From: Michael Paap

Ein Kommilitone wrote:

> *> Ich will mich mal nur auf diesen Teil der Diskussion beschränken und*
> *> eine Vermutung anstellen, warum eine Interface-Deklaration "per*
> *> Definitionem" auch ein Objekt ist:*
> *>*
> *> Auf diese weise hätte man die Möglichkeit, Interfaces an manchen*
> *Stellen*
> *> wie Objekte zu behandeln.*

Bitte... Ihr dürft mich ja gerne als krümelpickenden Pedanten verfluchen, aber es erleichtert die Diskussion wirklich ungemein, wenn man sich um eine exakte Sprache bemüht. "Interfaces" kann man nicht als Objekte behandeln, vom Reflection-Kontext jetzt mal abgesehen. Man kann Objekte eines Interface-Typs als Objekte behandeln.

> *> Beispielsweise könnte man mittels "instanceof*
> *> <Interfacename>" prüfen, ob eine Klasse ein bestimmtes Interface*
> *> implementiert und dann die eine im Interface beschriebene Methode*
> *> aufrufen, oder eben nicht.*

Das kann man, ja. Aber man kann noch viel mehr.

> *> Sollte meine Vermutung nicht zutreffen, wäre es vielleicht gut zu*
> *> wissen, warum jedes Interface "per Definitionem" ein Object ist - und*
> *> das stünde dann vielleicht auch sinnvollerweise im Script.*

Aehm... noch mal: es ist nicht ***jedes Interface*** ein Objekt. Aber jedes Objekt, welches einer Klasse angehört, die ein bestimmtes Interface implementiert, ist ein ***Object***. Denn jedes Objekt ist ein ***Object***. Ich glaube, das sollte man einfach mal ganz ohne zusätzliche Motivation und Begründung als Tatsache zur Kenntnis nehmen. :-)

Aber wenn es denn sein soll: Ich versuche mal zu beschreiben, was man u.a. mit Interfaces so anstellen kann. Ich denke, wenn man ***das*** verstanden hat, erledigt sich die Frage, warum Interfacetypen als Subtypen von Object angesehen werden, von selbst. Das stellt natürlich einen Vorgriff dar und mein Beispiel unten geht über den Kursstoff hinaus. Insofern muss sich niemand Sorgen machen, der das Ganze nicht komplett nachvollziehen kann.

Wie ich in einem anderen Posting schon schrieb: Die Verwendung von Interfaces als Typen von Variablen erlaubt einem, bzgl. der konkret verwendeten Klassen flexibel zu bleiben. Das ist wichtig: Man kann eine Bibliothek schreiben, die bestimmte Funktionen einer anderen Bibliothek benutzt, ohne überhaupt zu wissen, wie diese andere Bibliothek konkret aussieht. Die Klassenbibliothek besteht in einigen Ecken in erster Linie aus Interface-Deklarationen, deren Sinn darin besteht, dass Dritte Implementierungen liefern sollen. Manchmal gibt es dann von Sun eine Referenzimplementierung, manchmal aber auch nicht.

Ich kann ein Programm schreiben, ohne zu wissen, welche Implementierung der verwendeten Interfaces zur Laufzeit vorhanden sein wird, alleine deswegen, weil ich mich darauf beschränke "gegen das Interface" zu programmieren. Die konkreten Objekt kann dann z.B. eine sog. "Factory" liefern, welche zur Laufzeit entscheidet, welche konkrete Implementierung sie liefert. Die Entscheidung kann durch Interaktion mit dem Benutzer fallen, durch Aufrufparameter, Konfigurationsdateien oder schlicht dadurch, welche das Interface implementierenden Klassen in einem bestimmten Verzeichnis gerade ladbar vorhanden sind. Das "Plugin"-Konzept lässt grüßen.

Ich kann mein Programm auch schon schreiben und übersetzen, ***bevor*** überhaupt irgendeine Implementierung für die Interfaces existiert. Ich kann mein Programm völlig unabhängig von der Implementierung der Interfaces ausliefern und es dem Kunden überlassen, eine entsprechende

Bibliothek wo auch immer zuzukaufen.

Aber natürlich will ich dennoch die Objekte von dem Interfacetyp, mit denen ich operiere, in Variablen legen können. Ich will sie auch in eine LinkedList packen können oder in ein Array. Ersteres geht, weil diese Objekte alle auch vom Typ Object sind. Letzteres geht z.B., indem ich das Array als ein Array vom Interfacetyp deklariere oder auch als Object[].

Ich veranschauliche das Programmieren "gegen Interfaces" mal an einem Beispiel, in dem ich auch gleich noch einmal das Hantieren mit komplexen Packagestrukturen zeige. Noch mal: Das geht über den Kurs definitiv hinaus, es braucht sich also niemand sorgen zu machen, der es nicht vollständig versteht.

