Java I – Vorlesung 2
Imperatives Programmieren

3.5.2004

Variablen -- Datentypen -- Werte
Operatoren und Ausdrücke
Kontrollstrukturen: `if`

Imperatives Programmieren

- Im Kern ist Java eine imperative Programmiersprache.
- Programme sind Schritt-für-Schritt-Sequenzen von Anweisungen.
- Ausdrücke haben Werte.
- Werte können Variablen zugewiesen werden.
- Das wesentliche Werkzeug zur Steuerung des Programmablaufs sind Kontrollstrukturen.

„Größte Zahl“-Algorithmus als Java-Programm

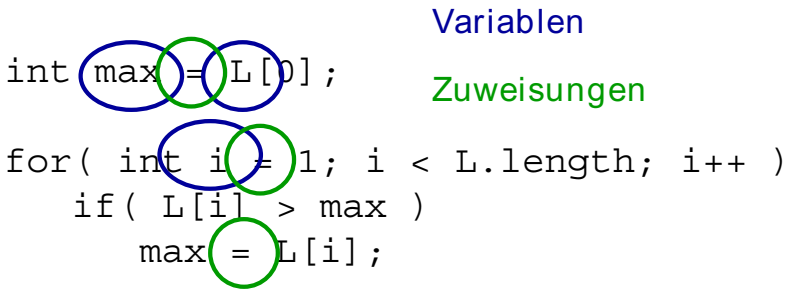
Gegeben eine Liste L von n natürlichen Zahlen; gesucht ist die größte Zahl in L.

```
int max = L[0];

for( int i = 1; i < L.length; i++ )
   if( L[i] > max )
      max = L[i];
```

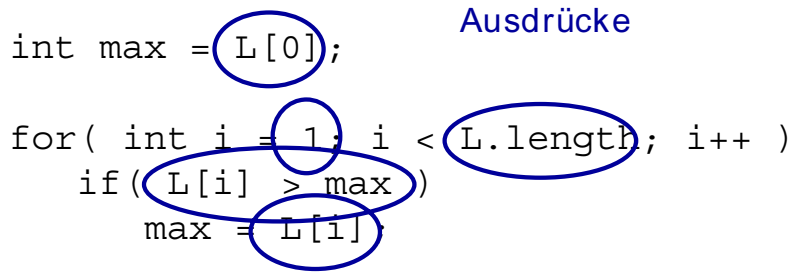

Elemente von imperativen Programmen

Gegeben eine Liste L von n natürlichen Zahlen; gesucht ist die größte Zahl in L.



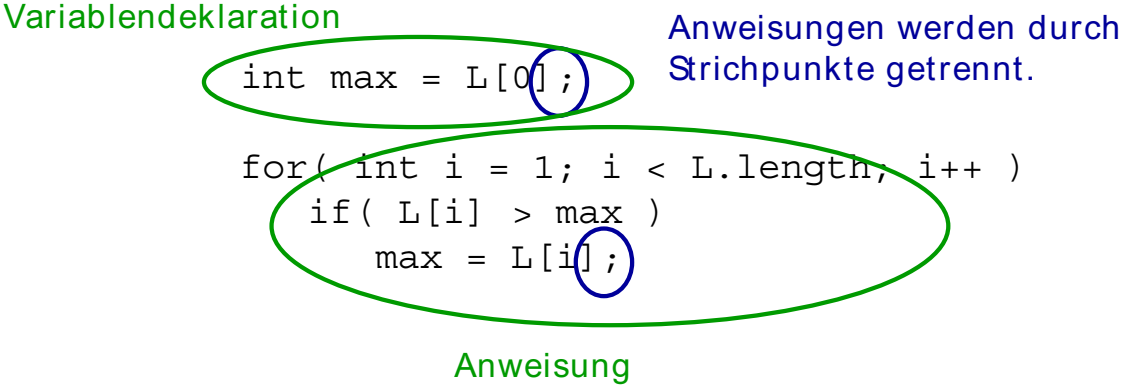
Elemente von imperativen Programmen

Gegeben eine Liste L von n natürlichen Zahlen; gesucht ist die größte Zahl in L.



