Java Einführung

Problem

Allgemein gilt, dass das Internet (fast) jedem nur Vorteile gebracht hat, es verbindet Leute auf der ganzen Welt und man hat somit als Programmierer auch die Möglichkeit einem sehr großen Publikum seine Machwerke mit einem sehr niedrigen Budget vorzuführen. Doch das Internet hat auch Nachteile, so handelt es sich bei dem Internet um einen Zusammenschluss der verschiedensten Rechnertypen mit den unterschiedlichsten Betriebssystemen. Für den Programmierer ist es nun nicht so einfach jedem gerecht zu werden. Entweder muss er nur die wichtigsten Betriebssysteme bedienen oder sich die unendliche Arbeit machen und für jedes einzelne Betriebssystem eine eigene Version zu erstellen, da diese untereinander nicht kompatibel sind.

Lösung mit Java

Mit der Programmiersprache Java ist es dem Programmierer einfacher gemacht worden. Warum alle Arbeit selbstmachen und die Programme selbst für die einzelnen Betriebssysteme anpassen; dies kann doch ein intelligenter Interpreter für einen machen. Ein Interpreter ist ein Programm dass den Quellcode Zeile für Zeile für das Betriebssystem übersetzt auf dem es läuft. Programme die mit Java geschrieben werden, werden nicht wie üblich durch einen Compiler gejagt, sondern liegen im sogenannten Bytecode vor.
Ein Compiler ist ein Programm, dass den Quellcode in Maschinensprache übersetzt.
Der Interpreter für den Bytecode ist speziell auf das Betriebssytem und der Bytecode ist universell verwendbar, das Programm ist also auf jedem Computer auf dem der Interpreter läuft lauffähig. Der Java-Interpreter, die Java Virtual Machine ist auf fast alle Betriebssystem verfügbar (Windows, Linux, Mac, Solaris, UNIX, BeOS, QNX, ...)

Java

Java selbst ist noch eine sehr junge Programmiersprache; die von Sun entwickelte Programmiersprache vereint verschiedene Fähigkeiten von anderen modernen Programmiersprachen in einer Programmiersprache. So wird vielen auffallen, dass der Aufbau sehr an C++ erinnert und auch Einflüsse auf ObjectPascal sind zu finden. Jedoch ist Java im Vergleich zu C++ einfacher zu erlernen, dafür ist Java aber auch um einige Fähigkeiten ärmer. Vor allem ist Java nicht so systemnah wie C++.

Möglichkeiten der Java-Programmierung

Java wird kostenlos von Sun vertrieben, jeder der mit Java programmieren will, kann sich die sogenannte JDK von der Website von Sun herunterladen. JDK steht für **J**ava **D**evelopment **K**it und wird von Zeit zu Zeit auf eine neuere Version geupdated (z.Z. ist die Version 1.4.1 der JDK aktuell). Jedoch wird die JDK mit ihren 45 MB Größe wohl für so manchen Modem und ISDN-User etwas groß sein, um es mal eben herunterzuladen. Solche Nicht-DSL-Nutzer empfehle ich sich nach einem Java-Buch mit CD-ROM umzusehen, auf denen befindet sich neben so manch nützlichem Programm auch oft eine Version der JDK. Wer sich das Buch nicht kaufen will, kann es in machen Büchereien ausleihen.
Jedoch wird das entwickeln mit der JDK für den Anfänger sehr unkomfortabel und kompliziert sein, da keine IDE (Integrierte Programmierumgebung) mitgeliefert wird. Man muss den Quellcode mit einem Texteditor, wie Notepad, schreiben, unter Datei.java abspeichern und anschließend mit der javac.exe in Bytecode umwandeln lassen (*javac Datei.java*). Mit java.exe kann man sich anschließend die Bytecode-Datei

interpretieren lassen (*java Datei*). Das ist sehr kompliziert!
Für alle die es etwas komfortabler wollen oder Anfänger sind, die sollten sich zusätzlich zur JDK noch eine IDE besorgen.

