"Hello World" in OOP

Zum Beitrag von H. von Lavergne in LOG IN 18 (1998) Heft 5

von
Siegfried Spolwig

Wer die Vorzüge einer OOP Sprache mit 'Hello World' offenbaren will, outet sich als unverbesserlicher Programmierer, dessen Wurzeln mindestens bis C unter Unix in den 70er Jahren zurück reichen. Das ist keine Schande, aber auch kein Vorteil, denn es zeigt mit einer unvermittelten Deutlichkeit die Probleme, die nahezu alle praktizierenden Informatiklehrer mit diesem Programmierstil haben. Sie erkennen ihn nicht und können daher auch nicht die Vorteile für den Unterricht darlegen. Bisweilen wird für diesen Stil den Begriff Paradigma verwendet, der nach z. B. *Kuhn* etwas anderes bedeutet. Wenn man aber sieht, daß es offenbar ebenso schwierig ist, ein traditionelles Programmierweltbild zu überwinden wie es damals der Abschied von der geozentrischen Sicht war, ist der Begriff vielleicht doch nicht - wenigstens in dieser Hinsicht - ganz falsch. [1)]

Betrachtet man die Quelltexte von HelloWorld, so muß man sich doch fragen, was ist das Objekt, auf welche seiner Eigenschaften wird da zugegriffen und welche seiner Methoden tut das? Davon ist weit und breit nichts zu sehen. Das Modul OUT, welches zur Erklärung dienen soll, ist ein Sammelsurium von Prozeduren (und nicht von Methoden, wie der Autor richtig bemerkt) und unterschiedlichen Typen. Was das mit OOP zu tun haben soll, wird ewig unerfindlich bleiben. Avergne wirbt für eine gute und richtige Sache, aber leider mit seltsamen Beispielen. Der Artikel gibt einen kenntnisreichen Einblick in einige Aspekte, aber wer soll damit überzeugt werden? Programmiersprachenfreaks wissen das eh, und der unkundige durchschnittlich interessierte Lehrer bekommt ein völlig falsches Bild von den an sich hervorragenden unterrichtlichen Möglichkeiten der OOP vermittelt.

Wer 20 Jahre lang aus seiner Sicht erfolgreich den Schülern writeln('Hello World') beigebracht hat - warum sollte der nun dafür Out.String('Hello World') und sogar noch eine 2. Zeile Out.Ln schreiben?

Insofern ist die Einladung der Redaktion zur Diskussion unbedingt notwendig, allerdings nicht mehr über die Frage "ob OOP im Unterricht", sondern nur noch "wie". Der Siegeszug von OOP ist in den letzten 10 Jahren längst von Forschung, Lehre und der Softwareindustrie entschieden, nur die Schule und die "müden" Informatiklehrer haben es nicht wahrgenommen. Dazu gehört auch dann die Frage, mit welchem Werkzeug? Jedoch nicht unter modischen Gesichtspunkten, sondern welches Werkzeug unterstützt die Lernziele am besten (vgl. Penon/Spolwig in: LOG IN 18 (1998) Heft 5). Zu den Vorzügen von OOP gehören zum einen die lernpsychologischen Vorteile im Unterricht und zum anderen die softwaretechnischen Möglichkeiten. Auf beides muß eingegangen werden.

Im Zentrum : der Bildungswert

Genauer:
Was ist der Bildungswert von 927 Zeilen Hauptprogramm zwischen *Begin* und *End*. ?

Das ist kein Scherz, sondern seit 1970 nahezu unverändert immer noch die Programmierwirklichkeit in vielen deutschen Gymnasien [2)]. Das menschliche Denken und Handeln läuft nicht nach formalisierten Algorithmen ab, sondern wir begegnen einer Umwelt von belebten und unbelebten Objekten, die durch Eigenschaften gekennzeichnet sind und die häufig auch ein eigenes Verhalten und einen bestimmten Zustand zeigen und mit denen wir in Kontakt treten. OOP hat schlicht den Anspruch und die Möglichkeiten, die Welt genau so abzubilden und nicht in einen 927zeiligen Algorithmus zu zwängen.

