

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

"Hello World!"

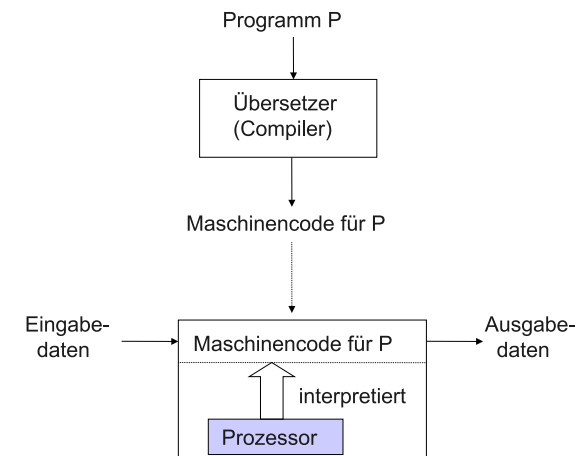
```
/* Ausgabe von "Hello World!" auf der Kommandozeile */
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Gespeichert in Textdatei namens Hello.java
- Einfaches Java-Programm
 - `main()` - Methode
 - Folge von Anweisungen, durch ; voneinander getrennt, bilden den Rumpf der Methode
- `System.out.println()` ruft eine Bibliotheksfunktion auf, die einen Text auf der Konsole ausgibt
- Befehlszeilenargumente können mittels `String[] args` übergeben werden (→ siehe nächste Vorlesung)

Unser erstes Java Programm

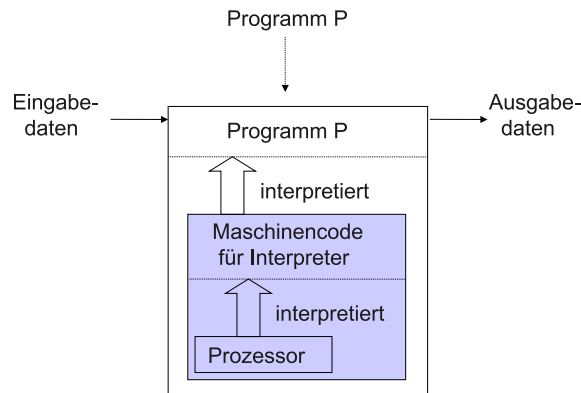
Prinzipielle Ausführungsvarianten I

1 Mittels Übersetzung:



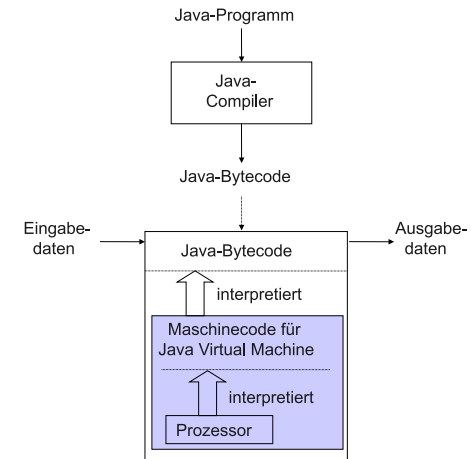
Prinzipielle Ausführungsvarianten II

2 Mittels direkter Interpretation:



Prinzipielle Ausführungsvarianten III

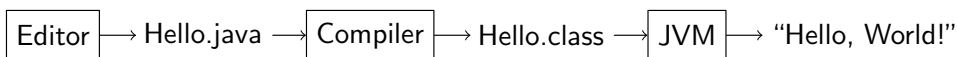
3 Mischformen aus Übersetzung und Interpretation (Beispiel):



Programmieren in Java

Schritte:

- Programm schreiben
- Programm übersetzen / compilieren
- Programm ausführen



Bemerkung:

- Zwei Schritte wurden zunächst vernachlässigt:
 - Integration/Komposition mehrerer Programmteile
 - Installation der Programme
 In der Praxis gewinnen diese Schritte an Bedeutung.

- Implementierung von Softwaresystemen heißt also:
 - Verfassen von Programmen bzw.
 - Anfertigen von Beschreibungen in Sprachen, die von Maschinen verarbeitet werden können.

Integrierte Datentypen

Literaturhinweis: Kapitel 1.2 aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Datentypen

- Programme verarbeiten Informationen in Form von Daten
- Art der Daten wird durch den entsprechenden **Datentyp** festgelegt
- Ein Datentyp hat einen Wertebereich und eine dazugehörige Menge an Operationen
- *Beispiel:*
 - Texte und Zeichenfolgen als String; können z.B. zusammengefügt oder in Großbuchstaben umgewandelt werden
 - Messwerte als Zahlen; können z.B. addiert, multipliziert, verglichen werden

Begriffsklärung: Wert

Werte (engl. *values*) in Java sind

- elementare Daten (Zahlen, Wahrheitswerte, Zeichen, ...)
- Referenzen auf Objekte (⇒ Vorlesung zum Thema "Objekte")

Wie für abstrakte Objekte oder Begriffe typisch, besitzen Werte

- keinen Ort,
- keine Lebensdauer,
- keinen veränderbaren Zustand,
- kein Verhalten.

Begriffsklärung: Typ

Ein **Typ** (engl. *type*) charakterisiert Werte, auf denen die gleichen Operationen zulässig sind.

Typisierte Sprachen besitzen ein Typsystem, das für jeden Wert festlegt, von welchem Typ er ist.

Bemerkungen

- Wir betrachten zunächst die Basisdatentypen und elementare Operatoren für Elemente dieser Datentypen, wie man sie in jeder Programmier-, Spezifikations- und Modellierungssprache findet.
- Die Basisdatentypen (engl. *basic / primitive data types*) bilden die Grundlage zur Definition weiterer Typen und Datenstrukturen.

Ganze Zahlen / Integer I

Dem Typbezeichner `int` ist in Java eine Wertemenge zugeordnet, die die ganzen Zahlen von -2^{31} bis $2^{31} - 1$ enthält.

Typbezeichner	<code>int</code>
Werte	ganzen Zahlen von -2^{31} bis $2^{31} - 1$
Typische Literale	<code>1</code> , <code>0</code> , <code>-99</code>
Operationen	<code>+</code> (Addition)
	<code>-</code> (Subtraktion)
	<code>*</code> (Multiplikation)
	<code>/</code> (Division)
	<code>%</code> (Modulo)

Ganze Zahlen / Integer II

Beispiele:

<code>5 - 3</code>	<code>2</code>	
<code>5 / 3</code>	<code>1</code>	Ganzzahlige Division!
<code>5 % 3</code>	<code>2</code>	
<code>1 / 0</code>		Laufzeitfehler
<code>3 - 5 - 4</code>	<code>-6</code>	Linksassoziativ
<code>3 - (5 - 4)</code>	<code>2</code>	
<code>3 * 5 - 4</code>	<code>11</code>	Priorität
<code>2147483647 + 1</code>	<code>-2147483648</code>	Überlauf!

- Innerhalb der Wertemenge sind die Operatoren auf beschränkten ganzen Zahlen gleich mit den üblichen mathematischen Operatoren.
- Außerhalb der Wertemenge ist ihr Verhalten nicht definiert.
- Weitere Datentypen für ganze Zahlen: `long` (64-bit Integer), `short` (16-bit Integer), `byte` (8-bit Integer)

Gleitkommazahlen I

Dem Typbezeichner `double` ist in Java die (endliche!) Wertemenge der Gleitkommazahlen zugeordnet.

Typbezeichner	<code>double</code>
Werte	reelle Zahlen (nach IEEE-Standard 754)
Typische Literale	<code>3.1415</code> , <code>1.98e20</code> , <code>-1.0</code>
Operationen	<code>+</code> (Addition)
	<code>-</code> (Subtraktion)
	<code>*</code> (Multiplikation)
	<code>/</code> (Division)

Gleitkommazahlen II

Beispiele:

<code>3.14 + .03</code>	<code>3.17</code>
<code>5.0 / 3.0</code>	<code>1.6666666666666667</code>
<code>1.0 / 0.0</code>	<code>Infinity</code>

- Die ganzen Zahlen sind in der Programmierung keine Teilmenge der reellen Zahlen!
- Keine exakten Berechnungen möglich
- Näherung an reelle Zahlen, aber nur endliche Darstellung \Rightarrow Präzision!
- Spezielle Werte: **Infinity** (wenn Darstellung zu groß), **NaN** (not a number; wenn Berechnung undefiniert)
- Weiterer Datentyp für Gleitkommazahlen: **float** (32-bit Gleitkommazahlen)

Boolesche Werte II

Semantik der Operationen:

- `a && b` ist **true**, wenn beide Operanden **true** sind; sonst **false**
- `a || b` ist **false**, wenn beide Operanden **false** sind; sonst **true**
- `!a` ist **true**, wenn `a` **false** ist; und **false**, wenn `a` **true** ist

Frage: Zu welchen Werten ergeben sich folgende Ausdrücke?

`7 >= 45 == true`

`sein || !sein` für geeignete Variable `sein`

Boolesche Werte I

Dem Typbezeichner **boolean** ist die Wertemenge der Wahrheitswerte $\{true, false\}$ zugeordnet.

Typbezeichner	boolean
Werte	wahr, falsch
Literale	true , false
Operationen	<code>&&</code> (Und)
	<code> </code> (Oder)
	<code>!</code> (Nicht)

Vergleichsoperatoren

Die folgenden Operatoren vergleichen primitive numerische Werte *gleichen* *Typs* miteinander und liefern als Ergebnis einen booleschen Wert.

<code>==</code>	Gleichheit
<code>!=</code>	Ungleichheit
<code>></code> , <code>>=</code>	größer bzw. größer gleich
<code><</code> , <code><=</code>	kleiner bzw. kleiner gleich

Begriffsklärung: Überladen I

Wenn unterschiedliche Funktionen oder andere Programmelemente den gleichen Bezeichner haben, spricht man vom **Überladen** des Bezeichners (engl. *overloading*).

Wie in den obigen Datentypen gezeigt, können Operatorbezeichner in Java **überladen** werden, d.h. in Abhängigkeit vom Typ ihrer Argumente bezeichnen sie unterschiedliche Funktionen.

Beispiele:

- + arbeitet z.B. auf `int` und `double`
- == arbeitet z.B. auf `int` und `boolean`

Strings I

Dem Typbezeichner `String` ist in Java die Wertemenge der Zeichenketten zugeordnet.

Typbezeichner	<code>String</code>
Werte	Zeichenketten, Text
Literale	"Hallo", "Die 7 Zwerge",
Operation	+ (Konkatenation, Verkettung)

Strings II

Beispiele:

```
"Hello, "+ "World!"    "Hello, World!"
"12 "+ "34"            "12 34"
"12"+"34"              "1234"
"Die "+ 7 + "Zwerge"   "Die 7 Zwerge"
```

- Wenn primitive Datentypen mit einem String konkateniert werden, werden sie automatisch in Strings umgewandelt.
- Strings sind in Java keine primitiven Datentypen, jedoch in die Sprachdefinition fest integriert.
- Strings können auf der Kommandozeile direkt ausgegeben oder auch von dort eingelesen werden.

Zu welchen Werten ergeben sich folgende Ausdrücke?

`"4" + "5"` ergibt sich zu

`" " + 9 + 7` ergibt sich zu

`"7" + "+" + "5"` ergibt sich zu

Praxisbeispiel: Schaltjahre

Wann ist ein Jahr ein Schaltjahr?

Bestimme für eine eingegebene Jahreszahl, ob es sich um ein Schaltjahr handelt!

Aus Wikipedia-Artikel *Schaltjahr*:

- Die durch 4 ganzzahlig teilbaren Jahre sind Schaltjahre. [...]
- Die durch 100 ganzzahlig teilbaren Jahre (z.B. 1700, 1800, 1900, 2100 und 2200) sind keine Schaltjahre. [...]
- Schließlich sind die ganzzahlig durch 400 teilbaren Jahre doch Schaltjahre. Damit sind 1600, 2000, 2400, ... jeweils wieder Schaltjahre. [...]

Implementierung: Schaltjahr.java

```
public class Schaltjahre {
    public static void main(String[] args) {
        String eingabe;
        int jahr;
        boolean istSchaltjahr;

        eingabe = args[0];
        jahr = Integer.parseInt(eingabe);

        istSchaltjahr = (jahr % 4 == 0) && (jahr % 100 != 0);
        istSchaltjahr = istSchaltjahr || (jahr % 400 == 0);

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

Bezeichner in Java

Variablen werden über **Bezeichner** (engl. *identifier*) referenziert.

Für Bezeichner muss in Java gelten:

- Bezeichner bestehen aus beliebig vielen Unicode-Zeichen (insbesondere Buchstaben, Ziffern und Unterstrich _).
- Sie starten nicht mit einer Ziffer.
- Bezeichner dürfen nicht mit Schlüsselwörtern oder Konstanten übereinstimmen.
- Bezeichner sind *case-sensitive*, d.h. es wird zwischen Groß- und Kleinschreibung unterschieden.

Variablendeklarationen

Eine Variablendeklaration hat in Java die Form:

```
<Typbezeichner> <Variablenbezeichner> ;
```

Sie definiert eine neue Variable vom angegebenen Typ mit dem angegebenen Bezeichner.

Beispiele:

```
int a;  
boolean b;  
double meineGleitkommavARIABLE;
```

Zuweisungen

Syntax in Java:

```
<Variable> = <Ausdruck> ;
```

Semantik:

Werte den Ausdruck aus und weise das Ergebnis der Variablen zu.

