Programmierkurs Java

UE 3 – Einführung

Dr.-Ing. Dietrich Boles

Gliederung

- Grundlagen
- Ausdrücke und Variablen
- Ein- und Ausgabe
- Bedingte Anweisung
- Alternativanweisung
- Wiederholungsanweisung
- Blockanweisung
- Boolesche Ausdrücke
- Fehler
- Zusammenfassung

Lexikalik

- Java zugrundeliegender Zeichensatz: **Unicode**
- 16-Bit-Zeichensatz (2^{16} Zeichen)
- erste 128 Zeichen: **ASCII** (7-Bit-Zeichensatz)

Möglichst nur ASCII-Zeichen bzw.

Zeichen auf der Tastatur verwenden !!!!!!!!!!!!

Token

- Token: lexikalische Einheiten
  - Symbole: `<, =, <=, ...`
  - Schlüsselwörter: `while, if, ...`
  - Bezeichner: *`Prozedurnamen`*`, `*`Klassennamen`*`, ...`
  - Literale: `true, 23, 24.5f, "hello world", ...`

- Trennung von Token:
  - Leerzeichen (Blank)
  - Tabulator
  - Zeilenende
  - Zeilenvorschub
  - Seitenvorschub

- Unterscheidung von Groß- und Kleinbuchstaben!

Schlüsselwörter

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	false	final	finally
float	for	goto	if
implements	import	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
strictfp	super	switch	synchronized
this	throw	throws	transient
true	try	void	volatile
while			

Bezeichner

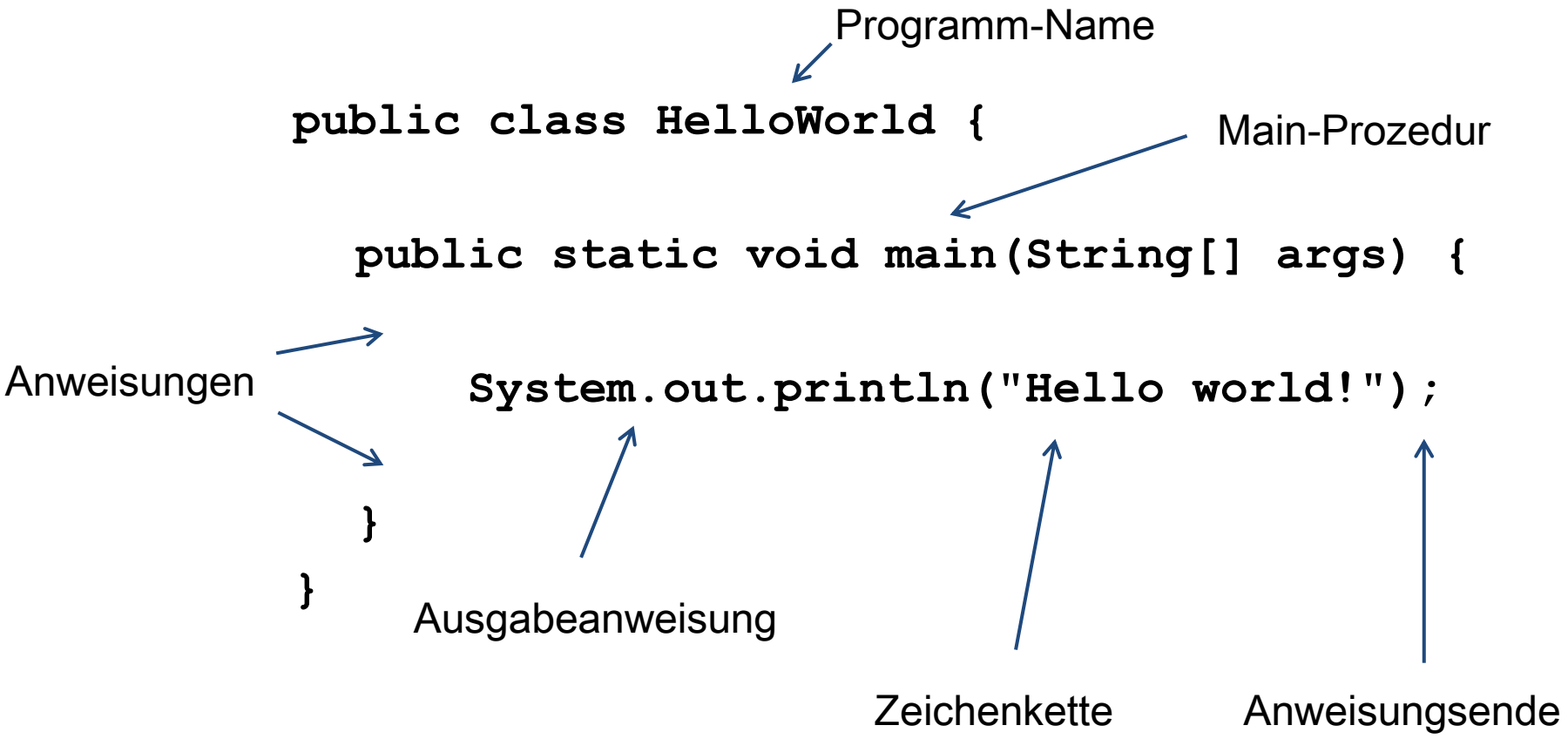
- Benennung von deklarierten Einheiten und Labeln:
  - Klassennamen
  - Variablennamen
  - Prozedurnamen
  - ...

