Schöne visuelle Welt?

- Objektorientierte Programmierung mit DELPHI und JAVA –

von
Johann Penon und Siegfried Spolwig

Einleitung

LOG IN veröffentlichte in letzter Zeit eine Reihe von Beiträgen, die mit der Entwicklung der Programmiersprache JAVA ein „Stimulans für den Informatikunterricht“ (Baumann) versprechen oder mit „JAVA jetzt – adieu PASCAL“ (Boettcher) die Neuzeit einzuleiten scheinen. Die mitgelieferten Beispiele zeigen jedoch eher die Probleme als die Lösungen auf. In einem Leserbrief wies G. Dick darauf hin, daß Schülerinnen und Schüler in erster Linie lernen sollten, in „Modellen zu denken und eben nicht in Programmiersprachen!“ Eine Aussage, der wir voll zustimmen.

In Schulbuchverlagen sind die ersten Titel dazu erschienen, deren Autoren offensichtlich auch eine Renaissance des schon oft totgesagten Programmierunterrichts sehen. Leider bieten diese Bücher überwiegend Miniprogramme an im Stile der beliebten Einführungswerke wie ‚DELPHI in 11 Tagen‘ und ‚Wie kann ich ... ?‘. Das Gefährliche an diesen Büchern ist, daß sie unter der Hand ein leicht bekömmliches Curriculum liefern, aber auf dem Niveau von Programmierkursen stecken bleiben. Vielfach lassen sie die alten Gespenster des mathematikorientierten Informatikunterrichts der frühen Jahre wieder auferstehen, wohl weil das jetzt alles viel besser aussieht.

Anlaß genug, um sich mit den Möglichkeiten moderner objektorientierter Programmierumgebungen zu beschäftigen. Seit vielen Jahren behandeln wir im Unterricht objektbasierte modulare Programme und haben damit ein Unterrichtskonzept entwickelt, das durchgängig von Anfangsunterricht bis zum Abitur reicht (vgl. SPOLWIG). Seit 1997 haben wir OOP mit Objekt-PASCAL (Turbo PASCAL 7) realisiert. Im Mittelpunkt des Unterrichts stand immer die Entwicklung des Datenmodells der Anwendung (Fachkonzept), nicht das systematische Erlernen einer bestimmten Programmiersprache. Die Ergebnisse waren durchaus zufriedenstellend, jedoch litten die Programme oder besser gesagt einige Programmiererinnen und Programmierer unter dem etwas dürftigen Aussehen der ASCII-Oberflächen, die auf den Windows-Bildschirmen so fatal an die alten DOS-Zeiten erinnern.

Wenngleich Objektorientierung für uns mittlerweile selbstverständlich geworden war, blieb die Frage, mit welchem Werkzeug gearbeitet werden sollte. Typischerweise präferierten drei überzeugte Kollegen natürlich drei verschiedene Sprachen:

- der Fachbereichsleiter DELPHI, weil das *das* zeitgemäße Werkzeug sei,
- der Webexperte JAVA wegen der Client-Server-Unterstützung,
- der Fachseminarleiter Objekt-PASCAL, weil es nicht mit einem gewaltigen Overhead und unwichtigen Gimmicks (das tolle Aussehen) vom eigentlichen Kern des Unterrichts ablenkt.

Also beschloß die Fachkonferenz am praktischen Beispiel zu untersuchen, welche Sprache geeigneter sei.

Zwei unvereinte Welten

Das Unbehagen gegen DELPHI, JAVA und ‚visuelle Programmierung' rührte nicht zuletzt daher, daß die bisher veröffentlichten Programmbeispiele oft genug eine völlig willkürliche und kaum nachvollziehbare Melange von Zugriffen auf GUI-Komponenten und Datenobjekte des Fachkonzeptes innerhalb aller möglichen Programmblöcke zeigen. In der Beliebtheitsskala führend ist mit weitem Abstand der Taschenrechner, der flott aus dem GUI-Baukasten zusammengeschoben werden kann und recht professionell aussieht. Was dabei als richtungweisend angeboten wird, ist meistens ein Schaf im Wolfspelz.

Was bekommt man zu sehen, wenn man den Pelz anhebt? Einerseits die Komponenten aus der Klassenhierarchie des GUI-Builders: Buttons, Editfelder, Checkboxen usw., also Elemente, die aus dem Konzept der objektorientierten Grafikoberflächen entstanden sind, andererseits eine Programmierweise, die den Begriff Architektur wohl kaum verdient und häufig bestenfalls als strukturierte Programmierung zu betrachten ist.

