Beispielprogramme und Ausführer

Ein ergänzendes Kapitel zum Buch „Java ist eine Sprache“ (von Ulrich Grude, illustriert von Boris Schaa), welches Anfang 2005 im Vieweg-Verlag erscheinen ist.

Dieses Kapitel wurde ergänzt/korrigiert/verändert an folgenden Zeitpunkten:

10.10.05 DrJava hinzugefügt, BeanShell neue Version.

Inhaltsverzeichnis

27 Beispielprogramme und Ausführer

Dieses Papier ist eine Ergänzung zum Buch „Java ist eine Sprache“ (von U. Grude, Vieweg Verlag, 2005). Dieses „27. Kapitel des Buches“ enthält Informationen, die vermutlich häufiger auf den neusten Stand gebracht werden müssen, als die übrigen Kapitel. Deshalb ist es nicht als Teil des Buches abgedruckt, sondern kann von der *Netzseite zum Buch* von folgender Adresse heruntergeladen werden:

`www.tfh-berlin.de/~grude/JavaIstEineSprache/Kap27Ausfuehrer.pdf`

In diesem Papier wird folgendes vorausgesetzt und angenommen:

Sie haben einen PC, der unter Linux oder unter einem Windows-Betriebssystem läuft und mit dem Internet verbunden ist. Ihnen ist das Buch „Java ist eine Sprache“ zugänglich und Sie haben mindestens das erste Kapitel („Einleitung“) bereits gelesen. Jetzt wollen Sie die Java-Beispielprogramme, die zu dem Buch gehören auf Ihren PC laden, einige davon ansehen und ausprobieren sowie selbst Java-Programme entwickeln und ausführen lassen. Sie wissen, dass Sie dazu einen *Java-Ausführer* benötigen und wollen einen solchen auf Ihrem PC einrichten.

27.1 Die Beispielprogramme herunterladen

Eine umfangreiche Sammlung von Beispielprogrammen (zum Buch „Java ist eine Sprache“) können Sie von der folgenden Adresse herunterladen:

`www.tfh-berlin.de/~grude/JavaIstEineSprache/BspJaSp.zip`

Das `zip`-Archiv `BspJaSp.zip` enthält ein Verzeichnis (oder: einen Ordner) namens `BspJaSp`. Darin finden Sie etwa 400 Java-Quelldateien (z. B. `Abbildungen01.java`, `Abbildungen02.java`, ..., `Hallo01.java`, `Hallo02.java`, ... etc.), ein paar `.bat`-Dateien (Windows-Kommando-Dateien, z. B. `Alle.bat`, `K10TstA.bat` etc.) und ein paar Unterverzeichnissen. Das `zip`-Archiv muss mit einem geeigneten Programm entpackt werden (z. B. mit `WinZip` oder `gunzip` etc.). Dabei sollten die Unterverzeichnisse nicht zerstört werden, sondern erhalten bleiben.

Empfehlung: Wenn man eines der Beispielprogramme (z. B. `Hallo01.java`) ausprobieren möchte, sollte man ein weiteres Verzeichnis z. B. mit dem Namen `Versuche` erzeugen und die Datei `Hallo01.java` dort hinein kopieren. Dann erst sollte man sie mit einem Compiler übersetzen (siehe unten) oder mit einem Quelltext-Interpreter direkt ausführen lassen (siehe unten). Wenn eine Quelldatei andere Quelldateien benutzt, sollte man auch diese Dateien in das Verzeichnis `Versuche` kopieren (z. B. benutzt die Quelldatei `Hallo05.java` die Quelldateien `Hallo03.java` und `Hallo04.java` und viele Quelldateien benutzen die Quelldatei `EM.java`). Die Quelldateien direkt im Verzeichnis `BspJaSp` auszuprobieren ist nicht empfehlenswert, weil es unübersichtlich viele Dateien enthält.

27.2 Überblick über verschiedene Java-Ausführer

Unter dem Begriff *Java-Ausführer* wird hier alles zusammengefasst, was man zum Schreiben und Ausführen von Java-Programmen benötigt. Ein Java-Ausführer kann z. B. aus einem *PC* (unter Linux oder Windows), einem *Editor* (zum Schreiben von Quelldateien), einem *Compiler* (zum Übersetzen von Quelldateien in Bytecodedateien) und einem *Bytecode-Interpreter* (zum Ausführen der Bytecodedateien) bestehen. Ein einfacherer Java-Ausführer besteht aus einem *PC* (unter Linux oder Windows), einem *Editor* und einem *Quelltext-Interpreter* (der Quelldateien direkt ausführen kann, ohne sie vorher zu übersetzen).

Häufig werden wir stillschweigend voraussetzen, dass ein PC unter Linux oder Windows vorhanden ist und werden nur die zusätzlich notwendigen Programme und Dateien als *Java-Ausführer* bezeichnen. Solche Ausführer kann man aus dem Internet herunterladen. Einen vollständigen Ausführer, der auch Hardware umfasst, kann man heute leider noch nicht direkt aus dem Internet laden :-).

Zuerst soll hier ein allgemeiner *Überblick* über einige besonders interessante Java-Ausführer gegeben werden, die im Internet kostenlos verfügbare sind. Dabei werden Fragen behandelt wie: Aus welchen Programmen besteht der Ausführer? Wie funktioniert er im Prinzip? Wodurch unterscheidet er sich grundsätzlich von anderen Java-Ausführern? In den folgenden Abschnitten (ab 27. 3) werden die Programme, aus denen die Ausführer bestehen etwas genauer beschrieben. Dort finden Sie zu jedem der Programme:

- Eine Internet-Adresse, von der Sie die benötigten Installationsdateien kostenlos herunterladen können.
- Hinweise zur Installation des Programms
- Hinweise zur Benutzung des Programms

27.2.1 Der Ausführer (javac, java) der Firma Sun

Die Firma Sun stellt allen Interessierten eine *Java-Entwicklungsumgebung* (engl. java development kit, JDK) zur Verfügung (in unterschiedlichen Versionen für Linux, für Windows und für weitere Betriebssysteme). Diese Entwicklungsumgebung enthält unter anderem einen *Java-Compiler* namens `javac` und einen *Java-Bytecode-Interpreter* namens `java`. Mit dem Compiler kann der Programmierer *Java-Quelldateien* dem Ausführer übergeben, prüfen und in sogenannte *Bytecodedateien* übersetzen lassen. Mit dem Interpreter kann der Benutzer Bytecodedateien *ausführen* lassen. Das Programmpaar (`javac`, `java`) wird im Folgenden einfach als *der Java-Ausführer von Sun* bezeichnet.

Wenn man nur fertige Bytecodedateien ausführen lassen möchte (aber keine Java-Quelldateien compilieren will), braucht man keine vollständige Entwicklungsumgebung (`JDK`), sondern nur eine Java *Ausführungsumgebung* (engl. Java runtime environment, `JRE`). Eine solche Ausführungumgebung (`JRE`) ist in jeder Entwicklungsumgebung (`JDK`) enthalten, wird von der Firma Sun aber auch als separate Einheit zum kostenlosen Herunterladen angeboten.

Mehrere Java-Ausführer (z. B. `DrJava`, `Eclipse` und `BeanShell`) basieren auf der Entwicklungsumgebung von Sun und ergänzen sie um weitere nützliche Programme.

27.2.2 Der Ausführer DrJava

Das Compilieren (Übersetzen) von Quellprogrammen hat Vor- und Nachteile: Es *beschleunigt* die Ausführung von Programmen, macht aber ihre Entwicklung komplizierter und schwerer zu durchschauen. Während man eine Programmiersprache *lernt* ist die Ausführungsgeschwindigkeit meist unwichtig, aber eine Vereinfachung des Entwicklunsvorganges besonders wünschenswert. Das Programm `DrJava` stellt dem Benutzer unter anderem einen *Quelltext-Interpreter* zur Verfügung, der einzelne Java-Befehle oder ganze Java-Quellprogramme direkt ausführen kann (ohne dass man sie vorher compilieren muss).

Außerdem ist DrJava eine *Integrierte Programm-Entwicklungsumgebung* (engl. IDE, integrated development environment), in der man Java-Quellprogramme wie üblich mit einem Editor erstellen, mit einem Compiler in Java-Bytecodedateien übersetzten und die Bytecodedateien mit einem Interpreter ausführen lassen kann.

Das Programm DrJava ist selbst in Java geschrieben und erwartet, dass man eine vollständige Entwicklungsumgebung (JDK) von Sun bereits installiert hat (nicht nur eine Ausführungsumgebung, JRE). DrJava kann wahlweise mit der neusten Version von Java (Java 6) oder mit einer älteren Version (Java 1.4) zusammenarbeiten.

27.2.3 Der Ausführer BeanShell

Das Programm BeanShell ist (ähnlich wie DrJava) ein Java-Quelltext-Interpreter, der einzelne Java-Befehle oder ganze Java-Quellprogramme direkt ausführen kann (ohne dass man sie vorher compilieren muss). Die BeanShell (Kurzname: bsh) ist aber (anders als DrJava) nur ein Quelltext-Interpreter und zur Zeit (Oktober 2005) noch nicht auf die Java-Version 5.0 umgestellt, weshalb man zur Zeit DrJava vorziehen sollte. Auch das Programm BeanShell ist selbst in Java geschrieben und erwartet, dass man eine Ausführungsumgebung (JRE, z. B. die von Sun) bereits installiert hat.

27.2.4 Der Ausführer Eclipse

Das Programm Eclipse (entwickelt als quelloffene Software unter Leitung von IBM) ist eine integrierte Entwicklungsumgebung für die Erstellung von integrierten Entwicklungsumgebungen und für verschiedene Sprachen (vor allem für Java, aber auch für C++ und weitere Sprachen). Eclipse ist ein sehr leistungsfähiges, aber auch ziemlich kompliziertes Werkzeug und wird hier nicht weiter behandelt. Weitere Informationen findet man unter der Adresse www.eclipse.org.

27.2.5 Die Editoren JEdit und JavaEditor

Java-Quellprogramme kann man im Prinzip mit jedem beliebigen Texteditor erstellen. Die Entwicklung eines Programms kann aber durch einen guten Editor erheblich vereinfacht und erleichtert werden.

Die Programme JavaEditor und JEdit sind zwei sehr leistungsfähige Editoren (und mehr), die sich zum Schreiben und Entwickeln von Java-Programmen besonders gut eignen. Der JavaEditor ist auf Java-Programme spezialisiert, mit dem JEdit kann man auch Programme in anderen Sprachen schreiben. Beide Editoren erwarten, dass eine Java-Entwicklungsumgebung von Sun (JDK) bereits installiert ist. Für Anfänger ist der JavaEditor besser geeignet, er läuft allerdings nur unter Windows. Der JEdit ist selbst in Java geschrieben (wie die Programme DrJava und BeanShell) und läuft somit überall, wo ein Java-Bytecode-Interpreter zur Verfügung steht (unter Linux, Windows und anderen Betriebssystemen).

27.2.6 Der Java-Ausführer (gcj, gij) des Gnu-Projekts

Die *Free Software Foundation* (siehe www.fsfeurope.org) ist ein weltweiter Verband, der für die Idee *quelloffener Software* wirbt und (seit 1984) im Rahmen des GNU-Projekts (siehe www.gnu.org) solche Software entwickelt und allen Interessierten kostenlos zur Verfügung stellt (einschliesslich der Quellprogramme, die man lesen und abändern kann). Im Rahmen des GNU-Projekts (GNU spricht man im Englischen GU-NU, mit zwei Us wie in Uhu) wurde unter anderem eine umfangreiche Sammlung von Ausführern für Sprachen wie Fortran, Pascal, C, C++, Ada, Java etc. entwickelt (gcc, GNU Compiler Collection). Der Java-Ausführer in dieser Sammlung besteht (ganz analog zum Java-Ausführer von Sun) aus einem Compiler gcj (GNU Compiler for Java) und einem Bytecode-Interpreter gij (GNU Interpreter for Java). Mit dem Compiler kann der Programmierer *Java-Quelldateien* (die er mit einem Editor geschrieben hat) dem Ausführer übergeben, prüfen und in sogenannte *Bytecodedateien* übersetzen lassen. Mit dem Interpreter kann der Benutzer Bytecodedateien *ausführen* lassen.

Der Compiler gcj hat zusätzlich noch eine weitere interessante Fähigkeit: Er kann Quelldateien auch direkt in sogenannte *ausführbare Dateien* (unter Windows: in exe-Dateien) übersetzen. Diese Dateien können von der betreffenden Plattform (PC unter Windows oder PC unter Linux) direkt und ohne die Hilfe eines Interpreterprogramms ausgeführt werden, können aber *nicht* auf eine andere Plattform (z. B. von Linux nach Windows oder umgekehrt) übertragen werden.

Die *Cygwin-Distribution* besteht aus zahlreichen quelloffenen Programmen, die aus der Unix/Linux-Welt in die Windowsenclave portiert wurden, und seitdem auch allen Windows-Benutzern zur Verfügung stehen. Zu dieser Distribution ge-

hören auch die zahlreichen Sprach-Ausführer des Gnu-Projekts und insbesondere der Java-Ausführer (`gcj`, `gij`).

27.2.7 Der Editor TextPad

Der `TextPad` ist einer der besseren Editoren für Textdateien und auch als Entwicklungsumgebung für Java-Programme recht gut geeignet. Leider läuft er nur unter Windows. Er wird als Shareware vertrieben, man kann ihn also beliebig lange ausprobieren, bevor man ihn bezahlt. Eine Einzellizenz kostet etwa 25 Euro.

Allgemeine Empfehlungen:

Installieren Sie zuerst die neuste Java-Entwicklungsumgebung (JDK) der Firma Sun auf Ihrem PC (je nach Ihrem Betriebssystem die Version für Linux oder die für Windows). Dadurch wird automatisch auch eine Java Ausführungsumgebung (JRE) installiert.

Falls auf Ihrem PC bereits eine ältere Version dieser Entwicklungsumgebung installiert ist, sollten Sie sorgfältig darauf achten, dass keine Konflikte zwischen der alten und der neuen Version auftreten. Es ist durchaus möglich, mehrere Versionen des Ausführers nebeneinander zu betreiben, aber Anfängern wird empfohlen, ältere Versionen zu löschen und erst dann die neue Version zu installieren (um so *eine* Quelle möglicher Probleme auszuschalten).

Bevor Sie weitere Teile eines Java-Ausführers (z. B. einen Editor) installieren, sollten Sie testen, ob der Ausführer (`javac`, `java`) richtig funktioniert. Tips dazu findet man im nächsten Abschnitt.

Wenn Sie den `JavaEditor` oder den `TextPad` (beide nur unter Windows) *nach* der Java-Entwicklungsumgebung (JDK) von Sun installieren, „verbinden sie sich“ automatisch mit der Java-Entwicklungsumgebung (d.h. Sie können Dateien compilieren und Programme ausführen lassen, ohne den Editor zu verlassen). Wenn Sie den `JavaEditor` oder den `TextPad` *vor* der Entwicklungsumgebung (JDK) installieren, müssen Sie die Verbindung „von Hand“ herstellen.

