Tutorium Java
Ein Überblick

Helge Janicke

26. Oktober 2000

1 Vorraussetzungen zum Programmieren mit Java.

Was braucht man, wenn man mit Java programmieren möchte?

- Einen Editor, mit dem man ASCII Texte erzeugen kann.
- Grundkenntnisse mit dem Umgang einer Konsole (DOS) oder Shell (Linux).
- Ein installiertes Java-software-development-kit.

1.1 Welcher Editor macht mir das Leben leichter?

Eigentlich ist die Wahl des Editors von den eigenen Vorlieben abhängig. Unter Windows, sowie unter Linux hat man immer eine recht gute Auswahl von Editoren, die sich für die Entwicklung von Java-Programmen eignen.
Für Windows kann ich hier TextPad und UltraEdit empfehlen, beide sind auch als Trial-Version erhältlich. Das TextPad-Trial hat den Vorteil, das es keiner zeitlichen Beschränkung unterliegt und man nur gelegentlich die Aufforderung zum Bestellen wegklicken muß. Für die Entwicklung unter Linux gibt es auch zwei Editoren, die ich getestet habe. JEdit und (wie sollte es anders sein) emacs (auch noch kwrite und nedit). Natürlich kann man auch die "Primitiv"-Editoren wie NotePad (WIN) und KEdit(Linux) verwenden, diese stellen allerdings kein Syntax-Highlighting zur Verfügung.
Abraten möchte ich noch von Editoren wie WordPad und ähnlichen Programmen, die in der Lage sind auch RTF-Texte zu erzeugen. Zu leicht kommen in diese Texte irgentwelche unsichtbaren Sonderzeichen rein, über die dann der Compiler stolpert.

1.2 Grundkenntnisse für die Konsole.

1.2.1 Windows / DOS

- **dir**
  Zeigt den Inhalt des aktuellen Verzeichnisses an. Ist das Verzeichnis zu groß, als das man alles lesen könnte helfen die Parameter: "/p" "/w" weiter. Mit "/h" als Parameter bekommt man eine kurze Hilfe zum Befehl dir.
  Beispiel:

```
dir /h
```

- **cd** *Verzeichnisname*
  Verzeichnis wechseln. "." ist aktuelles Verzeichnis. ".." entspricht vorherigen Verzeichnis.
  Beispiel:

```
cd c:\windows
```

- **mkdir** *Verzeichnisname*
  Neues Verzeichnis anlegen.
  Beispiel:

```
mkdir neuesVerzeichnis
```

1.2.2 Linux

- **ls** *-al*
  Zeigt den Inhalt des aktuellen Verzeichnisses an. Ist das Verzeichnis zu groß, als das man alles lesen könnte hilft der Zusatz "| more" .
  Beispiel:

```
ls -al | more
```

- **cd** *Verzeichnisname*
  Verzeichnis wechseln. "." ist aktuelles Verzeichnis. ".." entspricht vorherigen Verzeichnis. "∼" ist das Home-Verzeichnis.
  Beispiel:

```
cd ~/java
```

- **mkdir** *Verzeichnisname*
  Neues Verzeichnis anlegen.
  Beispiel:

```
mkdir neuesVerzeichnis
```

1.3 Das Java-software-development-kit (sdk, früher jdk)

Dies ist das wichtigste. Ob ihr auf eurem Rechner ein sdk installiert habt könnt ihr in der Konsole feststellen, wenn ihr "javac" eingebt. Entweder wird der Java Compiler gefunden, oder ihr müßt es erst noch richtig installieren. Ihr solltet mindestens die Version jdk1.2.2 haben.

2 Der compiler und die virtual machine

Java ist eine Betriebssystemunabhängige Sprache. Dies hat in der Entwicklung und vorallem in der Wartung von Programmen erhebliche Vorteile. Programme die in Java geschrieben sind laufen auf (fast) jedem Rechner. Genauer gesagt auf jedem Rechner, auf dem eine Java-Virtual-Machine (JVM) installiert ist. Dies ist ein Programm, was den Bytecode, der vom Java-compiler erzeugt wurde interpretiert und auf dem jeweiligen System laufen läßt. Für ein Programm bedeutet dies, das es nur eine Version des Bytecodes gibt, den die verschiedenen virtual machines interpretieren können.