Passend zur Jahreszeit wollen wir ein wenig grillen. Was genau heute auf den Grill kommt, wissen wir aber erst zur Laufzeit... ;-)

Nehmen wir an, wir haben die folgenden Klassen. Das Package liegt bei mir im Verzeichnis D:\eclipse\1618, d.h. dort gibt es also ein Verzeichnis de, in diesem ein Verzeichnis mpaap usw.

```
package de.mpaap.interfaceSpielchen;

public interface Grillbar {
   String getGrillanleitung();
}
```

```
package de.mpaap.interfaceSpielchen;

/*
 * Diese Klasse ist ein sogenanntes Singleton, d.h. es existiert
 * immer nur maximal ein Exemplar der Klasse, welches man
 * mit Hilfe der statischen Klasse getSingleInstance() erhalten kann.
 * Dass keine weiteren Exemplare erzeugt werden können,
 * wird durch den privaten Konstruktor gewährleistet.
 */
public class GrillbarFactory {

   private static GrillbarFactory singleton = new GrillbarFactory();

   private GrillbarFactory() {}

   public static GrillbarFactory getSingleInstance() {
      return singleton;
   }

   Grillbar getGrillBar() throws Exception {
      String grillbarType = System.getProperty("grillbarType");
      return (Grillbar) Class.forName(grillbarType).newInstance();
   }
}
```

```
package de.mpaap.interfaceSpielchen;

public class Grillfete {
   public static void main(String[] args) {
      GrillbarFactory factory = GrillbarFactory.getSingleInstance();
      try {
         Grillbar heutigesAbendessen = factory.getGrillBar();
         System.out.println("Heute Abend gibt es "
               + heutigesAbendessen.getClass().getName() + ".");
```

```
            System.out.println("Bitte bereiten Sie das Grillgut " +
                  "wie folgt zu:");
            System.out.println(heutigesAbendessen.getGrillanleitung());
         } catch (Exception e) {
            System.out.println("Heute ist leider ein Fastentag.");
         }
      }
}
```

Das Ganze compiliert man am einfachsten, indem man die Klasse GrillFete compiliert. Also (bei mir) von D:\eclipse\1618 aus:

```
javac de/mpaap/interfaceSpielchen/Grillfete.java
```

Eine Konkrete Implementierung von Grillbar wurde hierzu noch nicht benötigt.

So. Und nun kann jeder der will, sich seine eigene Klasse schreiben, die Grillbar implementiert, z.B. so:

```
package de.mpaap.grillen;

import de.mpaap.interfaceSpielchen.Grillbar;

public class SchweineNacken implements Grillbar {
   public String getGrillanleitung() {
      return "Schweinenacken in etwa 2 cm dicke Scheiben " +
            "schneiden,\nund in einer Mischung aus Öl, " +
            "Knoblauch und allerlei Gewürzen 20 Minuten\n" +
            "ziehen lassen. Danach gut abtropfen lassen und" +
            " auf den Grill damit!";
   }
}
```

Diese Packagestruktur liegt bei mir in D:\.

Wenn ich nun SchweineNacken compilieren will, benötige ich Zugriff auf das Interface Grillbar, also muss das Wurzelverzeichnis von de.mpaap.interfaceSpielchen.Grillbar im Classpath sein. Also von D:\ aus:

```
javac -classpath "D:\eclipse\1618" de/mpaap/grillen/SchweineNacken.java
```

Und nun will ich die GrillFete starten und dabei der VM den Namen der zu verwendenden Implementierung von Grillbar mitteilen. Dazu gibt es den VM-Parameter -D, dessen Verwendung ich am Beispiel zeige. Natürlich muss das Wurzelverzeichnis der Klasse de.mpaap.grillen.SchweineNacken zum Ausführen im Classpath sein, denn die brauchen wir ja jetzt. Also wieder ins Verzeichnis D:\eclise\1618 und

```
java -cp ".;D:/" -DgrillbarType=de.mpaap.grillen.SchweineNacken
de.mpaap.interfaceSpielchen.GrillFete
```

Das soll eine Zeile sein. Der Parameter -DgrillbarType sorgt dafür, dass der ausführenden VM ein Parameter grillbarType mit dem angegebenen Wert zur Verfügung steht und ausgelesen werden kann (siehe GrillbarFactory).