Es werden zwei völlig unterschiedliche Ansätze miteinander vermengt, die sich eigentlich gegenseitig ausschließen: die funktionale Abstraktion und die Datenabstraktion. Einfacher ausgedrückt: Statt eine Meldung mit writeln (′Hello World′) auf dem Bildschirm auszugeben, wird eine Messagebox.ShowMessage (′Hello World′) an derselben Stelle im Programmcode des Datenmodells bemüht, und das ganze Programm wird untergebracht in einem einzigen Modul main oder so ähnlich Das ist dann alles und leider sowohl mit DELPHI als auch mit JAVA möglich. Die Tatsache allein, daß ein Modul mit ‚class' beginnt, ist beileibe noch keine Garantie für OOD und OOP.

Woher rühren diese Ergebnisse? Einer der Gründe ist sicherlich, daß das OOP Konzept bislang noch wenig bekannt ist und ein weiterer Grund wird sein, daß der Software Life Cycle bei der visuellen Programmierung sehr aktionsbetont ist, aber unzulässigerweise auf folgende Vorgehensweise verkürzt wird:

1. Oberfläche mit dem GUI-Builder gestalten
2. Ereignisprozeduren auf die Komponenten legen
3. Überlegen, was in den Prozeduren passieren soll.

Bei dieser Reihenfolge ergeben sich dann zwangsläufig solche Ergebnisse.

Die gezeigten Beispiele sind durchweg inakzeptabel, weil sie einen Rückfall in die Strukturierte Programmierung der 70er Jahre bedeuten. Wir fanden auch nicht ansatzweise ein Beispiel für ein objektorientiertes Datenmodell, das mit DELPHI oder JAVA realisiert wurde. Deshalb haben wir selbst den Versuch unternommen, objektorientierte Programme mit grafischer Oberfläche zu entwickeln.

Ziel war es herauszufinden, ob und wie objektorientierte Analyse (OOA) und objektorientiertes Design (OOD) mit diesen Programmiersprachen in Einklang zu bringen sind, ob sie das Erreichen der Unterrichtsziele fördern oder es durch hohen technischen Aufwand behindern. Daneben sollte untersucht werden, ob die Sprachen eine natürliche Lesbarkeit des Quellcodes erlauben, so wie es PASCAL als Lehrsprache in einem hohen Maße garantiert.

Unser Ausgangsbeispiel war ein simples Ratespiel aus dem Anfangsunterricht, das modularisiert in Objekt-PASCAL vorlag (s. Bild_1).