IDEs für Java

Wie fast immer ist die Wahl der IDE abhängig vom Geldbeutel und vom Einsatzzweck. Es gibt sehr bekannte IDEs wie Microsoft Visual J++, IBM Visual Age for Java, Borland JBuilder, SunONE und Symantec/WebGain Visual Café. Doch diese IDE sind zum Teil sehr teuer und bieten Fähigkeiten die der normale Programmierer nicht benötigt. Zwar gibt es den JBuilder und SunONE auch als kostenlose Private Editions, doch ich empfehle ein anderes Programm. Außerdem ist SunONE etwas instabil.
Bei meiner Empfehlung handelt es sich im Gegenteil zu den zuvor genannten Programmen um eine IDE die standardmäßig keine GUIs (grafische Benutzeroberflächen) erzeugt. Es handelt sich nur um einen Quelltexteditor. Mein Programm nennt sich JCreator. Das Programm wird in zwei verschiedenen Versionen vertrieben: In einer kostenlosen LE-Version und in der kostenpflichtigen Pro-Version. Für unseren Einsatzzweck reicht jedoch die LE-Version völlig aus. Außerdem ist der JCreator wesentlich kleiner als die zu Anfang aufgeführten Programme, es ist gerade mal etwas größer als 2 MB.

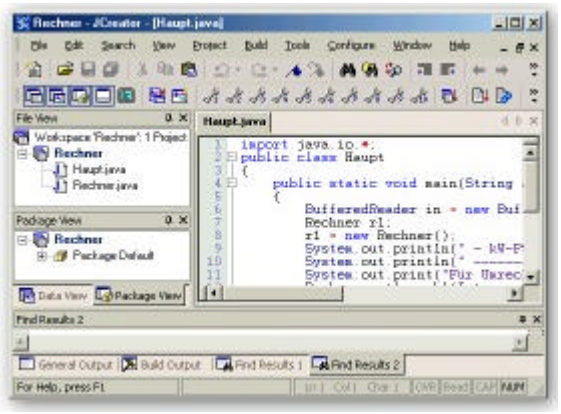
Wichtige Fähigkeiten vom JCreator

Bevor ich wirklich in die Oberfläche einsteige und euch zeige wie man mit dem JCreator arbeitet, möchte ich euch die Fähigkeiten nennen, die diese IDE so gut machen.
Im Gegensatz zu der Arbeit mit Notepad ist zu allererst einmal die Quellcodeformatierung und -hervorhebung zu nennen. Dieses Feature sticht einem zuerst ins Auge, wenn man das Programm gestartet hat. Wird eine Klasse oder eine Bedingungsabfrage mit einer geschweiften Klammer eingeleitet (näheres dazu später) so werden die folgenden Zeilen automatisch eingerückt, so dass der Quellcode übersichtlich und besser lesbar wird. Außerdem lässt sich mit einem kleinen Pulssymbol Quelltext in geschweiften Klammern Platz sparend minimieren. Außerdem werden Klammern und geschweifte Klammern automatisch ergänzt. Schlüsselwörter werden farbig hervorgehoben, so dass sie sofort ins Auge stechen wenn man den Quellcode betrachtet. Außerdem fällt einem die Möglichkeit auf, dass an die Java-Dateien in einem Workspace und einem Projekt zusammenfassen kann, wenn man mit vielen Klassen verteilt auf verschiedene Dateien arbeitet hilft einem dies immens. Die Dateien werden dann übersichtlich dargestellt. Auch zeigt das Programm Klassen grafisch in einer Baumstruktur an. Für Anfänger wird die Assistenten gestützte Programmerstellung interessant sein. Wenn man dann das Programm in Bytecode umwandeln lassen will, will man schnell das Feature des Anzeigens von Fehlern, deren Position und die Beschreibung nicht mehr missen.
Profi-Programmnutzer werden über die schnelle Geschwindigkeit des Programms erstaunt sein, da halten die trägen Profiprogramme nicht mehr mit.