- Beginn mit **Buchstabe**, **Unterstrich (_)** oder **$-Zeichen**
- anschließend: **Buchstaben**, **Ziffern, Unterstriche, $-Zeichen**
- Möglichst keine Umlaute und kein ß verwenden!
- Keine Schlüsselwörter erlaubt!
- Beispiele:
  - `nimm2`
  - `Kaetzchen`
  - `_zahl`
  - `$money`

Nicht erlaubt:

- `4gewinnt`
- `#dibo`
- `dibo@uni-oldenburg.de`

Programme


Programm-Name
public class HelloWorld {
Main-Prozedur
public static void main(String[] args) {
Anweisungen
System.out.println("Hello world!");
}
}
Ausgabeanweisung
Zeichenkette
Anweisungsende

Anweisungssequenzen

- Anweisungen einer Anweisungssequenz werden nacheinander ausgeführt!

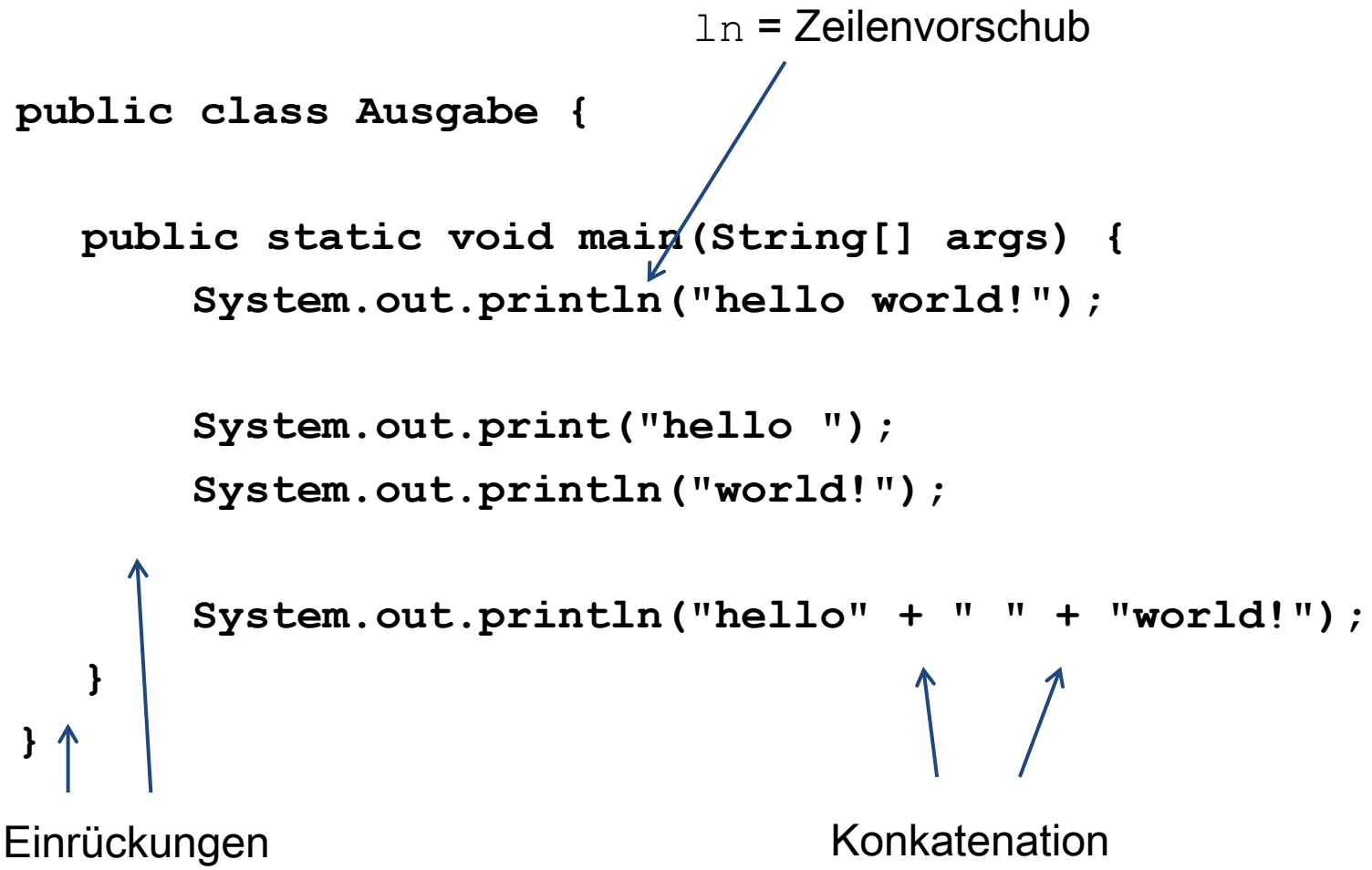
```
public class Anweisungssequenzen {

   public static void main(String[] args) {
        System.out.println("Eigenschaften von Java:");
        System.out.println("- einfach");
        System.out.println("- objektorientiert");
        System.out.println("- robust");
   }

}
```

Ausgabeanweisungen

- Ausgabe erfolgt auf die Console!



Kommentare

- 3 Typen
  - Zeilenkommentare
  - Bereichskommentare
  - Dokumentationskommentare (später)

```
public class Kommentare {