2.1 javac und java

Dies ist der Name des compilers. Sobald ihr ein Programm (Xyz.java) geschrieben habt, speichert ihr es ab. Wechselt in das Verzeichnis und ruft den Java-compiler auf.
Beispiel:

```
javac Xyz.java
```

Der compiler erzeugt eine Datei names "Xyz.class". Dies ist die "ausführbare" Datei. Möchtet ihr das Programm laufen lassen startet ihr die Virtual-Machine und übergebt ihr euer Program als Parameter.
Beispiel:

```
java Xyz
```

3 Java ist Objektorientiert

Objektorientierte Programmiersprachen (wie z.B. C++, Java, Delphi ...) basieren auf einem anderem Konzept, als prozedurale Sprachen (C, Pascal ...). Bei einer Prozeduralen Sprache ist es ist möglich Funktionen zu definieren, die eine bestimmte Aufgabe erfüllen. Diese Funktionen benötigen zum arbeiten meist Funktionsparameter, die beim Aufruf übergeben werden. Eine Funktion hat immer einen Rückgabewert, der allerdings auch leer sein kann. Alle Informationen die eine Funktion zum ausführen braucht müssen ihr übergeben werden, oder global definiert sein.
Eine Objektorientierte Sprache dagegen werden Funktionen (Methoden) und Variablen in einen Zusammenhang mit einem bestimmten Objekt gesetzt. Einen solchen Verbund bezeichnet man als Klasse. Am besten ist dies an einem Beispiel zu erklären.
Nehmen wir einen Jabberwok [1], ein urzeitliches drachenähnliches Wesen. Ein Jabberwok hat folgende Eigenschaften:

- ist hungrig
- ist wütend
- speit Feuer
- hat 365034 Schuppen

Nun natürlich lassen sich für ein solches Wesen noch andere Eigenschaften definieren, aber diese sollten fürs Beispiel genügen. Wie können wir einen solchen Jabberwok nun mit Hilfe eines Programms fassen und noch schlimmer, was hat denn ein Jabberwok mit Objektorientiertheit zu tun?
Betrachten wir den Jabberwok als Objekt mit oben genannten Eigenschaften. Die Eigenschaften des Jabberwok lassen sich als Variablen verwirklichen, der Jabberwok als solches ist eine Klasse. Wir könnten also schreiben:

```
public class Jabberwok {
  boolean istHungrig;
  boolean istWütend;
  boolean speitFeuer;
  int schuppen;
```

[1] Quelle: Beispiel aus "Java in 21 Tagen"

```
}
```

Nun macht es aber auch Sinn bestimmte Methoden mit einem Jabberwok in Verbindung zu bringen. Zum Beispiel:

- fresse
- spiele
- dusche
- tutSichWeh

Nun können wir unsere Klasse Jabberwok vervollständigen. Methoden, die wir mit unserem Jabberwok in Verbindung bringen, haben natürlich auch Auswirkungen auf die Eigenschaften unseres Jabberwoks.

```
public class Jabberwok {
  boolean istHungrig;
  boolean istWuetend;
  boolean speitFeuer;
  int schuppen;

  public void fresse() {
    // nun ist er nicht mehr hungrig.
    istHungrig = false;
  }
  public void spiele() {
    // das beruhigt ihn!
    istWuetend = false;
  }
  public void dusche() {
    // Wasser scheint gegen Feuerspeien zu helfen, aber es macht hungrig!
    istHungrig = true;
    speitFeuer = false;
  }
  public void tutSichWeh() {
```

```
      // ohhh wie gemein, das macht ihn wütend und er speit Feuer!!!!
      istWuetend = true;
      speitFeuer = true;
    }
  }
```

Nun haben wir Methoden und Variablen zusammen mit einem Jabberwok zu einer Klasse verknüpft. Eine Klasse ist sozusagen eine Beschreibung eines beliebigen Jabberwok's. Sozusagen eine Art "Bauanleitung". Doch wie kommt unser Jabberwok zur Welt. Bei der "Geburt" unseres Jabberwoks, hat er bestimmte Eigenschaften, die ihn kennzeichnen. Für eine Klasse bedeutet das, daß bestimmte Variablen vorgegebene Werte haben. Soll ein Jabberwok zum Beispiel immer hungrig zur Welt kommen, so ist das Einfachste zu schreiben:

```
boolean istHungrig = true;
```