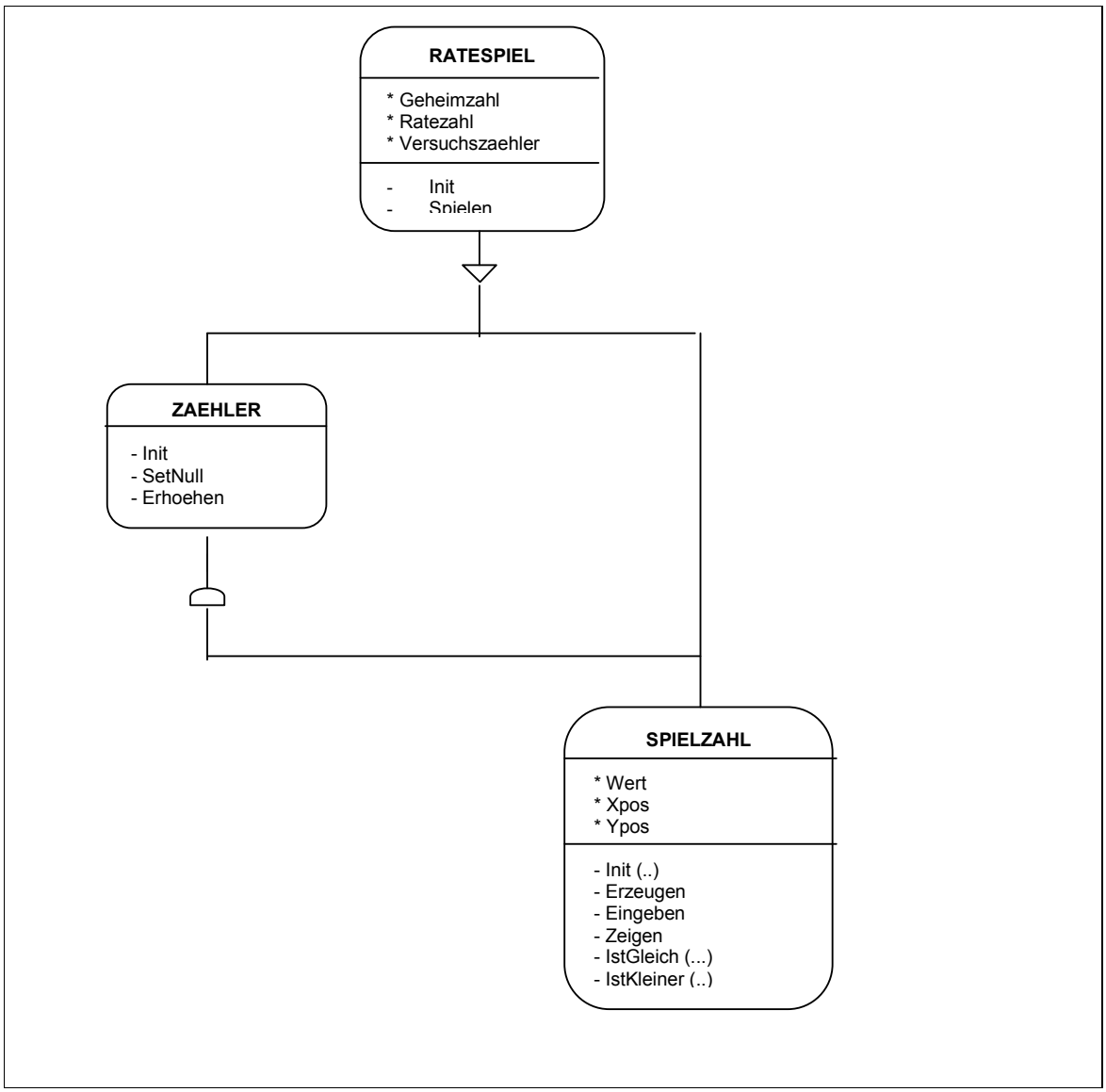


Bild 1: Ausgangsbeispiel. Klassen mit Turbo-PASCAL. Die Klassenhierarchie bleibt in DELPHI und JAVA gleich.

Hatten wir bisher nur *eine* Klassenhierarchie des Fachkonzeptes, die sich aus der OOA ergab, so haben wir beim Einsatz von GUI-Bibliotheken ein zweites Klassensystem mit grafischen Bildschirmkomponenten. Durch das Zusammenspiel der beiden Klassensysteme erhöht sich die Komplexität noch einmal erheblich.

Fernab aller gängigen DELPHI- und JAVA-Beispiele fanden wir mit dem wiederentdeckten Model-View-Controller Konzept (MVC) aus der SMALLTALK Welt (vgl. Balzert) einen erfolgversprechenden Hinweis. Absprachegemäß benutzte einer von uns DELPHI, der andere JAVA. Abgestimmt waren die Klassen mit Attributen und Methoden wie in Bild 1. Das Ergebnis wurde anschließend verglichen. Zu unserer großen Überraschung stimmten die Ergebnisse sowohl in der Konzeption der grafischen Oberflächen als auch über weite Strecken im Codeaufbau überein. Damit ist unseres Erachtens der Nachweis erbracht, daß es für die Frage,

wie OOA und OOD mit diesen Sprachen vereinbar sind, völlig gleichgültig ist, welche Sprache im Unterricht eingesetzt wird, wenn die Konzepte im Vordergrund stehen.

Das Model-View-Controller Konzept

Jede interaktive Anwendung besteht aus Eingaben, einem Datenmodell und der Darstellung der Daten auf dem Bildschirm. Eine Möglichkeit der Realisation besteht darin, diese drei Elemente jeweils nach Bedarf in einer Klasse zusammenzufassen (s. Bild 2 und das Ausgangsbeispiel in PASCAL).