   public static void main(String[] args) {

        // Start des Programms
        System.out.println("Kommentare");

        /* Dies ist
             das Ende
             des Programms */
   }
}
```

Zeilenkommentar

Bereichskommentar

Codekonventionen

- auf Lesbarkeit des Codes achten
- `public static void main...` in eine Zeile
- `}` unterhalb des `p` von `public`
- innere Anweisungen um 4 Spalten einrücken
- pro Zeile **eine** Anweisung
- Leerzeile vor Kommentaren
- Bereichskommentare folgendermaßen strukturieren:

```
/*
 * Kommentar
 */
```

Ausdrücke

- Berechnen von Werten („Ganze Zahlen“)
  - int-Literale (repräsentieren Zahlenwerte)
  - Operatoren (Priorität, Assoziativität, …)
  - Klammerung
  - Arithmetische Ausdrücke: berechnen und liefern einen Zahlenwert

```
public class Ausdruecke {

   public static void main(String[] args) {
        System.out.println(3 + 4);                  // 7
        System.out.println(5 + (6 * 7));            // 47
        System.out.println(-34 - 15 / 6);           // -36
        System.out.println(13 / 2 + 13 % 2);        // 7

        System.out.println("4 + 3 = " + 7);
        System.out.println("4 + 3 = " + (4 + 3));
   }
}
```