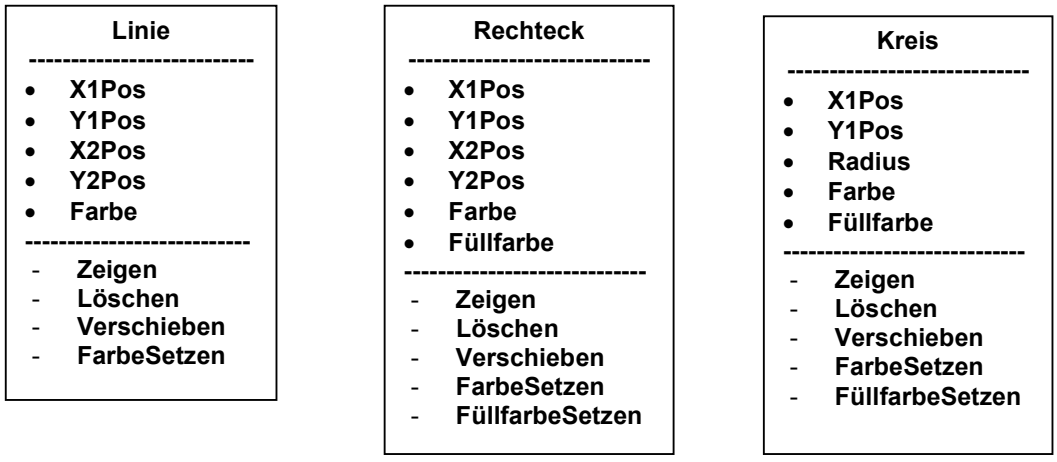
Schwill, Crutzen/Hein und andere haben dargelegt, daß objektorientierte Programmierung gegenüber anderen Stilen lerntheoretische Vorteile besitzt, daß sich damit auf informatischer Ebene Dinge beschreiben lassen, wie sie uns auch in der täglichen Wahrnehmung und im Umgang mit ihnen erscheinen. Dieser Vorteil wird aber erst und nur dann dem Schüler offenbar, wenn erstens *die Programmiersprache diese Beschreibung genauso leicht durch geeignete Konstrukte zuläßt* und zweitens, wenn im Unterricht diese Möglichkeit *auch richtig benutzt wird*. Und daran hapert es häufig, auch bei den wohlmeinenden Versuchen, wenn eine problematische Sprache und ein völlig ungeeigneter Programmierstil eingesetzt werden, die diese Vorteile zudecken und von daher wirkungslos machen.

Eine Einführungsstunde in Programmierung (grafische Objekte, die alsbald in Schülerübungen *benutzt* werden sollen) kann etwa folgendermaßen verlaufen:

An einem mitgebrachten Tennisball wird der Begriff Objekt eingeführt:

Ball	➔ Objekt
* hat eine Farbe, Form, Gewicht usw.	➔ Attribute
- kann rollen, springen, liegen usw.	➔ Methoden

Danach zeichnet der Lehrer mit farbigem Stiften eine Gerade, ein Rechteck, einen Kreis auf die Tafel mit dem Hinweis, daß diese "Objekte" auf dem Bildschirm dargestellt werden sollen. Aufgabe ist, die Attribute und Methoden für diese Objekte zu finden, um sie in einem Programm zu benutzen.
Alle Schüler, auch die schwächsten, kommen nahezu mühelos zu der Lösung:



Nur das Problem der Positionierung ist gemeinsam zu klären.

Es ist auch keine große Überraschung, daß die Objekte sowohl in den Attributen als auch in den Methoden bis auf wenige Abweichungen übereinstimmen. Daher ist es nicht weit, nach einem übergeordneten Begriff (FIGUR) zu suchen, der die gemeinsamen Attribute und Methoden in sich vereinigt und den als "Stammvater" der geometrischen Figuren zu betrachten. Damit sind unter der Hand wesentliche Merkmale von OOP (Klasse, Oberklasse, Vererbung) eingeführt worden, ohne sie an dieser Stelle bereits theoretisch zu thematisieren. Wichtig ist aber, daß die Vorgehensweise als völlig natürliche heraus gearbeitet und sofort auch reflektiert und verankert wird.