Eine andere Möglichkeit besteht darin in der Bauanleitung solche Vorbesetzungen zu treffen. Dies geschieht in einem sogenannten Konstruktor. Ein Konstruktor wird immer beim Erzeugen eines neuen Objektes aufgerufen, sozusagen bei der Geburt unseres Jabberwoks. Dort kann man auch Eigenschaften zuweisen.

```
  public class Jabberwok {
    boolean istHungrig;
    boolean istWuetend;
    boolean speitFeuer;
    int schuppen;

    /** Konstruktor */
    public Jabberwok() {
      istHungrig = true;
      istWuetend = false;
      speitFeuer = false;
      schuppen = 365034;
    }
```

```
  public void fresse() {
    // nun ist er nicht mehr hungrig.
    istHungrig = false;
  }
  public void spiele() {
    // das beruhigt ihn!
    istWuetend = false;
  }
  public void dusche() {
    // Wasser scheint gegen Feuerspeien zu helfen, aber es macht hungrig!
    istHungrig = true;
    speitFeuer = false;
  }
  public void tutSichWeh() {
    // ohhh wie gemein, das macht ihn wütend und er speit Feuer!!!!
    istWuetend = true;
    speitFeuer = true;

    // und er verliert ein paar Schuppen.
    schuppen = schuppen - 10;
  }
}
```

4 Ein erstes Programm

So könnte euer erstes Java-Programm "HelloWorld.java" aussehen:

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

4.1 Die Bedeutung der einzelnen Zeilen.

4.1.1 Das Schlüsselwort public

public bedeutet soviel wie "öffentlich". Es zeigt an, daß andere Klassen auf diese Klasse zurückgreifen können. Andere Schlüsselwörter die eine ähnliche Bedeutung haben sind:

- public
  allgemein zugänglich.
- package
  ist nur in einem engeren Rahmen (dem package) meist innerhalb eines Projektes bekannt.
- protected
  ist nur Subklassen bekannt.
- private
  ist nicht allgemein bekannt, kann nur von einer "Äußeren Klasse" oder der Klasse selbst benutzt werden.

4.1.2 class

class ist ein Schlüsselwort, das angibt, das nun eine Klasse folgt.

4.1.3 static

static bedeutet, das diese Methode auch benutzt werden kann, ohne das man schon eine Instanz der Klasse besitzt. Innerhalb einer solchen Methode kann man nur auf die statischen Variablen und Methoden zugreifen. Variablen, die ein spezielles Objekt beschreiben, oder Methoden können von hier nicht aufgerufen werden. Es ist auch möglich dieses SChlüsselwort vor class zu schreiben.

4.1.4 public static void main(String[] args)

Dies ist die komplette Zeile des Hauptprogramms, also dem Teil, der beim Starten des Programms ausgeführt wird. String[] steht für einen Array von Strings (Zeichenketten), die dem Programm beim Aufruf als Parameter übergeben werden. Der Name des Arrays ist "args".

4.1.5 System.out.println("HelloWorld")

Diese Zeile schreibt "HelloWorld" auf die Konsole. System ist eine Klasse, die grundlegende Methoden und Variablen zum Umgang mit dem Betriebssystem enthalten. So zum Beispiel die Variable "out", einem Stream, der die Standart-Ausgabe des Programms repräsentiert. Für diesen Stream ist die Methode "println(...)" definiert, die eine ganze Zeichenkette ausschreibt. In unserem Fall "HelloWorld".

4.2 Die Namensgebung der Klasse

WICHTIG!
Wenn ihr ein Programm abspeichert nennt die Datei unbedingt genauso wie ihr die Klasse genannt habt und hängt noch die Endung ".java" hintendran. Ansonsten kann der Compiler die Datei nicht übersetzen! Groß und Kleinschreibung beachten!
Übrigens, es ist Konvention, das alle Klassennamen mit einem Großbuchstaben beginnen sollten. Man tut gut daran dies einzuhalten.