KAPITEL 3

jetzt lerne ich

Grundsätzliches zu Object-Pascal

Für jede Programmiersprache gibt es eine Reihe von Regeln, an die der Programmierer sich halten muss, wenn der Compiler ihn verstehen soll. Um diese Grundregeln wollen wir uns in diesem Kapitel kümmern.

3.1 Der Aufbau von Programmen

Um uns die Regeln deutlich zu machen, die für Object-Pascal gelten, schauen wir uns doch am besten ein von einem Delphi-Experten erzeugtes Programm an – eines, das von Delphi selbst erzeugt wurde! Wenn wir Delphi starten, begrüßt es uns ja schließlich mit einem leeren Formular – wenn man dieses compiliert, erscheint ein Fenster, das man verschieben, in der Größe ändern und wieder schließen kann. Nicht viel, sicher – aber ein komplettes Programm.

Wenn Delphi gestartet wird, sieht man das leere Formular, genannt »Form1«. Drückt man nun die Taste F12, so sieht man den dazugehörigen Quelltext – also das (genauer: ein Teil des) von Delphi bereits erzeugte Programm.

Delphi 1 generiert folgenden Quelltext:

```
unit Unit1;

interface

uses
SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs;

type
TForm1 = class(TForm)
private
{ Private-Deklarationen }
public
{ Public-Deklarationen }
end;

var
Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Ab Delphi 2 sehen die beiden Zeilen hinter dem Wort »uses« etwas anders aus:

```
...
uses
Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs;
...
```

Oft ist es sinnvoller, nur einen kleinen Ausschnitt eines Quelltextes zu zeigen. Schlicht und einfach, weil der Rest nicht interessiert. In diesem Fall finden Sie vor oder hinter dem Quelltext drei Punkte »...«. Das bedeutet: »Hier kommt noch mehr.« Sie könnten das aber so nicht eingeben, Delphi könnte damit nichts anfangen.

Wagen wir doch einmal einige Experimente: Ändern wir die erste Zeile in

```
UNIT UNIT1;
```

(also nur Großbuchstaben) und lassen Delphi das Programm erneut übersetzen – dazu muss Strg F9 gedrückt werden. Einen Moment lang verwandelt sich der Mauszeiger in eine Sanduhr, aber das war es auch schon – denn Delphi hat das Programm erneut übersetzt, aber – vor allem – ohne Klage! Wir sehen:

Groß- bzw. Kleinschreibung spielt keine Rolle!

Den Compiler hätte auch UnIt Unit1; nicht gestört – das ist keineswegs selbstverständlich, denn in anderen Programmiersprachen wie zum Beispiel in C spielt Groß-/Kleinschreibung sehr wohl eine Rolle.

Überhaupt ist Pascal in dieser Hinsicht sehr tolerant – auch ohne es auszuprobieren, können Sie mir glauben, dass der Compiler auch folgende Zeilen schluckt:

```
Unit
      Unit1
;
```

Das heißt:

Leerräume und neue Zeilen (White-Spaces) werden ignoriert.

Wann wird der Compiler denn nun intolerant? Beispielsweise dann, wenn wir das Semikolon am Ende der Zeile entfernen und dann einen erneuten Versuch mit Strg + F9 starten. Nun werden wir mit der Fehlermeldung *Fehler im Modul Unit1: Modul-Header fehlt oder ist fehlerhaft* belohnt.

Woran liegt das? Zwei Anweisungen – komplette Befehle sozusagen – werden in Pascal immer durch Semikolons getrennt. Die erste Zeile sagt dem Compiler, dass es sich bei diesem Programmstück um die Unit mit dem Namen Unit1 handelt (was eine Unit ist, dazu später mehr). Nehmen wir ihm nun das Semikolon weg, so glaubt Delphi, der Befehl wäre noch nicht zu Ende. Da aber das »Interface« in Zeile 2 keinen Sinn im Zusammenhang mit der ersten Anweisung macht, erhalten wir eine Fehlermeldung. Also:

Das Semikolon trennt in Pascal zwei Anweisungen. Man sollte an das Ende eines Befehls immer ein Semikolon setzen, auch wenn das in Ausnahmefällen nicht immer notwendig ist.

Ein weiteres Experiment: Hängen wir an das Unit noch ein »s« an, so dass die erste Zeile so aussieht:

