Inhalt

4 Einführung in die Programmiersprache Java (Teil I) 4-2

4.1 Hallo Welt 4-2

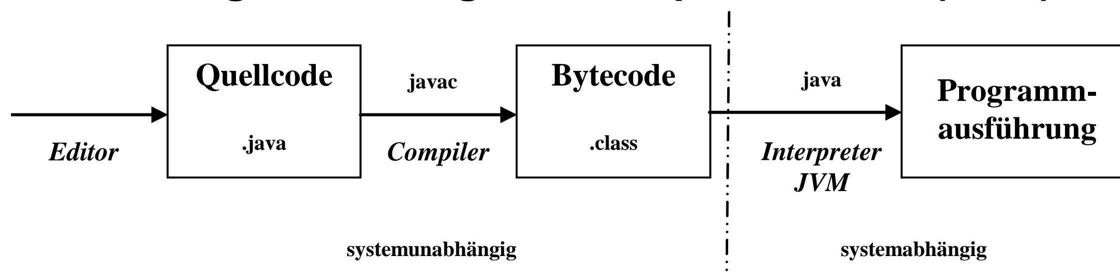
4.2 Grundelemente der Sprache 4-3

4.2.1 Alphabet 4-3
4.2.2 Bezeichner 4-3
4.2.3 Kommentare 4-4
4.2.4 Elementardatentypen 4-6
4.2.5 Konstanten 4-6
4.2.6 Variablen 4-7
4.2.7 Ausdrücke 4-8
4.2.8 Zusammenfassung 4-10

4.3 Ein- und Ausgaben 4-11

4.3.1 Ausgabe 4-11
4.3.2 Eingabe 4-11
4.3.3 Methoden der Klasse `IOTools` 4-12

4 Einführung in die Programmiersprache Java (Teil I)



4.1 Hallo Welt

Eine Anwendung (**Application**) ist ein *eigenständiges* Programm. Sie kann *unabhängig* vom Internet durch einen *Java-Interpreter* ausgeführt werden.

1. **Quellcode (*Editor*)**:

HalloWelt.java

```
public class HalloWelt
{
  public static void main( String[] args)
  {
    System.out.println( "Hallo Welt!");
  }
}
```

Ein Java-Programm ist in **Blöcke** strukturiert. Blöcke sind Anweisungen, die in geschweifte Klammern **{** und **}** eingeschlossen werden. Im Beispiel sind zwei Blöcke *ineinander geschachtelt*:

`class`

Eine Klasse `HalloWelt` ist die **oberste Struktureinheit** des Programms. Deren Name *muss* mit dem Namen des Programms übereinstimmen, `HalloWelt.java`.
Das Schlüsselwort `public` sagt aus, dass die Klasse `HalloWelt` eine **öffentlich** zugängliche Klasse ist.

`main`

Innerhalb einer Klasse gibt es **untergeordnete Struktureinheiten, wie Attribute und Methoden**, auf die wir später eingehen werden. Jede Klasse, die wie diese ein ausführbares Programm darstellen soll, besitzt die **Hauptmethode** `main`.

2. **Bytecode (*Compiler*)**:

```
$ javac HalloWelt.java
```
⇒ *HalloWelt.class*

3. **Ausführung (*Interpreter*)**:

```
$ java HalloWelt
Hallo Welt!
```

4.2 Grundelemente der Sprache

4.2.1 Alphabet

Unicode[1] (*Univeral Character Set*), **16-Bit-Zeichensatz**.

