3 Die Programmiersprache Java

Im letzten Kapitel haben wir die theoretischen Grundlagen der Programmierung diskutiert. Jetzt werden wir mit „Java“ eine konkrete Programmiersprache kennen lernen.

Die Sprache Java wurde ab 1991 von einem kleinen Team unter Leitung von James Gosling bei SUN unter dem Arbeitstitel OAK (Object Application Kernel) entwickelt. Ursprünglich wollte man eine Programmiersprache entwerfen, die sich in besonderer Weise zur Programmierung von elektronischen Geräten der Konsumgüterindustrie eignen sollte – also von Kaffeemaschinen, Videogeräten, Decodern für Fernsehgeräte etc. 1993 wurde die Zielrichtung des Projektes geändert: Eine Programmiersprache zu entwickeln, die sich in besonderer Weise zur Programmierung auf verschiedensten Rechnertypen im Internet eignen sollte. Als neuer Name wurde *Java* gewählt. Java, die Hauptinsel Indonesiens, ist im Amerikanischen ein Synonym für guten Bohnenkaffee. Der Name hat also keinen direkten Zusammenhang mit den Zielen des Projektes.

Seit 1995 bot SUN kostenlos den Kern eines Programmiersystems JDK (Java Development Kit) zusammen mit einer Implementierung des Java-Interpreters (Java Virtual Machine) an. Seitdem hat diese Sprache sich schneller verbreitet als jede andere neue Programmiersprache der letzten Jahre. Einige Ursachen für dieses Phänomen sind:

- Java Programme sind portabel, sie können also ohne jede Änderung auf unterschiedlichen Rechnern eingesetzt werden.
- Java ist ein modernisiertes C++. Diese Sprache hatte sich zuvor zum *Industriestandard* entwickelt, daher gibt es viele Programmierer, die ohne großen Aufwand auf Java umsteigen können. Java ist weniger komplex als C++, verbietet den unkontrollierten Umgang mit Zeigern und verkörpert moderne Konzepte objektorientierter Programmierung.
- Java hat sich an Universitäten verbreitet, weil in dieser Sprache viele der Konzepte enthalten sind, die Sprachen wie Pascal, Modula und Oberon für die Lehre so populär gemacht haben. Anders als die vorgenannten Sprachen konnte sich Java aber auch in der Praxis durchsetzen.
- Java Entwicklungsumgebungen von hoher Qualität sind zum Teil kostenlos verfügbar. Hinter Java stand zunächst die Firma SUN Microsystems, ein bedeutender Hersteller von Servern und Workstations. Mittlerweile wurde SUN von der Firma Oracle übernommen. Oracle engagiert sich aber genau so wie SUN für Java. Das *Java Development Kit* (JDK), ein Java-Interpreter (Java Virtual Machine), Werkzeugen und Dokumentationen zu Java werden von SUN/Oracle entwickelt und im Internet bereitgestellt. Derzeit (Mitte 2012) ist die Version 1.7 des JDK aktuell. Diese Version wird seit einiger Zeit als JDK 7 bezeichnet. Mit der Einführung von JDK 7 war die Einführung neuer Konzepte (u.a. Closures) erwar-

tet worden, auf die wir am Ende des Kapitels näherer eingehen. Die Einführung von closures wurde jedoch auf eine spätere Version verschoben.

- Neu in der Version JDK 7 ist eine Überarbeitung der Ein- und Ausgabe von Dateien, eine Schnittstelle zur parallelen Nutzung mehrerer Prozessoren (Fork/Join Framework) sowie viele weitere Verbesserungen.

Java wurde in der Vergangenheit in drei verschiedenen Ausgaben angeboten, einer Standard-Edition *SE*, einer Micro-Edition *ME* und einer Enterprise-Edition *EE*. Bereits SUN hatte eigenständige Entwicklung der Micro-Edition aufgegeben und diese in die Standard-Version integriert, Oracle hat offenbar ähnliche Pläne für die Enterprise-Edition. Derzeit sind kostenlos erhältliche Programmierumgebungen wie *NetBeans* und *Eclipse* populär – es handelt sich um sehr große und umfangreiche Systeme. Schlanke, aber nicht minder geeignete Systeme sind *JCreator* und das *BlueJ* System. Einfach ist auch die Integration von Java in einen frei erhältlichen Editor wie z.B. NotePad++. In allen Fällen ist aber Vorraussetzung, dass das bei *www.oracle.com* erhältliche Java-Development-Kit (JDK) installiert ist.