Und schon weiß ich, wie ich den Schweinenacken grillen kann. Und wer lieber etwas anderes grillen will, der entwerfe eine Implementierung von Grillbar und probiere es aus. ;-)

Natürlich ließe sich das Ganze auch so ausbauen, dass dynamisch in einem bestimmten (z.B. per -D übergebenen) Verzeichnis nach Klassen gesucht wird, die Grillbar implementieren und diese dann dem Benutzer in einem

Menü angeboten werden. Der Benutzer wählt dann aus, was er grillen will,
es wird dynamisch je ein Exemplar erzeugt und alle Exemplare werden in
ein Grillbar[] abgelegt, so dass man hübsch darüber iterieren kann. Und
auf jedem davon kann man z.B. toString() aufrufen, denn jedes ***ist*** ein
Object und hat deshalb diese Methode. Und auf jedem kann man
getGrillanleitung() aufrufen, denn jedes ist ein Grillbar.

Gruß,
Michael Paap

From: Sylvia Schütze

Michael Paap schrieb:

> *> Ich veranschauliche das Programmieren "gegen Interfaces" mal an einem*
> *> Beispiel, in dem ich auch gleich noch einmal das Hantieren mit*
> *komplexen*
> *> Packagestrukturen zeige. Noch mal: Das geht über den Kurs definitiv*
> *> hinaus, es braucht sich also niemand sorgen zu machen, der es nicht*
> *> vollständig versteht.*

Sorgen machen nicht, aber verstehen würd' ich es gern.

> [... interface]
> *> /*
> *> * Diese Klasse ist ein sogenanntes Singleton, d.h. es existiert*
> *> * immer nur maximal ein Exemplar der Klasse, welches man*
> *> * mit Hilfe der statischen Klasse getSingleInstance() erhalten kann.*

wäre "statische Methode" nicht richtig?

> *> * Dass keine weiteren Exemplare erzeugt werden können,*
> *> * wird durch den privaten Konstruktor gewährleistet.*
> *> */*
> *> public class GrillbarFactory {*
> *>*
> *>   private static GrillbarFactory singleton = new GrillbarFactory();*
> *>*
> *>   private GrillbarFactory() {}*
> *>*
> *>   public static GrillbarFactory getSingleInstance() {*
> *>   return singleton;*
> *>   }*

Das der private parameterlose Konstruktor verhindert, dass von irgend
jemanden über new GrillFactory() ein neues GrillFactory-Objekt erzeugt
wird, verstehe ich. Die Variable singleton ist ebenfalls private, also
auch vor Zugriffen geschützt. Aber die Methode getSingleInstance() ist
doch public und könnte doch mehrfach aufgerufen werden, und eifrig
viele GrillbarFactory's liefern?

> *>*
> *>   Grillbar getGrillBar() throws Exception {*
> *>   String grillbarType = System.getProperty("grillbarType");*

Was passiert hier genau? grillbarType ist ein String und im Beispiel
nach Ausführung von getProperty("grillbarType") mit
"de.mpaap.grillen.Schweinenacken" belegt. Also ist nur der
Laufzeitparameter der VM -DgrillbarType= ... entscheidend und hat
Einfluss auf die system properties?

Gibt es irgendwo eine "schöne" Übersicht über die Parameter java/javac?

> *>   return (Grillbar) Class.forName(grillbarType).newInstance();*

Das ist dann wieder klar. Hab ich den vollständigen Namen, kann ich
ein Objekt erzeugen.

Haupterkenntnis: Habe ich ein Interface, kann ich "dagegen"
Programmieren und mich erst zur Laufzeit entscheiden, welche

Implementierung dieses Interfaces ich verwenden möchte. Den
Interfacetyp kann ich verwenden, um zu schreiben und zu compilieren,
möchte ich aber auf ein konkretes Objekt des Interfacetypes zugreifen,
muss schon eine Implementierung da sein, oder "Heute ist leider
Fastentag." :-) Ist das so richtig?

Gruß Sylvia.

From: Michael Paap

Sylvia Schuetze wrote:

> *Sorgen machen nicht, aber verstehen würd' ich es gern.*

Ok. ;-)

> *wäre "statische Methode" nicht richtig?*

Natürlich... sorry.

> *Das der private parameterlose Konstruktor verhindert, dass von irgend*
> *jemanden über new GrillFactory() ein neues GrillFactory-Objekt erzeugt*
> *wird, verstehe ich. Die Variable singleton ist ebenfalls private, also*
> *auch vor Zugriffen geschützt. Aber die Methode getSingleInstance() ist*
> *doch public und könnte doch mehrfach aufgerufen werden, und eifrig*
> *viele GrillbarFactory's liefern?*

Nein, denn sie liefert ja immer dasselbe Exemplar, nämlich das, welches
in der ***statischen*** Variablen singleton referenziert ist. Die
initialisierung dieser Variablen erfolgt ***einmal*** beim Laden der Klasse.