Steuerzeichen
Sie dienen der *Steuerung von Ausgabegeräten*, wie Bildschirm und Drucker:

\b	Versetzen um eine Position nach links (**backspace**)
\f	Seitenvorschub (**formfeed**)
\n	Zeilenvorschub (**linefeed, new line**)
\r	Positionierung am Zeilenanfang (**carriage return**)
\t	Horizontaler Tabulator (**horizontal tab**)

Die folgenden Anweisungen erzeugen dieselbe Konsolenausgabe:

```
System.out.println( "Hallo Welt!");
System.out.print( "Hallo Welt!\n");
```

Entwerter
Weitere Escapesequenzen erzeugen *druckbare Zeichen* und dienen der *Entwertung von Metazeichen*:

\'	Entwerten des Abschlusssymbols für Zeichenkonstante
\"	Entwerten des Abschlusssymbols für Zeichenkettenkonstante
****	Entwerten des Backslashs als Escapesequenz

4.2.2 Bezeichner

Für die Bezeichnung (Identifier) der *Objekte*, ihrer *Klassen*, *Attribute* und *Methoden*, *Variablen* und *Konstanten* werden **Namen** benötigt:

Schlüsselwörter sind spezielle Bezeichner der Sprache Java (`int`, `if`, `public`, ...).

Namen sind frei wählbare, beliebig lange Wörter aus *Buchstaben* beliebiger Sprachen, *Ziffern*, dem *Unterstrich* und dem *Dollarzeichen*, die *nicht* mit einer Ziffer beginnen dürfen und keinem der *48 Schlüsselwörter* entsprechen.

Bezeichner sind *ein* oder aber auch *mehrere Namen*, verbunden durch einen *Punkt* (`setLampe`, `System.out.println`).

Groß- und Kleinbuchstaben sind *signifikant*.

[1] Unicode-Konsortium http://www.unicode.org/

Faustregeln Namensgebung

- ⇨ **Namen beschreiben den Inhalt: *"Sprechende" Namen*.**
- ⇨ ***Klassennamen* beginnen mit Großbuchstaben: `HalloWelt`.**
- ⇨ ***Variablen-*, *Objekt-*, *Attribut-* und *Methodennamen* beginnen mit Kleinbuchstaben: `lampe`.**
- ⇨ **Bei *zusammengesetzten Namen* beginnen weitere Wörter jeweils mit einem Großbuchstaben: `setLampe`.**
- ⇨ ***Konstanten* werden mit Großbuchstaben bezeichnet: `MAX`.**
- ⇨ **Bei *zusammengesetzten* Konstanten werden weitere Worte jeweils durch einen Unterstrich getrennt: `MAX_VALUE`.**

4.2.3 Kommentare

Kommentare dienen der *Lesbarkeit* und *Verständlichkeit* von Programmen und ihren Quelltexten. *Sie werden vom Compiler ignoriert*, haben somit keinen Einfluss auf Größe und Geschwindigkeit des ausführbaren Programms. Man unterscheidet Kommentare für **interne** und **externe Dokumentationen**. Java kennt drei verschiedene Arten von Kommentaren.

HalloWelt.java

```
// HalloWelt.java                                MM 2009
/*              Mein erstes Programm                */

/**
 * Konsolenausgabe des Schriftzugs "Hallo Welt!".
 */
public class HalloWelt
{
/**
 * Hauptmethode, erzeugt Bildschirmausschrift
 */
  public static void main( String[] args)
  {                                  // Konsolenausgabe
    System.out.println( "Hallo Welt!");
  }
}
```

Interne Dokumentation

1. `// Zeilenkommentar` (*Programmbeschreibung*)
   Alle dem Kommentarzeichen `//` folgende Zeichen bis Zeilenende werden vom Compiler ignoriert. Sie dienen vor allem dem Programmierer zur *Programmbeschreibung*.

```
// HalloWelt.java                                MM 2009
// Konsolenausgabe
```

2. `/* Kommentar */` (*Auskommentieren von Testcode*)
   Alle Zeichen zwischen `/*` und `*/` werden als Kommentar behandelt. Dieser kann über mehrere Zeilen gehen, darf aber *nicht* geschachtelt auftreten und wird zum *Auskommentieren von Programmteilen* vor allem in der *Testphase* verwendet.

