Institut für Informatik der
Technischen Universität München

Programmieren in Java

Ein Ergänzungskurs zur
Einführung in die Informatik

**Wintersemester
1998/99**

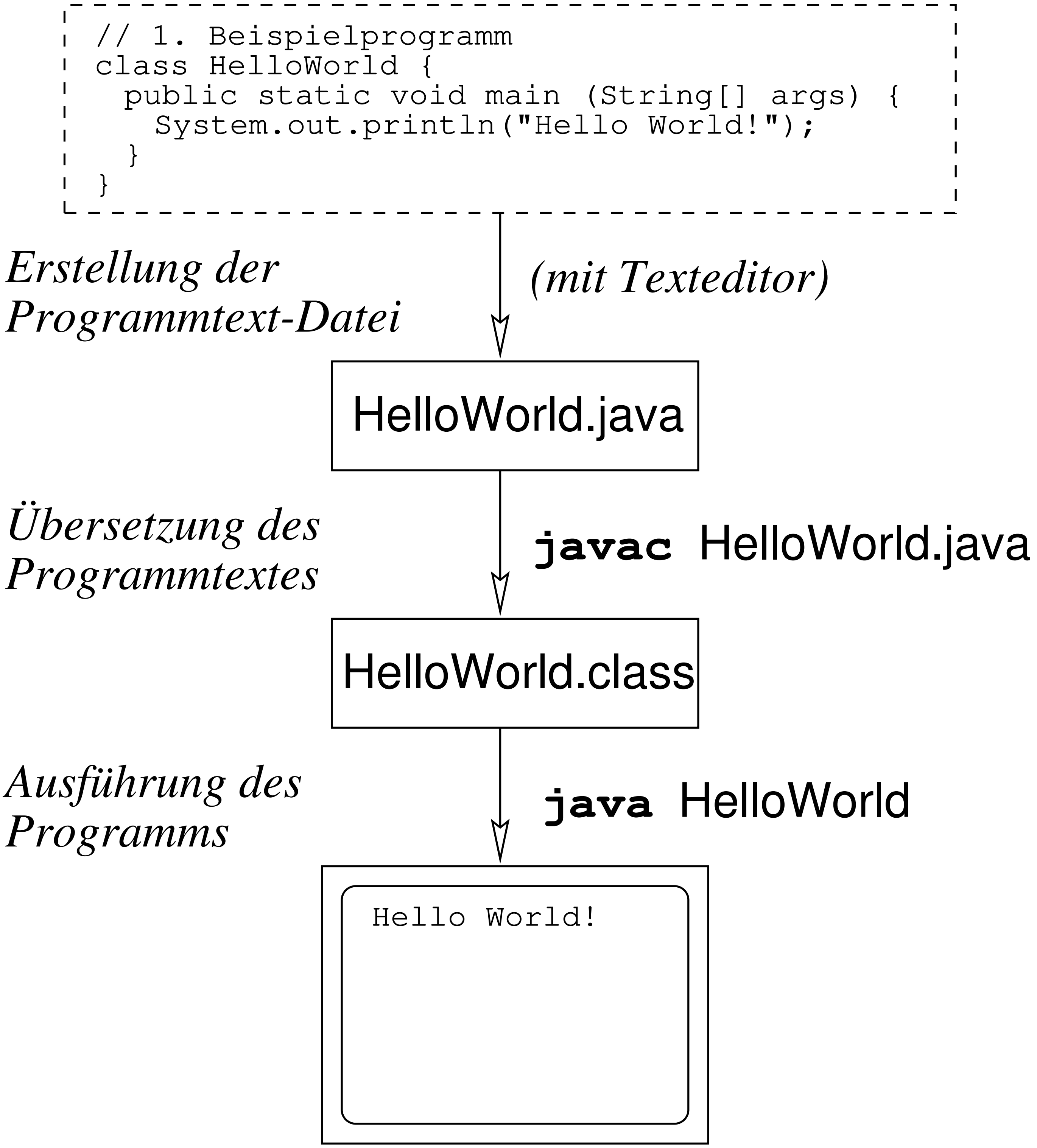
Programmieren in Java

Ein Ergänzungskurs zur
Einführung in die Informatik

Teil 1: Prozedurale Konzepte

- Übersetzung und Ausführung von Java-Programmen
- Klassen als ausführbare Programme
- Namen und Bezeichner
- Datentypen: Grundtypen und Felder
- Variablen
- Methoden
- Rekursion
- Zuweisung von Werten
- Ausdrücke und Operatoren
- Anweisungen: Fallunterscheidung und Schleifen

Übersetzung und Ausführung von Java-Programmen



Klassen

- Syntax: `class` *Klassenname* {
  Variablendeklarationen
  Methodendeklarationen
  }

- *Variablen*- und *Methodendeklarationen* in beliebiger Reihenfolge möglich

- Zunächst nur Klassenmethoden und -variablen (mit vorangestelltem Modifikator `static`)

- Klassen als Anwendungsprogramme
  → Deklaration einer Methode `main`:
  ```
  public static void main (String[] args) {
    ...
  }
  ```

Bezeichner

- Namen für Klassen, Methoden und Variablen
- 1. Zeichen muß Buchstabe sein
- Schlüsselwörter als Bezeichner *nicht* zulässig
- Beispiele für korrekte Bezeichner:
  `i`, `x123`, `EinLangerBezeichner`
- Beispiele für unzulässige Bezeichner:
  `2mal`, `class`
- Zusammengesetzte Bezeichner
  - Syntax: *Klassenname.Variablenname* bzw.
    Klassenname.Methodenname
