

Vorwort

Diese Anleitung soll euch den Einstieg in Java etwas erleichtern. Es wird Schritt für Schritt erklärt, wie man auf einem Windows System das Java Development Kit installiert und dieses einsetzt um Quelltexte zu compilieren und die so entstandenen Programme zu starten. Auf den Einsatz einer Entwicklungsumgebung, wie z. B. Eclipse, wird dabei zunächst absichtlich verzichtet um die Systematik des in Java essentiellen Paketsystems in Verbindung mit Klassenpfaden zu verdeutlichen.

Installation SDK

Um mit Java programmieren zu können muss zunächst das sogenannte Java Development Kit (kurz JDK) installiert werden. Dieses kann von der Seite <http://java.sun.com/j2se/1.5.0/download.jsp> nach kostenlos heruntergeladen werden. Es muss lediglich ein ebenfalls kostenloser „Sun Online Account“ angelegt werden bevor man das JDK dann auch wirklich herunterladen darf. Die ebenfalls auf der Seite angebotene Java Runtime Environment (kurz JRE) ist im JDK bereits enthalten und muss nicht extra installiert werden.

Die umfangreiche Dokumentation ist einer der großen Vorzüge von Java, weshalb es sich empfiehlt diese auch gleich mit herunterzuladen (J2SE Documentation unten).



<http://java.sun.com/j2se/1.5.0/download.jsp>

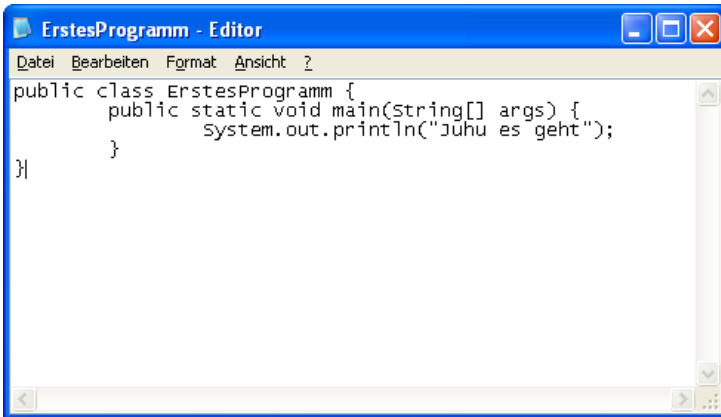
Installation

Hat man den Installer fertig heruntergeladen muß man diesen nur starten und sich durch die folgenden Dialoge klicken. Dabei wird zunächst das Development Kit installiert und anschließend die zum starten von Java Programmen nötige Runtime Environment. Da die meisten diese Runtime Environment wohl sowieso schon installiert haben kommt während der Installation eine Warnung (There is already ... reinstall?), die man getrost mit „yes“ bestätigen kann. Wer gerne den Installationspfad ändern möchte kann dies natürlich tun, in den folgenden Beispielen wird aber vom Standardinstallationspfad, also C:\Programme\Java\jdk1.5.0_06 für das Development Kit und C:\Programme\Java\jre1.5.0_06 für die Runtime Environment ausgegangen.

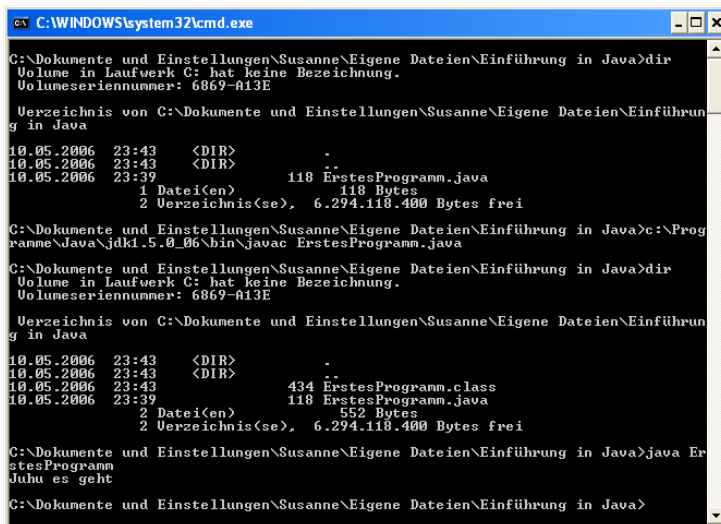
Ergebnis

Nach der Installation befinden sich oben genannte Verzeichnisse auf der Festplatte. Das für uns interessantere von beiden ist sicherlich das JDK Verzeichnis und darin das „bin“ Verzeichnis. In diesem befinden sich alle wesentlichen Programme, die wir zur Erstellung von Software mit Java benötigen, vor allem der Compiler „javac.exe“.