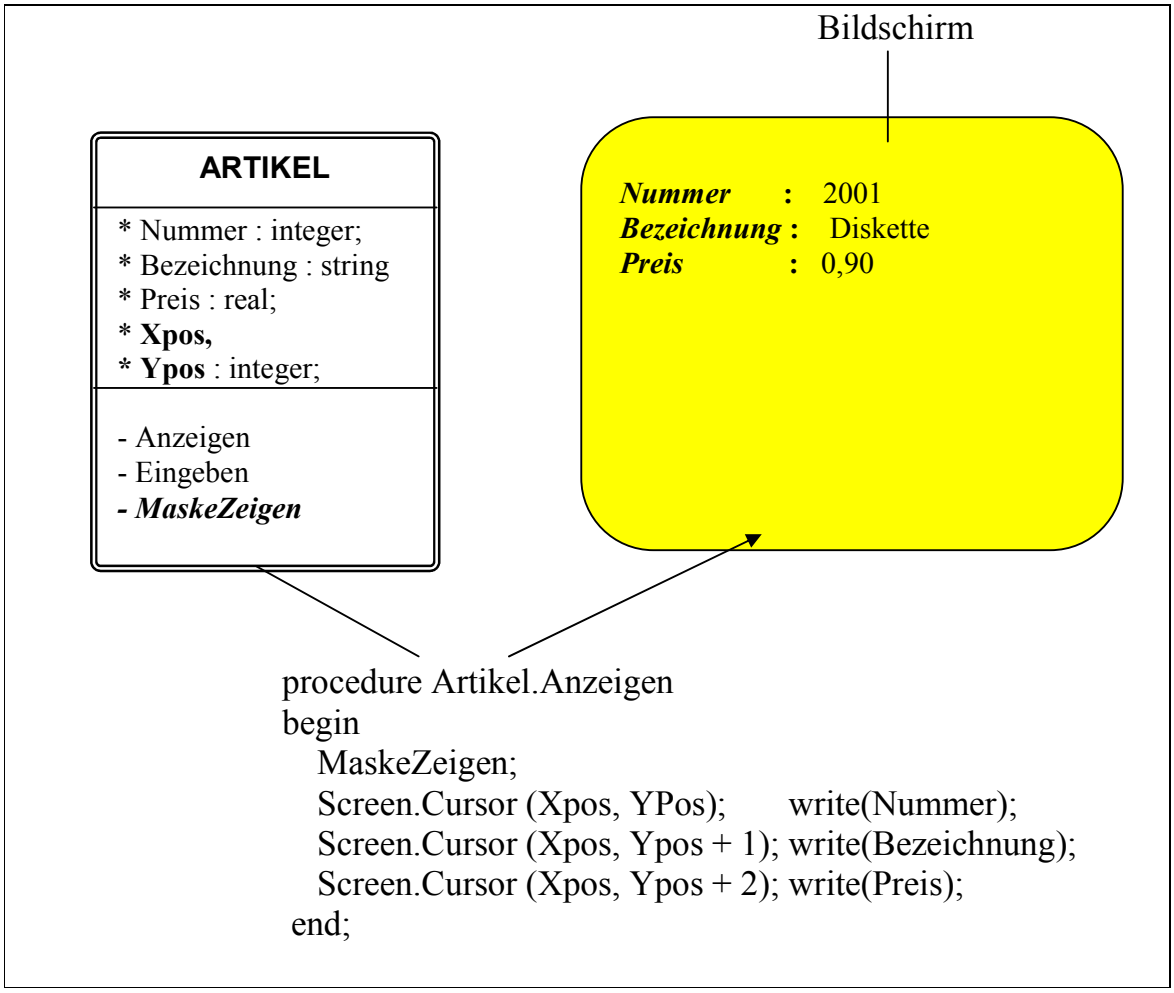


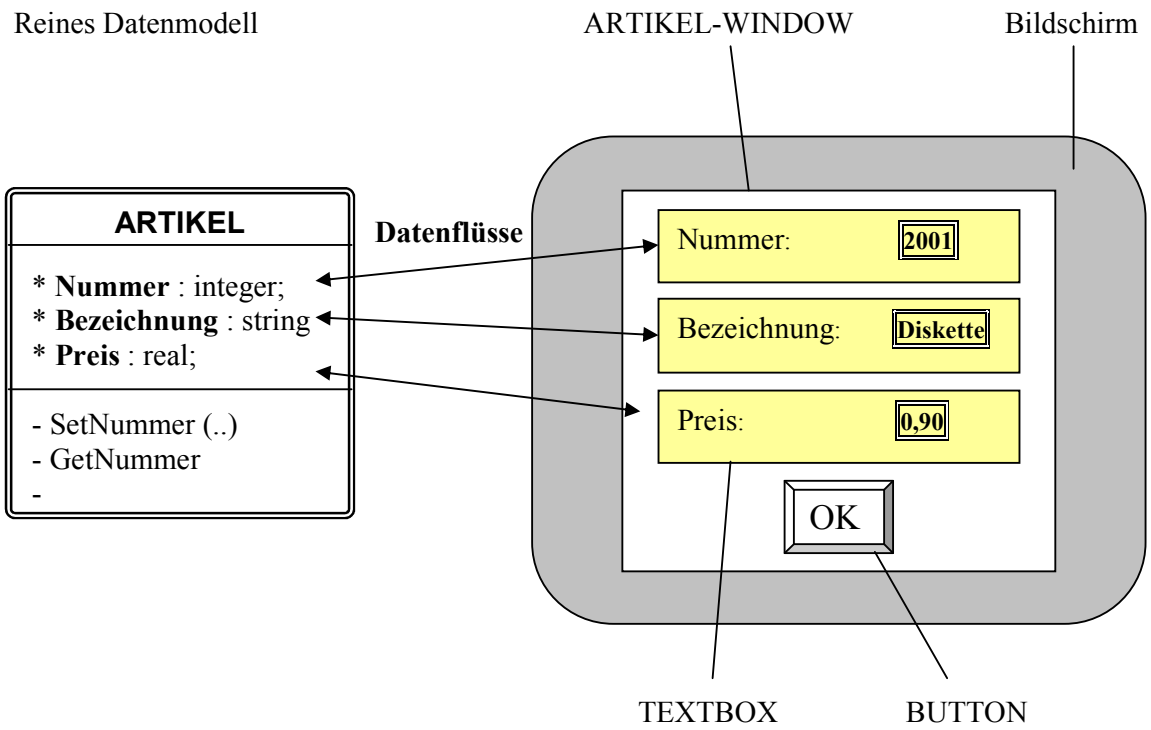
Bild 2: Software Design – integrative Lösung. Daten, Eingaben und Ausgaben sind in einer Klasse vereint.