> *>> Grillbar getGrillBar() throws Exception {*
> *>> String grillbarType =*
> *System.getProperty("grillbarType");*
>
> *Was passiert hier genau? grillbarType ist ein String und im Beispiel*
> *nach Ausführung von getProperty("grillbarType") mit*
> *"de.mpaap.grillen.Schweinenacken" belegt. Also ist nur der*
> *Laufzeitparameter der VM -DgrillbarType= ... entscheidend und hat*
> *Einfluss auf die system properties?*

Genau. Das was Du per System.getProperty erhältst, sind zum einen
etliche vordefinierte Eigenschaften des Systems, zum anderen die mit -D
gesetzten Parameter.

```
import java.util.*;

public class SystemProperties {
   public static void main(String[] args) {
      Properties p = System.getProperties();
      Enumeration enum = p.propertyNames();
      while (enum.hasMoreElements()) {
         String element = (String) enum.nextElement();
         System.out.println(element + "   " +
               System.getProperty(element));
      }
   }
}
```

> *Gibt es irgendwo eine "schöne" Übersicht über die Parameter java/javac?*

In den Tooldocs, der Doku zu den Tools and Utilities:
http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html unter "Basic Tools".

> *Haupterkenntnis: Habe ich ein Interface, kann ich "dagegen"*

> *Programmieren und mich erst zur Laufzeit entscheiden, welche*
> *Implementierung dieses Interfaces ich verwenden möchte. Den*
> *Interfacetyp kann ich verwenden, um zu schreiben und zu compilieren,*
> *möchte ich aber auf ein konkretes Objekt des Interfacetypes zugreifen,*
> *muss schon eine Implementierung da sein, oder "Heute ist leider*
> *Fastentag." :-) Ist das so richtig?*

Korrekt.
Was mich interessieren würde: Konntest Du anhand meiner Beschreibung das Beispiel zum Laufen bekommen oder war immer Fastentag? ;-)

Gruß,
Michael Paap

From: Sylvia Schütze

Hallo Michael,

Michael Paap schrieb:
> *>>Das der private parameterlose Konstruktor verhindert, dass von irgend*
> *>>jemanden über new GrillFactory() ein neues GrillFactory-Objekt erzeugt*
> *>>wird, verstehe ich. Die Variable singleton ist ebenfalls private, also*
> *>>auch vor Zugriffen geschützt. Aber die Methode getSingleInstance() ist*
> *>>doch public und könnte doch mehrfach aufgerufen werden, und eifrig*
> *>>viele GrillbarFactory's liefern?*
>
> *Nein, denn sie liefert ja immer dasselbe Exemplar, nämlich das, welches in der* ****statischen**** *Variablen singleton referenziert ist. Die initialisierung dieser Variablen erfolgt* ****einmal**** *beim Laden der Klasse.*

Oh, ja. Das hab ich übersehen.
> *>*
> *> import java.util.*;*
> *>*
> *> public class SystemProperties {*
> *> public static void main(String[] args) {*
> *> Properties p = System.getProperties();*
> *> Enumeration enum = p.propertyNames();*
> *> while (enum.hasMoreElements()) {*
> *> String element = (String) enum.nextElement();*
> *> System.out.println(element + " " +*
> *> System.getProperty(element));*
> *> }*
> *> }*
> *> }*
> *>*

Danke für das Beispiel. Habe es gleich mal in die Grillfete eingefügt und es schreibt mir neben vielen anderen properties auch:
grillbarType ke3.de.mpaap.grillen.Schweinenacken
auf die Konsole.

> *Was mich interessieren würde: Konntest Du anhand meiner Beschreibung das*
> *Beispiel zum Laufen bekommen oder war immer Fastentag? ;-)*

Ja, anhand Deiner Beschreibung habe ich das Beispiel zum Laufen gebracht. Allerdings habe ich eclipse verwendet. 2 Hürden gab's für mich trotzdem:
Vollständige Klassennamen erfordern die Auflistung aller Paketnamen. Ich dachte, es reicht, wenn ich im Paket k3 sitzend, nur de.mppap. .. eingebe, nur ich sitze halt nicht in k3.

Die Stelle in eclipse, an der man die Parameter für die VM eingeben
kann, habe ich nicht sofort gefunden.
Ein paar Rezept mehr hättest Du ja schon anbieten können. Und was
bitte sind denn "allerlei" Gewürze. :-)

Gruß Sylvia.