27.3 Die Java-Entwicklungsumgebung von Sun

Die Installationsdateien für die Java-Entwicklungsumgebung von Sun finden Sie unter der folgenden Netzadresse:

`java.sun.com/javase/downloads/index.jsp`

Relevant sind vor allem die folgenden Dateien:

JDK 6u1 (der Java Development Kit Version 6 update 1)

```
jdk-6u1-windows-i586-p.exe (für Windows, ca. 56 MB) oder
jdk-6u1-linux-i586-rpm.bin (für Linux,   ca. 58 MB)
```

Java SE 6 Documentation (die Online-Dokumentation)

```
jdk-6-doc.zip    (englische  Version, ca. 52 MB) oder
jdk-6-doc-ja.zip (japanische Version, ca. 57 MB)
```

Die Installationsdatei für Linux ist ein *selbstextrahierendes Archiv*, man muss diese Datei also nur ausführen lassen und alles weitere passiert dann (mehr oder weniger) automatisch.

Die Installationsdatei für Windows funktioniert ähnlich, auch sie muss man nur ausführen lassen.

Die *Dokumentation* der Entwicklungsumgebung, umfasst insbesondere die Dokumentation der *Java Standardbibliothek*. Es handelt sich dabei um ein `zip`-Archiv, welches ein System von HTML-Dateien enthält.

27.3.1 Den Java-Ausführer (javac, java) von Sun installieren

Lassen Sie zuerst die Installationsdatei (`jdk-6u1-linux-i586-rpm.bin` bzw. `jdk-6u1-windows-i586-p.exe`) ausführen. Daraufhin wird Ihnen ein Installationsverzeichnis vorgeschlagen. Das können Sie akzeptieren oder ändern.

Empfehlung: Der vorgeschlagene Name des Installationsverezeichnisses enthält einen Hinweis auf die installierte Version von Java. Ändern Sie den Namen zu `Java`, so dass er *keinen* solchen Hinweis mehr enthält. Erst wenn Sie später eine neuere Version zusätzlich installieren wollen, sollten Sie das Verzeichnis `Java` umbenennen, z.B. zu `Java_6.0`, und die neue Version (`6.1` oder `7.0` etc) wieder in ein Verzeichnis namens `Java` installieren. Alle Editoren und anderen Programme, sollten immer auf das Verzeichnis `Java` (und somit auf die neuste installierte Version) zugreifen.

Im Folgenden wird das Verzeichnis, in das Sie die Entwicklungsumgebung installiert haben, mit JAVA_HOME bezeichnet. Unter Windows könnte JAVA_HOME z. B. gleich c:\Programme\Java sein. Im Verzeichnis JAVA_HOME sollten Sie nach der Installation der Entwicklungsumgebung etwa folgende Unterverzeichnisse finden:

```
JAVA_HOME
   jdk1.6.0
      bin
      db
      demo
      include
      jre
      lib
      sample
   jre1.6.0
      bin
      lib
```

Prüfen Sie, ob (unter Windows) im Verzeichnis JAVA_HOME\bin unter anderem die Programme javac.exe (der Java-Compiler) und java.exe (der Bytecode-Interpreter) stehen.

Unter Linux sollten entsprechend im Verzeichnis JAVA_HOME/bin die Programme javac und java stehen.

Empfehlung: Tragen Sie das Verzeichnis JAVA_HOME/bin in die Umgebungsvariable PATH ein. Auch unter Windows können Sie dabei normale Schrägstriche / anstelle von Rückwärtsschrägstrichen \ verwenden. Falls Sie nicht wissen, wie man ein Verzeichnis in die PATH-Variable einträgt, sollten Sie sich von jemandem beraten lassen, der es weiss. Unterschiedliche Versionen von Windows erfordern unterschiedliche Schritte um eine Umgebungsvariable wie PATH zu verändern. Grundsätzlich gilt (unter Linux und Windows): Wenn der Benutzer versucht, ein bestimmtes Programm zu starten, etwa das Programm javac mit einem Kommando wie

```
> javac --help
```

dann sucht das Betriebssystem nach einer entsprechenden ausführbaren Datei, aber nur in *den* Verzeichnissen, die in der Umgebungsvariablen PATH eingetragen sind.

27.3.2 Die Installation des Ausführers testen

Nachdem Sie die Entwicklungsumgebung von Sun (im Kern den Ausführer (`javac`, `java`)) installiert haben, sollten Sie die folgenden beiden Tests durchführen:

Test 1:

Öffnen Sie ein Kommandoeingabefenster (unter Windows eine DOS-Eingabeaufforderung, unter Linux eine Shell) und geben Sie das folgende Kommando ein:

```
> java -version
```

Eine dreizeilige Meldung (die mit `java version "1.6.0_01"` oder einem ähnlichen Text beginnt) sollte ausgegeben werden. Falls statt dessen eine Fehlermeldung erscheint die erkennen lässt, dass das Programm `java` nicht gefunden wurde, können Sie probieren, den absoluten Pfadnamen des Programms `java` anzugeben, etwa so:

```
> JAVA_HOME\bin\java -version
```

Anstelle von `JAVA_HOME` müssen Sie dabei den entsprechenden Pfad angeben. Falls dieser Versuch gelingt, sollten Sie die Umgebungsvariable `PATH` überprüfen. Wahrscheinlich müssen Sie das Verzeichnis `JAVA_HOME/bin` dort noch eintragen, oder dafür sorgen, dass die Eintragung *wirksam* wird (indem Sie das Kommandoeingabefenster schliessen und wieder öffnen oder Ihren Rechner neu starten). Falls dieser Versuch mit dem absoluten Pfadnamen nicht gelingt, sollten Sie eine Fachperson zu Rate ziehen.

Wenn Sie den Java-Bytecode-Interpreter `java` erfolgreich aufgerufen haben, können Sie sich mit folgendem Befehl ein weiteres Erfolgserlebnis verschaffen:

```
> javac -help
```

Der Compiler `javac` wird dann sehr wahrscheinlich eine kurze Bedienungsanleitung ausgeben (ca. 20 Zeilen).

Ende von Test 1.

Anmerkung: *Technische* Mitarbeiter der Firma Sun unterscheiden die folgenden Java-Versionen: 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6. Mitarbeiter der *Marketing*-Abteilung von Sun unterscheiden dagegeben die Versionen 1.0, 1.1, 2, 5 und 6. Die „technische Version“ 1.6 ist identisch mit der „Marketing-Version“ 6 und eine Version wie 1.6.0 ist identisch mit 6.0.0 etc.

Test 2:

Versuchen Sie, das Beispielprogramm `Hallo01.java` (siehe 27.1) oder ein ähnlich simples Programm dem Ausführer zu übergeben und ausführen zu lassen, etwa mit den folgenden Schritten:

1. Kopieren Sie die Quelldatei `Hallo01.java` in ein leeres Verzeichnis, z. B. in das Verzeichnis `d:\Versuche` (bzw. `/Versuche`).

2. Öffnen Sie ein Kommandoeingabefenster (unter Windows eine DOS-Eingabeaufforderung, unter Linux eine Shell) und machen Sie das Verzeichnis `d:\Versuche` (bzw. `/Versuche`) zu Ihrem aktuellen Arbeitsverzeichnis.

3. Überzeugen Sie sich mit dem Kommando `dir` (bzw. `ls`) davon, dass die Datei `Hallo01.java` sich wirklich im aktuellen Arbeitsverzeichnis befindet.

4. Übergeben Sie das Programm dem Ausführer (d.h. compilieren Sie es) mit dem folgenden Kommando:

```
1   d:\Versuche> javac Hallo01.java
```

Die Nummerierung der Kommandozeilen soll ihre Erläuterung erleichtern, gehört aber nicht zu den Kommandos. Die Zeichenkette `d:\Versuche` stellt das aktuelle Arbeitsverzeichnis dar.

Achtung: Auch unter Windows wird bei solchen Kommandos zwischen Groß- und Kleinschreibung unterschieden! Z. B. sind `hallo` und `Hallo` verschiedene Namen.

Wenn das Kommando in Zeile 1 keine Fehlermeldungen auslöst, hat der Ausführer das Programm `Hallo01` akzeptiert und im aktuellen Arbeitsverzeichnis eine Bytecodedatei namens `Hallo01.class` erzeugt. Danach können Sie ihm mit folgendem Kommando befehlen, das akzeptierte Programm `Hallo01` (in der Datei `Hallo01.class`) auszuführen:

```
2   d:\Versuche> java Hallo01
```

Dieses Kommando sollte die folgenden beiden Ausgabezeilen produzieren:

```
Hallo Welt!
Wie geht es?
```

Ende von Test 2.

Empfehlung: Verändern Sie die Quelldatei `Hallo01.java` ein bisschen, übergeben Sie sie wieder dem Ausführer und lassen Sie die neue Version des Programms `Hallo01` ausführen. Verändern Sie die Quelldatei auch so, dass der Ausführer das Programm *ablehnt* und Fehlermeldungen ausgibt (indem Sie z. B. eine

öffnende oder eine schliessende Klammer entfernen oder das Wort `String` durch `StringX` ersetzen etc.). Versuchen Sie, die Fehlermeldungen zu verstehen. Ein englisches Wörterbuch in Griffweite oder eine Verbindung zu einem Lexikon im Netz (z.B. `www.leo.org`) sind dabei möglicherweise nützlich.

27.3.3 Die HTML-Dokumentation installieren

Zur Abrundung der Installation sollten Sie die HTML-Dokumentation der Java-Standardbibliothek (und der Werkzeuge von Sun etc.) installieren. Erstellen Sie dazu im Verzeichnis `JAVA_HOME` ein weiteres Unterverzeichnis namens `docs` und entpacken Sie da hinein das Archiv `jdk-6-doc.zip`. Danach sollte das Verzeichnis `JAVA_HOME` und das Unterverezeichnis `docs` etwa wie folgt aussehen:

```
JAVA_HOME
   jdk1.6.0
      bin
      db
      demo
      docs
         api
            ...
            index.html
            ...
         guide
         images
         ...
         index.html
         ...
      include
      jre
      ...
```

Das neue Unterverzeichnis `docs` sollte insbesondere eine Datei namens `index.html` enthalten, mit der die sehr umfangreiche Dokumentation der Java-Standardbibliothek (insgesamt ca. 100 MB) beginnt. Diese `html`-Datei (und damit die gesamte Dokumentation) kann man sich mit einem Webbrowser (z.B. FireFox, Opera, InternetExplorer, ... etc.) oder einem ähnlichen Programm ansehen.

Empfehlung (unter Windows): Legen Sie sich eine *Verknüpfung* mit der Datei `index.html` im Verzeichnis `JAVA_HOME\docs` auf den Desktop (indem Sie die Datei bei gedrückter rechter Maustaste über eine freie Stelle des Desktop ziehen, dort die rechte Maustaste loslassen und im daraufhin sich öffnenden Menü den Punkt `Verknüpfung hier erstellen` mit der linken Maustaste anklicken). Ein Doppelklick auf das Verknüpfungssysmbol öffnet Ihnen dann ganz einfach und schnell die Dokumentation der Java-Standardbibliothek. Beim Programmieren braucht man häufig nur *den* Teil der Dokumentation, der mit der `html`-Datei `JAVA_HOME\docs\api\index.html` beginnt, so dass es sich lohnt, eine Verknüpfung mit dieser Datei auf dem Desktop anzulegen (als Ersatz für die erste Verknüpfung oder zusätzlich). Unter Linux kann man ganz ähnlich vorgehen.

Nachdem Sie die Entwicklungsumgebung von Sun (im Kern also den Java-Ausführer (`javac`, `java`)) installiert und erfolgreich getestet haben, können Sie ihn durch die Installation weiterer Programme ausbauen und komfortabler gestalten. Die Reihenfolge der weiteren Schritte können Sie frei bestimmen, aber den Ausführer von Sun sollten Sie immer *zuerst* installieren.

27.3.4 Der Ausführer (javac, java) für Fortgeschrittene (ohne Pakete)

Einfache Beispielprogramme (siehe 27.1) kann man mit dem Java-Ausführer (`javac`, `java`) der Firma Sun bearbeiten wie im Abschnitt 27.3.2 anhand des Programms `Hallo01` beschreiben wurde. Falls mehrere Klassen zu einem Programm gehören (z. B. die Klassen `Hallo03`, `Hallo04` und `Hallo05`) sollte man ihre Quelldateien (`Hallo03.java`, `Hallo04.java` und `Hallo05.java`) in ein gemeinsames Verzeichnis kopieren und dort bearbeiten.

Erst wenn man mit *Paketen* arbeitet (siehe das Kapitel 17 im Buch „Java ist eine Sprache“) muss man sich mit bestimmten „Gewohnheiten und Fähigkeiten“ des Ausführers (`javac`, `java`) vertraut machen, die in diesem und dem folgenden Abschnitt erläutert werden.

In diesem und dem folgenden Abschnitt sind mit *Klassen* immer *Klassen und/oder Schnittstellen* gemeint, die der Programmierer vereinbart hat (keine Standardklassen wie `java.lang.String` oder `java.util.ArrayList`).

In diesem Abschnitt betrachten wir das Beispielprogramm `Hallo05` (aus der im Abschnitt 27.1 beschriebenen Sammlung), welches aus drei Klassen besteht, einer Hauptklasse `Hallo05` und zwei Nebenklassen `Hallo03` und `Hallo04`. Alle drei Klassen gehören zum *namenlosen Paket*. (d. h. ihre Quelldateien beginnen *nicht* mit einem `package`-Befehl). Jede Klasse ist in einer eigenen Quelldatei (`Hallo03.java`, `Hallo04.java` bzw. `Hallo05.java`) vereinbart. Das Programm `Hallo05` soll mit dem Compiler `javac` compiliert und mit dem Bytecode-Interpreter `java` ausgeführt werden. Je nachdem, *wo* die drei Quelldateien abgelegt wurden, sind dazu unterschiedliche Kommandos erforderlich.

Beispiel-01: Alle Quelldateien stehen im aktuellen Arbeitsverzeichnis `d:\Aka`

```
3   d:\Aka> javac Hallo05.java
4   d:\Aka> java  Hallo05
```

Die Kommandos werden hier so angegeben, wie man sie unter Windows z. B. in einer Dos-Eingabeaufforderung eingeben sollte. Die entsprechenden Linux-Kommandos sehen ganz ähnlich aus, meist muss man nur die Laufwerksbuchstaben wie

`d:` weglassen (auf weitere Unterschiede wird im Text kurz hingewiesen). Auch hier soll die Nummerierung der Kommandozeilen ihre Erläutern erleichtern, gehört aber nicht zu den Kommandos. Am Anfang jeder Kommandozeile wird das aktuelle Arbeitsverzeichnis angezeigt (im Beispiel-01: `d:\Aka`).

Dem Compiler `javac` muss man (in Zeile 1) nur die Quelldatei `Hallo05.java` der Hauptklasse angeben. Beim Prüfen dieser Datei erkennt er, dass die Hauptklasse unter anderem von der Klasse `Hallo03` abhängt. Daraufhin sucht er (im aktuellen Arbeitsverzeichnis) nach einer *Bytecodedatei* `Hallo03.class` *und* nach einer *Quelldatei* `Hallo03.java`. Wenn er keine der beiden findet, meldet er einen Fehler. Wenn er nur die *Bytecodedatei* findet, nimmt er sie zum Prüfen der Klasse `Hallo05` („Sind alle Abhängigkeiten zwischen `Hallo05` und `Hallo03` in Ordnung?“). Wenn er nur die *Quelldatei* `Hallo03.java` findet, übersetzt er sie automatisch in eine Bytecodedatei `Hallo03.class` und nimmt die für seine Prüfungen.