Elemente von imperativen Programmen

Gegeben eine Liste L von n natürlichen Zahlen; gesucht ist die größte Zahl in L.



Elemente von imperativen Programmen

Gegeben eine Liste L von n natürlichen Zahlen; gesucht ist die größte Zahl in L.

Kontrollstrukturen
(for- Schleife; if- Bedingung)

```
int max = L[0];

for( int i = 1; i < L.length; i++ )
    if( L[i] > max )
        max = L[i];
```


Variablen, Werte und Datentypen

- Java unterscheidet Werte von verschiedenen Datentypen (starke Typisierung).
- Variablen sind Positionen im Hauptspeicher, an denen Werte gespeichert werden.
- Jede Variable hat einen eindeutigen Datentyp und kann nur Werte dieses Typs aufnehmen (statische Typisierung).

Datentypen

- Ein Datentyp repräsentiert eine Menge von möglichen Werten und gibt an, wie ein Stück Speicher interpretiert wird.
- Java unterscheidet zwei Arten von Typen:
  - primitive Datentypen
  - Referenz-Datentypen
- Heute: Primitive Typen. Klassen (nächste Woche) sind Referenz-Typen.

Primitive Datentypen: Ganze Zahlen

- Es gibt fünf Datentypen für ganze Zahlen, die sich in Wertebereich und Speicherbedarf unterscheiden.
- Die vier folgenden Typen repräsentieren Zahlen mit Vorzeichen:

`byte`	8 Bits
`short`	16 Bits
`int`	32 Bits
`long`	64 Bits

- Wertebereich: $-2^{b-1},\ldots,2^{b-1}-1$
  (b ist Anzahl von Bits)

Primitive Datentypen: Ganze Zahlen

- Der fünfte Datentyp, `char`, hat 16 Bits und kein Vorzeichen.
- Repräsentiert zunächst die Zahlen von 0 bis 65535 (also ohne Vorzeichen).
- Interpretation als Zeichen in der Unicode-Codierung, die fast alle internationalen Alphabete abdeckt.
- Java ist konsequent auf Unicode ausgerichtet: Z.B. können Variablennamen Umlaute enthalten.

Primitive Datentypen: Fließkommazahlen

- Es gibt zwei Datentypen zur Repräsentation von Zahlen mit Hinterkommastellen:

`float`	32 Bits
`double`	64 Bits

- Codierung der Werte im Speicher nach IEEE-Standard für Fließkommazahlen (Vorzeichen, Exponent, Mantisse).
- Wertebereich ca. $\pm 10^{-45} .. \pm 10^{38}$ (`float`) bzw. $\pm 10^{-324} .. \pm 10^{308}$ (`double`).
- Vorsicht Rundungsfehler!

Primitive Datentypen: Wahrheitswerte

- Der achte primitive Datentyp, `boolean`, repräsentiert die Wahrheitswerte `true` und `false`.

Ausdrücke

- Werte können nicht direkt aufgeschrieben werden. Sie werden durch Ausdrücke dargestellt.
- Es gibt vier wesentliche Arten von Ausdrücken:
  - Literale
  - Variablen
  - Ausdrücke mit Operatoren
  - Methodenaufrufe (nächste Woche)

Literale

- Literale sind Ausdrücke, aus denen der Wert direkt (ohne zu rechnen) abzulesen ist.
- Beispiele: `2`, `3.14`, `true`, `"Hallo"`
- Jedes Literal hat einen eindeutigen Datentyp.
- Hier zunächst Ganzzahl-, Fließkomma- und Boolean-Literale. String-Literale nächste Woche.