Erstes Programm

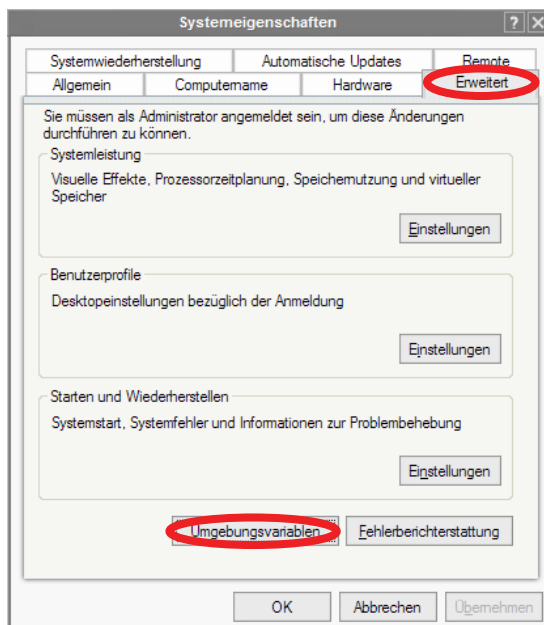


Sourcecode im Editor



Ordnerinhalt vor und nach dem compilieren, Ausführung

Umgebungsvariablen



Java Source Code kann mit jedem Editor erstellt werden. Deshalb öffnen wir zunächst einfach den „normalen“ Windows Editor und geben links stehenden Programmcode ein.

Das ganze speichert man als „ErstesProgramm.java“ in einem beliebigen Verzeichnis. Anschließend öffnet man die Windows Eingabeaufforderung (Ausführen -> cmd -> Enter) und wechselt in das Verzeichnis in dem man die ErstesProgramm.java Datei abgelegt hat. Befindet man sich in diesem Verzeichnis gibt man folgenden Befehl ein:

```
<path>\javac ErstesProgramm.java
```

wobei <path> durch den Pfad zu ersetzen ist in dem sich das Programm javac.exe befindet, also im Falle der Standardinstallation durch C:\Programme\Java\jdk.1.5.0_6\bin. Durch diesen Befehl wurde die Datei „ErstesProgramm.class“ erzeugt. Bei dieser Datei handelt es sich bereits um ein ausführbares Java Programm, das wir mit folgendem Befehl starten können:

```
java ErstesProgramm
```

Dies führt auf der Konsole zur Ausgabe:

```
Juhu es geht
```

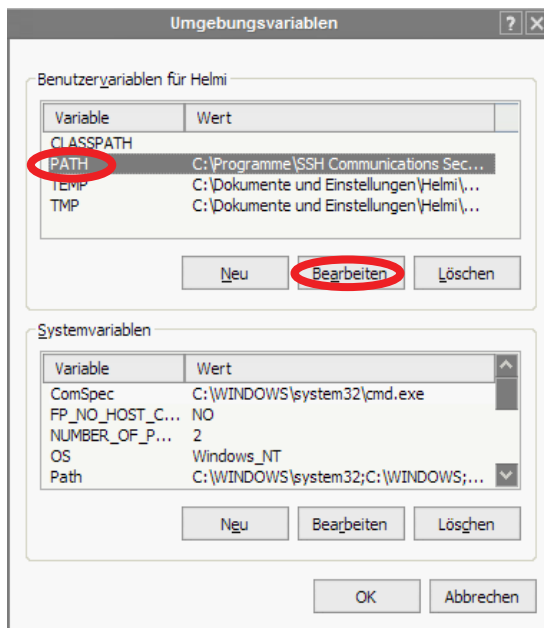
Links ist der Übersetzungsvorgang mit anschließender Ausführung beispielhaft dargestellt.