Aufgabe-01: Was sollte der Compiler Ihrer Ansicht nach machen, wenn er beim Suchen nach der Klasse `Hallo03` eine Bytecodedatei `Hallo03.class` *und* eine Quelldatei `Hallo03.java` findet? Seien Sie ruhig anspruchsvoll (aber erwarten Sie nicht, dass der Compiler Ihnen einen Kaffee kocht). Was der Compiler tatsächlich macht, findet man am Ende dieses Abschnitts (oder in der Dokumentation [HTML_Doc] im Verzeichnis `tooldocs`).

Ganz so, wie mit der Klasse `Hallo03`, verfährt der Compiler auch mit jeder anderen Klasse, von der die Hauptklasse `Hallo05` abhängt (im Beispiel ist das nur noch die Klasse `Hallo04`).

Nachdem der Compiler das Kommando in Zeile 1 erfolgreich ausgeführt hat, stehen im aktuellen Arbeitsverzeichnis `d:\Aka` drei Bytecodedateien `Hallo03.class`, `Hallo04.class` und `Hallo05.class`.

In Zeile 2 darf man dem Interpreter nur den *Namen der Hauptklasse* (`Hallo05`) angeben. Der Interpreter sucht daraufhin (im aktuellen Arbeitsverzeichnis) eine Bytecodedatei `Hallo05.class`, lädt sie, erzeugt die darin vereinbarte Klasse `Hallo05` und führt ihre `main`-Methode aus. Wenn er die Klasse `Hallo03` zum ersten Mal braucht, sucht er (im aktuellen Arbeitsverzeichnis) eine Bytecodedatei `Hallo03.class`, lädt sie und erzeugt die darin vereinbarte Klasse `Hallo03`. Ganz entsprechend verfährt er mit der Klasse `Hallo04`.

Wenn man nicht ausdrücklich etwas anderes angibt, gelten folgende Standard-Festlegungen:

Std-Fest-1: Wenn eine zu compilierende Klasse von einer weiteren Klasse `WK` abhängt, sucht der Compiler `javac` die Klasse `WK` nur im *aktuellen Arbeitsverzeichnis*. Falls nötig, compiliert er automatisch die Quelldatei der Klasse `WK`.

Std-Fest-2: Der Compiler `javac` legt jede von ihm erzeugte Bytecodedatei (z. B. `Hallo03.class`) im selben Verzeichnis ab, in dem die zugehörige Quelldatei (`Hallo03.java`) steht.

Std-Fest-3: Der Interpreter `java` sucht die Bytecodedateien aller Klassen, die er benötigt, im *aktuellen Arbeitsverzeichnis*.

Beispiel-02: Alle Quelldateien stehen in *einem* Verzeichnis `d:\Q` (Q wie Quellen), welches *nicht* das Arbeitsverzeichnis ist.

```
5  d:\Aka> javac -cp d:/Q d:/Q/Hallo05.java
6  d:\Aka> java  -cp d:/Q      Hallo05
```

Wenn der Compiler `javac` nicht im aktuellen Arbeitsverzeichnis, sondern in einem anderen Verzeichnis `d:\Q` nach weiteren Klassen suchen soll, muss man ihm dieses Verzeichnis nach `-cp` (wie class path) angeben. Entsprechendes gilt auch für den Interpreter `java`.

Anmerkung: In den `javac`- und `java`-Kommandos darf man auch unter Windows *Schrägstriche* `/` anstelle vom *Rückwärtsschrägstrichen* `\` angeben. Im Folgenden werden Pfadnamen *im Text* immer Windows-spezifisch mit Rückwärtsschrägstrichen notiert, in den *Kommandos* dagegen in der (für Windows und Linux einheitlichen) Notation mit Schrägstrichen (wie in Zeile 3 und 4).

Nach `-cp` darf man auch *mehrere* Verzeichnisse angeben (siehe das folgende Beispiel). Diese Verzeichnisse werden hier *CP-Verzeichnisse* genannt. Der Compiler sucht in den CP-Verzeichnissen nach weiteren Klassen, die er benötigt (d. h. nach den entsprechenden Bytecode- und Quelldateien). Der Interpreter sucht in den CP-Verzeichnissen nach allen Klassen, die er benötigt (aber nur in Form von Bytecodedateien).

Auch im Beispiel-02 legt der Compiler jede neu erzeugte *Bytecodedatei* in *dem* Verzeichnis ab, in dem die zugehörige *Quelldatei* steht (weil nicht ausdrücklich etwas anderes festgelegt wurde, siehe oben Std-Fest-2).

Man beachte, dass der Compiler (in Zeile 3) die zu compilierende Quelldatei `Hallo05.java` *nicht* im CP-Verzeichnis sucht, sondern nur anhand des angegebenen Pfadnamens `d:\Q\Hallo05.java`. Der Interpreter sucht dagegen auch die Bytecode-Datei der angegebenen Hauptklasse `Hallo05` in dem (in Zeile 4 angegebenen) CP-Verzeichnis.

Beispiel-03: Die Quelldateien stehen in drei verschiedenen Verzeichnissen, etwa so: `d:\Q3\Hallo03.java`, `d:\Q4\Hallo04.java`, `d:\Q5\Hallo05.java`. Keines dieser drei Quellverzeichnisse ist das aktuelle Arbeitsverzeichnis.

```
7  d:\Aka> javac -cp d:/Q3;d:/Q4  d:/Q5/Hallo05.java
8  d:\Aka> java  -cp d:/Q3;d:/Q4;d:/Q5  Hallo05
```

Wenn man nach `-cp` *mehrere* CP-Verzeichnisse angibt, muss man sie unter Windows durch Semikolons `;` und unter Linux durch Doppelpunkte `:` voneinander trennen.

Dem Compiler muss man (in Zeile 5) nur *zwei* CP-Verzeichnisse angeben, weil er die zu compilierende Datei ja anhand des angegebenen Pfadnamens `d:\Q5\Hallo05.java` sucht. Dem Interpreter muss man dagegen (in Zeile 6) *drei* CP-Verzeichnisse angeben, damit er auch die Bytecodedatei der Hauptklasse `Hallo05` finden kann.

Häufig ist es empfehlenswert, Quelldateien und Bytecodedateien, nicht gemeinsam sondern in *separaten Verzeichnissen* aufzubewahren (unter anderem deshalb, weil man Quelldateien regelmäßig datenssichern sollte, was bei Bytecodedateien viel weniger dringlich ist, weil man sie im Notfall aus den Quelldateien regenerieren kann). Zu diesem Zweck sollte man irgendwo ein separates Verzeichnis nur für Bytecodedateien anlegen, z. B. das Verzeichnis `c:\Bcd` (wie Bytecode).

Beispiel-04: Die Quelldateien stehen in drei verschiedenen Verzeichnissen, von denen keines das Arbeitsverzeichnis ist (genau wie im vorigen Beispiel). Alle Bytecodedateien sollen im Verzeichnis `c:\Bcd` abgelegt werden

```
9  d:\Aka> javac -cp d:/Q3;d:/Q4  -d c:/Bcd  d:/Q5/Hallo05.java
10 d:\Aka> java  -cp c:/Bcd  Hallo05
```

Mit der Option `-d` muss man für den Compiler (in Zeile 7) das Verzeichnis `c:\Bcd` als *dort-ablegen-Verzeichnis* (kurz: als *D-Verzeichnis*) festlegen. Dem Interpreter braucht man dann nur dieses eine D-Verzeichnis als CP-Verzeichnis anzugeben (in Zeile 8), denn dort findet er ja alle Bytecodedateien.

Das Kommando in Zeile 7 funktioniert richtig, ist aber *nicht effizient*. Denn wenn man es mehrmals ausführen lässt (nachdem man etwa *eine* der Quelldateien ein bißchen verändert hat), werden immer *alle drei* Quelldateien übersetzt, auch die, die gar nicht verändert wurden. Die Datei `Hallo05.java` wird auf jeden Fall übersetzt, weil wir sie in Zeile 7 als zu compilierende Datei angegeben haben. Die anderen beiden Quelldateien werden jedesmal compiliert, weil der Compiler ihre Bytecodedateien (die er selbst erzeugt hat) nicht findet. Er sucht ja nur in den bei-

den Quellverzeichnissen `d:\Q3` und `d:\Q4`, legt aber Bytecodedateien nicht dort, sondern im D-Verzeichnis `c:\Bcd` ab. Das lässt sich aber leicht verbessern.

Beispiel-05: Alles wie im vorigen Beispiel, aber effizienter

```
11 d:\Aka> javac -cp c:/Bcd;d:/Q3;d:/Q4 -d c:/Bcd d:/Q5/Hallo05.java
12 d:\Aka> java  -cp c:/Bcd                          Hallo05
```

Hier wird dem Compiler das Bytecode-Verzeichnis `c:/Bcd` als CP- *und* als D-Verzeichnis angegeben. Damit wird das Verzeichnis zu einer Art „Gedächtnis des Compilers“, in das er neue Erkenntnisse (Bytecodedateien) einträgt und in dem er alte sucht.

Die zahlreichen Blanks in Zeile 10 (vor `Hallo05`) sollen die Lesbarkeit der Kommandos verbessern. Sie sind erlaubt, können aber auch z. B. durch ein einziges Blank ersetzt werden.

Beispiel-06: Alle Quelldateien stehen im aktuellen Arbeitsverzeichnis `d:\Aka` (wie im Beispiel-01), aber die Bytecodedateien sollen in einem separaten Verzeichnis `c:\Bcd` abgelegt werden

```
13 d:\Aka> javac -cp c:/Bcd;. -d c:/Bcd Hallo05.java
14 d:\Aka> java  -cp c:/Bcd             Hallo05
```

Dem Compiler `javac` werden hier zwei CP-Verzeichnisse angegeben: Das Bytecode-Verzeichnis `c:\Bcd` und das aktuelle Arbeitsverzeichnis mit dem unauffälligen Namen `.` (Punkt). Wenn das Kommando in Zeile 13 zum ersten Mal ausgeführt wird, ist das Bytecode-Verzeichnis `c:\Bcd` noch leer. Der Compiler findet aber alle nötigen *Quelldateien* im aktuellen Arbeitsverzeichnis und compiliert sie. Bei weiteren Ausführungen des Kommandos findet der Compiler alle schon erzeugten Bytecodedateien im Verzeichnis `c:\Bcd` und erzeugt nur *die* erneut, bei denen es wirklich nötig ist (weil ihre Quelldatei verändert wurde).

Zum Abschluss dieses Abschnitts noch eine Lösung für die Aufgabe-01:

Lösung-01: Wenn der Compiler auf der Suche nach der weiteren Klasse `Hallo03` eine Bytecodedatei `Hallo03.class` *und* eine Quelldatei `Hallo03.java` findet, prüft er, ob die Bytecodedatei „veraltet“ ist (d. h. ob sie *vor* der letzten Veränderung der Quelldatei erzeugt wurde). Wenn nein, nimmt er die Bytecodedatei wie sie ist. Sonst erzeugt er aus der Quelldatei eine neue, nicht-veraltete Bytecode-Datei. Mit dieser Technik werden unnötige Compilationen vermieden, was vor allem bei großen Systemen (bei denen das Compilieren *aller* Quelldateien viele Stunden oder sogar Tage dauern kann) sehr nützlich ist.

27.3.5 Der Ausführer (javac, java) für Fortgeschrittene (mit Paketen)

Im vorigen Abschnitt gehörten alle Klassen zum namenlosen Paket. Das namenlose Paket ist gut geeignet zum Schreiben von relativ kleinen Programmen oder „wenn man schnell mal etwas ausprobieren will“. Die Klassen von größeren und „ernsthaften Programmen“ sollten dagegen immer in *Paketen-mit-Namen* zusammengefasst werden (siehe dazu im Buch „Java ist eine Sprache“ das Kapitel 17 über Pakete).

In diesem Abschnitt betrachten wir ein Beispielprogramm `K10Tst`, welches aus sechs Klassen besteht, die zu drei Paketen mit den vollen Namen `p00`, `p00.p10` und `p00.p20` gehören. An diesen Namen kann man erkennen: `p00` ist ein Top-Paket (d.h. in keinem anderen Paket enthalten) und `p10` und `p20` sind Unterpakete von `p00`.

Wir gehen auch hier davon aus, dass die Hauptklasse `K10Tst` in einer eigenen Quelldatei namens `K10Tst.java` vereinbart wurde, und dass Entsprechendes auch für die Nebenklassen des Programms (`K10`, `K11`, `K12`, `K21` und `K22`) gilt.

Es folgt eine Übersicht darüber, welche Klassen (oder: Quelldateien) zu welchem Paket gehören. Diese Zuordnung wurde vom Programmierer des Programms festgelegt und soll hier nicht begründet oder in Frage gestellt werden.

Übersicht-1: Zu welchem Paket gehören welche Klassen bzw. Quelldateien

zum Paket	gehören die Klassen	bzw. die Quelldateien
`p00`	`K10Tst`	`K10Tst.java`
`p00.p10`	`K12, K11, K10`	`K12.java, K11.java, K10.java`
`p00.p20`	`K22, K21`	`K22.java, K21.java`

Die Quelldateien des Programms sollten in einem *Baum von Quellverzeichnissen* abgelegt werden, der wie folgt zu erzeugen ist:

Zuerst legt man irgendwo ein *oberstes Quellverzeichnis* an (als Wurzel des Baumes), z. B. das Verzeichnis `d:\Qcd` (wie Quellcode). In diesem Verzeichnis legt man für jedes Top-Paket ein gleichnamiges Unterverzeichnis an, und darin für jedes im Top-Paket enthaltenen Paket ein gleichnamiges Unterverzeichnis etc. Wenn man den Baum von Quellverzeichnissen erstellt hat, legt man darin die Quelldateien entsprechend ihrer Paketzugehörigkeit ab.

Übersicht-2: In welches Verzeichnis gehören welche Quelldateien

Dem Paket	entspricht das Verzeichnis	in das folgende Quelldateien gehören
	`d:\Qcd`	
`p00`	`d:\Qcd\`**`p00`**	`K10Tst.java`
`p00.p10`	`d:\Qcd\`**`p00\p10`**	`K12.java, K11.java, K10.java`
`p00.p20`	`d:\Qcd\`**`p00\p20`**	`K22.java, K21.java`

Mit dieser Anordnung kann man das Programm besonders einfach dem Ausführer übergeben (d. h. compilieren) und ausführen lassen.

Zur Erinnerung: Unter Windows beginnt ein *absoluter Pfadname* immer mit einem Laufwerk (z. B. `c:\`, `d:\` oder `x:\`). Ein *relativer Pfadname* beginnt dagegen mit dem Namen eines Verzeichnisses oder einer Datei. Beispiele für absolute Pfadnamen: `d:\`, `d:\Qcd\p00\p10`, `d:\Qcd\p00\K10Tst.java`. Beispiele für relative Pfadnamen: `p00`, `p10\K10.java`, `K10.class`.

Beispiel-01: Das gesamte Programm `K10Tst` (alle sechs Quelldateien) dem Ausführer übergeben und ausführen lassen. Quelldateien und Bytecodedateien in denselben Verzeichnissen

```
1  d:\Qcd> javac p00/K10Tst.java
2  d:\Qcd> java  p00.K10Tst
```

Um die Kommandos zu vereinfachen, wurde hier das *oberste Quellverzeichnis* `d:\Qcd` zum *aktuellen Arbeitsverzeichnis* gemacht.