Die Schüler haben damit Objekte (genauer Klassen) beschrieben und können folgerichtig erwarten, daß diese Beschreibung auch so in der Programmiersprache umgesetzt wird. Genau dieses kann OOP realisieren und darin besteht der große Vorteil für den Unterricht und die Lernerfolge bei den Schülern im Gegensatz zu anderen prozeduralen, logischen und funktionalen Programmiersprachen, die eine künstliche und neu zu erlernende Denkweise erzwingen.

Aber auch genau an dieser Stelle beginnt das Problem bei DELPHI, JAVA & Co. Um ein rotes Rechteck auf den Bildschirm zu bringen, wären die adäquaten OOP-Methodenaufrufe (Aufträge) [3] :

```
Rechteck.FarbeSetzen(Rot);
Rechteck.Zeigen.
```

Schöner noch Rechteck.ZeigeDich, um auch den Auftrag (message) zu verdeutlichen.

Bei DELPHI z. B. wird ein Rechteck aber als Methode (!) von TCanvas aufgerufen; in voller Schönheit und Länge:

```
Form1.Image1.Canvas.Pen.Color := clRed;
Form1.Image1.Canvas.Rectangle(10,10,100,100);
```

Wenn ich das den Schülern als Lösung anbieten muß, ist mein OOP-Ansatz erledigt!

Es wäre völlig aussichtslos, den Schülern an dieser Stelle erklären zu wollen, wie die beiden Zeilen zustande kommen. Was wie ein Objekt aussieht (Rectangle) ist eine Methode, für den Zugriff auf ein Attribut (Color) gibt es bei DELPHI keine Methode, sondern es geht nur mit einer Zuweisung (die Properties sind ebenso ein Krampf). Color ist natürlich auch kein Attribut von Rectangle, sondern von Pen. Hier wird es unmittelbar klar, daß die Sprache an dieser Stelle in keiner Weise geeignet ist, den einfachen Sachverhalt 1:1 abzubilden [4] .

Wenn man also DELPHI u.a. für OOP einsetzen will, wird man als Lehrer einiges für einen erfolgreichen Unterricht für die Schüler (oder ev. mit fortgeschrittenen Schülern) umbrechen müssen. Dieses Problem ist meines Erachtens bisher nicht gesehen oder als nebensächlich erachtet worden. Es ist aber das Schlüsselproblem für den unterrichtlichen Erfolg von OOP. Andersherum ausgedrückt, wenn es nicht in einer Programmiersprache gelingt, die wenigen aussagekräftigen Begriffe Klasse, Objekt, Attribut, Methode, Botschaft genauso einfach zu notieren, dann gehört OOP nicht in den Unterricht, zumindest aber nicht eine solche Sprache. Konzepte, die sich schimärenhaft in kryptischen Implementierungen verbergen, sollten dann besser dem Fachstudium vorbehalten bleiben.