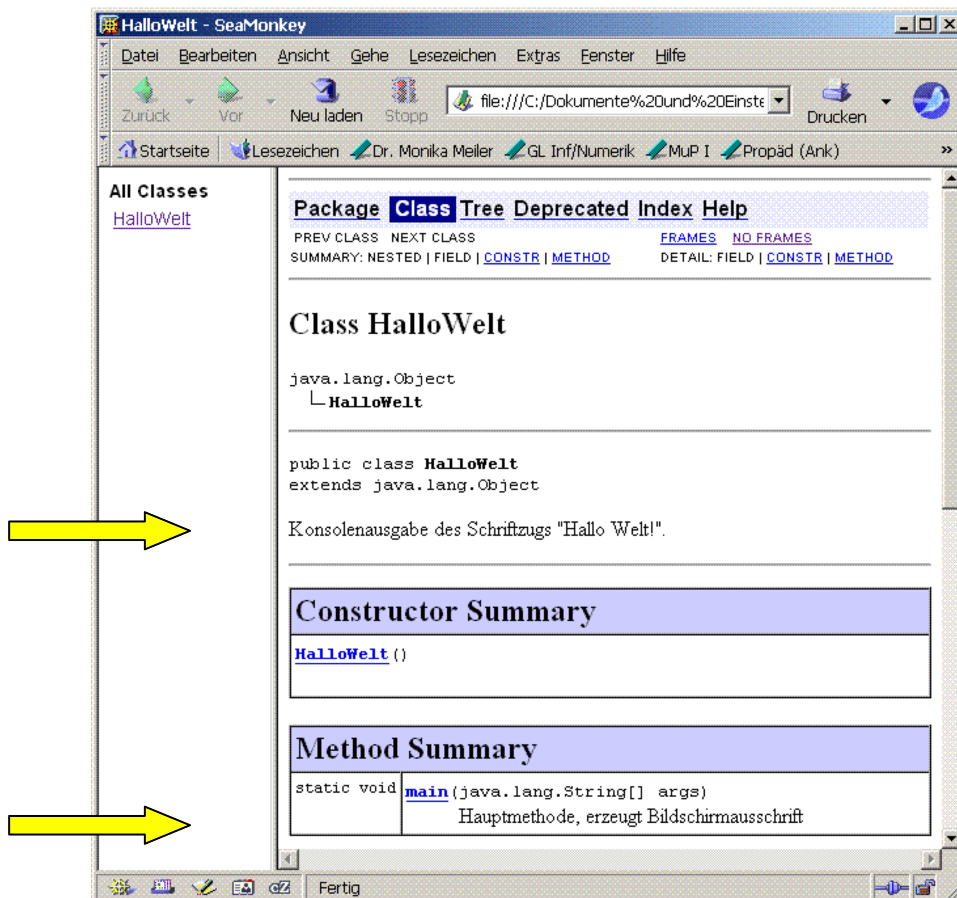
```
/*                Mein erstes Programm                */
```

Externe Dokumentation

3. `/** doc-Kommentar */` (*Online-Dokumentation*)
   Aus Kommentaren, die mit `/**` und `*/` eingeschlossen werden, können automatisch HTML-Seiten generiert werden. Das *Sun-Programm* `javadoc` generiert anhand der Struktur einer Klasse und den in ihr enthaltenen *Dokumentationskommentaren* eine *Online-Dokumentationen* und legt diese im aktuellen Verzeichnis ab. Externe Kommentare stehen stets unmittelbar vor den dokumentierten Klassen, ihren Attributen und Methoden.

```
/**
 * Konsolenausgabe des Schriftzugs "Hallo Welt!".
 */
/**
 * Hauptmethode, erzeugt Bildschirmausschrift
 */
```

`$ javadoc HalloWelt.java`



Java Plattform, All Classes

4.2.4 Elementardatentypen

Java kennt vordefinierte **Elementardatentypen** für **ganze Zahlen**, **Gleitpunktzahlen** und **logische Werte**. Aus diesen können komplexe **strukturierte Datentypen** zusammengesetzt werden.