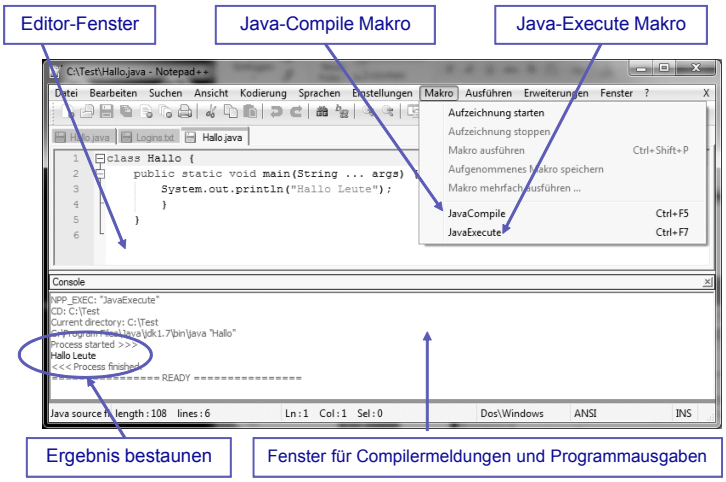

Abb. 3.1: Entwicklung von Java Programmen mit NotePad++

Den vielen Vorteilen von Java steht allerdings auch ein kleiner Nachteil gegenüber: Die *Portabilität* wird durch eine interpretative Programmausführung erreicht. Das bedeutet einen Verzicht auf optimale Programmlaufzeiten. Die Messergebnisse für die Laufzeit von Sortieralgorithmen im nächsten Kapitel belegen jedoch, dass dieser Nachteil auf modernen Prozessoren geringer ausfällt als vermutet.

Microsoft hat der Sprache Java und der zugehörigen Bibliotheks-Infrastruktur eine eigene Sprache, C# mit der sogenannten .NET-Platform, gegenübergestellt. Für den Benutzer sind Java und C# zunächst sehr ähnliche Sprachen, allerdings ist die Weiterentwicklung von Java in den letzten Jahren merklich ins Stocken geraten, so dass gegenwärtig C# viele moderne Konzepte implementiert, die Java derzeit noch fehlen.

3.1 Die lexikalischen Elemente von Java

Die meisten Programmiersprachen basieren auf dem weit verbreiteten ASCII-Zeichensatz. Landesspezifischen Zeichen, wie z.B. ö, ß, æ, ç oder Ã, sind dabei nicht zugelassen. Da alle ASCII-Zeichensätze 7 oder 8 Bit für die Darstellung eines Zeichens verwenden, ist die Anzahl der codierbaren Zeichen auf 256 beschränkt.

Java legt den neueren Zeichensatz *Unicode* zugrunde, der praktisch alle weltweit geläufigen Zeichensätze vereint, siehe auch S. 13. Die einzelnen Zeichen von Unicode werden durch *Attribute* als *Buchstaben* oder *Ziffern* klassifiziert. Aufbauend auf dieser Klassifikation kann man folgende Java-spezifische lexikalische Elemente definieren:

- ***Buchstaben*:** Alle in Unicode zulässigen Buchstaben und aus „historischen“ Gründen auch der Unterstrich „ _ “ sowie das Dollarzeichen „$“.
- Die ***Ziffern*** von 0 bis 9.
- ***Zeilenende*:** Eines der Zeichen Wagenrücklauf (CR=*carriage return*: ASCII-Wert 13) oder Zeilenwechsel (LF=*line feed*: ASCII-Wert 10) oder deren Kombination CR LF.
- ***Leerzeichen*** (*Whitespace*): Das Leerzeichen selbst (SP=*space*: ASCII-Wert 32), eines der folgenden *Steuerzeichen*: Tabulator (HT=*horizontal tabulator*: ASCII-Wert 9), Formularvorschub (FF=form feed: ASCII-Wert 9) oder ein Zeilenende.
- ***Trennzeichen*:** `( ) { } [ ] ; , .`
- ***Operatoren*:** `= > < ! ~ ? : == <= >= != && || ++ -- + - * / & | ^ % << >> >>> += -= *= /= &= |= ^= %= <<= >>= >>>=`
- ***Kommentare***, ***Bezeichner*** und ***Literale***.