Interessant dabei ist, dass wir im Gegensatz zum übersetzen mit javac die Dateiendung .class bei Ausführen mit java nicht angeben mussten. Dies liegt daran, dass JavaProgramme nicht mit Dateien gestartet werden sondern mit Klassen. Die Datei ErstesProgramm.class stellt dabei nur eine Art Container für die eigentliche Klasse „ErstesProgramm“ dar.

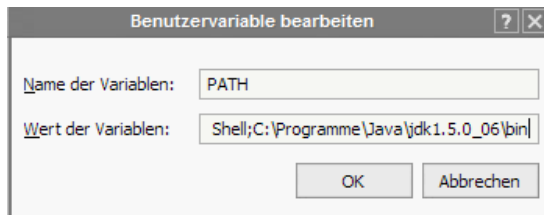
Um sich das Eintippen des kompletten Pfades (C:\...\javac) zur Ausführung des Programmes „javac“ zu sparen gibt es in Windows die Möglichkeit sog. Umgebungsvariablen zu setzen. Für uns intressent ist vor allem Die PATH Variable. In ihr stehen alle Verzeichnisse, die bei Eingabe eines Kommandos in der Eingabeaufforderung nach ausführbaren Dateien durchsucht werden.

Um diese Variable für unsere Zwecke zu verändern geht man wie folgt vor:

- Systemsteuerung öffnen
- Reiter Erweitert auswählen
- Button Umgebungsvariablen unten links anklicken
- Die Zeile mit dem Eintrag „PATH“ markieren
- auf Bearbeiten klicken
- den Wert der Variablen mit einem „;<path>“ erweitern Wobei path wieder durch den Pfad zum „bin“ Verzeichnis (für die Standardinstallation also: „c:\Programme\Java\jdk1.5.0_06\bin“) zu ersetzen ist. Wichtig ist dabei auch der Strichpunkt, mit dem in der Variablen die einzelnen Pfade getrennt werden.



Danach kann der Compiler einfach mit dem Befehl `javac <datei>.class` gestartet werden. Sollte es die PATH-Variable noch nicht geben einfach auf Neu klicken und diese wie oben beschrieben neu anlegen.



Packages

Java Klassen können mit Hilfe von Packages organisiert werden. Diese Packages kann man sich so ähnlich vorstellen wie Verzeichnisse. Dabei kann jedes Package Klassen enthalten oder wieder Subpackages. Dadurch ergibt sich eine Baumstruktur ähnlich eines Dateisystems.

Die einzelnen Ebenen eines Packages werden nur nicht durch ein „/“ sondern durch einen Punkt voneinander getrennt. Betrachtet man z. B. die Klasse Vector diese befindet sich im Package java und dort im Unterpaket util also nach Java schreibweise in `java.util`. Will man diese Klasse nun verwenden gibt man einfach in seinem Programm den Pfad zu dieser Klasse mit an und schreibt `java.util.Vector v;`

Oder man teilt dem Compiler mittels einer Import Anweisung mit, dass man in dieser Datei mit Vector immer den Vector aus `java.util` meint und nicht einen anderen, der vielleicht in `java.2d` liegt.

Das ist eben auch der Sinn dieser Packagestruktur. Es soll damit vermieden werden, dass solche Namenskonflikte entstehen.

blablabla heruntergeladen werden kann. Dort befindet sich ebenfalls die zu diesem Package gehörende Dokumentation.

Das Package enthält lediglich eine Klasse namens Kuehlschrank, die einige Kuehlschrankspezifische Meth-

Fremde Packages einbinden

```

/*
**Datei KTest.java
*/

//import-Anweisung um Klasse benutzen zu können.
import de.uniulm.informatik.sgi.progwerkstatt.*;

public class KTest {
    public static void main (String[] args) {
        Kuehlschrank k = new Kuehlschrank();
        k.rein("Gemuese");
        k.rein("Wurst");
        k.rein("Bierle");
        k.rein("Bierle");
        k.raus("Bierle");
        k.printInhalt();
    }
}