Ganze Zahlen

Typ	kleinster Wert (MIN)	größter Wert (MAX)	Byte	Codierung
`byte`	-128 (-2^7)	127 (2^7-1)	**1**	**Direktcode / Komplement**
`short`	-32´768 (-2^{15})	32´767 ($2^{15}-1$)	**2**	
`int`	-2´147´483´648 (-2^{31})	2´147´483´647 ($2^{31}-1$)	**4**	
`long`	-9´223´372´036´854´775´808 (-2^{63})	9´223´372´036´854´775´807 ($2^{63}-1$)	**8**	
`char`	0	65535 ($2^{16}-1$)	**2**	**Unicode**

Der Datentyp **`char`** wird speziell für **Zeichen** genutzt, intern als *ganze Zahlen* im Unicode UTF-16 dargestellt. Er gehört also zu den *ganzen Zahlen*.

Gleitpunktzahlen

Typ	kleinster Wert (MIN)	größter Wert (MAX)	Byte	Codierung
`float`	±1.40239846 E -45	±3.40282347 E +38	**4**	**IEEE 754**[2]
`double`	±4.940656458412465 E -324	±1.797693138462315750 E +308	**8**	

Logische Werte

Der Datentyp **`boolean`** kennt entsprechend der Booleschen Logik nur zwei Werte, **`true`** für *wahr* und **`false`** für *falsch*.

Wert	*Bedeutung*
`true`	wahr
`false`	falsch

4.2.5 Konstanten

Unbenannte Konstanten (explizit angegebene Daten)

Ganzzahlige Konstanten (integer-constant)

29 *Datentyp*: **`int`** 3000000000L *Datentyp*: **`long`**

Gleitpunktkonstanten (floating-constant)

18. = 18e0 = 1.8e1 = .18E2 = 0.018E3 = +18000E-3 *Datentyp*: **`double`**

Zeichenkonstanten (character-constant)

```
char a = 'a', b = 98, c = 0x63, d = '\u0064';
```

Zeichen	*dezimal*	*hexadezimal*	*Unicode*	*Bitfolge*
`'a'`	`97`	`0x61`	`'\u0061'`	`0000 0000 0110 0001`
`'1'`	`49`	`0x31`	`'\u0031'`	`0000 0000 0011 0001`
`'\n'`	`10`	`0x0A`	`'\u000A'`	`0000 0000 0000 1010`

Logische Konstanten (boolean-constant) **`true`**, **`false`**

Zeichenkettenkonstanten (string-literal)

Zeichenketten sind in Java *Objekte einer Klasse* namens `String`. Darauf können wir hier noch nicht genauer eingehen. **Konstanten** dieses Typs werden als Folge von Zeichen aus dem verfügbaren Zeichensatz, eingeschlossen in Ausführungszeichen, dargestellt.

`"Hallo Welt!"`

[2] http://www.h-schmidt.net/FloatApplet/IEEE754de.html

Benannte Konstanten

Jeder ändernde Zugriff auf die ganzzahlige Konstante `MAX` wäre unzulässig.

```
final int MAX = 100;
```

4.2.6 Variablen

Beim Euklidischen Algorithmus werden als mathematische Objekte natürliche Zahlen verarbeitet. Diese müssen im Rechner abgespeichert werden und auch wieder aufgefunden werden. Drei Fragen sind zu klären:

Wo befindet sich im Speicher die Zahl?
Wie viel Speicherplatz benötigt sie?
Wie wird sie abgespeichert?

Daten werden als **Bitfolgen** abgespeichert, d.h. Folgen aus 0 und 1. **Variablen** dienen als *Platzhalter* im Speicher für die Daten und besitzen *drei Grundbestandteile*:

Name	symbolischer Bezeichner für die Anfangsadresse der Bitfolge
Typ	Länge der Bitfolge und ihre Interpretationsvorschrift
Wert	die interpretierte Bitfolge

```
short b = 106;
```

Arbeitsspeicher

Name *Symbolische Adresse*	**Adresse** *Adresse im Speicher*	**Wert** *Inhalt der Speicherzellen*	**Typ** *Interpretations-vorschrift*
	...	...	
b	5e	00	`short` (2 Byte)
	5f	6a	
	...	...	