3.1.1 Kommentare

Java kennt drei Arten von *Kommentaren*. Die erste Form beginnt mit `//` und erstreckt sich bis zum Ende der Zeile:

```
// Dieser Kommentar endet automatisch am Zeilenende
```

Die zweite Form des Kommentars kann sich über mehrere Zeilen erstrecken. Er besteht aus allen zwischen den Kommentarbegrenzern `/*` und `*/` stehenden Zeichen. Kommentare dieser Form dürfen nicht geschachtelt werden:

```
/*  Die folgende Methode berechnet
    eine schwierige Funktion
    auf die ich sehr stolz bin   */
```

Eine Variante beginnt mit `/**` und endet mit `*/`. Solche Kommentare werden von dem Zusatzprogramm *javadoc* erkannt, und in eine standardisierte Dokumentation übernommen. Durch zusätzliche Marken wie z.B. `@param`, `@result` sowie beliebige HTML-Formatierungsanweisungen kann der Benutzer die Strukturierung der Dokumentation beeinflussen. Die Dokumentation der Java-API (siehe Abb. 3.12) ist auf diese Weise automatisch erzeugt.

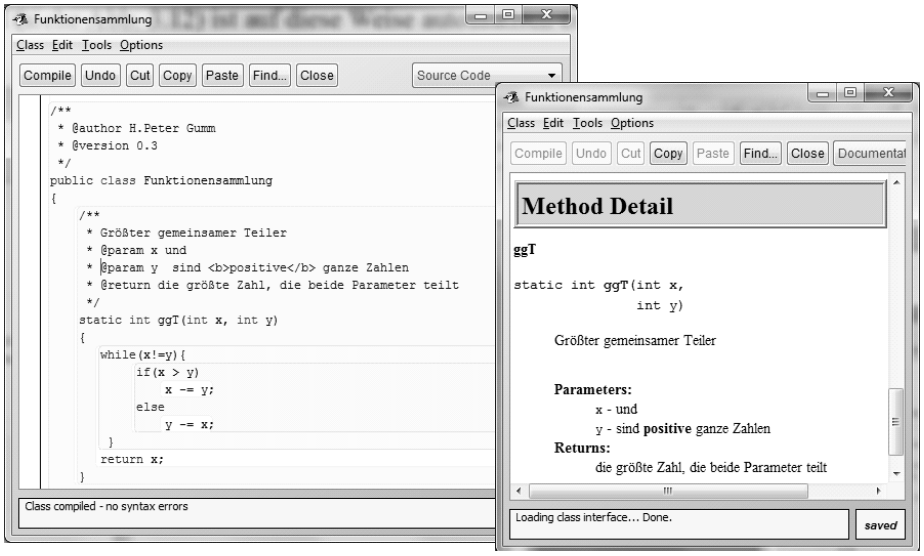

Abb. 3.2: JavaDoc-Kommentar und Teil der erzeugten HTML-Dokumentation (in BlueJ)

3.1.2 Bezeichner

Bezeichner beginnen mit einem Java-Buchstaben. Darauf können weitere Java-Buchstaben, Ziffern und Unterstriche folgen. Die Länge eines Bezeichners ist nur durch die maximal verwendbare Zeilenlänge begrenzt. Beispiele sind:

```
x y meinBezeichner Grüße Üzgür λ_halbe ελλασ èlmùt_çôl
```

Einige der Bezeichner haben eine besondere, reservierte Bedeutung und dürfen in keinem Fall anders verwendet werden. Dazu gehören die Schlüsselwörter der Programmiersprache Java und drei spezielle konstante Werte (Literale):

```
null, false, true.
```

Drei weitere besondere Bezeichner sind Namen vordefinierter Klassen:

```
Object, String, System.
```

Technisch gesehen könnte man diese Bezeichner auch mit einer anderen Bedeutung verwenden. Man sollte das aber vermeiden.

3.1.3 Schlüsselwörter

Die folgenden Bezeichner sind *Schlüsselwörter* der Programmiersprache Java:

```
abstract    assert      boolean    break      byte
case        catch       char       class      const
```

```
continue      default       do          double      else
enum          extends       final       finally     float
for           goto          if          implements  import
instanceof    int           interface   long        native
new           package       private     protected   public
return        short         static      strictfp    super
switch        synchronized  this        throw       throws
transient     try           void        volatile    while
```

Die Schlüsselwörter, `const` und `goto`, sind zwar reserviert, werden aber in den aktuellen Versionen von Java nicht benutzt. Das Schlüsselwort `enum` ist neu ab JDK 1.5.