Ganzzahl-Literale

- Standardmäßig werden Zahlen wie `17`, `0`, `-238723` im Quelltext als Literale von Typ int zur Basis 10 interpretiert.
- Literale, die mit 0 anfangen (außer 0 selbst), werden als Oktalzahlen (Basis 8) interpretiert: `013` repräsentiert Wert 11.
- Literale, die mit 0x anfangen, werden hexadezimal (Basis 16) interpretiert: `0x1cA` repräsentiert 458.
- Literale, die mit `L` oder `l` enden, werden als Typ long interpretiert und dürfen größere Werte repräsentieren.

Ganzzahl-Literale: Typ char

- Literale von Typ `char` sind einzelne Zeichen in einfachen Anführungszeichen: `'a'`, `'Ä'`.
- Sonderzeichen werden mit Backslash `\` gekennzeichnet:
  - `'\n'` Zeilenumbruch
  - `'\''` einfaches Anführungszeichen
  - `'\\'` Backslash selbst
- Man kann auch direkt den Unicode-Wert angeben: `'\u102'` ist das Zeichen Ă.

Fließkomma-Literale

- Zahl-Literale wie `3.14` und `-2.6`, die einen Dezimalpunkt enthalten, haben Typ `double`.
- Fließkommaliterale dürfen Exponenten (zur Basis 10) angeben: `3.14e1` repräsentiert Wert 31.4, `-2.e-2` repräsentiert -0.02.
- Suffix `F` spezifiziert Typ `float`; Suffix `D` macht Typ `double` explizit.
- Mit Suffix oder Exponent darf man den Punkt weglassen: `0f`, `-2e-2` usw.

Andere Literale

- Die Schlüsselwörter `true` und `false` sind die beiden Literale des Datentyps `boolean`.

Ein Beispielprogramm

```
class Test {
  public static void main(String[] args) {
    System.out.println(2);
    System.out.println("Hallo");
    System.out.println(-2e-2);
  }
}
```

- Mit Literalen allein kann man noch nicht viel machen. Jetzt rechnen wir!

Operatoren

- Elementare Rechenoperatoren werden mit Operatoren dargestellt:

```
2 + 3   0.4 * 25.4 - 100   27 > -3
```

- Jeder Operator-Ausdruck hat einen eindeutigen Datentyp.
- Operatoren sind
  - arithmetische Operatoren
  - Vergleichsoperatoren
  - logische Operatoren
  - ein paar sonstige

Arithmetische Operatoren

- Die üblichen arithmetischen Operationen sind Operatoren:

```
a + b
a - b
a * b
a / b
a % b   (Rest bei ganzzahliger Division)
```

- (`a`, `b` sind andere Ausdrücke)
- Arithmetische Operatoren kombinieren Zahltypen und haben entsprechenden Zahltyp als Wert.

Vergleichs-Operatoren

- Die üblichen arithmetischen Vergleichs-Operationen sind Operatoren:

`a < b`	`a > b`	
`a <= b`	`a >= b`	(größer-gleich)
`a == b`		(ist-gleich)
`a != b`		(ist-ungleich)

- Größer, größer-gleich usw. nehmen Zahltypen als Operanden. Ist-gleich und Ist-ungleich nehmen beliebige Datentypen.
- Ergebnis ist immer ein Wert vom Typ `boolean`.

Erweiternde Typkonvertierungen

- Was passiert, wenn die beiden Operanden eines Operators verschiedene Typen haben?
- Wenn es alles Zahltypen sind, werden alle Operanden in denjenigen beteiligten Datentyp mit dem größten Wertebereich konvertiert:
  - Ganzzahlen in Ganzzahlen mit mehr Bytes
  - `float` in `double`
  - Ganzzahlen in `float` oder `double` (Vorsicht: mögliche Rundungsfehler!)