Konvertierung in Zeichenkette + Konkatenation

Variablen

- Speichern von Zahlenwerten
  - Variable = Behälter für **einen** Zahlenwert
  - Typ des Wertes
  - (eindeutiger) Name der Variablen (Bezeichner)
  - Initialisierung der Variablen mit einem (berechneten) Wert
  - Benutzung in Ausdrücken

```
public class Variablen {

   public static void main(String[] args) {
      int zahl1 = 12;
      int zahl2 = 4 + 3;
      int zahl3 = 6 - zahl2;
      System.out.println(zahl1);
      System.out.println("Zahl 3 = " + zahl3);
   }
}
```

Typ

Name

Initialisierung

Benutzung

Zuweisung

- Ändern von gespeicherten Zahlenwerten
  - Zuweisung eines neuen Wertes
  - Berechnung des Ausdrucks rechts vom = - Zeichen
  - Variablenname in einem Ausdruck wird ersetzt durch aktuell gespeicherter Wert dieser Variablen
  - Überschreiben des gespeicherten Wertes

```
public class Zuweisung {

   public static void main(String[] args) {
        int zahl1 = 7;              // 7
        int zahl2 = 4 + 8;          // 12
        int zahl3 = 16 - zahl2;     // 4

        zahl2 = zahl1 + 1;          // 8
        zahl3 = zahl2 / zahl3;      // 2

        System.out.println("Ergebnis = " + zahl3);
   }
}
```

Zuweisung

Ein- und Ausgabe mit Klasse IO

- Vordefinierte Java-Klasse **`IO`**
- Dateien:
  - **`IO.java`** (Java-Quellcode)
  - **`IO.class`** (Java-Bytecode)
  - **`IO.README`** (Informationen)
- Die Datei **`IO.class`** muss sich in demselben Verzeichnis befinden, wie ein Java-Programm, das ausgeführt werden soll !
- Eingabeanweisungen; z.B. **`int zahl = IO.readInt();`**
- Ausgabeanweisungen;
  - Java-Standard: **`System.out.println("hello!");`**
  - in `IO`, z.B.: **`IO.println("hello!");`**

Ein- und Ausgabe mit IO / Ausgabeanweisungen

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

     System.out.print(4711);
     System.out.println(" ist eine positive Zahl!");
     int fuenf = 5;
     IO.print(-fuenf);
     IO.println(" ist eine negative Zahl!");

  }
}
```

Ein- und Ausgabe mit IO / Eingabeanweisungen

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

     int zahl1 = IO.readInt("Zahl 1: ");
     int zahl2 = IO.readInt("Zahl 2: ");
     int produkt = zahl1 * zahl2;
     IO.println(zahl1 + "*" + zahl2 + "=" + produkt);

  }
}
```

Kontrollstrukturen

- Schema, welches die Reihenfolge der Abarbeitung von Anweisungen festlegt
- Typen:
  - Sequenz
  - Verzweigungen
  - Schleifen
- Steuerung durch Bedingungen (boolesche Ausdrücke)
- Motivation:
  - **Falls** der Benutzer einen positiven Wert eingegeben hat, soll dieser quadriert werden.
  - **Falls** der Benutzer einen geraden Wert eingegeben hat, soll dieser ausgegeben werden, **andernfalls** soll der Divisionsrest durch 10 ausgegeben werden.
  - **Solange** der Benutzer eine 0 eingibt, soll er erneut zur Eingabe aufgefordert werden.
  - **Solange** eine Variable größer als 0 ist, soll sie durch 2 dividiert werden.