  - Erlauben Zugriff auf Variablen/Methoden
    in anderen Klassen

Datentypen in Java

Grundtypen:

- Ganze Zahlen: `byte`, `short`, `int`, `long`
- Gleitkomma-Zahlen: `float`, `double`
- Boolesche Werte: `boolean`
- Textzeichen: `char`
- Kein Datentyp: `void` (nur bei Methoden)

Referenztypen: Datentypen für Felder und Objekte

Zeichenketten: Objekte (Datentyp `String`)

Literale (explizite Wertangaben)

Beispiele:

- Ganze Zahlen: `1`, `-5`, `0xFF` (hexadezimal)
- Gleitkomma-Zahlen: `1.0`, `1.05e2`
- Boolesche Werte: `true`, `false`
- Textzeichen: `'a'`, `97` (Unicode)
- Zeichenketten: `"Text"`, `""` (leere Zeichenkette)

Variablen

- Aufgabe: Speicherung von Werten (“Zwischenergebnissen”)
- Deklaration:
  - Syntax: *Datentyp Variablenname*;
  - Beispiel: `int zahl;`
  - *Variablenname* erlaubt Zugriff auf Variable
  - *Datentyp* bestimmt Speicherbedarf/Art des Variablenwerts
- Initialisierung:
  - Syntax: *Variablenname* = *Wert*;
  - Beispiel: `zahl = 42;`
  - Schreiben eines Anfangswerts
- Deklaration und Initialisierung in einem Schritt:
  Beispiel: `int zahl = 42;`
- Zunächst nur folgende Formen von Variablen:
  - Klassenvariablen (mit Modifikator `static`)
    Beispiel: `static int zahl = 42;`
  - lokale Variablen (innerhalb einer Methode deklariert)

(Klassen-)Methoden

Vollständige Deklaration einer Klassenmethode:
`static` *Methodenkopf* { *Methodenrumpf* }