Eine andere Möglichkeit (MVC) trennt die Ebenen in drei selbständige Einheiten. (s. Bild 3). Der Begriff *Model* wird für die Applikation ohne Benutzungsoberflächen (Fachkonzept), also für die interne Datenverarbeitung verwendet. Die einzelnen *Views* sind für die aktuelle Darstellung der Eingangs- und Ausgangsdaten in den jeweiligen Anzeigenobjekten (Editfelder, Menüs, Dialogboxen usw.) verantwortlich. Sie werden dabei von der Windowsverwaltung unterstützt.

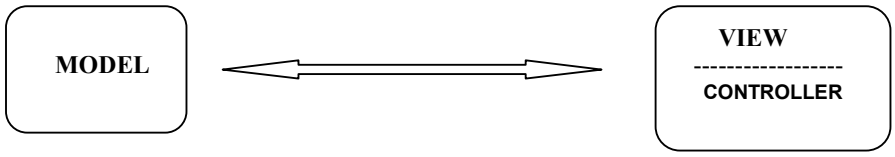
Der *Controller* überwacht alle Eingabegeräte. Eingabedaten werden an das zuständige Fenster weitergeleitet. Änderungen der Modelldaten werden also vom Controller eingeleitet. View und Controller bilden zusammen die Benutzungsoberfläche.

Diese Beziehungen bestimmen den möglichen Fluß der Botschaften und der Daten. Das besondere Kennzeichen der Architektur ist die Tatsache, daß das Model weder die Views noch den Controller kennt. (Praktisch heißt das, daß in den Datenklassen keine View– oder Controllerelemente aufgerufen werden dürfen!) Dies bedeutet, daß die interne Datenverarbeitung von der Benutzungsoberfläche gänzlich abgekoppelt ist. Änderungen in der Benutzungsoberfläche haben also keine Auswirkung auf die interne Verarbeitung der Daten und der Datenstruktur.

Die Controllerelemente werden bei grafischen Oberflächen mit Hilfe der Ereignissteuerung aktiviert. Der Ablauf des Programms folgt damit keinem starren Schema wie bei dem EVA-Prinzip, sondern wird durch frei wählbare beliebige Aktionen des Benutzers gesteuert.



- Daten, Repräsentation und Eingaben sind getrennt.



- Zu dem reinen Datenobjekt (Model) kommen selbständige GUI-Objekte (WINDOW, TEXTBOX, BUTTON) hinzu.
- VIEW und CONTROL werden häufig in einem Fensterobjekt zusammen-gefasst.
- Zu jedem MODEL gehört ein VIEW–CONTROLLER-Paar.
- MODEL weiß nichts über VIEW-CONTROLLER.
- VIEW-CONTROLLER kennen MODEL, holen von und schicken ihm die Daten.
- Zwischen den GUI-Objekten und dem Model-Objekt muß eine Verbindung hergestellt werden.

Bild 4: Software Design mit GUI-Objekten

Die Model-Klassen müssen entwickelt und entsprechend dem MVC-Mechanismus zu den View-Klassen in Beziehung gesetzt werden.

Der große Vorteil dieser Architektur besteht darin, daß eine bruchlose Verbindung zwischen den objektorientierten GUI-Klassen und einem objektorientierten Model möglich ist (vgl. Balzert). Allgemeiner betrachtet kann die Unterscheidung in Model und View mit den Kategorien Wesen und Erscheinung gleichgesetzt werden.

Hinzu kommt, daß schnell ein Prototyp der Applikation, bzw. der Benutzungsoberfläche zu erstellt werden kann, ohne daß das Model bereits fertig vorliegt. Unterstützt wird dies insbesondere dann, wenn die Entwicklungsumgebung das interaktive Erstellen von Oberflächen mit einem GUI-Builder unterstützt.

Beispiele in Objekt-PASCAL, DELPHI, JAVA

Um den Mechanismus dieser Architektur möglichst deutlich hervorzuheben, wurde bewußt ein sehr simples Beispiel gewählt. Die Sinnhaftigkeit des Beispiels steht hier nicht zur Debatte, es geht nur darum, Klassen und ihre Beziehungen und deren Implementation zu zeigen. Zwischen TZahl und TZaehler besteht eine Vererbungsbeziehung (*kind of*), während diese mit der Klasse TRatespiel durch eine Aggregationsbeziehung verbunden sind (TZahl und TZaehler sind *part of* Spiel). Assoziationen liegen im GUI-Fenster vor.