```

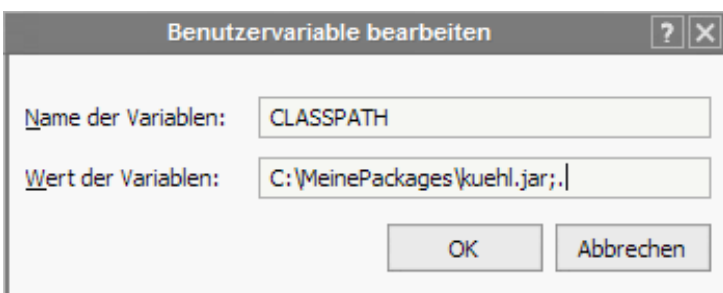
Sourcecode zum Programm KTest

```

C:\WINDOWS\system32\cmd.exe
C:\>cd KTest
C:\KTest>javac -classpath C:\MeinePackages\kuehl.jar KTest.java
C:\KTest>java -classpath C:\MeinePackages\kuehl.jar;. KTest
Gemuese
Wurst
Bierle
C:\KTest>_

```

KTest mit -classpath compilieren und ausführen



Anlegen der CLASSPATH Umgebungsvariable

Wichtig: Verweis auf aktuelles Verzeichnis nicht vergessen

Häufig reicht die Funktionalität der mit JDK ausgelieferten Packages nicht aus und man muss fremde Packages einsetzen. Dies ist zum Beispiel bei der FileOfRecord-Bibliothek aus der Vorlesung der Fall. Wie man solche Packages einsetzen kann wird in diesem Abschnitte anhand einer Kuehlschrank Bibliothek erläutert.

Diese kann unter www.blablabla heruntergeladen werden. Dort befindet sich ebenfalls die in Java übliche Dokumentation, die den Funktionsumfang des Paketes beschreibt. Das Kuehlschrank Package enthält dabei nur eine Klasse namens Kuehlschrank, die einige Kuehlschrankspezifische Methoden anbietet. Zunächst muss die kuehl.jar Datei heruntergeladen und in einem beliebigen Verzeichnis z. B. „C:\MeinePackages“ gespeichert werden.

Anschließend schreiben wir ein kleines Programm, dass ein Kuehlschrankobjekt anlegt und ein paar Sachen in den Kuehlschrank legt. Ein Blick in die Dokumentation verrät uns, dass die dazu benötigte Kuehlschrank-Klasse im Package „de.uniulm.informatik.sgi.progwerkstatt“ liegt. Um dem Compiler klar zu machen, dass wenn in unserem Programm von Kuehlschrank die Rede ist genau die Klasse in diesem Package gemeint ist importieren wir zuerst alle Klassen in diesem Package mit `import de.uniulm.informatik.sgi.progwerkstatt.*;` Anschließend können wir diese Klasse ganz „normal“ benutzen (s. links).

Anschließend wird das Programm compiliert. Dabei ist zu beachten, dass javac nur den Ort kennt an dem die Standardpackages liegen. Von unserem kuehl.jar Package weiss es nichts. Deshalb müssen wir diesen Ort bekannt machen. Das tut man mit der Option -classpath gefolgt vom Dateinamen unseres Archivs inklusive Pfadangabe. In unserem Beispiel ergäbe das:

```
javac -classpath C:\MeinePackages\kuehl.jar KTest.java
```

Dasselbe gilt für die Laufzeitumgebung und auch für das Program java gibt es eine -classpath Option. Hierbei ist aber zu beachten, dass java, sobald man diese Option einschaltet im aktuellen Verzeichnis nicht mehr nach Klassen sucht und deshalb auch unsere KTest Klasse nicht mehr findet. Wir müssen das aktuelle Verzeichniss dann explizit mit „.“ explizit noch mit angeben. Also:

```
java -classpath C:\MeinePackages\kuehl.jar;. KTest.java
```

Auch hier werden mehrere Pfadangaben wieder durch einen Strichpunkt getrennt.

Auch für Klassenpfade gibt es eine Umgebungsvariable, die von java und javac ausgewertet wird. Diese heisst CLASSPATH. Legt man diese an und speichert in ihr die betreffenden Pfade und jar Dateien muss man diese nicht mehr jedesmal über die classpath-Option angeben.