In Zeile 1 wird die zu compilierende Haupt-Quelldatei (in der die Hauptklasse vereinbart wird) durch den relativen Pfadnamen `p00\K10Tst.java` beschrieben. Der gilt relativ zum aktuellen Arbeitsverzeichnis `d:\Qcd`, entspricht also dem absoluten Pfadnamen `d:\Qcd\p00\K10Tst.java`. Unter diesem Namen sucht der Compiler die zu compilierende Datei.

Beim Prüfen der Quelldatei `K10Tst.java` erkennt er, dass die Hauptklasse unter anderem von einer weiteren Klasse mit dem vollen Namen `p00.p10.K10` abhängt. Daraufhin sucht er (im aktuellen Arbeitsverzeichnis) nach einer *Bytecodedatei* mit dem relativen Pfadnamen `p00\p10\K10.class` *und* nach einer *Quelldatei* mit dem relativen Pfadnamen `p00\p10\K10.java`. Diese Pfadnamen gelten relativ zum aktuellen Arbeitsverzeichnis `d:\Qcd`, entsprechen also den absoluten Pfadnamen `d:\Qcd\p00\p10\K10.class` bzw. `d:\Qcd\p00\p10\K10.java`.

Ab hier passiert ganz Entsprechendes wie im Beispiel-01 des vorigen Abschnitts: Wenn der Compiler *keine* der beiden Dateien findet, meldet er einen Fehler. Wenn er nur die *Bytecodedatei* findet, nimmt er sie zum Prüfen der Klasse `K10Tst` („Sind alle Abhängigkeiten zwischen `K10Tst` und `K10` in Ordnung?“). Wenn er nur die *Quelldatei* findet, übersetzt er sie automatisch in eine Bytecodedatei und nimmt die für seine Prüfungen. Was er macht, wenn er beide Dateien findet, wurde in der Lösung-01 am Ende des vorigen Abschnitts beschrieben.

Ganz so, wie mit der Klasse `K10`, verfährt der Compiler auch mit jeder anderen Klasse, von der die Hauptklasse `K10Tst` abhängt (im Beispiel sind das die Klassen `K11`, `K12`, `K21` und `K22`).

Wenn der Compiler das Kommando in Zeile 1 erfolgreich ausgeführt hat, stehen sechs Bytecodedateien in den Quellverzeichnissen, jede in *dem* Verzeichnis, in dem auch die zugehörige Quelldatei steht (weil in Zeile 1 kein dort-ablegen-Verzeichnis angegeben wurde).

Dem Interpreter `java` wird in Zeile 2 befohlen, das Programm mit der Hauptklasse `p00.K10Tst` auszuführen. Daraufhin sucht er (im aktuellen Arbeitsverzeichnis) eine Bytecodedatei mit dem relativen Pfadnamen `p00\K10Tst.class`, d. h. mit dem absoluten Pfadnamen `d:\Qcd\p00\K10Tst.class`. Wenn er später die Klasse `p00.p10.K10` zum ersten Mal benötigt, sucht er eine Bytecodedatei mit dem relativen Pfadnamen `p00\p10\K10.class`, d. h. mit dem absoluten Pfadnamen `d:\Qcd\p00\p10\K10.class` etc. etc.

Den *Baum von Quellverzeichnissen* muss der Programmierer anlegen. Der Compiler ist aber in der Lage, „ganz parallel dazu“ einen *Baum von Bytecodeverzeichnissen* anzulegen und alle Bytecodedateien darin abzulegen (statt im Baum von Quellverzeichnissen). Der Programmierer muss nur irgendwo ein *oberstes Bytecodeverzeichnis* anlegen, z. B. das Verzeichnis `c:\Bcd` (wie Bytecode) und dem Compiler als dort-ablegen-Verzeichnis angeben. Alles Weitere erledigt der Compiler dann automatisch.

Beispiel-02: Das gesamte Programm `K10Tst` (alle sechs Quelldateien) dem Ausführer übergeben und ausführen lassen. Quelldateien und Bytecodedateien in separaten Bäumen von Verzeichnissen

```
3  d:\Qcd> javac -d c:/Bcd p00/K10Tst.java
4  d:\Qcd> java -cp c:/Bcd p00.K10Tst
```

Der Compiler erzeugt im D-Verzeichnis `c:\Bcd` drei Unterverzeichnisse namens `p00`, `p00\p10` und `p00\p20` und legt die von ihm erzeugten Bytecodedateien darin ab, ganz „parallel“ zu den Quellverzeichnissen.

Übersicht-3: In welchem Verzeichnis legt der Compiler welche Bytecodedatei ab

Dem Paket	entspricht das Verzeichnis	in dem der Compiler folgende Bytecodedatei ablegt
	`c:\Bcd`	
`p00`	`c:\Bcd\`**`p00`**	`K10Tst.class`
`p00.p10`	`c:\Bcd\`**`p00\p10`**	`K12.class, K11.class, K10.class`
`p00.p20`	`c:\Bcd\`**`p00\p20`**	`K22.class, K21.class`

Dem Interpreter `java` muss man das D-Verzeichnis des Compilers als CP-Verzeichnis angeben (in Zeile 2), damit er die Bytecodedateien *dort* sucht (statt im aktuellen Arbeitsverzeichnis).

Das Kommando in Zeile 3 funktioniert *richtig*, ist aber noch ähnlich *uneffizient* wie das Compilationskommando im Beispiel-04 des vorigen Abschnitts: Wenn man es mehrmals ausführen lässt (nachdem man etwa *eine* der Quelldateien ein bißchen verändert hat), werden immer alle sechs Quelldateien übersetzt, weil der Compiler die bereits erzeugten Bytecodedateien nicht findet. Eine Verbesserung des Kommandos erfolgt ganz ähnlich wie im vorigen Abschnitt.

Beispiel-03: Alles wie im vorigen Beispiel, aber effizienter

```
5  d:\Qcd> javac -cp c:/Bcd;. -d c:/Bcd p00/K10Tst.java
6  d:\Qcd> java  -cp c:/Bcd            p00.K10Tst
```

Der Compiler sucht Quell- und Bytecodedateien in den beiden CP-Verzeichnissen `c:\Bcd` und `.` und legt alle Bytecodedateien im D-Verzeichnis `c:\Bcd` ab. Beim Suchen und Ablegen geht er allerdings immer von *relativen Pfadnamen* wie `p00\K10Tst.java` und `p00\p10\K10.class` etc. aus, die mit den betreffenden Paketnamen beginnen. Der Interpreter sucht mit ebensolchen relativen Pfadnamen im CP-Verzeichnis `c:\Bcd` nach allen Bytecodedateien, die er benötigt.

In der (im Abschnitt 27.1 beschriebenen) Sammlung von Beispielprogrammen findet man auch drei Dos-Batchdateien namens `K10TstA.bat`, `K10TstB.bat` und (Sie haben richtig vermutet) `K10TstC.bat`. Die Kommandos in der A-Datei ähneln denen im Beispiel-01 und die Kommandos in der B-Datei denen im Beispiel-03 (in einer anpassungsfreundlichen Form, für den Fall dass Ihre obersten Verzeichnisse *nicht* `d:\Qcd` und `c:\Bcd` heißen). Mit der C-Datei (C wie clear) kann man alle Bytecode-Dateien, die von der A- oder B-Datei erzeugt wurden, wieder löschen. Es wird empfohlen, für andere Programme die Verwendung entsprechender Batchdateien (in irgendeiner Skriptsprache formuliert) zu erwägen.

Wie wichtig *relative Pfadnamen* und die *vollen Namen* von Klassen bei den hier skizzierten Vorgängen sind, soll noch durch ein weiteres, negatives Beispiel verdeutlicht werden. Nachdem das Kommando in Zeile 5 erfolgreich ausgeführt wurde, steht die Bytecodedatei `K10Tst.class` im Verzeichnis `c:\Bcd\p00`.

Beispiel-04: Vier Kommandos, die *nicht* funktionieren

```
7   d:\Qcd>      java -cp c:\Bcd\p00 K10Tst
8   c:\Bcd\p00> java                 K10Tst
9   d:\Qcd>      java -cp c:\Bcd\p00 p00.K10Tst
10  c:\Bcd\p00> java                 p00.K10Tst
```

In Zeile 7 wird dem Interpreter befohlen, im Verzeichnis `c:\Bcd\p00` nach einer Klasse mit dem vollen Namen `K10Tst` zu suchen. Er findet dort zwar eine Bytecodedatei namens `K10Tst.class`, aber die Klasse darin hat den vollen Namen `p00.K10Tst`. Also lehnt er sie ab. Ganz Entsprechendes passiert auch in Zeile 8.

In Zeile 9 wird dem Interpreter befohlen, im Verzeichnis `c:\Bcd\p00` nach einer Klasse mit dem vollen Namen `p00.K10Tst` zu suchen. Daraufhin sucht er nach einer Datei mit dem absoluten Pfadnamen `c:\Bcd\p00\p00\K10Tst.class` und findet sie nicht (weil es ein Verzeichnisse `...p00\p00...` nicht gibt). Ganz Entsprechendes passiert auch in Zeile 10.

27.3.6 Die Umgebungsvariable CLASSPATH

Wenn man eine bestimmte Liste von CP-Dateien häufig braucht, kann man sie auch in die Umgebungsvariablen `CLASSPATH` schreiben (statt sie in jedem `javac-` und `java-`Kommando nach `-cp` anzugeben). Wenn man in einem Kommando die Option `-cp` angibt, wird dadurch die Variable `CLASSPATH` ausser Kraft gesetzt.

Beispiel-05: Den Wert der Umgebungsvariablen `CLASSPATH` (unter Windows) mit dem `echo`-Kommando anzeigen lassen und mit `set`-Kommando verändern.

```
1   d:\Aka> echo %CLASSPATH%
2   d:\Aka> %CLASSPATH%
3   d:\Aka> set CLASSPATH=c:/Bcd;.
4   d:\Aka> echo %CLASSPATH%
5   d:\Aka> c:/Bcd;.
6   d:\Aka> set CLASSPATH=
7   d:\Aka> echo %CLASSPATH%
8   d:\Aka> %CLASSPATH%
```

Hier sind die Ausgaben des Betriebssystems halbfett hervorgehoben. Mit `echo`-Kommandos wie in Zeile 1, 4 und 7 kann man sich den momentanen Wert einer Umgebungsvariablen anzeigen lassen. Die Ausgaben in Zeile 2 und 8 bedeuten, dass die Variable `CLASSPATH` zur Zeit *leer* ist. In Zeile 13 wird der Variablen `CLASSPATH` der Wert `c:/Bcd;.` (eine Liste von CP-Verzeichnisssen, siehe Beispiel-03) zugewiesen. Dabei darf man vor und hinter dem Gleichheitszeichen `=` keine Blanks notieren. In Zeile 3 wird die Variable wieder *geleert*. Wenn man die verwendete Dos-Eingabeaufforderung schliesst, geht die Wirkung aller `set`-Kommandos verloren.

Unter Windows 95 und 98 kann man `set`-Kommanods auch in die Datei `autoexec.bat` schreiben. Unter Windows NT, -2000 und -XP kann man die Werte von Umgebungsvariablen auch über Start > Systemsteuerung > System > Erweitert > Umgebungsvariablen festlegen. Um die Umgebungsvariable `CLASSPATH` unter Linux anzusehen und zu verändern, muss man entsprechende Shell-Kommandos eingeben oder in eine `rc`-Datei (z. B. `.bashrc`) schreiben.

27.4 Der Java-Ausführer DrJava

Installieren Sie zunächst die neuste Java-Entwicklungsumbung (JDK) der Firma Sun (siehe Abschnitt 27.3).

Unter der Adresse `www.drjava.org` finden Sie die benötigte Installationsdatei und eine (sehr einfache) Installationsanleitung für `DrJava`.

Starten Sie `DrJava` (das kann zahlreiche Sekunden dauern). Ein Fenster mit drei Teilfenstern sollte erscheinen:

Teilfenster 1 (oben links, ziemlich schmal)
Teilfenster 2 (oben rechts, ziemlich breit)
Teilfenster 3 (unten, breit, mit 3 Reitern (engl. tabs): `Interactions`, `Console` und `Compiler Output`).

27.4.1 Der Quelltext-Interpreter von DrJava

Klicken Sie im Teilfenster 3 auf den Reiter `Interactions`. Damit wird das Fenster zur Eingabe eines *Java-Quelltext-Interpreters*, der einzelne Java-Befehle und ganze Java-Programme direkt ausführen kann (ohne dass man sie vorher compilieren muss).

Einfache Befehle: Geben Sie im Teilfenster 3 z. B. den Java-Befehl `1+1` ein (und schließen Sie mit `Return` ab). Der Quelltext-Interpreter sollte daraufhin das Ergebnis `2` ausgeben. Probieren Sie weitere Ausdrücke wie `7/3`, `7%3`, `7.0/3.0`, `1+2+3+4+5` etc. aus.

Geben Sie dann z. B. den Java-Befehl

```
> System.out.printf("Hallo %+d%n", -17);
```

ein. Der Quelltext-Interpreter sollte daraufhin

```
Hallo -17
```

ausgeben.

Sie können auch Variablen vereinbaren und dann benutzen, etwa so:

```
> java.math.BigDecimal biggy = new java.math.BigDecimal(0.1);
> System.out.println(biggy);
```

Der Quelltext-Interpreter sollte daraufhin

```
0.1000000000000000055511151231257827021181583404541015625
```

ausgeben (das ist "der wahre Wert des Literals 0.1"). Auch `import`-Befehle (z. B. `import java.math.BigDecimal;` oder `import java.math.*;` oder `import static java.lang.Math.*;`) werden vom Quelltext-Interpreter korrekt ausgeführt.

Fortgeschrittene Befehle: Öffnen Sie (mit dem Befehl `Open...` im Menü `File`) die Java-Quelldatei `Hallo01.java` (oder eine ähnliche kleine Quelldatei mit `main`-Methode). Der Inhalt der Datei sollte im Teilfenster 2 erscheinen. Kopieren Sie den gesamten Inhalt der Datei (mit `Strg-c` und `Strg-v`) in das Teilfenster 3 und drücken Sie `Return`. Das veranlaßt den Quelltext-Interpreter die Klasse `Hallo01` zu erzeugen, veranlaßt ihn aber noch nicht dazu, die `main`-Methode dieser Klasse auszuführen ("man sieht noch nicht, dass etwas passiert ist").

Geben Sie jetzt (im Teilfenster 3) den Befehl `Hallo01.main(null);` ein. Das sollte die Ausführung der `main`-Methode in der Klasse `Hallo01` bewirken.

Ganz ähnlich wie die das kleine Programm `Hallo01` können Sie auch größere Programme vom Quelltext-Interpreter ausführen lassen.

27.4.2 Der Compiler und der Bytecode-Interpreter von DrJava

Statt einzelen Befehle oder ganze Programme vom Quelltext-Interpreter direkt ausführen zu lassen, können Sie Quelldateien auch compilieren und dann (die daraus entstandene Bytecodedatei) ausführen lassen. Entsprechende Befehle (`Com-`

`pile All Documents`, `Compile Current Document`, `Run Document's Main Method`) finden Sie im Menü `Tools` von `DrJava`. Die vom Compiler erzeugten Bytecodedateien (.class-Dateien) werden standardmäßig im aktuellen Arbeitsverzeichnis abgelegt.

27.5 Der JavaEditor (von Gerhard Röhner)

Der `JavaEditor` läuft nur unter Windows (nicht unter Linux). Seine Installationsdatei `javaeditor.zip` ist etwa 1,3 MB groß und kann von der folgenden Netzadresse heruntergeladen werden:

`www.bildung.hessen.de/abereich/inform/skii/material/java/editor.htm`

In dieser Adresse hat *skii* nichts mit Skifahren zu tun, sondern ist eine Abkürzung für *Sekundarstufe II*.