3.1.4 Literale

Literale sind unmittelbare Darstellungen von Elementen eines Datentyps. 2, –3 und 32767 sind Beispiele für Literale vom Typ *int*. Insgesamt gibt es folgende Arten von Literalen:

- Ganzzahlige Literale,
- Gleitpunkt-Literale,
- boolesche Literale: `false` und `true`,
- die *Null-Referenz*: `null`,
- Literale für Zeichen und Zeichenketten.

Ganzzahlige Literale

Für ganze Zahlen verwendet Java die Datentypen *byte* (8 Bit), *short* (16 Bit), *int* (32 Bit) und *long* (64 Bit). Für *ganzzahlige Literale* sind neben der Standardschreibweise auch noch die oktale und die hexadezimale Schreibweise erlaubt. Letztere wird mit den Zeichen `0x` eingeleitet, danach können normale Ziffern (0,...,9) oder hexadezimale Ziffern (A,...,F oder a,...,f) folgen. Als Beispiel wird in der Java-Literatur gerne die dezimale Zahl –889275714 hexadezimal als `0xCafeBabe` oder als `0xCAFEBABE` notiert.

Ganzzahlige Literale bezeichnen normalerweise Werte des Datentyps *int*. Will man sie als Werte des Datentyps *long* kennzeichnen, so muss man das Suffix `L` (oder `l`) anhängen. Beispiele für ganzzahlige Literale sind:

```
2    17    -3    32767    0x1FF    4242424242L    0xC0B0L
```

Gleitpunkt-Literale

Die Datenformate für reelle Zahlen in Java sind *float* (floating point number, 32 Bit) und *double* (double precision number, 64 Bit). *Gleitpunkt-Literale* werden als Dezimalzahlen mit dem im Englischen üblichen Dezimalpunkt notiert. Optional kann ein ganzzahliger Exponent folgen. Gleitpunkt-Literale bezeichnen normalerweise Werte des Datentyps *double*. Durch Anhängen eines der Suffixe F oder D (bzw. f oder d) spezifiziert man sie explizit als Werte der Datentypen float oder double. Beispiele für Gleitpunkt-Literale des Datenyps *double* sind:

```
3.14    .3    2.    6.23e-10    3.7d    1E+137
```

Beispiele für Gleitpunkt-Literale des Datenyps *float*:

```
3.14f   .3f    2.f   6.23e-10f    3.7F    1E+38F
```

Zeichen-Literale

Ein *Zeichen-Literal* ist ein einzelnes, in einfache Arophe eingeschlossenes Unicode-Zeichen. Falls das eingeschlossene Zeichen selbst ein Aroph oder ein \ sein soll, muss eine der folgenden Ersatzdarstellungen, auch *Escape-Sequenzen* genannt, verwendet werden:

- `\b` für einen Rückwärtsschritt (BS=*backspace*: ASCII-Wert 8)
- `\t` für einen horizontalen Tabulator (HT)
- `\n` für einen Zeilenwechsel (LF)
- `\f` für einen Formularvorschub (FF)
- `\r` für einen Wagenrücklauf (CR)
- `\"` für ein "
- `\'` für ein '
- `\\` für ein \
- `\uxxxx` für ein Unicode-Zeichen. `xxxx` steht dabei für genau 4 hexadezimale Ziffern.

Beispiele für Zeichen-Literale:

```
'a'  '%'  '\t'  '\\'  '\''  '\"'  '\u03a9' '\uFFFF' '\177' 'α'
```

Zeichenketten-Literale

Zeichenketten-Literale (meist *String-Literale* genannt) sind Folgen von Unicode-Zeichen, die in doppelte Anführungszeichen eingeschlossen sind. Falls ein Zeichen des Strings selber ein doppeltes Anführungszeichen oder ein \ sein soll, verwendet man eine der oben angegebenen Ersatzdarstellungen. Ein String-Literal muss auf der gleichen Zeile beginnen und enden. Längere Strings erhält man durch Konkatenation (Verkettung) von Strings mit dem +-Operator:

```
"Hallo Welt !"
"Erste Zeile \n zweite Zeile \n dritte Zeile."
"Dieser String passt nicht in eine Zeile, daher"
+ "wurde er mit \"+\" aus zwei Teilen zusammengesetzt."
```

3.2 Datentypen und Methoden

Wie bei vielen höheren Programmiersprachen gibt es auch in Java *einfache* und *strukturierte Datentypen*. Die strukturierten Datentypen werden auch als Referenzdatentypen bezeichnet.

***Einfache Datentypen*:**
boolean, *char* und die numerischen Datentypen: *byte*, *short*, *int*, *long*, *float*, *double*.