From: Michael Paap

Sylvia Schuetze wrote:

> *Ein paar Rezept mehr hättest Du ja schon anbieten können.*

Hey, es war irgendwann nach Mitternacht. Ich hatte gerade nach ein paar
Gläsern Wein zum Spaß einen meiner Wohngenossen gefragt, ob er ein
hübsches Beispiel für Interfaces wisse. Daraufhin wollte er wissen, was
um Himmels Willen ein Interface sei. Ich habe es dann versucht, ihm mal
eben zu erklären. Aber der Gute studiert VWL und hat kein Wort
verstanden... und meinte dann "Nimm irgendwas mit Grillschweinen, das
wird immer gern genommen." Das habe ich dann gemacht. Einen so guten Rat
soll man schließlich nicht in den Wind schlagen... aber für mehr als ein
Beispiel hat es dann doch nicht mehr gereicht.

> *Und was bitte sind denn "allerlei" Gewürze. :-)*

Das, was in der Packung "Steak Gewürzzubereitung, Gebr. Wichartz GmbH &
Co. KG, Wuppertal, 1 kg" drin ist. :-)

Gruß,
Michael Paap

From: Sylvia Schütze

Michael Paap schrieb:
> *... aber für mehr als ein*
> *Beispiel hat es dann doch nicht mehr gereicht.*
> *>*
> *>>Und was bitte sind denn "allerlei" Gewürze. :-)*
> *>*
> *Das, was in der Packung "Steak Gewürzzubereitung, Gebr. Wichartz GmbH &*
> *Co. KG, Wuppertal, 1 kg" drin ist. :-)*

1kg ... dann mal etwas "gesünderes":

```
package de.mpaap.grillen;

import de.mpaap.interfaceSpielchen.Grillbar;

public class Thueringer_Rostbraetel implements Grillbar {
   public String getGrillanleitung() {
      return "1 kg Schweinekamm\n1\2l dunkles Bier\n"
            + "5 EL Öl\n750 g Zwiebeln\n1 EL Senf\nSalz,"
            + "Pfeffer, Petersilie\n\n- Kammscheiben (ca. "
            + "150g= klopfen, mit Pfeffer würzen, mit Öl "
            + "bestreichen\n- Bier, Senf, etwas Öl. Salz und"
            + " Pfeffer zu einer Marinade rühren\n- Fleisch"
            + " und Petersilienstrauß chen 12 bis 14 h in der"
            + " Marinade ziehen lassen\n- abtropfen, grillen\n-"
            + " Zwiebeln in Öl gebraten dazu reichen";
   }

   public String getName() {
```

```
        return "Thüringer Rostbrätel";
    }
}
```

Aufruf mit: -DgrillbarType=de.mpaap.grillen.Thüringer_Rostbrätel

String heutigesAbendessen.getName() in Grillfete statt
heutigesAbendessen.getClass().getName() aufzurufen finde ich einfach
besser. ;-)

Guten Appetit

Sylvia.

From: Michael Paap

Sylvia Schuetze wrote:

> *Ein paar Rezept mehr hättest Du ja schon anbieten können.*

Hier noch ein Nachschlag für Interessierte. Zunächst ändere man die
Methode getGrillBar() der Klasse GrillbarFactory wie folgt:

```
    Grillbar getGrillBar() throws Exception {
        String grillbarType = System.getProperty("grillbarType");
        String grillbarURL = System.getProperty("grillbarURL");
        Class c;
        if (grillbarURL != null) {
            URL[] urls = {new URL(grillbarURL)};
            URLClassLoader ucl = new URLClassLoader(urls);
            c = ucl.loadClass(grillbarType);
        } else {
            c = Class.forName(grillbarType);
        }
        return (Grillbar) c.newInstance();
    }
```

Und dann starten mit den VM-Parametern

-DgrillbarType=de.mpaap.grillen.UeberraschungsGrillgut
-DgrillbarURL=http://www.mpaap.de/javademo/

Gruß,
Michael

From: Michael Paap

Sylvia Schuetze wrote:

> *1kg ... dann mal etwas "gesünderes":*

Ich verarbeite ja nicht das ganze Kilo, ich kaufe das Zeug nur im
Großhandel.

[Thüringer Rostbrätel]

Hmmm. Auch lecker. ;-)

> *String heutigesAbendessen.getName() in Grillfete statt*
> *heutigesAbendessen.getClass().getName() aufzurufen finde ich einfach*
> *besser. ;-)*

Ok... allerdings musst Du dann auch Grillbar um diese Methode erweitern,

sonst: "The method getName() is undefined for the type Grillbar".
Ansonsten gefällt mir die Idee, insbesondere in Verbindung mit meinem Überraschungsgrillgut. Ich werde das gleich mal auf das neue Interface anpassen.