Logische Operatoren

- Logische Operatoren verknüpfen Werte von Typ `boolean` zu Werten von Typ `boolean`:

`a && b`	"und"
`a || b`	"oder"
`!a`	"nicht"

Präzedenz von Operatoren

- Operatoren können geschachtelt werden:
  `2*3+4`
- Man kann Klammern setzen, um die Schachtelung eindeutig zu machen:
  `(2*3)+4` `2*(3+4)`
- Ohne Klammern wendet Java "Punkt vor Strich" an. Die Position in der Punkt-vor-Strich-Reihenfolge heißt Präzedenz eines Operators.

Präzedenz von Operatoren

Operatoren	Assoziativität
`()  []  .`	von links
`++  --` (als Präfix)	von rechts
`++  --` (als Postfix) `-  !` (unär)	von rechts
`*  /  %`	von links
`+  -`	von links
`<<  >>` (bitweises Shift)	von links
`<  <=  >  >= instanceof`	von links
`==  !=`	von links
`&` (bitweises AND)	von links
`^` (bitweises XOR)	von links
`|` (bitweises OR)	von links
`&&` (logisches AND)	von links
`||` (logisches OR)	von links
`?:` (Konditional)	von rechts
`= += -= *= /= %= &= ^= |= <<= >>=`	von rechts

Java als Taschenrechner

```
class Test {
  public static void main(String[] args) {
    System.out.println(2+3*4);
    System.out.println(27*(9872%5) > 98*2772);
    System.out.println(-2e-2 - 103.2382);
    System.out.println(false || (3 > 2));
  }
}
```

- Immerhin können wir jetzt rechnen!

Variablen

- Jede Variable enthält Werte eines bestimmten Datentyps.
- Variablen müssen deklariert werden, damit der Compiler ihren Datentyp kennt.
- Variablen können Werte eines Ausdrucks zugewiesen bekommen.
- Variablen können ausgewertet werden, um ihren Wert in einem Ausdruck zu verwenden.

Variablen: Ein Beispiel

```
class Test {
  public static void main(String[] args) {
    int zahl;

    zahl = 2;
    zahl = zahl + 1;
    System.out.println(zahl);
  }
}
```

Zuweisung

Zuweisung

Deklaration als int-Variable

Auswertung

Auswertung

Variablennamen

- Variablennamen (allgemeiner: Bezeichner) können aus beliebigen Unicode-Zeichen bestehen.
- Das erste Zeichen eines Namens darf keine Ziffer sein, und der Name darf kein Schlüsselwort sein.
- Groß- und Kleinbuchstaben werden unterschieden.
- Beispiele:

```
x   String   abc27   üß_27Π
```


Deklaration

```
typ var1 [= init1] [, var2 [= init2], ...];
```

- Mehrere Variablen des gleichen Datentyps können auf einmal deklariert werden.
- Variablen können bei der Deklaration mit dem Wert eines Ausdrucks initialisiert werden:

```
int a = 2, _b100 = 27, meine_var;
```

Zuweisungen

```
var = expr;
```

- Der Ausdruck `expr` wird ausgewertet, dann wird der Wert in `var` gespeichert.
- Wert kann erweiternd in den Typ der Variable konvertiert werden.
- Falls der Wert von `expr` zur Compilezeit berechnet werden kann (z.B. Literal) und in den Typ von `var` passt, kann er auch verengend in den Typ konvertiert werden.

Variablen: Das Beispiel

```
class Test {
  public static void main(String[] args) {
    int zahl;

    zahl = 2;
    zahl = zahl + 1;
    System.out.println(zahl);
  }
}
```

- Damit kann man schon ziemlich viel machen!

Variablen: Ein Additionsprogramm

```
class Addieren {
  public static void main(String[] args) {
    int x = Integer.parseInt(args[0]),
        y = Integer.parseInt(args[1]);

    int summe = x + y;

    System.out.println(summe);
  }
}
```

- Der Ausdruck `Integer.parseInt(args[i])` wertet zum (i+ 1)- ten Kommandozeilen-Argument aus, als Zahl gelesen.