Der weitere Unterricht verläuft dann so, daß der Lehrer bis zur nächsten Stunde einen 'Programmierauftrag' gemäß dieser vereinfachten Spezifikation erhält und zur Folgestunde ein Grafikpaket abliefert, mit dem die Schüler dann arbeiten können [5].

```
unit uGrafik;
(* ********************************************************************* *)
(*                                                                       *)
(* K L A S S E : TZeichenblatt u.a.                                      *)
(* --------------------------------------------------------------------- *)
(* Version     : 1.02                                                    *)
(* Autor       : S. Spolwig, OSZ-Handel I, 10997 Berlin                  *)
(*                                                                       *)
(* Beschreibung: stellt ein Objekt ZEICHENBLATT vom Typ TZeichenblatt   *)
(*               zur Verfügung, das in einer TImage-Komponente liegt.   *)
(*               Es wird mit Zeichenblatt.Init(..) mit der Image-Kompo- *)
(*               nente verknüpft.                                        *)
(*               Die anderen Klassen/Objekte beziehen sich auf ZEICHEN- *)
(*               BLATT.                                                  *)
(*               X1 und Y1 ist der Bezugspunkt jeweils links (oben),    *)
(*               beim Kreis der Mittelpunkt.                             *)
(*               Der Ursprung ist [0,0]                                  *)
(*               Die Farben sind vom Typ TColor in DELPHI.               *)
(* Compiler    : DELPHI 3.0                                              *)
(* Aenderung   : V. 0.9   13-NOV-98                                      *)
(*                  1.0   14-NOV-98   polymorphe Methoden                *)
(*                  1.01  04-DEC-98   Bug in Textpos; Farben gesetzt in *)
(*                                    Zeigen, sonst falsche Werte       *)
(*                  1.02  14-JAN-99   GetBreite, GetHoehe für Zeichblatt*)
(* ********************************************************************* *)
interface
uses graphics, extctrls;

type
TFigur  = class (TObject)
            protected
             x1, y1, x2, y2 : longint;
             Farbe,
             Fuellfarbe     : Tcolor;

            public
             constructor Init(ax1, ay1, ax2, ay2 : longint);
             function    GetXpos : longint;          virtual;
             function    GetYpos : longint;          virtual;
             procedure   SetFarbe  (F : TColor);     virtual;
             function    GetFarbe: TColor;           virtual;
             procedure   SetFuellFarbe (F : Tcolor); virtual;
             function    GetFuellfarbe : TColor;     virtual;
             procedure   PosVersetzenUm (dx, dy : longint); virtual;
             procedure   Zeigen;                     virtual;
             procedure   Loeschen;                   virtual;
             procedure   Entfernen;                  virtual;
          end;


TZeichenblatt = class (TFigur)
                  constructor Init (Im : TImage);
                  function GetBreite : longint; virtual;
                  function GetHoehe  : longint; virtual;
                  procedure Zeigen;             override;
                end;
```

```
TLinie    = class (TFigur)
              procedure Zeigen;   override;
            end;


TRechteck = class (TFigur)
              procedure Zeigen;   override;
            end;

  .....

(* ------------------------------------------------------------------- *)
```

OOP zeigt im Unterricht seine Stärken dann, wenn durch Abstraktion in der OOA geeignete Klassen, und zwar im strengen Sinne der Lehre, heraus gearbeitet werden und sie dann systemkonform implementiert und benutzt werden. Dazu ein weiteres Beispiel:

Für eine Karteiverwaltung findet man in der strukturierten Programmierung häufig solche Zeilen:

```
        gotoxy(10,10); write('Vorname : ') ; readln(adressenfeld[index+1].datensatz.vorname);
        gotoxy(10,11); write('Name : ') ;     readln(adressenfeld[index+1].datensatz.name);
....
```

In OOP (Turbo-Pascal) könnte das so geschrieben werden:

```
        Karteikarte.EintraegeEinlesen;
        Karteikasten.HinzuFuegen(KarteiKarte);
```

Es gibt zwei Objekte: Karteikarte und Karteikasten mit je einer passenden Methode. Die vollständige Aktion 'Erfassen' würde lauten:

```
        Statusmeldung.Zeigen (' - E i n g a b e - ');
        Karteikarte.EintraegeEinlesen;
        Karteikasten.Hinzufuegen(KarteiKarte);
        Sysmeldung.Einblenden (' Weiter mit RETURN-Taste ');
        KarteiKarte.Loeschen;
```

Ein klares Objektmodell und eine Notation, die auch Tante Frieda verstände. Bereits in DELPHI und JAVA u.ä. Programmiersprachen, die Formulare und GUI-Builder benutzen, läßt sich dieser einfache Zusammenhang nicht mehr so klar wie mit Turbo-Pascal (Objekt Pascal) realisieren. Dort sind andere Techniken erforderlich (vgl. Penon/Spolwig, a.a.O.).

Was ist noch geheim an *Form1.Image1.Canvas.Pen.Color := clRed* ?

Der zweite Aspekt der mißbräuchlichen Benutzung von OOP läßt sich sehr viel kürzer darstellen. Der Grund für die Entwicklung moderner Programmiersprachen und OOP ist nicht zuletzt in den Anforderungen an Qualität und Sicherheit von Softwaresystemen zu sehen. Der Schlüssel dazu liegt in den Prinzipien Kapselung und Geheimnisprinzip. Kapselung ist ein konstitutives Merkmal von OOP: Methoden kapseln Daten. - Punktum. Ohne wenn und aber.