```
Units Unit1;
```

… so gefällt auch das dem Compiler nicht: Wieder erhalten wir dieselbe Fehlermeldung. Beim Eintippen des »s« wird das Wort plötzlich nicht mehr, wie vorher, **fett** dargestellt. Dies ist eine Besonderheit des Delphi-Editors. Fett werden nämlich die **reservierten Worte** angezeigt, die Worte, die Object-Pascal als Befehle erkennt. Und »Unit« ist eben so ein Befehl, »Units« aber nicht. Daraus erklärt sich auch die Fehlermeldung.

Object-Pascal kennt bestimmte Befehle. Im Editor werden sie fett dargestellt.

Solche Befehle lassen sich natürlich nicht für eigene Bezeichnungen innerhalb des Programms verwenden.

Doch nun zur ganzen Wahrheit: Wir haben uns bisher überhaupt nicht das eigentliche Programm angesehen, sondern einen ganz bestimmten Teil, nämlich (aha!) eine Unit. Wie sieht man nun das eigentliche, das *Hauptprogramm*?

Hierzu gibt es im Menü ANSICHT den Punkt PROJEKT-QUELLTEXT (bzw. ab Delphi 4.0 das Menü PROJEKT mit dem Punkt QUELLTEXT ANZEIGEN), der Folgendes zu Tage fördert. Delphi 1 generiert hier folgenden Quelltext:

```
program Project1;

uses
Forms,
Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}

begin
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

Ab der Version 2 von Delphi existiert zwischen den Wörtern »begin« und »end« eine weitere Zeile:

```
...
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

Dies ist in Wirklichkeit das wahre Programm – jedenfalls sein Hauptteil. Werfen wir doch einen näheren Blick darauf:

Das Programm beginnt mit dem Befehl *program* und dem Namen des Programms, nämlich *project1*. Da der Compiler nun weiß, dass das Programm Projekt1 heißen soll, ist der Befehl zu Ende, und es folgt – richtig – ein Semikolon.

Dieses Programm braucht aber noch etwas anderes, damit es funktioniert: die beiden Units *Forms* und *Unit1*. Die *Unit1* kennen wir schon, schließlich ist das der Teil, den wir uns zuerst angesehen haben. *Forms* wird zu Delphi mitgeliefert und enthält einige Dinge, die fast jedes Delphi-Programm benötigt – mehr dazu später. Der Befehl *uses* (benutzt) teilt dem Programm also mit, dass es diese beiden Units benutzen soll. Aber nicht nur das: Mit `in 'UNIT1.PAS'` wird ihm sogar mitgeteilt, in welcher Textdatei es die Unit zu suchen hat. Jetzt wäre der Befehl eigentlich zu Ende, doch vor dem Semikolon kommt noch der Text `{Form1}`.

Dabei handelt es sich um eine Freundlichkeit von Delphi für den Programmierer, nämlich um einen Kommentar. Mit anderen Worten – das, was zwischen den geschweiften Klammern steht, interessiert den Compiler nicht, der Programmierer kann sich dazwischen sozusagen Notizen machen. Mit diesem Kommentar gibt Delphi uns den Hinweis, dass die Unit1 für das Formular Form1 zuständig ist.

Kommentare im Programmcode stehen zwischen { }.

Es gibt noch eine zweite Möglichkeit für Kommentare: (* *) statt der geschweiften Klammern. Die Zeile hätte also genauso gut

```
Unit1 in 'UNIT1.PAS' (* Form1 *);
```

lauten können. Anschließend folgt noch ein Spezialkommentar. Es handelt sich um einen besonderen Kommentartyp, dessen Inhalt der Compiler eben doch beachtet (keine Regel ohne Ausnahme). Wir wollen dies aber erst einmal ignorieren und die restlichen vier Zeilen unseres Programms betrachten:

Ab Delphi 3.0 können Sie einen Kommentar auch durch zwei Divisionszeichen einleiten, etwa so:

```
// dies ist ein Kommentar
```

Ein solcher Kommentar reicht bis ans Zeilenende.

Der Befehl *begin* wird Ihnen noch sehr, sehr oft begegnen, vorausgesetzt, Sie programmieren in Zukunft mit Pascal. Mit ihm wird eine Anweisungsfolge eingeleitet, die logischerweise mit *end* endet. Alle Befehle bzw. Anweisungen zwischen *beginn...end* werden sozusagen zu einem einzigen Block zusammengefasst.

In diesem Fall handelt es sich bei dem Block um das komplette Programm, das mit der ersten Anweisung hinter b*egin* anfängt und mit *end* wieder aufhört.

Aber Achtung: Warum steht hinter *end* kein Semikolon – schließlich ist ein *beginn...end*-Block doch ein Befehl? Richtig, eigentlich würde man hier ein Semikolon erwarten, aber der Punkt hinter *end* hat eine ganz besondere Bedeutung, nämlich: »Hier ist das Programm zu Ende.«

Davon kann man sich leicht überzeugen: Tippen Sie ein oder zwei Zeilen unter dem *end*. irgendetwas ein, zum Beispiel »blablabla«. Obwohl das sicher (nicht nur) für den Compiler keinen Sinn macht, so beschwert er sich trotzdem nicht darüber. Denn mit dem *end.* ist das Programm für ihn zu Ende. Den Rest schaut er sich gar nicht mehr an. Und damit Punkt.

Innerhalb eines Programms fasst man mit *Begin...End* Anweisungen zu einem *Block* zusammen. Ein solcher Block wird wie eine einzelne Anweisung betrachtet, daher trennt man mehrere Blöcke bzw. Anweisungen durch Semikolons.

Das Programm selbst beginnt mit *begin* und endet mit *end*, auf das letzte *end* folgt jedoch kein Semikolon, sondern ein Punkt.

Kurz gesagt: In den beiden Zeilen

```
Application.CreateForm(TForm1, Form1);
Application.Run;
```

bzw. ab der Delphi-Version 2

```
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
```

steckt die gesamte Funktionalität des Programms. Wir werden später sehen, wie das möglich ist.

Der prinzipielle Aufbau eines Object-Pascal-Programms sieht also (nach unserem bisherigen Wissen, das wir später noch ergänzen werden) folgendermaßen aus:

```
Program Name ;
Uses unit1, unit2, ..., Unitx ;
            Begin
                  Befehle
            End.
```

Syntaktisch korrekt wäre es auch, wenn die *Befehle* fehlen würden – das Programm würde dann gar nichts tun.

Wenn man dem Programm keinen Namen zuweisen möchte, so kann sogar die »program«-Anweisung weggelassen werden – und wenn man keine Units benötigt, auch der »Uses«-Befehl. Ein Programm muss aber mit »begin« beginnen und mit »end.« enden.

3.2 Was sind Standardanweisungen und reservierte Wörter?

Reservierte Wörter haben wir bereits kennen gelernt: *program, begin, end, unit, uses*. All dies sind Wörter, die für den Delphi-Compiler eine ganz bestimmte Bedeutung haben – sie sind für einen bestimmten Zweck reserviert. Im Folgenden erhalten Sie eine Übersicht über die reservierten Wörter in Delphi. Kursiv geschriebene Wörter sind dabei neu gegenüber Borland Pascal 7.0, fett geschriebene Wörter sind neu in Delphi 2 (gegenüber Delphi 1), fett und kursiv geschriebene Wörter sind neu ab Delphi 3.

and	*exports*	*library*	shl
array	file	mod	shr
as	**finalization**	nil	string
asm	*finally*	not	then
begin	for	object	to
case	function	of	**threadvar**
class	goto	*on*	*try*
const	if	or	type
constructor	implementation	packed	unit
destructor	in	procedure	until
div	inherited	program	uses
do	*initialization*	*property*	var
downto	inline	*raise*	while
else	interface	record	with
end	*is*	repeat	xor
except	label	set	

Es gibt jedoch noch weitere Begriffe, die für den Delphi-Compiler eine vordefinierte Bedeutung haben: Aber sie werden nicht ganz so streng behandelt wie die reservierten Wörter, denn es ist möglich, sie im Programm neu zu definieren. Dies ist aber so gut wie nie sinnvoll. Hier sind die so genannten *Standardanweisungen*:

absolute	export	near	**register**
abstract	external	*nodefault*	resident
assembler	far	*override*	*stdcall*
at	forward	private	**stored**
automated	index	*protected*	virtual
cdecl	interrupt	public	write
default	*message*	*published*	
dynamic	name	read	

3.3 Reservierte Zeichenfolgen

Das war aber keineswegs alles, was der Compiler für sich reserviert, denn auch bestimmte Sonderzeichen haben in Pascal einen bestimmten Zweck. Zwei reservierte Zeichen kennen wir ja schon – das Semikolon und den Punkt. Beide haben eine bestimmte Bedeutung, dürfen also nicht nach Belieben verwendet werden. Auch für andere Zeichen ist dies unmittelbar einsichtig: + für Addition oder = für Vergleiche leuchten ein. Gut, zugegeben: Der Sinn von := oder # ist vielleicht weniger einsichtig. Aber später mehr dazu. Hier eine Übersicht über alle reservierten *Sonderzeichenfolgen*:

+	.	<>	^
-	,	<	(
*	;	>	)
/	:	>=	..
[	]	<=	@
:=	=	{	}
$	#	(*	*)
(.	.)		

3.4 Was sind Bezeichner?

Bisher haben wir eine ganze Menge davon gehört, was verboten ist – nämlich welche Wörter und Zeichen Delphi für sich selbst reserviert hat. Aber wir dürfen eine ganze Menge von Dingen so benennen, wie wir möchten, unter anderem:

Konstanten, Variablen, Typen, Funktionen, Programme, Units

Diesen Dingen können wir Namen geben, die für uns aussagekräftig sind. Solche Namen werden *Bezeichner* genannt. Aber – wie kann es auch anders sein – es gelten natürlich auch für Bezeichner bestimmte Regeln, nämlich:

- Bei Bezeichnern werden maximal 63 Zeichen unterschieden.
- Ein Bezeichner muss mit einem Buchstaben oder einem Unterstrich beginnen.
- Ein Bezeichner kann keine Leerzeichen oder White-Spaces enthalten.

3.5 Zusammenfassung

- Groß- und Kleinschreibung spielt keine Rolle.
- White-Spaces sind mehrere Leerräume, Tabulatoren oder neue Zeilen.
- White-Spaces werden ignoriert.
- Befehle werden im Editor hervorgehoben dargestellt.
- Mehrere Anweisungen werden durch Semikolons getrennt. (Merkregel: Das Semikolon schließt eine komplette Anweisung ab.)
- Kommentare setzt man in (* *), in { } oder ab Delphi 3 hinter //.
- Bezeichner müssen mit einem Buchstaben oder einem Unterstrich (_) beginnen.
- Der Projekt-Quelltext ist das Hauptprogramm – die gesamte Anwendung besteht aus Hauptprogramm und Units.
- Mehrere Pascal-Anweisungen werden mit *begin* und *end* zu einem Block zusammengefasst.
- Hinter dem letzten *end* folgt ein Punkt.
- Einige Worte und Zeichenfolgen sind *reserviert* – es handelt sich um Anweisungen der Sprache Pascal.
- Standardanweisungen sind reservierte Worte, die aber neu definiert werden können.

3.6 Testaufgaben

3.1 Ist die Position des Semikolons in diesen Anweisungen korrekt?

a)
```
Program test
;
```

b)
```
Program ; test ;
```

c)
```
; program test;
```

d)
```
program test (* abcd *) ;
```

e)
```
program test ; (* abcd *)
```

f)
```
program test (* abcd ; *)
```

3.2 Welche der folgenden Anweisungen sind nicht korrekt?

```
UNit unIT1;
Unit unit1
unit unit;
unit
unit1;
```

3.3 Muss der Befehl *program* am Anfang eines Programms angegeben werden?

3.4 Welche der Zeilen sind nicht korrekt?

```
Uses Unit1 in Unit1.PAS (* Form1 *);
Uses Unit1 in 'Unit1.pas' ; (Form1)
```

3.5 Was ist an diesem »Programm« falsch?

```
program test;
begin
end;
```

3.6 Welche der folgenden Bezeichner sind gültig?

```
_12Test
2ter_Aufruf
_2ter Aufruf
März
```