Adresse b ⇒ $\&b = (5e)_{16} = (0101\ 1110)_2 = 64 + 16 + 8 + 4 + 2 = 94$

Wert von b ⇒ $b = (00\ 6a)_{16} = (0000\ 0000\ 0110\ 1010)_2 = 64 + 32 + 8 + 2 = 106$

```
char b = 'j';
```

Wäre die obige Bitfolge als *Zeichen* zu interpretieren, d.h. als Wert vom Typ `char` (2 Byte), so wäre unter *b* das Zeichen 'j' abgespeichert.

Variablendeklaration und Initialisierer

Typ Variablenname [= Ausdruck], Variablenname [= Ausdruck], . . . **;**

```
int anzahl;               // Variable hat den Wert 0
float zahl, summe;  // Variablen haben den Wert 0.0
float a = 1e10, b = 1e-10;       // Initialisierung
float c = a + b;   // a, b deklariert und definiert
```

Eine Variablendeklaration ist an jeder Stelle im Programm erlaubt, jedoch vor dem ersten Verwenden der Variablen notwendig.

4.2.7 Ausdrücke

Entsprechend der üblichen Syntax, gegebenenfalls unter Verwendung der Klammer „(“ und „)“, lassen sich *Variablen* und *Konstanten* mittels *Operatoren* zu **Ausdrücken** verknüpfen. Je nach dem **Hauptverknüpfungsoperator** unterscheidet man:

Java-Ausdrücke
- **Arithmetische Ausdrücke**
- **Bitausdrücke**
- **Wertzuweisungen**
- **Inkrementieren und Dekrementieren**
- **Logische Ausdrücke (Bedingungen)**
- **Bedingte Ausdrücke**

Arithmetische Ausdrücke

```
      7 / 3   ⇒                   2        int
  7.0 / 3.0   ⇒  2.3333333333333335     double
    7.0 / 3   ⇒  2.3333333333333335     double
  1 / 2 * 2   ⇒                   0        int
```

Bitausdrücke

	&		|		^		~
	0	1	0	1	0	1	
0	0	0	0	1	0	1	1
1	0	1	1	1	1	0	0

In den folgenden Beispielen seien Operanden und Ergebnisse vom Datentyp `byte`.

Negation
```
~ 1     :  ~ 0000 0001              ⇒  1111 1110  ⇒  -2
~~ 1    :  ~ -2                                   ⇒   1
```
Und
```
1 & 3   :  0000 0001 & 0000 0011    ⇒  0000 0001  ⇒   1
```
Oder
```
1 | 3   :  0000 0001 | 0000 0011    ⇒  0000 0011  ⇒   3
```
Exklusiv-Oder
```
1 ^ 3   :  0000 0001 ^ 0000 0011    ⇒  0000 0010  ⇒   2
```
Verschieben

<< bitweises Verschieben um angegebene Stellenzahl nach links, Auffüllen mit 0-Bits

>> bitweises Verschieben um angegebene Stellenzahl nach rechts,
Auffüllen mit dem höchsten Bit
(vorzeichenerhaltend, arithmetisches Verschieben)

>>> bitweises Verschieben um angegebene Stellenzahl nach rechts, Auffüllen mit 0-Bits
(nicht vorzeichenerhaltend, logisches Verschieben)

```
-1 << 3  :  1111 1111 << 3          ⇒  1111 1000  ⇒  -8
-1 >> 3  :  1111 1111 >> 3          ⇒  1111 1111  ⇒  -1
```

Wertzuweisungen

```
x = y = 5;                           ⇒ x = 5, y = 5
x += y + ( z = 1 );          ⇒ z = 1, x = 11, y = 5
```

Inkrementieren und Dekrementieren

```
y = z = 5;                              ⇒ y = 5, z = 5
x = --y + 5;                            ⇒ x = 9, y = 4
x *= y++ + ++z;                 ⇒ x = 90, y = 5, z = 6
--x;                                    ⇒ x = 89
```

Logische Ausdrücke (Bedingungen)