27.5.1 Den JavaEditor installieren

Um den JavaEditor zu installieren, gehen Sie folgendermaßen vor:

1. Entpacken Sie die Datei `javaeditor.zip` in irgendein (leeres) Verzeichnis.
2. Danach sollte dieses Verzeichnis unter anderem eine Datei namens `setup.exe` enthalten. Lassen Sie diese Datei ausführen (z.B. durch einen Doppelklick).
3. Folgen Sie den wenigen und einfachen Anweisungen auf dem Bildschirm.

Dann können Sie prüfen, ob der `JavaEditor` sich bei der Installation richtig mit dem Java-Compiler `javac` und dem Java-Bytecode-Interpreter `java` der Firma Sun (siehe oben Abschnitt 27.3) verbunden hat. Starten Sie dazu den `JavaEditor` und wählen Sie im Menü `Fenster` den Menüpunkt `Konfiguration`. Ein Fenster mit etwa 17 „Aktenreitern“ (a tabbed window with some 17 tabs) sollte aufgehen. Wählen Sie den Reiter `Compiler` und prüfen Sie, ob in der Zeile `Java-Compiler` der richtige Pfad (zum Compiler `javac` von Sun) eingetragen ist. Klicken Sie dann auf den Reiter `Interpreter` und prüfen Sie, ob in der Zeile `Interpreter` der richtige Pfad (zum Bytecode-Interpreter `java`) eingetragen ist. Korrigieren Sie die Einträge falls notwendig.

Nachdem Sie diese Überprüfung erfolgreich abgeschlossen haben, sollten Sie den Hallo-Test durchführen. Öffnen Sie dazu im `JavaEditor` das Beispielprogramm `Hallo01.java` und übergeben Sie es dem Ausführer, indem Sie im Menü `Starten` den Menüpunt `Compilieren` wählen (oder die Tastenkombination `<Strg>-F9` drücken oder das kleine gelbe Ordnersymbol neben dem roten Ordnersymbol anklicken). Im Ausgabefenster des Compilers sollte eine Meldung der Art `“... Hallo01.java erfolgreich compiliert“` erscheinen. Falls das Ausgabefenster des Compilers gerade nicht sichtbar ist, müssen Sie auf den entsprechenden Aktenreiter ganz unten links im Editor klicken.

Wenn die Übergabe des Programms funktioniert hat, können Sie das Programm *ausführen* lassen, indem Sie das kleine grüne Dreieck gleich neben dem roten Ordnersymbol anklicken.

Besonders hilfreich ist beim `JavaEditor` das Hilfe-Menü. Allerdings sind nicht alle Menüpunkte darin von Anfang an „belegt". Einige funktionieren erst nach bestimmten *Zusatzinstallationen*, die in den folgenden Abschnitten beschrieben werden. Dabei wird angenommen, dass Sie den JavaEditor in das Verzeichnis `C:\Programme\JavaEditor` installiert haben.

27.5.2 Der Hilfe-Menüpunkt Tutorial

Um im Hilfe-Menü den Menüpunkt `Tutorial` „richtig zu belegen", brauchen Sie die Datei `tut-winhelp.zip` von der Firma Sun. Diese Datei ist ca. 12 MB groß und kann von der folgenden Netzadresse heruntergeladen werden:

`java.sun.com/docs/books/tutorial/information/download.html`

Entpacken Sie diese Datei in irgendein Verzeichnis (z.B. in das Verzeichnis `C:\Programme\JavaEditor\tutorial`). Dieses Verzeichnis sollte dann eine Windows-Hilfe-Datei namens `turorial.chm` enthalten (`.chm` steht für `compiled HTML help file`). Starten Sie den `JavaEditor`, wählen Sie im Menü `Fenster` den Menüpunkt `Konfiguration`. Ein Fenster mit etwa 17 „Aktenreitern" sollte aufgehen. Wählen Sie den Reiter `Dokumentation` und tragen Sie in der Zeile `Tutorial` den absoluten Pfadnamen der Datei `tutorial.chm` (z. B. `C:\Programme\JavaEditor\tutorial\tutorial.chm`) ein. Wenn Sie jetzt im Menü `Hilfe` den Menüpunkt `Tutorial` anklicken, sollte ein vernünftiger HTML-Text angezeigt werden.

27.5.3 Der Hilfe-Menüpunkt Javabuch

Um im Hilfe-Menü den Menüpunkt `Javabuch` „richtig zu belegen", brauchen Sie die beiden Dateien `hjp3html.zip` (ca. 3.5 MB) und `hjp3exam.zip` (ca. 0.6 MB), die Sie von der folgenden Netzadresse herunterladen können:

`www.javabuch.de`

Legen Sie irgendwo ein Verzeichnis an (z.B. das Verzeichnis `C:\Programme\JavaEditor\javabuch`) und in diesem Verzeichnis zwei Unterverzeichnisse namens `html` und `examples`. Entpacken Sie die Datei

hjp3html.zip in das Unterverzeichnis html und die Datei hjp3exam.zip in das Unterverzeichnis examples. Starten Sie dann den JavaEditor, wählen Sie im Menü Fenster den Menüpunkt Konfiguration. Ein Fenster mit etwa 17 „Aktenreitern“ sollte aufgehen. Wählen Sie den Reiter Dokumentation und tragen Sie in der Zeile Javabuch den absoluten Pfadnamen C:\Programme\JavaEditor\javabuch\html\cover.html ein. Wenn Sie jetzt im Menü Hilfe den Menüpunkt Javabuch anklicken, sollte die HTML-Version des Buchs „Handbuch der Java-Programmierung“ von Guido Krüger erscheinen.

Dieses HTML-Buch können Sie an einer Stelle noch verbessern, indem Sie es korrekt mit der Dokumentation der Java-Standardbibliothek auf Ihrem Rechner verbinden. Öffnen Sie dazu mit einem Texteditor (z.B. mit dem JavaEditor) die Datei C:\Programme\JavaEditor\javabuch\html\hjp3lib.js) und suchen Sie (ziemlich oben in der Datei) die beiden Zeilen

```
var jdkdocs = "file:///C|/jdk1.4/docs/";
var apidocs = "file:///C|/jdk1.4/docs/api/";
```

In diesen Zeilen steht C| für das Laufwerk C:. Korrigieren Sie die Pfadnamen, so dass sie auf die richtigen Unterverzeichnisse docs und docs/api im Verzeichnis JAVA_HOME zeigen (die im Abschnitt 27.3.3 erwähnt wurden), etwa so:

```
var jdkdocs = "file:///C|/Programme/java/docs/";
var apidocs = "file:///C|/Programme/java/docs/api/";
```

27.5.4 Der Hilfe-Menüpunkt Mindstorm

Um im Hilfe-Menü den Menüpunkt Mindstorm „richtig zu belegen“, sollten Sie zuerst einmal Interesse dafür entwickeln, Lego-Roboter (mit einem kleinen Computer vom Typ RCX darin) mit Hilfe von Java-Programmen zu steuern. Wenn dieses Interesse dann nicht mehr zu bremsen ist, brauchen Sie die beiden Dateien tutorial.zip (ca. 0.5 MB) und lejos_win32_2_1_0.doc.zip (ca. 1.2 MB), die Sie von der folgenden Netzadresse herunterladen können:

lejos.sourceforge.net

Die Datei tutorial.zip sollten Sie unabhängig vom JavaEditor entpacken und die darin enthaltene Datei index.html mit einem Browser gründlich ansehen. Für den JavaEditor sollten Sie irgendwo ein Verzeichnis anlegen, z.B. C:\Programme\JavaEditor\Lejos und die Datei lejos_win32_2_1_0.doc.zip dort hinein entpacken. Starten Sie dann den

`JavaEditor`, wählen Sie im Menü `Fenster` den Menüpunkt `Konfiguration`. Ein Fenster mit etwa 17 „Aktenreitern“ sollte aufgehen. Wählen Sie den Reiter `Mindstorm` und tragen Sie in der Zeile `Manual` den absoluten Pfadnamen `C:\Programme\JavaEditor\Lejos\apidocs\index.html` ein. Wenn Sie jetzt im Menü `Hilfe` den Menüpunkt `Mindstorm` anklicken, sollte eine HTML-Dokumentation für bestimmte Java-Klassen erscheinen, die zur Steurung von Lego-Robotern entwickelt wurden.

Mit dem `JavaEditor` haben Sie übrigens auch einen schlichten, aber recht brauchbaren *Webbrowser* installiert. Falls Ihr Internet-Explorer wieder mal in die Werkstatt muss, weil neue Sicherheitslöcher gestopft werden müssen, können Sie den `JavaEditor` starten und im Menü `Hilfe` den Menüpunkt `Webseite` wählen. Ab da können Sie dann mehr oder weniger wie gewohnt durchs Internet surfen.

27.6 Der Java-Ausführer BeanShell (von Pat Niemeyer)

Mit dem Programm `BeanShell` kann man Java-Quellprogramme direkt ausführen lassen, ohne Sie vorher zu compilieren oder auf andere schwer zu durchschauende Weise verändern zu müssen.

Das Programm `BeanShell` wurde in Java geschrieben und mit einem Compiler in Bytecode übersetzt. Verbreitet wird es meist in Form einer sogenannten `jar`-Datei. Dabei steht `jar` für `java archive` und ist gleichzeitig eine Dateinamenerweiterung (wie `.txt` oder `.java` etc.). Eine `jar`-Datei ist im Wesentlichen eine `zip`-Datei (meistens eine komprimierte Zusammenfassung von *mehreren* Dateien), die eine Art Inhaltsverzeichnis (eine sogenannte *Manifest-Datei*) enthält. Eine `jar`-Datei kann man mit jedem ZIP-Dekomprimier-Programm dekomprimieren (und im Prinzip auch mit jedem ZIP-Komprimier-Programm erstellen, aber dann muss man die Manifest-Datei selbst von Hand schreiben). Ein Programm namens `jar.exe` (im Verzeichnis `JAVA_HOME\bin`) kann Dateien zu einer `jar`-Datei zusammenfassen und komprimieren und später wieder dekomprimieren, und kümmert sich dabei automatisch um die Manifest-Datei.

Ein Java-Bytecode-Interpreter (wie z. B. der namens `java` von Sun) kann auch `jar`-Dateien ausführen (und entnimmt der Manifest-Datei, bei welcher Datei er mit der Ausführung beginnen soll). Somit läuft ein Programm wie die `BeanS-`

hell überall, wo ein Java-Bytecode-Interpreter zur Verfügung steht (unter Linux, Windows und anderen Betriebssystemen).

27.6.1 Die BeanShell installieren

Um den Java-Ausführer BeanShell (Version 2.0 beta 4) auf Ihrem PC zu installieren, brauchen Sie die Datei bsh-2.0b4.jar (ca. 0.3 MB), die Sie von der folgenden Netzadresse herunterladen können:

www.beanshell.org

Kopieren Sie diese Datei (so wie sie ist und ohne sie zu dekomprimieren) in irgendein Verzeichnis. Außerdem sollten Sie eine Verknüpfung mit dieser Datei auf Ihrem Desktop erzeugen (wie das geht wurde in der Empfehlung im Abschnitts 27.3.3 beschrieben).

Eine sehr gute Dokumentation (ca. 70 Seiten, auf Englisch) zum BeanShell-Programm findet man in den Dateien bshmanual.html (gut zum Lesen auf einem Bildschirm geeignet) und bshmanual.pdf (gut zum Ausdrucken geeignet), ebenfalls unter der Netzadresse www.beanshell.org.

Das Programm BeanShell kann man wahlweise *mit* oder *ohne* grafische Benutzeroberfläche (Grabo) starten.

27.6.2 Die BeanShell mit Grabo starten

Durch einen Doppelklick auf die Verknüpfung (die Sie sich auf den Desktop gelegt haben und die mit der Datei bsh-2.0b2.jar verbunden ist) starten Sie die Grabo-Version des Programms BeanShell (die Version *mit* grafischer Benutzeroberfläche). Falls das Doppelklicken nicht die gewünschte Wirkung hat, müssen Sie ein Kommandoeingabefenster (unter Windows eine DOS-Eingabeaufforderung, unter Linux eine Shell) öffnen und z.B. folgendes Kommando eingeben:

```
> java -jar d:/meineDateien/Editoren/bsh-2.0b2.jar
```

Auch unter Windows können Sie dabei normale Schrägstriche / anstelle von Rückwärtsschrägstrichen \ verwenden.

Daraufhin sollte (nach einer Verzögerung von einigen Sekunden) ein Fenster mit dem Titel BeanShell Desktop 1.1 aufgehen und in diesem Fenster ein weiteres *Workspace-Fenster* mit dem Titel Bsh Workspace: 0.

Jetzt können Sie den Hallo-Test durchführen. Wählen Sie dazu im `File`-Menü des Workspace-Fensters den Menüpunkt `Workspace Editor`. Dadurch sollte ein *Editor-Fenster* mit dem Titel `Editor for: Bsh Worspace: 0` aufgehen. Öffnen Sie in diesem Editor-Fenster die Datei `Hallo01.java` (Menü `File`, Menüpunkt `Open` etc.) und vergrößern Sie alle Fenster so weit, dass Sie den gesamten Programmtext (alle 5 Zeilen) und das Workspace-Fenster unbehindert und vollständig sehen können.

Wählen Sie im `Evaluate`-Menü des Editor-Fensters den Menüpunkt `Eval in Workspace`. Dadurch wird „der Inhalt des Editor-Fenster ausgeführt". Wenn dadurch nichts Sichtbares passiert, ist wahrscheinlich alles in Ordnung. Denn im Editor-Fenster steht (außer einigen Kommentarzeilen) nur eine 5 Zeilen lange *Vereinbarung* einer Klasse namens `Hallo01`. (d. h. ein Befehl, den man etwa wie folgt ins Deutsche übersetzen kann: „Erzeuge eine Klasse namens `Hallo01`, die die folgenden Elemente enthält: ..."). Diese Vereinbarung wurde gerade ausgeführt, d.h. der Ausführer hat eine Klasse namens `Hallo01` (mit einer `main`-Methode darin) erzeugt und ab jetzt *existiert* die Klasse (und darin die `main`-Methode). Mit dem Inhalt des Editor-Fensters wird dem Ausführer *nicht* befohlen, die `main`-Methode in der Klasse `Hallo01` auszuführen.

Um ein bißchen mehr Äkschen zu sehen sollte man im File-Menü des Workspace-Fensters den Menüpunkt `Capture System in/out/err` wählen. Dadurch wird vor allem die Standardausgabe (`System.out`), die bisher noch mit einer Art Datenmülleimer verbunden war, mit dem Workspace-Fenster verbunden. Danach sollten Sie im Workspace-Fenster den Befehl eingeben, dass die `main`-Methode der Klasse `Hallo01` ausgeführt werden soll, und zwar so:

```
bsh % Hallo01.main(null);
```

Nach einem Druck auf die Return-Taste sollte die Ausgabe der `main`-Methode (`Hallo Welt!`) in der folgenden Zeile des Workspace-Fensters erscheinen.

Als nächsten Schritt nehmen wir im Editor-Fenster eine kleine Veränderung an der Klasse `Hallo01` vor, indem wir das `String`-Literal `"Hallo Welt!"` z. B. zu `"Hallo Welt!!!"` verändern.