Die Beispiele in DELPHI und JAVA zeigen die Implementation der GUI-Fenster, in denen auf das Datenmodell zugegriffen wird. Die Verbindung zwischen Model und View/Controller wird mit den Methoden `DatenAktualisieren` und `MaskeAktualisieren` hergestellt.

In Objekt-PASCAL

```
UNIT uSpielZahl;
(* ********************************************************************* *)
(* K L A S S E : TSpielzahl                                              *)
(* --------------------------------------------------------------------- *)
INTERFACE

USES BILDSCHIRM, TXTBOX;

type
   TSpielzahl = object
                  Wert  : 0..100;
                  XPos,
                  YPos  : integer;

                  constructor Init (eineXPos, eineYPos : integer);
                  procedure ZufaelligErzeugen;
                  procedure Eingeben;
                  procedure Zeigen;
                  procedure Loeschen;
                  function  IstGleich (andereZahl : TSpielzahl ) : boolean;
                  function  IstKleiner (andereZahl : TSpielzahl) : boolean;
                end;

IMPLEMENTATION
...
END.
```

Bild 5 : Turbo-PASCAL 7; Ein- und Ausgaben sind in der Klasse integriert.

In DELPHI

```
UNIT uSpielZahl;
(* ********************************************************************** *)
(* K L A S S E : TSpielZahl                                               *)
(* ---------------------------------------------------------------------- *)
INTERFACE

type
   TSpielzahl = class (Tobject)
                  Wert : 0..100;
                  constructor Init ;
                  procedure Setwert ( w: integer);
                  function  Getwert : integer;
                  procedure ZufaelligErzeugen;
                  function  IstGleich (andereZahl : TSpielzahl ): boolean;
                  function  IstKleiner (andereZahl : TSpielzahl): boolean;
                end;

IMPLEMENTATION
(* ====================================================================== *)

 constructor TSpielzahl.Init ;
(* ---------------------------------------------------------------------- *)
 begin
   Wert := 0;
 end;

 procedure TSpielzahl.Setwert (w: integer);
(* ---------------------------------------------------------------------- *)
 begin
   Wert := w;
 end;

 function TSpielzahl.Getwert : integer;
(* ---------------------------------------------------------------------- *)
 begin
   Result := Wert;
 end;

...
END.
```

Bild 6: Klasse Tspielzahl mit DELPHI: Standardmethoden ohne GUI-Zugriffe.

```
unit uSpielFenster;
(* ********************************************************************** *)
(* K L A S S E : TSpielFenster                                            *)
(* ---------------------------------------------------------------------- *)

interface

uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
     Dialogs,StdCtrls,
     uRatespiel; (* import: TRatespiel *)

type
  TSpielFenster = class(TForm)
                    EingabeLbl   : TLabel;
                    Eingabefeld  : TEdit;
                    ZaehlerLbl   : TLabel;
                    EndeBtn      : TButton;
                    Zaehler      : TLabel;
                    MeldeLeiste  : TLabel;
                    GeheimLbl    : TLabel;
                    Programmkopf : TLabel;

                    procedure Init;
                    procedure Beenden (Sender: TObject);
                    procedure TipEingeben (Sender: TObject; var Key: Char);
                    procedure Zeigen (Sender: TObject);

                    procedure DatenAktualisieren (sp: TRatespiel);
                    procedure MaskeAktualisieren (sp: TRatespiel);
                  end;

implementation
...


procedure TSpielFenster.DatenAktualisieren (sp: TRatespiel);
(* ---------------------------------------------------------------------- *)
begin
  with sp do
  begin
    Ratezahl.Setwert(strToInt(EingabeFeld.Text));
    VersuchsZaehler.Erhoehen;
  end;
end;


procedure TSpielFenster.MaskeAktualisieren (sp : TRatespiel);
(* ---------------------------------------------------------------------- *)
begin
  with sp do
  begin
    Zaehler.Caption := inttostr (Versuchszaehler.GetWert);
  end;
end;

END.
```

Bild 7: GUI-Klasse SpielFenster in DELPHI zusätzlich zum Datenmodell mit den zwei Methoden, die den Datentransport zum Model erledigen.