Bedingte Anweisung

- Syntax:

  "if" "(" <Bedingung> ")" <true-Anweisung>

- Semantik:
  - Werte Bedingung aus
  - Nur wenn Bedingung wahr ist, führe true-Anweisung aus

```
public class BedingteAnweisung {

   public static void main(String[] args) {
        int zahl = IO.readInt("Zahl: ");
        if (zahl < 0)
            zahl = -1 * zahl;
        System.out.println("Absolutwert = " + zahl);
   }
}
```

Alternativanweisung

- Syntax:

  "if" "(" <Bed.> ")" <true-Anw> "else" <false-Anw>

- Semantik:
  - Werte Bedingung aus
  - Wenn Bedingung wahr ist, führe true-Anweisung aus
  - Wenn Bedingung falsch ist, führe false-Anweisung aus

```
public class Alternativanweisung {
   public static void main(String[] args) {
     int zahl = IO.readInt("Zahl: ");
     if (zahl < 0)
         System.out.println("Eingabe einer negativen Zahl");
     else
         System.out.println("Eingabe einer nicht-negativen Zahl");
     System.out.println("Quadratzahl = " + (zahl * zahl));
   }
}
```

Wiederholungsanweisung

- Syntax:

  "while" "(" <Bedingung> ")" <Schleifenanweisung>

- Semantik:
  (1) Werte Bedingung aus
  (2) Falls die Bedingung falsch ist, beende die Wiederholungsanweisung.
  (3) Falls die Bedingung wahr ist, führe Schleifenanweisung aus und fahre bei (1) fort.

```
public class Schleife {
    public static void main(String[] args) {
        int zahl = IO.readInt("Zahl (>=0): ");
        while (zahl < 1000)
            zahl = zahl * 2;
        System.out.println(zahl);
    }
}
```

Blockanweisung

- Syntax:

  "{" {<Anweisung>} "}"

- Sinn: Zusammenfassung mehrerer Anweisungen zu einer (zusammengesetzten) Anweisung

```
public class Blockanweisung {
    public static void main(String[] args) {
        int zahl = IO.readInt("Zahl (>=0): ");
        int spiegelzahl = 0;
        while (zahl > 0) {
            spiegelzahl = spiegelzahl * 10 + zahl % 10;
            zahl = zahl / 10;
        }
        System.out.println(spiegelzahl);
    }
}
```

Boolesche Ausdrücke

- Bedingungen = boolesche Ausdrücke
- Liefern Wahrheitswert (wahr/true, falsch/false)
- Vergleichsoperatoren:

```
==      if (zahl == 4) ...
!=      if (zahl != 4) ...
<       if (zahl1 < zahl2 - 4) ...
<=      if (2 * zahl1 <= 3 * zahl2) ...
>       if (zahl > 0) ...
>=      if (zahl / 2 >= 1) ...
```

- Boolesche Operatoren:

```
!       if (!(zahl == 4)) ...
&&      if (zahl >= 0 && zahl < 10) ...
||      if (zahl < 0 || zahl >= 10) ...
```

Codekonventionen

- Variablenname: Anfangsbuchstabe klein; Anfangsbuchstaben neuer Wortbestandteile groß
- Variablenname: aussagekräftig !!!
- Vor und hinter duadischen Operatoren ein Leerzeichen
- Hinter `if` und `while` ein Leerzeichen
- Für true-, false- und Schleifenanweisung möglichst immer die Blockanweisung verwenden
- falls Blockanweisung:
  - `)` und `{` in dieselbe Zeile, durch Leerzeichen getrennt
  - `}` unter `i` von `if` bzw. `w` von `while`
- innere Anweisungen um 4 Spalten einrücken
- Alternativanweisung: `} else {` (in eine Zeile)

Fehler

- Syntaxfehler

```
While (i = 8) i plus 1;
```

- Laufzeitfehler (= Exceptions)

```
int i = IO.readInt(); // 0
int j = 8 / i;
IO.print(j);
```

- Endlosschleifen

```
int i = 10;
while (i > 0)
    IO.println(i);
    i = i - 1;
```

- Logische Fehler

Zusammenfassung

- Ausdruck: Vorschrift zur Berechnung von Werten
- Variable: Behälter zum Speichern eines Wertes
- Kontrollstruktur: Schema, welches die Reihenfolge der Abarbeitung von Anweisungen festlegt
- Sequenz: Ausführung von Anweisungen nacheinander
- if-Anweisung: Ausführung von Anweisungen abhängig von einer Bedingung
- while-Anweisung: wiederholte Ausführung von Anweisungen, abhängig von einer Bedingung