Wenn wir danach im Workspace-Fenster denselben Befehl eingeben wie vorhin (`Hallo01.main(null);`) bekommen wir auch dieselbe Ausgabe wie vorhin: `Hallo Welt!` Das liegt daran, dass trotz unserer Änderung im Editor-Fenster noch die alte Klasse `Hallo01` mit der alten `main`-Methode darin existiert. Um eine neue, geänderte Klasse erzeugen zu lassen wählen wir im `Evaluate`-Menü

des Editor-Fensters erneut den Menüpunkt `Eval in Workspace` und geben danach im Workspace-Fenster wieder unseren Befehl

```
bsh % Hallo01.main(null);
```

ein. Diesmal sollte die veränderte Ausgabe `Hallo Welt!!!` erscheinen.

Größere Veränderungen unserer Klasse `Hallo01` lassen sich ganz entsprechend ausprobieren, etwa so:

Unter oder über dem Befehl `System.out.println("Hallo Welt!!!");` fügen wir einen weiteren ähnlichen Befehl ein, z.B. `System.out.println ("Wie geht's!");`. Dann lassen wir die Klassen-Vereinbarung im Editor-Fenster erneut ausführen (Menü `Evaluate`, Menüpunkt `Eval in Workspace`). Schließlich geben wir im Workspace-Fenster wieder den Befehl ein, die `main`-Methode der Klasse `Hallo01` auszuführen (`Hallo01.main(null);`). Dadurch wird auch der neu eingefügte Befehl ausgeführt.

Zu einem Workspace-Fenster kann man sich beliebig viele Editor-Fenster öffenen und im Prinzip ist es möglich, darin ein ganzes Programm zu entwickeln (aber es gibt deutlich leistungsfähigere und komfortablere Editoren, z.B. den JavaEditor).

Beenden können wir das Programm BeanShell z. B. indem wir im Workspace-Fenster den Java-Befehl `System.exit(0);` (oder das kürzere `BeanShell`-Kommando `exit();`) eingeben.

27.6.3 Die Umgebungsvariable CLASSPATH

Wenn ein Windows-Betriebssystem eine *ausführbare Datei* braucht (z.B. eine `exe`-Datei oder eine `bat`-Datei etc.), sucht es diese Datei in allen Verzeichnissen, die in der Umgebungsvariablen `PATH` eingetragen sind. Wenn der Interpreter `java` eine `class`-Datei (d. h. eine Bytecodedatei) braucht, sucht er sie in allen CP-Verzeichnissen. Das sind alle Verzeichnisse, die man beim Aufruf des Interpreters mit der Option `-cp` angegeben hat oder (wenn der Interpreter ohne diese Option aufgerufen wurde) alle Verzeichnisse, die in der Umgebungsvariablen `CLASSPATH` eingetragen sind. Ausser Verzeichnissen kann man auch `jar`-Archive in die Umgebungsvariable `CLASSPATH` eintragen.

Ein `jar`-Archiv enthält in der Regel mehrere `class`-Dateien und wird vom Java-Ausführer genau wie ein *Verzeichnis* behandelt. Indem man den Pfadnamen eines `jar`-Archivs in die Umgebungsvariable `CLASSPATH` einträgt, stellt man sicher,

dass der Java-Ausführer die im Archiv enthaltenen `class`-Dateien findet, wenn er sie sucht.

Die `jar`-Datei `bsh-2.0b2.jar` des BeanShell-Programms enthält vor allem zwei `class`-Dateien namens `Console.class` (das BeanShell-Programm *mit* Grabo) und `Interpreter.class` (das BeanShell-Programm *ohne* Grabo). Um das Starten dieser Klassen zu erleichtern, sollte man die `jar`-Datei `bsh-2.0b2.jar` in die Umgebungsvariable `CLASSPATH` eintragen. Danach könnte diese Umgebungsvariable unter Windows z. B. folgenden Wert enthalten:

```
C:/Klassen;D:/meineDateien/Editoren/bsh-2.0b2.jar;.
```

Dieser `CLASSPATH` bewirkt, dass der Java-Ausführer `class`-Dateien in zwei Verzeichnissen und in einer `jar`-Datei sucht, zuerst in `C:\Klassen`, dann in `D:\meineDateien\Editoren\bsh-2.0b2.jar` und schliesslich im aktuellen Arbeitsverzeichns (welches durch einen Punkt `.` bezeichnet wird). Man beachte, dass auch unter Windows die Verzeichnisse im `CLASSPATH` mit normalen Schrägstrichen `/` notiert werden sollten, und nicht mit Rückwärtsschrägstrichen `\`. Falls Ihre Schrägstrich-Taste klemmt, können Sie anstelle *eines* Schrägstrichs `/` auch *zwei* Rückwärtsschrägstriche `\\` angeben.

Unter Linux könnte der Wert der `CLASSPATH`-Variablen z. B. so aussehen:

```
/Klassen:/meineDateien/Editoren/bsh-2.0b2.jar:.
```

Hier werden die einzelnen Pfadnamen durch Doppelpunkte `:` voneinander getrennt, unter Windows durch Semikolons `;` .

27.6.4 Die BeanShell mit bzw. ohne Grabo starten

Nachdem man die `jar`-Datei `bsh-2.0b2.jar` der BeanShell in die Umgebungsvariable `CLASSPATH` eingetragen hat (wie im vorigen Abschnitt beschrieben), kann man das Programm durch Eingabe der folgenden Kommandos (in einer DOS-Eingabeaufforderung bzw. in einer Shell) starten:

```
> java bsh.Console        // mit  Grabo (wie im Abschnitt 27.5.2)
> java bsh.Interpreter    // ohne Grabo
```

Wenn man den Interpreter startet, sollte er eine kleine Meldung vom Autor Pat Niemeyer ausgeben und danach das Kürzel `bsh` und ein Prozentzeichen `%` als Aufforderung zur Eingabe von Befehlen, etwa so:

```
(BeanShell 2.0b2 - by Pat Niemeyer (pat@pat.net)
bsh %
```

Wenn man jetzt einzelne Java-Befehle eingibt, werden sie sofort ausgeführt. Eine sofort *sichtbare Wirkung* haben aber nur Befehle, die Daten zur Standardausgabe ausgeben (z. B. der Befehl `System.out.println("Hallo!")`) und erst nachdem man die Standardausgabe mit dem aktuellen Fenster verbunden hat (indem man im Menü `File` den Menüpunkt `Capture System in/out/err` wählt).

Ein Dialog mit dem Interpreter kann etwa so aussehen (die Eingaben des Benutzers sind halbfett hervorgehoben:

```
bsh % String s1 = "Hallo!";
bsh % System.out.println(s1 + " Wie geht's?");
Hallo! Wie geht's?
bsh % exit();
```

Mit dem BeanShell-Kommando `exit();` (oder dem Java-Befehl `System.exit(0);`) kann man die BeanShell beenden.

Anstelle des Java-Befehls `System.out.println` kann man auch das kürzere BeanShell-Kommando `print` verwenden, welches meistens das Gleiche leistet und manchmal sogar etwas mehr, z. B. beim Ausgeben von Reihungen:

```
bsh % int[] ir = {5, 2, 8};
bsh % print(ir);
int []: {
5,
2,
8,
}
bsh % System.out.println(otto);
[I@128e20a
bsh % print(Integer.toHexString(otto.hashCode()));
128e20a
```

In der Ausgabe des Befehls `System.out.println` ist `[I` ein etwas kryptischer Name für den Typ `int[]` (Reihung von `int`).

Mit dem BeanShell-Kommando `source` kann man sich Java-Quelldateien in den Interpreter reinholen (statt sie noch einmal einzutippen). Danach kann man die in den Quelldateien vereinbarten Klassen und ihre Methoden etc. benutzen, z. B. so:

```
bsh % source("Hallo03.java");
bsh % Hallo03.druckeStrich();
**********************************
bsh % Hallo03.druckeProdukt(3, 7);
**********************************
3 mal 7 ist gleich 21
bsh %
```

Auf diese Weise lassen sich die Methoden einer Klasse einfach interaktiv testen.

Normalerweise bewirken Audrücke wie `17` oder `n + 17` etc. *keine* Ausgaben. Mit dem BeanShell-Kommando `show()` kann man den Interpreter dazu veranlassen, die Werte von Ausdrücken automatisch zu zeigen (d.h. auszugeben), etwa so:

```
bsh % int n = 3;
bsh % show();                  // schaltet ein
<true>
bsh % 17;                      // wird ausgegeben
<17>
bsh % n + 17;
<20>
bsh % Math.log10(100);
<2>                            // Der 10-er-Logarithmus von 100 ist 2
bsh % Math.sin(Math.PI));
<1.2246467991473532E-16>
bsh % show();                  // schaltet aus
bsh % 17;                      // wird nicht ausgegeben
bsh % print(17);
17
```

Der erste Aufruf von `show()` schaltet die automatische Ausgabe von Werten von Ausdrücken *an* und wird vom Interpreter mit `<true>` quittiert. Der zweite Aufruf von `show()` schaltet die automatische Ausgabe wieder *aus* und wird nicht quittiert. Die Zahl `1.2246467991473532E-16` sollte man trotz ihrer vielen Ziffern als „eine 0 mit einem winzigen Fehler“ lesen.

Beim Aufruf des Interpreters ohne Grabo kann man auch gleich ein Java-Programm (oder ein BeanShell-Skript) angeben, welches ausgeführt werden soll, etwa so:

```
> java bsh.Interpreter Hallo01.java
Hallo Welt!
Wie geht es?
```

Falls das auszuführende Programm (im Beispiel `Hallo01`) weitere Klassen benötigt, sollten die über die Umgebungsvariablen `CLASSPATH` erreichbar sein. Als Beispiel wird hier das Programm `Hallo08.java` ausgeführt, in dem Methoden der Klasse `EM` (wie „Eingabe-Modul“) aufgerufen werden:

```
> java bsh.Interpreter Hallo08.java
A Bitte geben Sie eine Ganzzahl ein: 12
```

```
B 12 mal 12 ist gleich 144
C Bitte geben Sie einen String  ein: abc
D Warum ausgerechnet abc ???
```

Diese Ausführung funktioniert nur, wenn vorher die Quelldatei `EM.java` in eine Bytecodedatei `EM.class` übersetzt wurde (z.B. mit dem Compiler `javac` von Sun, siehe Abschnitt 27.3) und die Datei `EM.class` in einem Verzeichnis steht, welches in der Umgebungsvariablen `CLASSPATH` eingetragen ist.

27.6.5 Bekannte Fehler der BeanShell (Version 2, beta 2)

1. Methoden, die Daten von der Standardeingabe *einlesen* (wie z. B. das im vorigen Abschnitt erwähnte Beispielprogramm `Hallo08`) werden von der BeanShell *ohne* Grabo (`bsh.Interpreter`) *korrekt* ausgeführt, führen aber in der BeanShell *mit* Grabo (`bsh.Console`) zum „Hängenbleiben“ des Programms.

2. Mit `String`-Literalen intitialisierte `String`-Variablen werden nicht *internalisiert* (vom `bsh.Interpreter` und von der `bsh.Console`). Das bedeutet konkret, dass z.B. die folgenden Befehle die Ausgabe `false` produzieren statt `true`:

```
bsh % String s1 = "ABC";
bsh % String s2 = "ABC";
bsh % print(s1 == s2);
false
```

Wenn man die Internalisierung ausdrücklich befiehlt, wird `true` ausgegeben:

```
bsh % String s1 = "ABC".intern();
bsh % String s2 = "ABC".intern();
bsh % print(s1 == s2);
true
```

3. Bestimmte („harmlos aussehende“) Java-Klassen lösen Fehlermedlungen aus und werden nicht richtig interpretiert, z.B. die Klasse `Zaehler01`. Diese Klasse enthält Attribute der Typen `int` und `long`. Wenn man alle `long`-Attribute zu `int`-Attributen macht, wird die Klasse problemlos und korrekt interpretiert. Vielleicht ist dieser Fehler in der nächsten Version der BeanShell schon behoben.

Das BeanShell-Programm ist zur Zeit (Ende 2004) noch nicht an die neue Java-Version 5.0 angepaßt. Deshalb kann es keine *Quelldateien* (`.java`-Dateien) ausführen, in denen z. B. die neuen Methoden namens `System.out.printf` aufgerufen, Aufzählungstypen (`enum types`) vereinbart oder die neuen parametrisierte Typen wie `ArrayList <String>` benutzt werden.

Programme, die die neuen Methoden namens `System.out.printf` benutzen, kann man trotzdem von der BeanShell ausführen lassen, wenn man wie folgt vorgeht:

1. Man übersetzt die Quelldatei `AM.java` (aus der Sammlung der Beispielprogramme, siehe Abschnitt 27.1) in eine Bytecodedatei `AM.class` (z. B. mit dem Compiler `javac` von Sun, siehe Abschnitt 27.3).
2. Man sorgt dafür, dass die Datei `AM.class` über die Umgebungsvariable `CLASSPATH` auffindbar ist (d.h. in einem Verzeichnis steht, welches im `CLASSPATH` eingetragen ist).
3. In den Quelldateien, die man von der BeanShell ausführen lassen will, ersetzt man Aufrufe der Methode `System.out.printf` durch entsprechende Aufrufe der Methode `AM.printf`.

Die BeanShell kann bereits *übersetzte* Aufrufe der Methoden `System.out.printf` (die in *Bytecodedateien* wie z. B. `AM.class` stehen) ausführen, nur keine noch-nicht-übersetzte Aufrufe in *Quelldateien*.

Ein Quelltext-Interpreter wie die BeanShell ist einfacher und direkter zu verstehen und zu bedienen, als ein compilierender Ausführer (wie z. B. der Ausführer (`javac`, `java`) von Sun). Beim Lernen einer Sprache ist die Einfachheit der Werkzeuge wichtiger als die Geschwindigkeit, mit der die Übungsprogramme ausgeführt werden.

27.7 Der Editor JEdit (von Slava Pestov)

Der Editor JEdit ist, ähnlich wie die BeanShell (siehe Abschnitt 27.5) in Java geschrieben. Um ihn zu installieren brauchen Sie *eine* der folgenden beiden Dateien

```
jedit42install.exe   (ca. 2 MB, nur für Windows)
jedit42install.jar   (ca. 2 MB, für alle Plattformen)
```

die Sie von der folgenden Netzadresse herunterladen können:

www.jedit.org

27.7.1 Den JEdit installieren

Um den JEdit zu installieren, müssen Sie eine der beiden Dateien `jedit42install.exe` oder `jedit42install.jar` ausführen lassen.

Die Datei `jedit42install.exe` kann man (nur unter Windows) durch einen Doppelklick darauf ausführen lassen. Sie ist für den Fall gedacht, dass Sie mit dem JEdit z. B. C++-Programme editieren wollen und keinen Java-Bytecode-Interpreter (wie den namens `java` von Sun) installiert haben.

Die Datei `jedit42install.jar` kann man ausführen lassen, indem man sie in irgendein Verzeichnis kopiert, auf dem Desktop eine Verknüpfung auf die Datei erzeugt und auf diese Verknüpfung einen Doppelklick ausführt.