Bevor es losgehen kann

Bevor ich mit der Beschreibung beginnen kann muss ich vor dem ersten Start noch einige Einstellungen machen. JCreator ist zwar eine IDE für Java, jedoch wird immer noch die JDK von Sun benötigt. Diese wird aber nicht mit dem Programm mitgeliefert. Zuvor muss also dies erst einmal installiert sein. Doch dies reicht immer noch nicht, denn der JCreator braucht davon noch den Pfad. Genau diesen verlangt das

Programm dann auch beim ersten Start. Gib also den Pfad des Homeverzeichnisses des JDK an und der weiteren gefragten JDK-Programmteile an. Erst danach wird dich folgendes Bild erfreuen (natürlich ohne das Programm):



JCreator im Detail

Zuerst muss ein neues Projekt erstellt werden. Dies geht indem du auf *File/New* klickst und im dann zu sehenden Fenster *Empty Project* auswählst. Nun musst du rechts dem Projekt noch einen Namen geben. Nun ist ein Projekt erstellt worden, dieses enthält jedoch noch keine Dateien. Java-Dateien fügt man ein, indem man *File/New* ein das Fenster aufruft und dort *Java File* auswählt, nachdem man rechts der Java-Datei einen Namen gegeben hat, wird die Java-Datei angelegt.

Nun erst kann ich dir die Oberfläche von JCreator erklären, also das Fenster des JCreators besteht aus 3 kleineren Fenstern und einem großen Fenster. In das große Fenster wird der Quelltext eingegeben, automatisch eingerückt und markiert. An der linken Seite des Fensters ist die aktuelle Zeile eingeblendet, das vereinfacht die Orientierung und macht die Fehlersuche später einfacher. Im linken oberen Fenster ist ein Projekt-Explorer, der dir die Dateien deines Projektes und des Workspaces anzeigt. Darunter werden die in der Java-Dateien befindlichen Klassen übersichtlich angezeigt. Im langen Fenster unten werden später die Ergebnisse der Umwandlung in Bytecode angezeigt und natürlich auch etwaige Fehler. Ein Klick an den Fehler führt zum Sprung in die Zeile, in der sich der Fehler befindet.

Wie bei allen Windows-konformen Programmen gibt es auch noch eine Symbolleiste, die neben den üblich Icons für die Standartoperationen speichern, öffnen, usw. auch JCreator spezifische Icons beinhaltet. Die wichtigsten dieser Knöpfe befinden sich im hinteren Drittel. Es handelt sich um die Knöpfe (); mit dem ersten dieser Knöpfe kann man die aktuelle Java-Datei kompilieren; mit dem zweiten wird die aktuelle kompilierte Java-Datei ausgeführt; mit dem dritten werden alle Dateien des Projekts kompiliert und mit dem letzten dann folgerichtig alle kompilierten Java-Dateien des Projekts ausgeführt.

Und wie gehts weiter?

Ja, nun das war das wichtigste was du über die Arbeit mit dem JCreator wissen müsstest. Sollte es aber noch Probleme geben, so hilft dir die Hilfe des JCreators eventuell weiter (leider wie das gesamte Programm auf englisch).

Projekt erstellen

Ich habe es zwar bei der Beschreibung der Programmoberfläche schon mal erklärt,

doch so ganz so einfach ist es nicht. Also noch mal: Klicke auf *File/New* nach dem der JCreator gestartet ist. Nun erscheint ein Fenster, in diesem wähle dann den Punkt *Empty Project* aus. Gebe deinem Projekt noch einen schönen Name, ich würde für unsere erste Übung "***Hello***" vorschlagen. Das Eingabefeld dafür findest du rechts.
Danach wirst du ein recht leeres Fenster vorfinden. Es befinden sich nämlich noch keine Java-Dateien in unserem Projekt. Füge also unsere erste Java-Dateien in das Projekt ein, indem du das Dateierstellungsfenster mit *File/New* aufrufst. Wähle dort den Punkt *Java File.* Nun musst du noch einen Namen vergeben, ich würde wieder "***Hello***" vorschlagen. Gebe dies also rechts an. Sehr wichtig ist, dass du dir den Namen der Java-Datei merkst, da jede Java-Datei aus einer Klasse bestehen muss, die so heißt wie die Java-Datei.