Wenn man mit den Schülern größere Softwareprojekte entwickeln will, wie es z. B. der Berliner Rahmenplan für das Projekthalbjahr vorsieht, dann sind diese Prinzipien im Unterricht auch darzustellen und umzusetzen, insbesondere dann, wenn arbeitsteilig vorgegangen wird.
Ein direkter Zugriff mit einer Zuweisung auf ein Objektattribut wie oben, der auch noch die gesamte Datenstruktur offenlegt, muß schlicht tabu sein. Selbst bei den abgestuften Möglichkeiten der Zugriffsberechtigungen (private, protected, public ..) sollte im Unterricht nur Rechteck.FarbeSetzen(Rot) erlaubt sein. Wie kann man sonst den Schülern glaubhaft diese immens wichtigen Prinzipien vermitteln, wenn sie in ihren Programmen laufend dagegen verstoßen dürfen?

Zusammenfassend läßt sich sagen:

- OOP ist eine neue und erfolgreiche Denkweise für den Informatikunterricht, nicht eine neue Runde "Wir lernen Oberon" oder "JAVA - jetzt!"

- OOP von der ersten Stunde an nicht nur in der Programmierung, sondern als didaktische Basis und Erklärungsmodell für informatische Erscheinungen.

- Konsequente Anwendung der OOP-Konstrukte, keine hybrid programmierten Nachlässigkeiten oder Erleichterungen im Stil von 'MyUtils' oder 'DLH' (Dirty little Helpers). Die Schule erfordert kein schlechtes professionelles 'quick and dirty', sondern hat die Zeit, das Exemplarische in den Mittelpunkt des Unterrichts zu stellen. Das mag manchmal zu etwas langatmigen Methodenzugriffen führen, verschafft aber Klarheit im Denken.

- Im Mittelpunkt des Unterrichts stehen OOA und OOD, d. h. Abstraktion, Klassenbildung, Modellierung, komplexe Systeme (und auch noch ein bißchen Algorithmik ;-).

- Dazu brauchen wir gute Beispiele, die zeigen wie ein anspruchsvoller, aber auch zufriedenstellender Unterricht für Schüler und Lehrer gemacht werden kann. [6)]

---

[1] Ein Kollege berichtet, daß, nachdem in einer Fortbildungsveranstaltung mit gestandenen und engagierten Informatiklehrern in der OOA alle notwendigen Klassen spezifiziert waren, die das Anwendungsproblem insgesamt lösten, die Teilnehmer darauf bestanden, daß jetzt aber noch Prozeduren für die Ablaufsteuerung gefunden werden müßten! Das zeigt, wie schwer die eingefahrene ablauforientierte Programmierung überwunden werden kann.

[2] Das schaffen natürlich nicht alle und einige hören schon bei, sagen wir mal, 132 Zeilen auf, weil das dann noch so schön übersichtlich bleibt. Solche Beispiele kann man u. a. auch vom LOG IN Server herunterladen. Nebenbei gesagt, eine Schande für ein Fach, das gerne von sich sagt, daß es die höchste Innovationsrate habe.

[3] Schöner wäre noch 'Rechteck.ZeigeDich', um den Auftrag (message) an das Objekt zu verdeutlichen.

[4] C, JAVA usw. sind von verstockten Programmierern für verstockte Programmierer entwickelt worden, die wahrscheinlich auch nachts in C träumen können, aber nicht für die Schule und zu Ausbildungszwecken. Deshalb ist PASCAL auch immer noch für den Unterricht unschlagbar. Bei OBERON mußte Wirth wohl diesen Aspekt aufgeben, weil er gleichzeitig damit ein Betriebssystem entwarf. Deshalb ist OBERON in dieser Hinsicht auch keinen Deut besser als geeignet als C++, DELPHI oder JAVA.

[5] Natürlich gab es unter den Kollegen eine Diskussion darüber, ob man so etwas machen könne, denn später würden die Schüler als Programmierer eine derartige Grafikbibliothek in keiner Sprache wiederfinden. Das zeigt deutlich den Spagat zwischen didaktischer und professioneller Entwicklersicht.

[6] Einige Versuche hierzu findet man unter http://oszhdl/be.schule.de/gymnasium/informatik/index.htm