Falls die `jar`-Datei sich nicht durch einen Doppelklick ausführen lässt, muss man dafür sorgen, dass sie über die Umgebungsvariable `CLASSPATH` auffindbar ist. Das kann man z. B. dadurch erreichen, dass man einen Punkt `.` (der das jeweils aktuelle Arbeitsverzeichnis bezeichnet) als ein Verzeichnis in den `CLASSPATH` einträgt, die `jar`-Datei in irgendein Verzeichnis kopieren, ein Kommondoeingabefenster öffnet, das Verzeichnis mit der `jar`-Datei darin zum aktuellen Arbeitsverzeichnis macht und folgendes Kommando eingibt:

```
> java -jar jedit42install.jar
```

Wenn es einem gelungen ist, eine der beiden Dateien (`jedit42install.exe` oder `jedit42install.jar`) zu starten, muss man ein paar einfache Fragen beantworten und dann sollte die Installation ohne Probleme durchgeführt werden. Dabei wird normalerweise auf dem Desktop eine Verknüpfung mit dem dem JEdit erstellt, so dass man ihn durch einen Doppelklick auf diese Verknüpfung starten kann. Sonst muss man ihn ähnlich wie Datei `jedit42install.jar` (siehe oben) mit einem Kommando wie

```
> java -jar jEdit.jar
```

starten. Eventuell muss man dabei den absoluten Pfadnamen der `jar`-Datei angeben, z. B. so:

```
> java -jar C:\Programme\JEdit\jEdit.jar
```

Der installierte Editor belegt auf der Festplatte etwa 12 bis 15 MB. Der Kern des Editors, die Datei `jedit.jar`, ist ca. 3 MB groß.

Nachdem man den JEdit gestatrtet hat, kann man mit der Funktionstaste `F1` (oder indem man in Menü `Help` den Menüpunkt `jEdit Help` wählt) eine umfangreiche HTML-Hilfedatei öffnen. Die gleichen Informationen stehen auch in der Datei

`jedit42manual-a4.pdf`

die man sich ebenfalls von der Netzadresse `www.jedit.org` herunterladen kann. Diese `pdf`-Datei enthält etwa 130 Seiten und ist auch gut zum Ausdrucken geeignet.

27.7.2 Der PlugIn-Manager

Zum Editor JEdit gibt es etwa 80 *Ergänzungen* (plugins). Nicht alle davon sind unentbehrlich und die Leistungen einiger überlappen sich stark. Besonders bequem ist, dass man allein mit dem JEdit (ohne einen Browser oder andere Hilfsprogramme) weitere Ergänzungen aus dem Internet laden und installieren lassen kann.

Wenn man im Menü `Plugins` den Menüpunkt `Plugin Manager` wählt, sieht man zuerst eine Liste aller bereits *installierten* Ergänzungen und kann dann z.B. den Reiter `Install` anklicken, um eine Liste aller verfügbaren Ergänzungen aus dem Internet laden und eventuell weitere Ergänzungen installieren zu lassen.

Empfehlung: Installieren Sie die Ergänzungen `Console`, `JCompiler`, `JavaStyle` und `ErrorList` (wenn sie nicht schon vorinstalliert sind). Im Menü `Plugin Manager` können Sie zu jeder installierten Ergänzung eine Hilfedatei öffnen.

27.7.3 Erste Schritte mit dem JEdit

Zur Programmierung und Dokumentierung des JEdit haben sehr viele Personen beigetragen (siehe Menü `Help`, Menüpunkt `About jEdit ...`) und viele sind an der Weiterentwicklung beteiligt. Das hat wohl dazu beigetragen, dass die Be-

nutzeroberfläche nicht ganz einfach ist und man sich eine Weile daran gewöhnen muss, ehe man „flüssig damit arbeiten“ kann. Hier können nur ein paar Hinweise gegeben werden, wie man ein Programm mit dem JEdit dem Ausführer übergeben und wie man es ausführen lassen kann.

1. Starten Sie den JEdit.

2. Öffnen Sie die Datei `Hallo01.java` (Menü `File`, Menüpunkt `Open` etc.). Setzen Sie sich eine Sonnenbrille auf, falls die Syntaxhervorhebungen auf Ihrem Bildschirm zu grell sind.

3. Um die Datei dem Ausführer zu übergeben (d. h. zu compilieren) gehen Sie wie folgt vor: Wählen Sie im Menü `Plugins` den Menüpunkt `JCompiler` und dann den Menüpunkt `Compile File` . Ein Fenster `Console` sollte aufgehen, bei dem im Textfeld links oben `JCompiler` ausgewählt ist (und nicht `System` oder `BeanShell` etc).

4. Bei dem einfachen Programm `Hallo01` sollten alle Meldungen im `Console`-Fenster *grün* sein (zu roten Fehlermeldungen siehe unten den Punkt 7.).

5. Direkt in der `Console` ausführen lassen kann man ein Programm nur, wenn es keine Daten von der Standardeingabe einliest (das `Console`-Fenster ist ein reines Ausgabefenster, kein Eingabefenster). Um das `Hallo01`-Programm direkt in der `Console` ausführen zu lassen gehen Sie so vor: Wählen Sie im Menü `Plugins` den Menüpunkt `Console` und dann `Run Current Buffer`. Ein Fenster `Commando` öffnet sich und Sie klicken darin auf `OK`. Das Fenster `Console` öffnet sich wieder (oder erhält den Fokus, wenn es vom Compilieren her noch offen ist). Diesmal ist im Textfeld links oben `System` ausgewählt (und nicht `JCompiler` wie vorhin beim Compilieren). Falls sich oben rechts in diesem `Console`-Fenster ein kleines Ying-Yang (bestehend aus zwei Pfeilen) dreht, klicken Sie auf den Knopf unmittelbar rechts danaben (d.h. auf den laufenden Mann mit dem gekrümmten Pfeil über der Schulter). Die Ausgaben des Programms sollten jetzt schwarz im `Console`-Fenster erscheinen (mit ein paar grünen Meldungen drumrum).

6. Programme, die Daten von der Standardeingabe einlesen, müssen z. B. von einem separaten Kommandoeingabefenster (einer DOS-Eingabeauffoderung oder einer Shell) gestartet werden.

7. Falls nach dem Compilieren (siehe oben 4.) *rote* Fehlermeldungen im `Console`-Fenster erscheinen, sollten Sie sich die auch in der speziellen *Fehlerliste* ansehen. Wählen Sie dazu im Menü `Plugins` den Menüpunkt `ErrorList`. Wenn

Sie darin auf einen Fehler klicken, bekommt automatisch das Fenster mit dem Programmtext den Fokus und der Cursor sprint in die fehlerhafte Zeile.

Empfehlung: Verändern Sie die Größen und Positionen der einzelnen Fenster so, dass das Hauptfenster (mit dem Programmtext darin), das `Console`-Fenster und das `Error List` Fenster sich nicht überlappen und Sie alle drei jederzeit sehen können.

Empfehlung: Lernen Sie jede Woche ein bis zwei weitere Befehle, Optionen oder andere Möglichkeiten des JEdit, dann haben Sie schon nach wenigen Jahren einen ziemlich guten Überblick über diesen vielseitigen Editor.

27.8 Die Java-Entwicklungsumgebung von GNU/Cygwin

Unter Linux steht einem der GNU-Java-Ausführer (`gcj`, `gij`) mehr oder weniger automatisch zur Verfügung. Falls er nicht vorinstalliert ist, muss man ihn von der Linux-Distributions-CD oder von der SuSE-Netzseite (`www.suse.com`) oder von der RedHat-Netzseite (`www.redhat.com`) oder einem anderen Ort her installieren. Der Java-Ausführer basiert auf einem Programm namens `gcc` („das gemeinsame Frontend der GNU Compiler Collection“). Dieses Programm muss eine Version größer 3 haben (z. B. 3.1.0 oder 3.3.1 oder 3.4.1 etc.), sonst muss man es wahrscheinlich auch gegen eine neuere Version austauschen. Mit dem Kommando

```
gcc -dumpversion
```

können Sie die Version Ihres `gcc`-Programms herausfinden.

Die *Cygwin-Distribution* besteht aus zahlreichen quelloffenen Programmen, die aus der Unix/Linux-Welt in die Windowsenclave portiert wurden, und seitdem auch allen Windows-Benutzern zur Verfügung stehen. Zu dieser Distribution gehören auch die zahlreichen Sprach-Ausführer des Gnu-Projekts und insbesondere der Java-Ausführer (`gcj`, `gij`). Die neuste Version der Cygwin-Distribution finden Sie unter der Netzadresse

`www.cygwin.com`

Laden Sie von dort zuerst nur die kleine Datei `setup.exe` auf Ihren PC unter Windows und starten Sie dann dieses Programm.

27.8.1 Die Cygwin-Distribution installieren

Mit dem Programm `setup.exe` können Sie die Cygwin-Distribution (teilweise oder vollständig) wahlweise in *einem* Schritt oder in *zwei* Schritten auf Ihrem Windows-PC installieren. Dazu bietet Ihnen das Programm drei mögliche Arbeitsschritte an:

1. Sie können die Cygwin-Distribution direkt aus dem Netz auf Ihrem PC installieren lassen (ohne dass die Installationsdateien dauerhaft auf Ihrem PC gespeichert werden). Wenn man eine schnelle und zuverläßige Verbindung zum Internet hat, ist diese Möglichkeit durchaus empfehlenswert.

2. Sie können die Installationsdateien der Cygwin-Distribution erstmal nur aus dem Netz auf Ihren PC herunterladen lassen.

3. Sie können die zuvor heruntergeladenen Installationsdateien der Cygwin-Distribution auf Ihrem PC installieren lassen.

Wenn man die Cygwin-Distribution *vollständig* installiert, belegt sie auf der Festplatte etwa 1,5 GB. Die Distribution besteht aus etwa 20 *Kategorien* (Categories). Eine Kategorie besteht aus (etwa 3 bis 30) *Paketen* und die Pakete enthalten die einzelnen *Programme* und anderen *Dateien*. Beim Installieren können Sie ganze Kategorien und/oder einzelne Pakete zum Installieren auswählen (ein Paket ist die kleinste installierbare Einheit). Abhängigkeiten zwischen den Paketen werden automatisch berücksichtigt (deshalb sollten Sie ein automatisch zur Installation ausgewähltes Paket nicht wieder abwählen, sonst fehlt es nachher).

Eine genaue Bedienungsanleitung für das Programm `setup.exe` findet man auf der Cygwin-Netzseite (`www.cygwin.com`) in der Nähe des Textes `get help on using setup.exe`. Standardmäßig (by default) wird nur eine minimale Teilmenge der Distribution (ohne Java-Ausführer) installiert. Damit auch der Java-Ausführer installiert wird, müssen Sie im entsprechenden Auswahlfenster in der Kategorie `Devel` (wie „development“) das Paket `gcc-java: Java compiler` durch Anklicken mit einem Kreuzchen versehen.

Empfehlung: Wenn Sie 150 oder 200 MB auf Ihrer Festplatte leicht entbehren können, sollten Sie in den Kategorien `Archive`, `Base`, `Devel`, `Doc`, `Editors` und `Interpreters` *alle* Pakete installieren lassen, das vereinfacht die Auswahl erheblich. Wenn Sie 1 GB übrig haben, können Sie auch *alle* Pakete in *allen* Kategorien (mit Ausnahme der Kategorie `X11`) installieren lassen. Die Kategorie `X11` belegt allein ca. 500 MB und sollte nur installiert werden, wenn man sie braucht. Die Pakete der Kategorie `X11` sind nicht erforderlich, um Java zu lernen.

Lassen Sie am Ende der Installation vom Programm `setup.exe` ein Icon auf dem Desktop erstellen. Das Icon ist eine Art schwarzes C mit einer grünen Pfeilspitze darin. Wenn man darauf klickt, öffnet sich ein *Cygwin-Fenster*, in dem man DOS-Kommandos wie `cd`, `dir`, `echo %PATH%` etc. und Unix-Kommandos wie `ls`, `df`, `ps`, `echo $PATH` etc. eingeben kann.

Anmerkung: Ein Cygwin-Fenster ähnelt einer DOS-Eingabeaufforderung (etwa so wie ein Adler einem Spatz ähnelt). In einer DOS-Eingabeaufforderung kommuniziert man mit einem simplen Programm namens `CMD.EXE`. In einem Cygwin-Fenster kommuniziert man mit einem deutlich leistungsfähigeren Programm namens `bash.exe`. Statt „ein Cygwin-Fenster öffnen“ sagen Linux-Benutzer einfach „eine Shell öffnen“ oder (weil es auch noch andere Shell-Programme gibt) „eine `bash`-Shell öffnen“.

Das Verzeichnis, in das Sie die Cygwin-Distribution installiert haben (z.B. `c:\Programme\cygwin`) wird im Folgenden mit `CYGWIN_HOME` bezeichnet. Nach der Installation sollten Sie das `bin`-Verzeichnis der Installation (damit ist das Verzeichnis `CYGWIN_HOME\bin`, konkret also z. B. `c:\Programme\cygwin\bin`) in die Umgebungsvariable `PATH` eintragen. Dadurch wird vieles leichter.

Testen Sie die Installation, indem Sie ein Cygwin-Fenster öffnen und folgendes Kommando eingeben:

```
$ gcj -dumpversion
```

Der Java-Compiler sollte daraufhin seine Versions-Nummer ausgeben (z. B. `3.3.1` oder so ähnlich). Falls das `bash`-Programm das Kommando `gcj` nicht erkennt, sollten Sie prüfen, ob das Verzeichnis `CYGWIN_HOME\bin` wirklich in der Umgebungsvariablen `PATH` eingetragen ist, z. B. mit folgendem Kommando:

```
$ echo $PATH
```

In der (möglicherweise langen und unübersichtlichen) Ausgabe sollte irgendwo der Pfadname des `bin`-Verzeichnisses vorkommen.

Falls die `PATH`-Variable in Ordnung ist, aber im `bin`-Verzeichnis kein Programm namens `gcj.exe` steht, ist der Java-Ausführer nicht richtig installiert worden. Haben Sie beim Installieren in der Kategorie `devel` wirklich das Paket `gcc-java: Java compiler` durch Anklicken mit einem Kreuzchen versehen? Man kann mit dem `setup.exe`-Programm leicht einzelne Pakete *nachinstallieren*.

27.8.2 Eine java-Datei in eine class-Datei übersetzen und ausführen

Kopieren Sie das Beispielprogramm `Hallo01.java` in ein (möglichst leeres) Verzeichnis, öffnen Sie ein Cygwin-Fenster und machen Sie das Verzeichnis zum aktuellen Arbeitsverzeichnis. Prüfen Sie mit dem DOS-Kommando `dir` oder mit dem Shell-Kommando `ls`, ob das gelungen ist.

Um die Quelldatei dem Ausführer zu übergeben (zu compilieren), geben Sie folgendes Kommando ein:

```
$ gcj -C Hallo01.java
```

Mit der Option `-C` befehlen Sie dem Ausführer, eine `class`-Datei namens `Hallo01.class` (und nicht eine direkt ausführbare Datei namens `Hallo01.exe`) zu erzeugen. Überzeugen Sie sich mit einem `ls`-Kommando davon, dass die `class`-Datei ornungsgemäß erzeugt wurde. Mit dem folgenden Kommando können Sie sie ausführen lassen:

```
$ gij Hallo01
```

Wenn ein Programm aus mehreren Dateien besteht, kann man diese einzeln oder zusammen dem Ausführer übergeben (compilieren) und dann die Hauptklasse (die mit der `main`-Methode darin) ausführen lassen, etwa so:

```
$ gcj -C Hallo12.java Hallo10.java Hallo11.java
$ gij    Hallo10
```

Hier werden die drei Dateien `Hallo10.java`, `Hallo11.java`, `Hallo12.java` gemeinsam dem Ausführer übergeben (die Reihenfolge spielt keine Rolle). Dann wird die Hauptklasse `Hallo10` ausgeführt.