Jetzt kann's los gehen

Jetzt ist alles dafür vorbereitet mit dem Java programmieren zu beginnen. Darum wollen wir dann auch einmal anfangen. Ich will das Programm Hello modular erweitern. Zuerst einmal soll es nur "Hello World" ausgeben, sonst nichts.
Klicke also in das Quellcodefenster und gibt folgende Codezeilen ein, die ich an schließend erläutern werde:

```
public class Hello
{
   public static void main (String argv[])
   {
      System.out.println("Hello World");
   }
}
```

Die erste Zeile ist die erwähnte benötigte Klasse, die so heißt wie die Java-Datei. Aber wir wollen uns die Zeile noch etwas genauer ansehen. Die Zeile beginnt mit dem Schlüsselwort *public*, das heißt, dass die Klasse im ganzen Projekt gültig ist. Mit *class* wird angezeigt, dass es sich um eine Klasse handelt und Hello ist der Name der Klasse. Somit lässt sich allgemein folgende Deklaration einer Klasse ableiten: *public/private class Klassenname*

Sehr wichtig ist auch, dass man die folgenden Anweisungen in geschweifte Klammern setzt.
Innerhalb der Klasse (also innerhalb der zwei geschweiften Klammern) befindet sich das *main()*-Objekt, dass in der Startdatei des Java-Projekts vorkommen soll, da wir nur eine Java-Datei haben, muss sie in dieser Datei stehen. Dieses Objekt lassen wir mal weit gehend unkommentiert, da darin z.T. Techniken angewendet werden, auf die ich erst später eingehen möchte, also einfach ohne nachzudenken in die Java-Datei einfügen. Eins möchte ich aber schon verraten, mit void legt man fest, dass dieses Objekt keinen Rückgabewert besitzt, d.h. dass nach der Abarbeitung der in ihr stehenden Anweisungen das Objekt kein Ergebnis zurückgibt. Das Objekt umklammert mit geschweiften Klammern die in ihr stehenden Anweisungen; dies ist üblich, wenn danach mehr als eine Anweisung folgt (normalerweise steht mehr als nur eine Anweisung im *main()*-Objekt).
Nach diesem geheimnisvollen *main()*-Objekt kommt nun die eigentliche Anweisung die den Text "Hello World" auf dem Bildschirm ausgibt.

Es geht weiter!

Das war aber noch sehr einfach. Mit so wenig wollen wir uns auf keinen Fall zufrieden geben; wie ich schon erwähnt habe, werden wir unser Programm modular erweitern. Genau dies wollen wir nun im Folgenden tun.

Soeben haben wir ein klitzekleines Programm mit der Fähigkeit Ausgaben auf dem Bildschirm zu machen erstellt. Was legt also nun näher, als genau das Gegenteil zu tun und nun Eingaben des Benutzers in unser Programm zu implementieren. Wir müssen dafür unser Programm nur sehr marginal verändern. Hier erst mal der neue Quelltext (nur die fetten Zeilen sind neu):

```
import java.io.*;
public class Hello
{
    public static void main (String argv[]) throws IOException
    {
        String eing;
        BufferedReader eingabe = new BufferedReader(New InputStream-
        Reader(System.in));
        System.out.print("Bitte Name eingeben:");
        eing = eingabe.readLine();
        System.out.println("Hello World! Hello " + eing);
    }
}
```

Mit der ersten neuen Zeile werden Befehle aus der Bibiliothek io in das Programm importiert, aus dieser benötigen wir Eingabefunktionen (io = **I**nput/**O**utput). Dies wird wie gut zu sehen ist nach dem Schema import *java.Bibliothek* gemacht. Mit dem Asterix (.*) teilt man mit, dass man alle Funktionen importiert werden sollen.

Die *main()*-Methode müssen wir auch noch geringfügig abändern. Mit dem Befehl throws *IOException* teilen wir dem Programm mit, dass es sich nicht um eventuelle Ein- und Ausgabefehler kümmern soll. Da Java ein Programmiersprache ist die sehr auf Sicherheit und Fehlerfreiheit wert legt ist, stellt die Ein- und Ausgabe für sie ein enormes Fehlerpotential dar, weshalb eine solche Anweisung notwendig ist. Grundsätzlich werden Fehler zu Gruppen zusammengefasst. Bei der Eingabe mögliche Fehler sind folglich in der Gruppe *IOException* enthalten.