	&, &&		**|, ||**		**!**
	false	**true**	**false**	**true**	
false	**false**	**false**	**false**	**true**	**true**
true	**false**	**true**	**true**	**true**	**false**

```
int x = 2; boolean y, z;
y = 0 < x & x <= 2;          ⇒ true & true ⇒ y = true
z = x == 4;                                 ⇒ z = false
```

Optimierung &&, ||

```
// Division durch 0 wird vermieden:
double x = 1, y = 0, toleranz = 0.1; ...
if( y == 0 || x / y > toleranz ) ...
```

Bedingte Ausdrücke

max(x, y**)** `max = ( x > y) ? x : y;`

4.2.8 Zusammenfassung

Die Menge aller ***Ausdrücke*** wird wie folgt zusammengefasst:

1. ***Konstanten***, ***Variablen*** und ***Methodenaufrufe*** (später) sind ***Ausdrücke***.
2. ***Ausdrücke*** verknüpft mit ***Operatoren*** ergeben wieder Ausdrücke, wobei die übliche Syntax mit ***Klammerung*** erlaubt wird und folgende Vorrangregeln gelten:

Java-Operatoren mit Rangfolge und Assoziativitätsrichtung:

15	**( )** **[ ]** **.**	Ausdrucksgruppierung Auswahl der Feldkomponenten Auswahl der Klassenkomponenten	→
14	**! ~** **++ --** **+ -**	Negation (logisch, bitweise) Inkrementieren, Dekrementieren (Präfix oder Postfix) Vorzeichen	←
13	**(** *Typ* **)**	explizite Typumwandlung	←
12	*** /** **%**	Multiplikation, Division Rest bei ganzzahliger Division	→
11	**+ -**	Summe, Differenz	→
10	**<< >>** **>>>**	bitweise Verschiebung nach links, rechts	→
9	**< <=** **> >=**	Vergleich auf kleiner, kleiner oder gleich Vergleich auf größer, größer oder gleich	→
8	**== !=**	Vergleich auf gleich, ungleich	→
7	**&**	Und (bitweise, logisch)	→
6	**^**	exklusives Oder (bitweise)	→
5	**|**	inklusives Oder (bitweise, logisch)	→
4	**&&**	Und (logisch)	→
3	**||**	inklusives Oder (logisch)	→
2	**? :**	bedingte Auswertung (paarweise)	←
1	**=** **°=**	Wertzuweisung zusammengesetzte Wertzuweisung (*=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=, >>>=)	←

4.3 Ein- und Ausgaben

Getreu dem **EVA-Prinzip** (*Eingabe - Verarbeitung - Ausgabe*) der elektronischen Datenverarbeitung ist es notwendig, Ein- und Ausgaberoutinen zur Verfügung zu stellen.

4.3.1 Ausgabe

Textausgabe
Eine Methode zur **Textausgabe** wurde schon in ***HalloWelt.java*** verwendet.

- Bildschirmausgabe mit anschließendem Zeilenvorschub:

```
System.out.println( "Hallo Welt!");      // Hallo Welt!
```

- Bildschirmausgabe ohne anschließendem Zeilenvorschub (+ mit drei verschiedenen Interpretationen):

```
int a = 3, b = 5;
System.out.print                    // Ich rechne: 3 + 5 = 8
( "Ich rechne: " + a + " + " + b + " = " + (a + b));
```

`System.out.println` und `System.out.print` sind **Methoden** aus einer Standardbibliothek mit denen man *Text* und *Zahlen* in einem Anwendungsfenster, einer sogenannten **Konsole**, ausgeben kann. Der auszugebende Text steht in der Methode als Argument in Anführungsstrichen. Der *Operator* + wird in Abhängigkeit der *Operanden* mit drei verschiedenen Interpretationen verwendet. Mit ihm können *Texte* verknüpft und *Zahlen* addiert werden. `'+'` kann aber auch als *Zeichen* in einem Text stehen.