In JAVA

```
/* ********************************************************************** */
/* K L A S S E : TSpielZahl                                               */
/* ********************************************************************** */

import java.util.Random;

public class TSpielZahl
{
    int Wert;
    private static Random r;


public TSpielZahl()
{
    r = new Random();
    Wert = 0;
}


public void SetWert(int x)
{
    Wert = x;
}

public int GetWert()
{
    return Wert;
}
```

Bild 8: Klasse TSpielzahl mit JAVA: Standardmethoden ohne GUI-Zugriffe.

```
/* ********************************************************************** */
/* K L A S S E   : zraten (Spielefenster)                                 */
/* ********************************************************************** */

import sunsoft.jws.visual.rt.base.*;
import sunsoft.jws.visual.rt.shadow.java.awt.*;
import java.awt.*;

public class zraten extends Group {


public boolean DatenAktualisieren(Object Eingabe)
{
   try
   {
       Ratespiel.Ratezahl.SetWert(Integer.parseInt(Eingabe.toString()));
       Ratespiel.Versuchszaehler.Erhoehen();
       gui.EingabeFeld.set("text", "");
       return true;
   } /*try*/

   catch (NumberFormatException Ausnahme)
   {
       gui.MeldungLeiste.set("text", "Nur Zahlen erlaubt!");
       gui.EingabeFeld.set("text", "");
       return false;
   } /*catch*/
} /*DatenAktualisieren*/


public void MaskeAktualisieren()
{
   gui.ZaehlerFeld.set("text",
        new Integer(Ratespiel.Versuchszaehler.GetWert()).toString());
} /*MaskeAktualisieren*/
```

Bild 9: GUI-Klasse SpielFenster in JAVA zusätzlich zum Datenmodell mit den zwei Methoden, die den Datentransport zum Model erledigen.

Einschätzung zur Unterrichtstauglichkeit

Object-PASCAL

Mit Objekt-PASCAL (Turbo-PASCAL) lassen sich gut Programme realisieren, die von der OOA über OOD bis zu OOP ohne Brüche in einer Linie stehen. Die in der Analysephase gefundenen Klassen lassen sich relativ einfach und direkt in Programmcode umsetzen.

Der Aufwand für das Lernen zusätzlicher Syntax ist sehr klein. Vererbung und Polymorphie werden unterstützt; selbstprogrammierte Ereignissteuerung und ansprechende Benutzungsoberflächen sind mit dieser Sprache jedoch nur sehr aufwendig zu realisieren. Hier helfen selbstentwickelte Bibliotheksmodule. Das alte TURBO VISION lohnt den Lernaufwand nicht, schon weil die Oberfläche völlig veraltet ist. Wer Turbo-PASCAL benutzen will, sollte statt Turbo.exe den TPX.exe Compiler installieren, der OOP speziell unterstützt.

Daneben ist PASCAL, wenn es schüler/innen- und lernzieladäquat eingesetzt wird, als Lehr- und Lernsprache in der Schule immer noch unübertroffen. Allerdings verlangt OOP eine große Selbstdisziplin, um nicht in hybride Programmierweise zu verfallen. Ähnliches gilt für DELPHI und eingeschränkt für JAVA, solange keine Fachkonzeptklassen entworfen werden. Zu den Vorteilen von PASCAL gehört auch die weitgehende Verbreitung in den Schulen und der geringe Einsatz von Ressourcen.

DELPHI

Einen großen Vorteil bietet der GUI-Builder, mit dem in relativ kurzer Zeit einfache Benutzungsoberflächen erstellt werden können. Die Erfahrung zeigt, daß die Möglichkeit zur individuellen Oberflächengestaltung mit professionellem Aussehen, Bildern und Farben sehr zur Erhöhung der Motivation beiträgt. Aspekte der Softwareergonomie lassen sich für die gute Gestaltung von Benutzungsschnittstellen im Unterricht gut aufnehmen. Die Syntax von DELPHI baut auf (Objekt-)PASCAL auf und ist damit vielen bekannt.

DELPHI präsentiert sich nach dem Start mit vier Fenstern. Es organisiert Programme als Projekteinheiten und legt mindestens 5 verschiedene Dateien an, bevor auch nur eine Zeile Code geschrieben wurde. Deshalb ist eine Kenntnis der Arbeitsweise des Systems erforderlich, was einen zusätzlichen Lernaufwand bedingt. Verstöße gegen die Projektorganisation und (un)beabsichtigtes Verändern des automatisch erzeugten Codes führen häufig zur Unbrauchbarkeit und zum Verlust des Programms. Eine Reihe von GUI-Komponenten und DELPHI-Klassen sind hoch komplex und schwer verständlich, zumal die integrierte Dokumentation nur Kennern weiterhilft. Zu DELPHI als Programmiersprache ist viel Literatur vorhanden, zur objektorientierten Programmierung mit DELPHI so gut wie nichts.