Beispiele:

```
a = 27 % 23;  
b = true;  
meineGleitkommavARIABLE = 3.14;
```

Sprechweise:

“Variable b ergibt sich zu true” oder “Variable b wird true zugewiesen”.

Begriffsklärung: Ausdruck I

Ausdrücke sind das Sprachmittel zur Beschreibung von Werten.

Ein **Ausdruck** (engl. **expression**) in Java ist (u.a.)

- ein Literal,
- ein Bezeichner (Variable, Name),
- die Anwendung einer Operation auf einen oder mehrere Ausdrücke,
- oder aufgebaut aus Sprachmitteln, die erst später behandelt werden.

Begriffsklärung: Ausdruck II

Jeder Ausdruck hat einen Typ:

- Der Typ eines Literals ergibt sich aus dem Wertebereich.
- Der Typ eines Bezeichners ergibt sich aus dem Wert, den er bezeichnet.
- Der Typ einer Operation ist der Ergebnistyp der Operation.

Präzedenzregeln I

Wenn Ausdrücke nicht vollständig geklammert sind, ist im Allg. nicht klar, wie ihr Syntaxbaum aussieht und wie die Auswertung zu erfolgen hat.

Beispiele:

```
3 == 5 == true
false == true || true
false && true || true
```

Präzedenzregeln II

Präzedenzregeln legen fest, wie Ausdrücke zu strukturieren sind:

- Am stärksten binden unäre Operatoren in Präfixform.
- Binäre Infix-Operatoren in Java sind (in der Regel) linksassoziativ.
- Präzedenzregeln für Infix-Operatoren:
 - *, / , \% binden stärker als + und -
 - ! bindet stärker als &&, und && stärker als ||
- Boolesche Operatoren && und || sind nicht-strikt.

Eingabe/Ausgabe

In unseren Beispielen verwenden wir zunächst die Eingabe auf Kommandozeilebene bei Programmstart.

- Das erste Argument ist in der `main`-Methode unter `args[0]` verfügbar; das zweite Argument unter `args[1]` verfügbar, etc.
- Da die Argumente als `String`-Werte übergeben werden, müssen sie evtl. in einen anderen Datentyp umgewandelt werden (\Rightarrow siehe "Explizite Typumwandlung")
- Beispiel: Compilieren und Ausführen von `LeapYear` mit Parameter

```
> javac Schaltjahr.java
> java Schaltjahr 2004
true
> java Schaltjahr 1900
false
```

- Fehlt der Parameter beim Aufruf, wird ein Fehler ausgegeben!

Typumwandlungen: Explizite Typumwandlung

- Bisweilen muss man Werte eines Types in einen anderen Typen umwandeln.
- Zur Umwandlung von Strings in numerische Werte gibt es spezielle Bibliotheksmethoden:

<code>int Integer.parseInt(String s)</code>	Wandelt s in int-Wert um
<code>double Double.parseDouble(String s)</code>	Wandelt s in double-Wert um
<code>boolean Boolean.parseBoolean(String s)</code>	Wandelt s in boolean-Wert um

Typumwandlungen: Expliziter Cast

- Java bietet integrierte Typumwandlung für primitive Datentypen
- **Achtung: Informationsverlust**
- Auf Klammerung achten, da Casts stärker binden als arithmetische Operationen
- *Beispiel:* `(int)2.71828` ergibt den `int`-Wert 2

- Hinweis: Runden mittels der Bibliotheksfunktion `long Math.round(double d)` und explizitem Cast von `long` auf `int`
Beispiel: `(int) Math.round(2.71828)` liefert 3

Typumwandlungen: Automatische Promotion für Zahlen

- Daten von primitiven numerischen Typen können in einem Kontext verwendet werden, wenn in dem ein Wert eines Datentyps verwendet wird, der einen größeren Wertebereich abdeckt \Rightarrow kein Informationsverlust!
- *Beispiel:* Bei der Auswertung von `4.0 * 3` wird 3 zunächst automatisch in einen `double`-Wert umgewandelt, danach wird die Multiplikation auf `double` durchgeführt
- *Beispiel:* Bei der Auswertung von `"Ocean's " + 11` wird 11 zunächst automatisch in einen `String`-Wert umgewandelt, danach wird die Konkatination auf `String` durchgeführt

Zusammenfassung

- Compilieren und Ausführen von Java-Programmen
- Integrierte Datentypen
 - Primitive Datentypen (`int`, `double`, `boolean`)
 - Strings
- Ausdrücke
- Variablen: Deklaration und Zuweisung
- Typkonversion