Fehlerausgabe
Analog verwendet man die Methoden `System.err.println` und `System.err.print` zur **Fehlerausgabe** auf einer **Konsole**.

```
System.err.print( " Fehler!");
System.err.println( "Bitte nur ganze Zahlen eingeben!");
```

4.3.2 Eingabe

Etwas komplizierter sind Methoden zur Eingabe über Tastatur. Um das Eingabekonzept verstehen und anwenden zu können, sind umfangreiche Kenntnisse über objektorientiertes Programmieren und Datenströme notwendig. Deshalb ist es zweckmäßig, dem Anfänger eine Klasse zur Verfügung zu stellen, welche gängige Eingaberoutinen zusammenfasst. Da eine solche innerhalb eines Pakets[3] bereits existiert, nutzen wir diese im Sinne der Wiederverwendung.

Paket `Tools` — `Tools.zip`
Klasse `Tools.IO.IOTools` in `Tools` — Tastatureingaberoutinen

1. **Installationsschritte:**

[3] Dietmar Ratz, Jens Scheffler, Detlef Seese, *Grundkurs Programmieren in Java, Bd 1: Der Einstieg in Programmierung und Objektorientierung*, Hanser Verlag, 2001.

2. In ein Verzeichnis (hier: `Java`) wird das komprimierte Paket `Tools.zip` aus dem Netz heruntergeladen, ohne es auszupacken!
3. Zum Einbinden des Pakets `Tools` in andere Programme muss die Umgebungsvariable `CLASSPATH` auf das aktuelle Verzeichnis und auf `Tools.zip` gesetzt werden:

```
CLASSPATH=.;C:\...\Java\Tools.zip
```

4. Jedes Java-Programm, welches Klassen des Pakets `Tools` verwendet, muss diese am Anfang des Programms einbinden, für die Eingaberoutinen mit der Anweisung:

`import Tools.IO.IOTools;` oder auch
`import Tools.IO.*;`

4.3.3 Methoden der Klasse `IOTools`

Nach diesen Vorbereitungen kann die Klasse `IOTools` für Eingaben verwendet werden, dabei ist `prompt` ein Zeichenkette (`String`), der vor der Eingabe auf der Konsole erscheint.

Lesen eines Zeichen:	`readChar()`
mit Eingabeaufforderung:	`readChar( String prompt)`
Lesen eines Wortes:	`readString()`
mit Eingabeaufforderung:	`readString( String prompt)`
Lesen einer Zeile:	`readLine()`
mit Eingabeaufforderung:	`readLine( String prompt)`
Lesen einer kurzen ganzen Zahl:	`readShort()`
mit Eingabeaufforderung:	`readShort( String prompt)`
Lesen einer ganzen Zahl:	`readInteger()`
mit Eingabeaufforderung:	`readInteger( String prompt)`
Lesen einer langen ganzen Zahl:	`readLong()`
mit Eingabeaufforderung:	`readLong( String prompt)`
Lesen einer gebrochenen Zahl:	`readFloat()`
mit Eingabeaufforderung:	`readFloat( String prompt)`
Lesen einer gebrochenen Zahl doppelter Genauigkeit:	`readDouble()`
mit Eingabeaufforderung:	`readDouble( String prompt)`
Lesen eines Wahrheitswertes	`readBoolean()`
mit Eingabeaufforderung:	`readBoolean( String prompt)`

Das folgende Beispiel soll den Umgang mit der Klasse `IOTools` demonstrieren. Das Programm liest eine ganze Zahl mit der Methode `readInteger` der Klasse `IOTools`.

IntegerEingabe.java

```
// IntegerEingabe.java                           MM 2007

import Tools.IO.*;                                  // Eingabe

/**
 * Integereingabe unter Verwendung der Klasse IOTools.
 */
public class IntegerEingabe
{
/**
 * Eingabe und anschliessende Ausgabe
 * einer ganzen Zahl.
 */
  public static void main( String[] args)
  {
// Eingabe
    int eingabe = IOTools.readInteger
                  ( "Bitte eine ganze Zahl eingeben! ");
// Ausgabe
    System.out.println( "Eingegebene Zahl: " + eingabe);
  }
}
```