JAVA

Die auf dem Markt erhältlichen JAVA Entwicklungsumgebungen mit GUI-Builder sind ähnlich in Umfang und Bedienung wie in DELPHI. Damit gelten die Vor- und Nachteile bei der Arbeit mit GUI-Buildern gleichermaßen. Die grafischen Oberflächen lassen sich im Gegensatz zu DELPHI aber mit Hilfe von Layout-Managern plattform- und bildschirmunabhängig entwickeln. Zu bedenken ist, daß JAVA immer noch Weiterentwicklungen unterliegt mit teilweise erheblichen Änderungen in den Klassenbibliotheken für die grafischen Oberflächen.

JAVA basiert auf einer reduzierten C-Syntax, was für viele in der Schule einen höheren Lernaufwand mit sich bringt. Als rein klassenorientierte und nicht hybride Sprache unterstützt JAVA per se ein objektorientiertes Vorgehen. Da die Basisversion kostenlos erhältlich ist, kann sie auch an Schülerinnen und Schüler für Hausarbeiten weitergegeben werden.

JAVA ist auf vielen Maschinen verfügbar und die Programme lassen leicht ins WWW einbinden, was erheblich zur Motivation der Schülerinnen und Schüler beiträgt.

Fazit

- Trotz der beschriebenen Umstellungsschwierigkeiten auf DELPHI oder JAVA empfehlen wir für den Informatikunterricht der SEK II diesen Umstieg. *Sinnvoll ist ein solcher Umstieg aber nur dann, wenn er auch mit einem Wechsel im Programmierstil in Richtung OOP einhergeht.* Wer jedoch nur seine alten Programme mit diesen Sprachen neu verpacken will, sollte den damit verbunden Aufwand überdenken. Ohne Kenntnis und Verständnis von OOP und der mitgelieferten Klassenbibliothek stößt man bei der Implementierung sowohl in DELPHI als auch in JAVA häufig auf unüberwindliche Anfangsschwierigkeiten und Probleme, die ohne fremde Hilfe nicht lösbar sind.

- Wer sich noch nie mit OOP beschäftigt hat, sollte zuerst in der ihm vertrauten Programmiersprache, soweit diese es zuläßt, objektbasierte modulare Programme schreiben (vgl. Spolwig) und nach entsprechender Erfahrung auf andere Sprachen umsteigen.

- Bei konsequentem objektorientierten Vorgehen führen Ereignisorientierung und grafische Oberfläche zu einer erheblichen Zunahme der Komplexität. Die gewonnene Zeit, die bei der schnellen und eleganten Entwicklung der Oberflächen herausarbeitet wird, wird beim Realisieren und Verknüpfen der komplexen Klassenstrukturen mehr als aufgebraucht.

- Der konsequente Einsatz des MVC-Konzepts eignet sich hervorragend, um ein klares Fachkonzepts beim Anwendungsproblem herauszuarbeiten und View-/Controllerkomponenten im Sinne einer sauberen Softwarearchitektur davon zu trennen. Hierdurch sind unmittelbar Bezüge und andere Realisierungsmöglichkeiten im Sinne von Client – Server Anwendungen vorbereitet.

- Nach unseren Erfahrungen im Unterricht und in der Fortbildung resultieren *die Schwierigkeiten beim Einsatz dieser Sprachen in erster Linie aus der Datenmodellierung durch Klassenbildung und der Ablaufsteuerung durch Ereignisse, die vielen ungewohnt ist.* Der Fortbildungsbedarf liegt unseres Erachtens bei diesen Schwerpunkten und nicht - wie häufig angeboten und nachgefragt - bei reinen Programmierkursen in JAVA oder DELPHI.

Literatur

Balzert, H.: Lehrbuch der Software-Technik: Software-Entwicklung, Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford 1996.

Baumann, R.: JAVA – Stimulans für den Informatikunterricht. IN: LOG IN 17 (1997) Heft 5

Böttcher, K.: JAVA jetzt – adieu PASCAL. IN: LOG IN 17 (1997) Heft 6

Borkner-Delcarlo, Olaf : KDE 1.1 programmieren und anwenden, MITP Bonn 1998

Braun, W.: Einführung in die visuelle Projekterstellung mit DELPHI, Winklers Verlag 1997.

Dick, G.: Leserbrief zu ‚JAVA jetzt – adieu PASCAL'. IN: LOG IN 18 (1998) Heft 2.

Doberenz, W.; Kowalski, T.: Borland Delphi 3 für Einsteiger und Fortgeschrittene. Hanser Verlag, München, Wien 1997

Doberenz, W.: JAVA, Hanser Verlag, München, Wien 1996

Modrow, E.: Informatik mit DELPHI, Ferd. Dümmlers Verlag, Bonn 1998.

Neumann, Horst, A.: Objektorientierte Entwicklung von Software-Systemen. Addison-Wesley 1995

Spolwig, S.: Objektorientierung im Informatikunterricht. Ferd. Dümmlers Verlag, Bonn 1997.