Zuweisungen mit Rechnen

- Zuweisungen der Form `x = x + 1`, in der eine Variable `x` nur mit einem anderen Wert kombiniert und gleich wieder zugewiesen wird, sind sehr häufig.
- Abkürzende Syntax:

```
x += expr   x -= expr
x *= expr   x /= expr   x %= expr
```

- Ausdrücke `x++`, `x--`, `++x`, `--x` erhöhen bzw. vermindern Wert der Variable um 1. Wert des Ausdrucks ist alter (bei `x++`) bzw. neue (bei `++x`) Wert von `x`.

Zuweisungen mit Rechnen

```
class Test {
  public static void main(String[] args) {

    int zahl = 2;

    zahl *= 3;
    System.out.println(zahl);

    System.out.println(zahl++);

    System.out.println(--zahl);
  }
}
```

Wert von zahl	Ausgabe
2	
6	
	6
	6
7	
	6
6	

Anweisungen

- Ein Java-Programm (genauer: jede Methode) ist aus einer Sequenz von Anweisungen aufgebaut.
- Eine Anweisung entspricht etwa einem Schritt im Algorithmus.
- Bisher kennen wir:
  - Variablendeklarationen
  - Ausdrucks-Anweisungen (v.a. Zuweisungen)
- Mehrere Anweisungen werden durch Strichpunkt-Zeichen getrennt.

Kontrollstrukturen

- Manchmal will man Anweisungen mehrfach ausführen, oder nur unter bestimmten Bedingungen, oder zwischen verschiedenen Anweisungen auswählen.
- Das ist die Funktion von Kontrollstrukturen.
- Schleifen: `for` `while` `do ... while`
- Bedingungen: `if` `switch`
- Kontrollstrukturen bauen komplexe Anweisungen auf.

`if-else`

- Syntax:

```
if ( expr )
   anweisung1
[ else
   anweisung2 ]
```

- Semantik: Falls *expr* zutrifft (zu `true` auswertet), wird $anweisung_1$ ausgeführt, sonst $anweisung_2$ (falls der `else`-Zweig vorhanden ist).
- Beispiel:

```
if ( n >= 0 )
  System.out.println("positiv");
else
  System.out.println("negativ");
```

Blöcke

- Häufig will man mehrere Anweisungen hintereinander ausführen, wenn eine Bedingung erfüllt ist.
- Eine Sequenz von Anweisungen wird mit { und } zu einem Block zusammengefasst. Der ganze Block zählt dann als eine Anweisung.
- Beispiel:

```
if ( n >= 0 )
 {
  if ( a > b )   z = a;
 }
 else System.out.println("negativ");
```


`else-if`-Ketten

- Syntax:

```
if ( expr1 )
   anweisung1
else if ( expr2 )
   anweisung2
else if ( expr3 )
   anweisung3
[...]
else
   anweisungn
```

- Ausdrücke werden in angegebener Reihenfolge bewertet, bis einer zutrifft; dann wird die entsprechende Anweisung ausgeführt.

Nützliche Ausdrücke und Anweisungen

- (i+1)-tes Kommandozeilen-Argument als String:
  ```
  args[i]
  ```
- (i+1)-tes Kommandozeilen-Argument als int:
  ```
  Integer.parseInt(args[i])
  ```
- Erstes Zeichen des (i+1)-ten Arguments:
  ```
  args[i].charAt(0)
  ```
- Wert eines Ausdrucks ausgeben:
  ```
  System.out.println(expr);
  ```


Zusammenfassung

- Ausdrücke sind Stücke Programmtext, die einen Wert haben.
- Werte haben Datentypen.
- Variablen sind spezielle Ausdrücke, die für einen bestimmten Datentyp deklariert werden und denen Werte zugewiesen werden können.
- Mit dem `if`-Statement kann man zur Laufzeit entscheiden, welche Teile eines Programms ausgeführt werden sollen.