werden in der Regel verwendet, wenn man **Zahlen mit Nachkommaanteil** benötigt.
- Die **Genauigkeit von `double` ist für viele Anwendungen hinreichend**.
- Allerdings gibt es **Anwendungen, für welche die Genauigkeit von `double` nicht ausreicht**.
- Ein typisches Beispiel ist das Lösen großer Gleichungssysteme.
- Probleme tauchen aber auch bei Berechnungen im Finanzbereich auf, bei denen Rundungsfehler bis zur zweiten Nachkommastelle ausgeschlossen werden müssen.

Gemischte Arithmetik

- Dieselben Gesetze, die für die **Konvertierung** zwischen `int` und `long` gelten, finden auch für `float` und `double` Anwendung.
- Allerdings kann man auch Integer-Variablen Werte von Fließkommazahlen zuordnen und umgekehrt.
- Nur wenn es mit **keinem Informationsverlust** verbunden ist, kann eine **Zuweisung direkt** erfolgen.
- Andernfalls muss man das **Casting** verwenden.

```
double x = 4.5;
int i = (int) x;
x = i;
```

- Dabei wird bei der **Konvertierung von Fließkommazahlen nach Integer-Zahlen stets der Nachkommaanteil abgeschnitten**

Zusammenfassung

- Java stellt verschiedene elementare Datentypen für das „**Verarbeiten von Zahlen**“ bereit.
- Die **Integer-Datentypen** repräsentieren ganze **Zahlen**.
- Die Datentypen `float` und `double` repräsentieren **Fließkommazahlen**.
- **Fließkommazahlen** sind eine Teilmenge der rationalen und reellen Zahlen.
- Die **Werte dieser Datentypen** werden durch **Literale** beschrieben.
- Für die Konvertierung zwischen Datentypen verwendet man das **Casting**.
- Berechnungen mit Daten vom Typ `double` und `float` können **Rundungsfehler** produzieren.
- Dadurch entstehen häufig **falsche Ausgaben und Ergebnisse**.
- **You have been warned!**