- Methodenkopf:
  - Syntax:
    Ergebnistyp Methodenname `(` *Parameter* `)`
  - Beispiel: `int berechne(char c, long l)`
  - *Methodenname* erlaubt Aufruf der Methode
  - *Ergebnistyp*: Datentyp für Rückgabewert
    (bei Methoden ohne Rückgabewert: `void`)
  - Formale *Parameter*: Deklaration wie Variablen
    (jeweils durch Kommas getrennt)
- Methodenrumpf:
  - Festlegung der bei Aufruf der Methode auszuführenden Anweisungen
  - Deklaration lokaler Variablen
    (nur innerhalb des Methodenrumpfs gültig)
  - Formale Parameter: Zugriff wie auf lokale Variablen
  - `return`-Anweisung legt Rückgabewert fest
    (Ausführung des Methodenrumpfs wird beendet)
    Syntax: `return` *Rückgabewert*`;`

Methodenaufrufe

- Syntax:
  Variablenname `=` *Methodenname*`(` *Parameter* `);`
  Methodenname`(` *Parameter* `);`

- Beispiele:
  `zahl = berechne('a', 4711);`
  `berechne('b', 0);`

- Aktuelle *Parameter*: Übergabe von Werten

- Methodenergebnisse können verworfen werden

Rekursion

- Während Ausführung eines Methodenrumpfs erneuter Aufruf derselben Methode

- Terminierung: keine unendlichen Aufruffolgen!
  $\rightarrow$ Abbruchkriterium erforderlich

- Aufrufbaum: Darstellung von Methodenaufrufen

Beispielmethode:

```
static int fakultaet (int n) {
  if (n <= 1) return 1;
  return n * fakultaet(n-1);
}
```

```
fakultaet(4)
     |
fakultaet(3)
     |
fakultaet(2)
     |
fakultaet(1)
```

Aufruf-
reihenfolge

Zuweisungen

- Syntax: *Variablenname* `=` *Wert* ;
- Beispiel: `zahl = 42;`
- Schreiben eines Wertes in eine Variable
- *Wert*: Ergebnis der Auswertung eines Ausdrucks

Ausdrücke

- Auswertung eines Ausdrucks liefert Wert
- Mögliche Formen von Ausdrücken:
  - Literale
    (Beispiel: `zahl = 42;`)
  - Variablen
    (Beispiel: `zahl = andereZahl;`)
  - Methodenaufrufe
    (Beispiel: `zahl = berechnung();`)
  - Zusammengesetzte Ausdrücke
    (Beispiel: `zahl = andere Zahl + 42;`)

Operatoren

- arithmetische Operatoren (numerische Werte):
  `+` (Addition), `-` (Subtraktion), `*` (Multiplikation),
  `/` (Division), `%` (Divisionsrest)

- logische Operatoren (boolesche Werte):
  `!` (NOT), `&` (AND), `|` (OR), `^` (XOR),
  `&&` (nicht-striktes AND), `||` (nicht-striktes OR)

- Vergleichsoperatoren:
  `==` (gleich), `!=` (ungleich),
  `<` (kleiner), `<=` (kleiner oder gleich),
  `>` (größer), `>=` (größer oder gleich)

- Zuweisungen als Ausdrücke:
  Zuweisungsausdruck entspricht zugewiesenem Wert

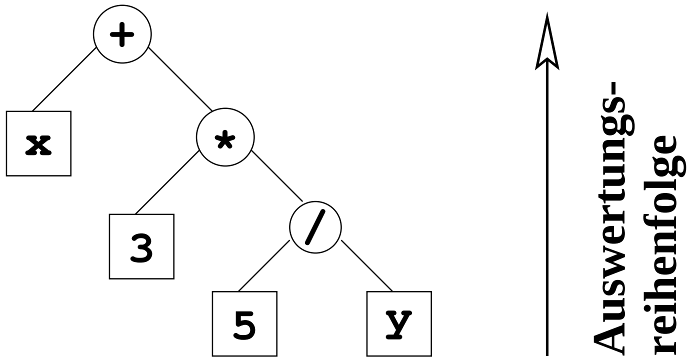
- Inkrement/Dekrement (numerische Variablen):
  `++` (Inkrement), `--` (Dekrement)
  (vor und nach der Variable verwendbar)