Ach ja, und ein Tip noch: Auch wenn es möglich ist, Bezeichner mit Umlauten etc. zu verwenden, würde ich doch davon abraten. In der Praxis führt das gerne zu allerlei hübschen Problemchen, wenn man auf unterschiedlichen Systemen editiert/compiliert/ausführt.

Ich glaube, wir sollten langsam mal aufhören oder nach de.rec.mampf umziehen. ;-)

Gruß,
Michael Paap

From: Michael Paap

Michael Paap wrote:

> *Ach ja, und ein Tip noch: Auch wenn es möglich ist, Bezeichner mit*
> *Umlauten etc. zu verwenden, würde ich doch davon abraten. In der Praxis*
> *führt das gerne zu allerlei hübschen Problemchen, wenn man auf*
> *unterschiedlichen Systemen editiert/compiliert/ausführt.*

Noch ein kleiner Hinweis:

return "1 kg Schweinekamm\n1\2l dunkles Bier\n"

führt zu Problemen, weil der \ von 1\2 als Escapezeichen gewertet wird.
Um das zu verhindern, musst Du den \ selbst escapen, also:

return "1 kg Schweinekamm\n1\\2l dunkles Bier\n"

Gruß,
Michael Paap, hungrig.

From: Sylvia Schütze

Michael Paap schrieb:
> *>*
> *Hier noch ein Nachschlag für Interessierte. Zunächst ändere man die*
> *Methode getGrillBar() der Klasse GrillbarFactory wie folgt: [...]*

klappt wunderbar. Nochmals Danke für den Exkurs. Habe nun auch alle Umlaute in ae ,ue, oe geändert bzw. brav \u00XX in den Strings verwendet. Ist dies (das Verwenden von \u00XX statt Umlaut im String) noch nötig oder im Jahr 2004 übervorsichtig?

Gruß Sylvia.

From: Michael Paap

Sylvia Schuetze wrote:

> *Habe nun auch alle*
> *Umlaute in ae ,ue, oe geändert bzw. brav \u00XX in den Strings*
> *verwendet. Ist dies (das Verwenden von \u00XX statt Umlaut im String)*
> *noch nötig oder im Jahr 2004 übervorsichtig?*

Ach je... das ist so eine Sache. Natürlich ist es ***furchtbar*** lästig, in allen String-Literalen alles, was nicht 7-Bit-ASCII ist zu ersetzen. Das kann man zwar automatisieren, aber lesbarer wird der Code davon auch nicht. Ich denke, man sollte wissen, welche Probleme es in diesem

Zusammenhang gibt (Achtung, das geht wieder mal über den Kurs hinaus!) und dann entscheiden, was man tut. Und wenn es mal irgendwo kracht, sollte man an solche Sachen wie Encoding/Charset/Locale denken, bevor man sich einen Wolf sucht... auch und gerade dann, wenn auf den ersten Blick kein Zusammenhang besteht. Alles Erfahrungssache. ;-/

Packe mal das hier

```
    System.out.println("Köln");
```

in ein Programm und starte von der Kommandozeile. Prompt wird das 'ö' verhunzt. Ursache: Die Windows-Kommandozeile verwendet aus Gründen der Kompatibilität zu DOS ein anderes Encoding (meist Cp850) als das Betriebssystem selbst (bei W2K z.B. Cp1252). Da beide Encodings Obermengen von 7-Bit-ASCII sind, fällt das bei Zeichen mit einem Code <= 127 nicht auf, bei denen darüber hingegen schon. Und da hilft dann auch kein Ersetzen von "Köln" durch "K\u00F6ln". Denn Java als vollwertiges 32-Bit-System benutzt das Encoding, das vom Betriebssystem vorgegeben wird. Ergebnis: Konsolenausgaben unter Windows sind verhunzt. Es gibt ziemlich einfache Workarounds, die aber meist dafür sorgen, dass das Programm nicht mehr wirklich plattformunabhängig ist (http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=umlaute+konsole+group%3Ade.comp.lang.java&btnG=Search).

Die von mir bevorzugte pragmatische Lösung besteht darin, dass ich mich bei Kommandozeilenausgaben üblicherweise auf 7-Bit-ASCII beschränke, also "Koeln", und mich ansonsten darüber freue, dass unter Java das Arbeiten mit graphischen Benutzeroberflächen recht einfach ist, so dass man bei "normalen" Programmen dem Benutzer lieber ein

```
   JOptionPane.showMessageDialog(null, "Köln");
```

entgegenwirft.