27.8.3 Eine java-Datei in eine exe-Datei übersetzen

Beginnen Sie wie im vorigen Abschnitt. Um das Quellprogramm dem Ausführer zu übergeben (zu compilieren), geben Sie folgendes Kommando ein:

```
$ gcj -o Hallo01.exe -Wa,-W --main=Hallo01 Hallo01.java
```

Mit der Option `-o Hallo01.exe` befehlen Sie dem Ausführer, eine Datei namens `Hallo01.exe` zu erzeugen (o wie output). Sie können die Ausgabedatei auch `emil.exe` nennen, wenn Sie das bevorzugen.

Mit den Optionen `-Wa,-W` verhindern Sie, dass bestimmte überflüssige Warnungen ausgegeben weden.

Mit der Option `--main=Hallo01` teilen Sie dem Ausführer mit, dass die Klasse `Hallo01` die Hauptklasse des zu erzeugenden Programms sein soll. Diese Angabe ist weniger überflüssig als es im ersten Moment scheinen mag, denn die Datei `Hallo01.java` könnte auch die Vereinbarungen von mehreren Klassen enthalten und auch die Nebenklassen eines Java-Programms dürfen `main`-Methoden enthalten. Überzeugen Sie sich mit einem `ls`-Kommando davon, dass die `exe`-Datei ornungsgemäß erzeugt wurde. Mit dem folgenden Kommando können Sie sie ausführen lassen:

```
$ Hallo01
```

Wenn Sie Datei `Hallo01.exe` auf einem anderen Windows-Rechner ausführen lassen wollen, müssen Sie auch die Datei `cygwin1.dll` (ca. 1,2 MB, aus dem Verzeichnis `CYGWIN_HOME\bin`) auf denen anderen Rechner übertragen und dort über die Umgebungsvariable `PATH` erreichbar machen.

27.8.4 Mehrere java-Dateien in eine exe-Datei übersetzen

Angenommen, Sie haben ein Cygwin-Fenster geöffnet und im aktuellen Arbeitsverzeichnis liegen die drei Quelldateien `H10.java`, `H11.java`, `H12.java` eines Programms. `H10` ist die Hauptklasse des Programms und es soll eine `exe`-Datei namens `H.exe` erzeugt werden.

```
$ gcj -o H.exe -Wa,-W --main=H10 H10.java H11.java H12.java
```

Mit `-o H.exe` wird der Name der zu erzeugenden Ausgabedatei festgelegt, `-Wa,-w` unterdrückt lästige Warnungen und mit `--main=H10` wird die Hauptklasse des Programms festgelegt.

27.9 Der Compiler jikes

Die Compiler `javac` (von Sun, siehe 27.3), `gcj` (aus dem Gnu-Projekt, siehe 27.7.2) und `jikes` leisten im Prinzip das Gleiche: Der Programmierer kann mit ihnen *Java-Quelldateien* (die er mit einem Editor geschrieben hat) dem Ausführer übergeben, prüfen und in sogenannte *Bytecodedateien* übersetzen lassen. Die Bytecodedateien kann man dann mit einem Bytecode-Interpreter wie `java` (von Sun) oder `gci` (aus dem Gnu-Projekt) ausführen lassen.

Der Compiler `jikes` ist nur „ein halber Java-Ausführer“ (weil man zusätzlich noch einen Bytecode-Interpreter benötigt). Unter den kostenlosen Java-Compilern

ist er aber deutlich der *schnellste* (und muss auch Vergleiche mit deutlich teureren Produkten nicht scheuen). Er wurde mit intensiver Unterstützung der Firma IBM entwickelt und wird jetzt im Rahmen eines quelloffenen Projekts gewartet und erweitert. Der `jikes` wurde in der Sprache C++ programmiert. Dadurch wurde seine Schnelligkeit möglich. Gleichzeitig wurde es deshalb nötig, verschiedene Versionen für verschiedene Betriebssysteme (Windows, Linux und andere) zu erstellen und zu verwalten (von der Sprache C++ gibt es für viele Plattformen unterschiedliche Dialekte, aber keinen Dialekt, der auf allen Plattformen gleich funktioniert).

Die neuste Version des Compilers `jikes` kann man sich von der folgenden Netzadresse herunterladen:

`www-124.ibm.com/developerworks/opensource/jikes/`

Die Distribution des Compilers (für die jeweilige Plattform) besteht im Wesentlichen aus einer ausführbaren Datei (namens `jikes.exe` unter Windows bzw. nur `jikes` unter Linux) und einer Dokumentationsdatei namens `manual.html`. Man installiert den Compiler, indem man die ausführbare Datei in irgendein Verzeichnis kopiert, welches in der Umgebungsvariablen `PATH` eingetragen ist (z. B. in das `bin`-Verzeichnis der Java-Entwicklungsumgebung von Sun: `JAVA_HOME\bin` bzw. `JAVA_HOME/bin`).

Aufgerufen und benutzt wird der `jikes` weitgehend so wie der Compiler `javac` von Sun (siehe 27.3.2 und 27.3.4). Es gibt nur *einen* grundlegenden Unterschied: Der `jikes` „weiss nicht“, wo die Java-Standardbibliothek liegt, mit der die von ihm erzeugten Bytecodedateien später ausgeführt werden sollen. Deshalb muss man ihm das entsprechende Verzeichnis ausdrücklich mitteilen. Im folgenden Beispiel wird angenommen, dass die Java-Entwicklungsumgebung von Sun (unter Windows) in das Verzeichnis `c:\Programme\java` installiert wurde. Dann kann man dem `jikes` die Quelldatei `Hallo01.java` wie folgt übergeben:

```
> jikes -cp c:/Programme/java/jre/lib/rt.jar Hallo01.java
```

Die Datei `rt.jar` (run time java archive) enthält die gesamte Standardbibliothek des Java-Ausführers (`javac`, `java`) von Sun, und wenn man dem `jikes` sagt, wo diese Bibliothek steht, kann er sie einfach „mitbenutzen“. Wenn man den `jikes` häufiger benutzt, ist es empfehlenswert, den absoluten Pfad der Datei `rt.jar` in die Umgebungsvariablen `CLASSPATH` einzutragen. Die Optionen `-cp` und `-d` funktionieren beim `jikes` genau so wie beim Compiler `javac` (siehe 27.3.5). Weitere Einzelheiten zur Handhabung des Compilers `jikes` findet man in seiner Dokumentationsdatei `manual.html`.

27.10 Den TextPad als Entwicklungsumgebung benutzen

Der Editor `TextPad` läuft nur unter Windows. Die Installationsdatei können Sie von folgender Adresse herunterladen (ca. 2,5 MB):

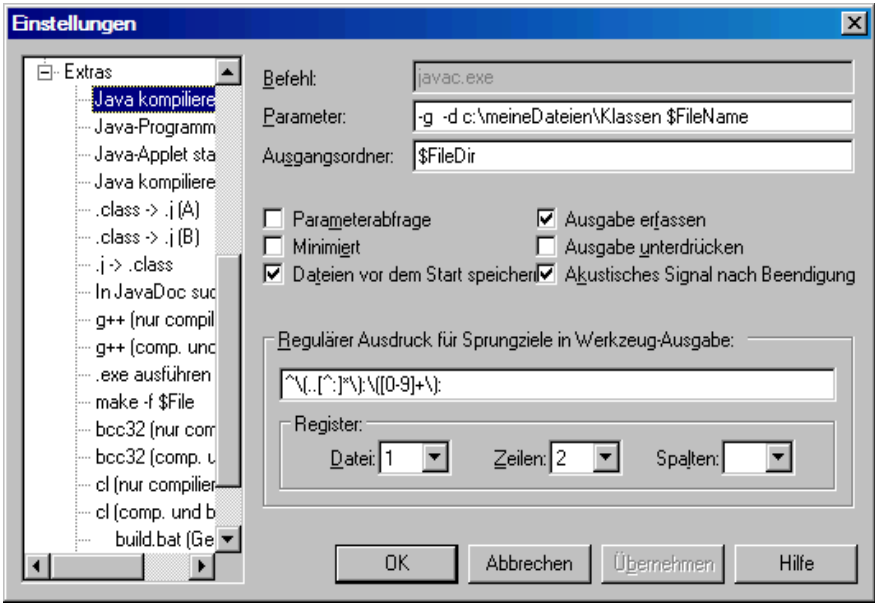
`www.textpad.com`

Installieren Sie aber zuerst die Java-Entwicklungsumgebung (JDK) der Firma Sun (siehe oben 27.3) und erst danach den `TextPad`. Dadurch ermöglichen Sie es dem `TextPad`, sich bei der Installation automatisch mit der Java-Entwicklungsumgebung (JDK) "zu verbinden".

Es wird hier empfohlen, im `TextPad` noch folgende Einstellungen vorzunehmen:

1. Für das Compilieren von .java-Dateien

Starten Sie den `TextPad` und klicken Sie im Menu `Konfiguration` auf `Einstellungen`, dort auf das Pluszeichen `+` neben `Extras` und schließlich auf `Java compilieren (javac)`. In dem sich dann öffnenden Fenster sollten Sie Einstellungen etwa wie die folgenden vornehmen:



Erläuterungen:

Die Angaben in der Zeile `Parameter:` bewirken Folgendes:

`-g` Der Compiler soll in jede `.class`-Datei, die er erzeugt, sogenannte Debug-Informationen einfügen (die in bestimmten Situationen das Finden von Fehlern erleichtern).

`-d c:\meineDateien\Klassen` Der Compiler soll alle `.class`-Dateien, die er erzeugt, in das Verzeichnis `c:\meineDateien\Klassen` schreiben (und nicht in das aktuelle Arbeitsverzeichnis). Natürlich können Sie nach `-d` (wie `directory`) ein beliebiges Verzeichnis angeben, müssen dieses Verzeichnis aber selbst anlegen. In dieses Verzeichnis sollten Sie selbst keine Dateien hineintun (nur der Java-Compiler `javac` sollte dort die von ihm erzeugten `.class`-Dateien ablegen).

`$FileName` Diese Angabe bezeichnet die Datei, die Sie sich mit dem `TextPad` gerade anseseh ("die aktuelle Datei, die gerade den Fokus hat"). Diese Datei sollte eine Java-Quelldatei sein (z.B. `Hallo01.java`). Sie wird compiliert, wenn Sie im Menü `Extras` den Eintrag `Java compilieren (javac)` wählen.

Wenn in der Zeile `Regulärer Ausdruck für Sprungziele in Werkzeug-Ausgaben` schon "eine komplizierte Zeichenkette" steht, können Sie diese erstmal stehen lassen oder durch die etwas kürzere Zeichenkette

```
^\(..[^:]*\):\([0-9]+\):
```

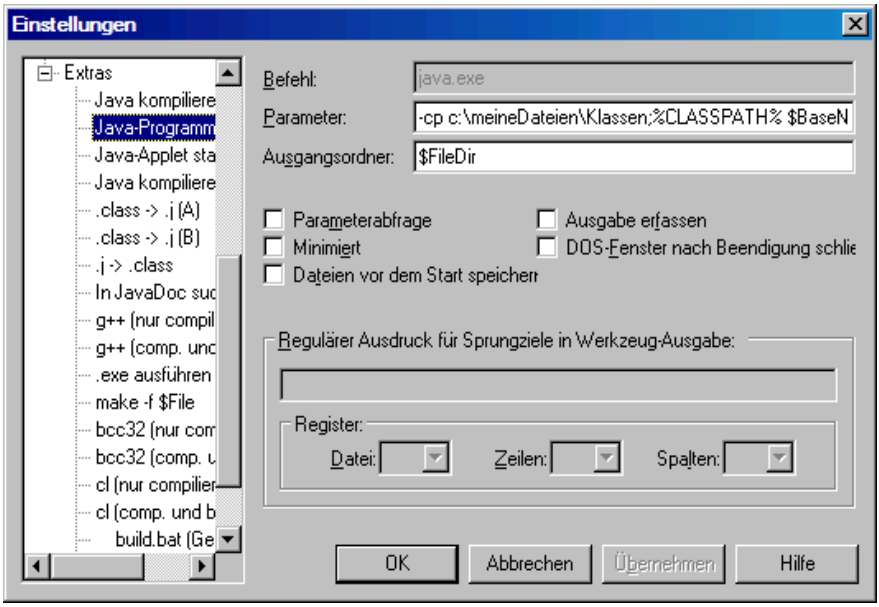
ersetzen. Diese Zeichenkette bewirkt Folgendes: Wenn Sie ein Java-Programm compiliert haben und der TextPad Ihnen als Ergebnis eine Liste von Fehlermeldungen anzeigt, z.B. so:

```
1  Hallo07.java:7: illegal start of expression
2     static public void main(String[] sonja) {
3     ^
4  Hallo07.java:17: ';' expected
5     } // main
6      ^
7  Hallo07.java:18: '}' expected
8  } // class Hallo07
9   ^
10 3 errors
```

dann können Sie z.B. irgendwo in der Zeile 1 *doppelklicken*. Daraufhin wird der `TextPad` ihnen automatisch die Zeile 7 der Datei `Hallo07.java` anzeigen. Ganz entsprechend können Sie auch in der Zeile 4 oder in der Zeile 7 doppelklicken (weil auch dort ein *Dateiname* und eine *Zeilennummer* angegeben ist).

2. Für das Ausführen von Java-Programmen

Starten Sie den `TextPad` und klicken Sie im Menu `Konfiguration` auf `Einstellungen`, dort auf das Pluszeichen `+` neben `Extras` und schließlich auf `Java-Programm starten`. In dem sich dann öffnenden Fenster sollten Sie Einstellungen etwa wie die folgenden vornehmen:



Erläuterungen:

In der Zeile `Parameter:` sollten Sie etwa Folgendes eintragen:

```
-cp c:\meineDateien\Klassen;%CLASSPATH% $BaseName
```

Dabei müssen Sie anstelle von `c:\meineDateien\Klassen` genau das selbe Verzeichnis angeben wie im Befehl für das Compilieren von Dateien (siehe oben).

Nach `-cp` (wie classpath) kann man eine Liste von Verzeichnissen angeben (wobei man die einzelnen Verzeichnisse durch Semikolons `;` voneinander trennen muss). Wenn beim Ausführen eines Java-Programms eine weitere Klasse benötigt wird, sucht der Ausführer in all diesen Verzeichnissen nach einer entsprechenden `.class`-Datei.

Üblicherweise schreibt man die Verzeichnisse, in denen der Java-Ausführer nach `.class`-Dateien suchen soll, in die Umgebungsvariable namens `CLASSPATH`. Die Angabe

```
-cp c:\meineDateien\Klassen;%CLASSPATH%
```

bewirkt, dass der Ausführer zuerst im Verzeichnis `c:\meineDateien\Klassen` sucht und dann in allen Verzeichnissen, die in der Umgebungsvariablen `CLASSPATH` stehen.

`$BaseName` Diese Angabe bezeichnet die Datei, die Sie sich mit dem `TextPad` gerade anseseh ("die aktuelle Datei, die gerade den Fokus hat"), aber ohne eine Erweiterung (wie `.txt` oder `.java` oder `.class` etc.). Wenn man ein Java-Programm starten will, muss man den Namen der Hauptklasse angeben, aber ohne Erweiterung.