Auswertung zusammengesetzter Ausdrücke

- Striktheit: Auswertung aller Teilausdrücke (unabhängig vom Einfluß auf Gesamtausdruck)
- Festlegung der Auswertungsreihenfolge durch Klammerung
- Ohne Klammerung Auswertungsreihenfolge festgelegt durch
  - Operatorpräzedenz (bei unterschiedlichen Operatoren)
  - Assoziativität (bei gleichen Operatoren)
- Operatorpräzedenz: “Reihenfolge” der Operatoren (z.B. “Punkt vor Strich”-Regel)
- Assoziativität: meist von links nach rechts (linksassoziativ)
- Operatorbaum: Darstellung von Ausdrücken

Beispielausdruck:

`x + 3 * ( 5 / y )`



Felder (Reihungen)

- Zusammenfassung mehrerer Variablen desselben Datentyps
- Deklaration:
  - Syntax: *Datentyp*[] *Variablenname*;
  - Beispiel: `int[] zahlenfeld;`
  - Festlegung des *Datentyps* der Feldelemente
- Initialisierung:
  - Syntax:
    Variablenname `= new` *Datentyp*[*Elementanzahl*];
  - Beispiel: `zahlenfeld = new int[10];`
  - Festlegung der *Elementanzahl*
- Zugriff auf Feldelemente:
  - Syntax: *Variablenname*[*Index*]
  - Beispiel: `int zahl = zahlenfeld[9];`
  - Elemente von 0 bis *Elementanzahl* − 1 indiziert
- Leere Felder: Initialisierung mit `null`
  (Beispiel: `int[] zahlenfeld = null;`)
- Elementanzahl eines Feldes: Feldattribut `length`
  (Beispiel: `int laenge = zahlenfeld.length;`)

Anweisungen

- Aufgabe: Steuerung des Programmablaufs
- Anweisungen nur in Methodenrümpfen zulässig
- Anweisungen liefern keinen Wert zurück
- wichtige Arten von Anweisungen:
  Zuweisungen, Methodenaufrufe ohne Rückgabewert, Fallunterscheidungen, Schleifen

Einfache Fallunterscheidung

- Syntax:
  `if (`*Bedingung*`)` $\mathit{Alternative}_{true}$
  `else` $\mathit{Alternative}_{false}$
- Beispiel:
  ```
  if (zahl < 10) zahl++;
  else { zahl--; zahl = zahl / 2; }
  ```
- Auswertung von *Bedingung* liefert Booleschen Wert
  $\rightarrow$ Ausführung der entsprechenden *Alternative*
  (andere *Alternative* wird nicht ausgeführt)
- `else`-Alternative kann weggelassen werden

Schleifen

- `while`-Schleife:
  - Syntax: `while (`*Bedingung*`)` *Schleifenkörper*
  - Beispiel:

    `while (zahl > 0) { zahl = zahl - 10;}`
  - Auswertung der *Bedingung* ergibt `true` →
    Ausführung des *Schleifenkörpers*
    Wiederholung der Auswertung der *Bedingung*
  - Auswertung der *Bedingung* ergibt `false` →
    Schleife wird beendet

- `for`-Schleife:
  - Syntax:

    `for(`*Initialisierung*`;` *Bedingung*`;` *Fortschaltung*`)`
      Schleifenkörper
  - Beispiel:

    ```
    for (i = 0; i < feld.length; i++)
      zahl = zahl + feld[i];
    ```
  - *Bedingung*/*Schleifenkörper*: siehe oben
  - *Initialisierung*: Vor 1. Auswertung der *Bedingung*
  - *Fortschaltung:* nach jeder Ausführung des *Schleifenkörpers*
  - besonders geeignet für Felddurchläufe u.ä.