Mit *String eing* deklariert man die Variable eing vom Datentyp *String*. Wobei die Deklarierung einer Variable vom Datentyp String eine Ausnahme ist, denn normalerweise deklariert man so: *datentyp variablenname*

Alle die sich jetzt wundern, was denn da anders sein soll, schließlich habe ich den Datentyp einfach nur klein geschrieben, sei gesagt, dass es genau so ist! Die kontextsensitive Programmiersprache Java möchte Datentypen und Variablenname kleingeschrieben haben. String ist eine Methode die dem Benutzer neben dem eigentlichen deklarieren noch weitere Funktionen.

Die nächste Zeile ist lang und kompliziert! Der BufferedReader ist ein Objekt aus der IO-Bibiliothek, die wir am Anfang des Quelltextes importiert haben. Mit "*BufferedReader eingabe*" erzeugt man das Objekt *eingabe*, dass die Eigenschaften der Klasse *BufferedReader* hat. Mit dem Rest wir das Objekt *eingabe* definiert und deklariert. Mit *new BufferedReader()* erzeugt man nun wirklich das neue Objekt. In der Klammer befinden sich Parameter, die das Objekt haben soll. In der Klammer wird der *InputStreamReader* aufgerufen, der ebenfalls in der IO-Bibiliothek zu finden ist.

Auch dieser hat einen Parameter. Der ist die Methode *System.in*.
Die nächste Zeile sollte auf den ersten Blick bekannt sein, jedoch habe ich sie geringfügig verändert, denn ich habe *.print* statt *.println* geschrieben. Was ist den der Unterschied zwischen diesen beiden Befehlen? Nun mit dem ersten - von mir hier gewählten, wir kein Zeilenumbruch nach der Zeile gemacht, beim zweiten Befehl ist dies der Fall. Warum soll aber kein Zeilenumbruch gemacht werden? Ganz einfach, es handelt sich nur um die Beschreibung was getan werden soll, die auf dem Bildschirm ausgegeben wird. Da wollen wir danach unsere Eingabe machen und uns nicht verwirren lassen, dass der Cursor in die nächste Zeile springt wenn wir etwas eingeben wollen. Ganz einfach, oder?
Während wir am Anfang damit beschäftigt waren das Objekt eingabe zu erstellen, werden wir es nun einsetzen. Mit der nächsten Zeile wird es erst benötigt. Wir weisen der Variable eing den Inhalt der Eingabe zu, die mit *readLine()* in den Speicher gelesen wird.
Mit der letzen Zeile schließlich wird dann unser Werk ausgegeben, Also "Hello World! Hello Name".

Programmieren die dritte

Nun setze ich ein paar Programmiergrundlagen ein um ein anspruchsvolles Projekt zu realisieren.

Vorüberlegung

Bevor man mit dem Programmieren anfängt, muss man sich einmal grundsätzlich mit der Problemstellung befassen. Was sind eigentlich Primzahlen? Also, Primzahlen sind Zahlen die nur durch sich selbst und eins teilbar sind. Wie stellt man nun fest ob es sich bei einer Zahl um eine Primzahl handelt? Man probiert aus, ob sie sich durch irgendeine Zahl teilen lässt. Wenn dies der Fall ist, dann handelt es sich um KEINE Primzahl.
Aus dieser Vorüberlegung ergibt sich folgende Vorgehensweise: Versuchen die eingegebene Zahl durch alle Zahlen zu teilen. Dies lässt sich noch weiter konkretisieren: Man muss nur Zahlen bis zur eingegebenen Zahl prüfen, alle darüber sind nicht glatt teilbar. Außerdem können wir uns auch noch sparen die Zahl durch eins zu teilen, da dies ja bei allen Zahlen möglich ist. Natürlich ist auch die Überprüfung der Zahl selbst als Divisor überflüssig, da ja eine Primzahl auch durch sich selbst teilbar ist.

Vorgehensweise

Ich habe, um das Problem zu lösen, eine Schleife entwickelt, die sich solange wiederholt wie gilt: eingegebene Zahl minus eins ist größer gleich der Divisor. Warum ich von der Zahl eins abziehe, habe ich in der Vorüberlegung klar gelegt: Man muss die Zahl selbst ja nicht überprüfen. Der Divisor ist eine Variable und mit zwei initialisiert. In der Schleife selbst führe ich eine Modulo-Division durch, die den Rest der Division aus gibt. Ist der Rest null, dann ist die Zahl durch den aktuellen Divisor teilbar und ich weiß, dass es sich um keine Primzahl handelt.