Aber es gibt auch wesentlich subtilere Probleme. Mal zwei (stark vereinfachte) Beispiele aus der Praxis... Sachen, die mich mangels Erfahrung richtig Zeit gekostet haben:

```
************************** WARNUNG **************************
*                                                           *
* Es folgen Stories aus meinem Leben, die von keiner Relevanz  *
* für den Kurs 1618 bzw. 1616 sind. Lesen auf eigene Gefahr;   *
* für sinnlos vertane Zeit wird keine Haftung übernommen.      *
*                                                           *
*************************************************************
```

1. Editieren und Compilieren auf verschiedenen Systemen.

Man nehme folgenden Code:

```
        char x = 'ä';
        char y = 'ö';
        System.out.println(x == y ? "gleich" : "ungleich");
```

Diesen Code habe ich unter Windows erstellt und getestet. Erwartungsgemäß gibt er "ungleich" aus. Ich habe dann den ganzen Kram per FTP (Source und Class-Dateien) auf ein Solaris-System (unixoid) geschleppt und dort laufen lassen: Keine Probleme.
Ein wenig später ergab es sich, dass ich das Zeug auf dem Solaris-Rechner neu compilieren musste. Und plötzlich verhielt sich mein Programm ausgesprochen seltsam. Den Fehler zu finden, hat mich eine satte Stunde gekostet (natürlich war das Programm etwas größer und der Fehler äußerte sich in einer Weise, die mich damals an alles denken ließ, nur nicht an Umlaute). Des Rätsels Lösung: Der Editor, mit dem ich das Programm erstellt hatte, arbeitete natürlich mit dem Windows-Encoding CP1252. Der Compiler unter Windows ging von eben diesem Encoding aus und erzeugte netten Bytecode, der auch unter Solaris ordnungsgemäß lief. Aber als ich dann unter Solaris (Encoding

ISO-8859-1) neu compilierte, interpretierte der Compiler die Bytes, die CP1252 für die Umlaute benutzt hatte jeweils als Fragezeichen, denn diese Bytes verwendet ISO-8859-1 nicht. De facto stand also in meinem Bytecode das da:

```
char x = '?';
char y = '?';
System.out.println(x == y ? "gleich" : "ungleich");
```

Und das sorgt dann logischerweise dafür, dass x und y gleich sind. Verschiedene Lösungen:

a) Immer nur Bytecode übertragen, nie Source.
b) Immer Unicode verwenden, also

```
char x = '\u00e4';
char y = '\u00f6';
System.out.println(x == y ? "gleich" : "ungleich");
```

c) Beim Compilieren per Compileroption -encoding das Encoding angeben, mit dem die Sourcedatei erstellt wurde.

2. Es gibt da ein Programm, welches auf Solaris brav seinen Dienst tat, nun aber unter Windows laufen sollte. Das Programm erstellt u.a. serverseitig dynamisch Webseiten, die dann auf Clients in Browsern dargestellt werden. Diese Webseiten enthalten auch eingebettete Dateien, z.B. Bilder im GIF-Format. Interessanterweise wurden in den unter Solaris erzeugten Webseiten die Bilder angezeigt, lief das Programm hingegen unter Windows, teilte der Browser schlicht mit, die Bilder seien keine gültigen GIF-Dateien. Nun denkt man ja im Zusammenhang mit Bildern nicht unbedingt an Encoding-Probleme. Also durchforstet man (qualvoll, da "gewachsener Code") das Programm, um herauszufinden, wie diese Bilder eigentlich von der Festplatte - wo sie in Ordnung sind - in die Response zum aufrufenden Browser gelangen, bzw. ab wo sie nicht mehr in Ordnung sind.

Wie sich herausstellte, war vor langer Zeit jemand auf die Idee gekommen, das Laden der Bilder von der Festplatte sei eine Performancebremse und die Bilder müssten daher gepuffert werden... in Anbetracht der Tatsache, dass sowohl das BS als auch der Servlet-Container selbst schon puffern, zwar ein eher verwegener Gedanke, er wurde aber umgesetzt. Also lud das Programm die Bilder in byte[]. Dann hatte man Objekte, und die konnte man in einer HashMap ablegen, z.B. mit dem Dateinamen als Schlüssel. Dass auf diese Weise früher oder später jedes je aufgerufene Bild im Hauptspeicher liegt, woraufhin vermutlich der Rechner anfängt zu swappen... naja, eine verwegene Idee halt. Bis dahin zwar reichlich schräg, aber noch keine Erklärung für die vermurksten Bilder.

Des Rätsels Lösung: Dem Programmierer war es aus irgendwelchen Gründen wichtig, nur mit Strings als Parameter zu arbeiten. Deshalb wurden die Bilder ***nicht*** als byte[] in die HashMap geworfen, sondern zuvor in Strings umgewandelt. Und später dann wieder zurück. Ein Encoding wurde natürlich für diese Spielchen nicht explizit angegeben... wozu auch: Wenn man kein Encoding angibt, wird für beide Umwandlungen das Standard-Encoding des BS benutzt, es kann also nichts passieren. Dachte man.

Und unter Solaris stimmte das auch, denn dort wird ISO-8859-1 verwendet, welches eine bijektive Abbildung zwischen allen 256 Byte-Werten und Zeichen im String vornimmt, egal, ob diese nun dargestellt werden können oder nicht. Unter Windows stimmte es nicht mehr. Denn CP1252 nimmt eine Überprüfung vor, ob die Bytes auch im Zeichensatz vorkommen. Und wenn nicht, dann werden diese nicht erst bei der ***Darstellung*** zu Fragezeichen (normal), sondern schon bei der ***Umwandlung*** von Byte nach String. Beim Zurückwandeln hat man dann für alle nicht in CP1252 vorkommenden Bytes die Bytes für das Fragezeichen und damit ungültige GIFs.

Lösung:

Bei Umwandlungen von String nach byte[] oder umgekehrt ***immer*** ein Encoding vorgeben, vorzugsweise eines, welches auch alle Byte-Werte umkehrbar codiert.

Das hätte ein Neuschreiben des Codes erfordert... der enthielt derlei Konvertierungen an Dutzenden Stellen, mit unabsehbaren Nebenwirkungen. Das wollte niemand. Also ein Würgaround: Man übergibt beim Ausführen der VM den Parameter -Dfile.encoding=ISO-8859-15 und lässt den Code, wie er ist. ***Schauder***. Läuft aber. So weit ich weiß, immer noch. ;-)

Und derlei Späßchen gibt es ohne Ende...

Gruß,
Michael Paap

From: Sylvia Schütze

Michael Paap schrieb:

> *System.out.println("Köln");*
> *JOptionPane.showMessageDialog(null, "Köln");*

Ja, klar. Konsolenausgaben sind sicher ziemlich selten.

> *Aber es gibt auch wesentlich subtilere Probleme.*
> *1. Editieren und Compilieren auf verschiedenen Systemen.*
> *Man nehme folgenden Code:*

```
char x = 'ä';
char y = 'ö';
System.out.println(x == y ? "gleich" : "ungleich");
```

> *Diesen Code habe ich unter Windows erstellt und getestet.[...]*

> *Ein wenig später ergab es sich, dass ich das Zeug auf dem Solaris-Rechner neu compilieren musste. Und plötzlich verhielt sich mein Programm ausgesprochen seltsam. Den Fehler zu finden, hat mich eine satte Stunde gekostet*

Eine Stunde ist doch nicht schlecht.

> *a) Immer nur Bytecode übertragen, nie Source.*

Das wär mir am symphatischsten.

> *b) Immer Unicode verwenden, also*

```
char x = '\u00e4';
char y = '\u00f6';
System.out.println(x == y ? "gleich" : "ungleich");
```

schlecht zu lesen.

> *c) Beim Compilieren per Compileroption -encoding das Encoding angeben, mit dem die Sourcedatei erstellt wurde.*

Gut zu wissen, dass es das gibt. Ja, ich wollte mir ja mal die Parameter ansehen.

> *2. Es gibt da ein Programm, welches auf Solaris brav seinen Dienst [...]*
> *früher oder später jedes je aufgerufene Bild im Hauptspeicher liegt, woraufhin vermutlich der Rechner anfängt zu swappen...*

köstlich,

> *nur mit Strings als Parameter zu arbeiten. Deshalb wurden die*
> *Bilder ***nicht*** als byte[] in die HashMap geworfen, sondern zuvor in*
> *Strings umgewandelt. Und später dann wieder zurück.[...]*

> *Und unter Solaris stimmte das auch, [...]*
> *Unter Windows stimmte es nicht mehr.*

Die Plattformunabhängigkeit von Java erfordert doch mehr Überlegungen
bei der Programmierung als ich mir vorstellte. An so etwas :

> *Bei Umwandlungen von String nach byte[] oder umgekehrt ***immer*** ein*
> *Encoding vorgeben, vorzugsweise eines, welches auch alle Byte-Werte*
> *umkehrbar codiert.*

muss man erst mal denken.

> *Und derlei Späßchen gibt es ohne Ende...*

Jaaa? Soll ich Knabberzeug holen? :-)

Gruß Sylvia.

From: Michael Paap

Sylvia Schuetze wrote:

> *Jaaa? Soll ich Knabberzeug holen? :-)*

Nee, lass mal. Vielleicht sollten wir die Einrichtung von
feu.informatik.kurs.1618.ss2004.talk initiieren. ;-)

Gruß,
Michael Paap