Umsetzung in Java

Ich habe ja soeben detailliert die Vorgehensweise beschrieben, so dass die Umsetzung ich Java keine große Kunst mehr ist:

```
import java.io.*;
public class Primzahl
{
  public static void main(String argv[]) throws IOException
```

```
{
  BufferedReader in = new BufferedReader (new InputStream-
  Reader(System.in));
  int zahl;
  int div;
  int erg;
  boolean prim;
  prim = true;
  div = 2;
  System.out.println(" - Primzahl-Test -");
  System.out.println(" -------------");
  System.out.print("Bitte Zahl eingeben:");
  zahl = Integer.parseInt(in.readLine());
  while((zahl-1)>=div)
  {
   erg = zahl % div;
   if (erg == 0)
   {
     System.out.println(zahl+ " ist KEINE Primzahl");
     prim = false;
     break;
   }
   div++;
  }
  if (prim==true)
  System.out.println(zahl+ " ist eine Primzahl");
 }
}
```

Mit der ersten Zeile wird wie gehabt die Java-Bibliothek io geladen, in der sich Elemente für die Ein- und Ausgabe mit Java befinden. Danach geht es mit dem Block der Hauptklasse Primzahl weiter (immer daran denken, dass der Klassenname der Hauptklasse exakt mit dem Dateiname der .java-Datei übereinstimmt).
Wie bei jeder Eingabe muss der *main*-Block durch throws IOException erweitert werden. Danach kommt ebenfalls typisch für die Eingabe die Deklaration des Objekts *in* vom Typ BufferedReader.
In den folgenden vier Zeilen werden alle benötigten Variablen deklariert. *zahl* soll später die eingegebene Zahl aufnehmen; *div* wird den aktuellen Divisor aufnehmen, *erg* das Ergebnis der Modulo-Division und *prim* soll uns sagen ob es sich um eine Primzahl handelt oder nicht. In den nächsten zwei Zeilen werden die Variablen dann deklariert. *prim* soll als Anfangswert *true* haben, wir gehen also davon aus dass es sich um eine Primzahl handelt. Der Divisor soll aus bekannten Gründen (durch eins sind alle Zahlen teilbar => unnötig zu überprüfen!) mit dem Anfangswert '2' arbeiten.
Die nächsten zwei Zeilen verschönern unser Programm etwas, es wir eine Überschrift ausgegeben und diese wird auch noch schön unterstrichen. Erst die nächste Zeile hat wieder eine Funktion. Es soll eine Eingabeaufforderung an den Benutzer ausgegeben werden, in der er gebeten wir die Zahl einzugeben die überprüft werden soll. Diese Zahl wird dann in den Datentyp Integer konvertiert (*Integer.parseInt()*) und der Variable *zahl* zugewiesen.

Nun fangen wir auch schon gleich mit dieser Variable an zu arbeiten. Wir schreiben eine *while*-Schleife die sooft wiederholt wird bis nicht mehr gilt, dass die eingegebene Zahl minus eins größer als der aktuelle Divisor ist.
Nun führen wir eine Modulo-Division durch deren Ergebnis wir der Variable *erg* zuweisen. Mit der folgenden *if*-Kontrollstruktur wird überprüft ob der Rest der Modulo-Division null war, d.h. ob sich die Zahl glatt durch den aktuellen Divisor teilen lässt. (Achtung: Vergleiche werden in Java mit doppeltem Gleichheitszeichen gemacht!) Ist dies der Fall, so handelt es sich bei der Zahl nicht um eine Primzahl. Dies geben wir mit der folgenden Zeile aus und vermerken es in der Variable *prim*. Mit *break* verlassen wir die Schleife, schließlich haben wir schon herausgefunden, dass es sich um keine Primzahl handelt. Weitere Überprüfungen können wir uns ersparen.
Bei jedem Schleifendurchlauf erhöhen wir den Divisor *div* um eins und das geht sehr einfach mit dem Postinkrementoperator (++).
Sollte bei allen Schleifendurchläufen bei der Modulo-Division ein Ergebnis ungleich null erzielt worden sein, so wird bei der nächsten if-Kontrollstruktur eine Meldung ausgegeben, dass es sich um eine Primzahl handelt.