2 Programmentwicklung mit Delphi

Die Programmentwicklung mit Delphi wird in drei Schritten eingeführt:
- Zuerst wird die prinzipielle Arbeitsweise geschildert,
- danach wird das Zusammenspiel der erzeugten Module beschrieben, insbesondere die zeitliche Reihenfolge der Aktionen,
- und zuletzt wird ein sehr reduziertes Beispielprogramm gebracht.

2.1 Arbeiten mit Delphi

Delphi ist ein Werkzeug im Sinne von Kapitel 1.1.2, Seite 19 zur Entwicklung von Programmen, die unter dem Betriebssystem ©*Microsoft Windows* laufen. Es beruht auf der Programmiersprache *Object-Pascal*. Die Grundidee des Systems besteht darin, den Programmierern fertige Programmstrukturen und darin lauffähige Objektklassen für die Standardaufgaben eines Windows-Programms bereitzustellen. Die Programmierer erzeugen – auch hier von Delphi unterstützt – Instanzen der vorgegebenen Objektklassen, deren Datenfelder sie mit konkreten Werten füllen. Das geschieht weitgehend interaktiv am Bildschirm, indem die Instanzen entweder mit der Maus bearbeitet werden, um z. B. die Größe und Position zu bestimmen, oder durch das Setzen bestimmter Eigenschaften (*Properties*) wie Farbe oder Schriftart mit Hilfe eines *Objektinspektors*.

Windows-Objekte agieren (meist) nicht selbst, sondern reagieren auf Ereignisse (*Events*), die z. B. durch einen Mausklick oder einen Tastendruck auf der Tastatur ausgelöst werden. Das Windows-Betriebssystem empfängt solche Ereignisse und reicht sie an die Elemente auf dem Bildschirm weiter. Ein typisches *Mausereignis* ist ein Doppelklick auf das Kreuz-Symbol in der oberen rechten Ecke eines Fensters, das z. B. eine Meldung <Doppelklick an der Position (400,120)> erzeugt, die dann der Reihe nach an die Fenster des *Desktops* durchgereicht wird. Stellt das angeklickte Fenster fest, dass sein Kreuzchen getroffen wurde, dann reagiert es darauf, indem es sich selbst schließt.

Die Delphi-Objekte müssen somit

- dem Windows-Systems bekannt gemacht werden, um die Botschaften des Systems zu empfangen, und
- über Methoden verfügen, um auf die Standardereignisse des Systems zu regieren.

Beides wird erreicht, indem alle Objekte von einer Mutterklasse *tObject* abgeleitet werden, die über solche Eigenschaften verfügt und sie an die Tochterklassen vererbt. Die Reaktion auf die Standardereignisse beruht typischerweise darin, nichts zu tun.

Weil die Elemente eines Windows-Programms ebenso wie die Standardereignisse festgelegt sind, kann das Delphi-System fertige Programmschablonen erstellen, die die vom Programmierer am Entwicklungsbildschirm zusammengestellte Oberfläche aus Fenstern, Beschriftungen, Knöpfen, Ein- und Ausgabefeldern, ... erzeugt, ohne dass eine einzige Zeile Pascal-Quelltext selbst geschrieben werden muss. Damit ist aber nur festgelegt, wie die Oberfläche aussieht, aber nicht, was sie tut. Bis auf wenige Standardreaktionen (z. B. Fenster verkleinern oder schließen) reagiert das Programm gar nicht!

Das Programmieren unter Delphi besteht deshalb weitgehend daraus, die anfangs leeren Ereignisbehandlungsmethoden (*Event-Handler*) auszufüllen, indem Pascal-Quelltext eingegeben wird, der beschreibt, wie das gerade bearbeitete Objekt auf einzelne Ereignisse zu reagieren hat:

- Was passiert, wenn ein bestimmter Knopf angeklickt wird? *(Das Programm wird beendet.)*
- Was passiert, wenn ein anderer Knopf angeklickt wird? *(Eine Datei wird gespeichert.)*
- Was passiert, wenn eine Taste gedrückt wird? *(Je nach gedrückter Taste wird anders reagiert.)*
- ...

Das Schreiben von Delphiprogrammen geschieht im allgemeinen in drei Phasen:

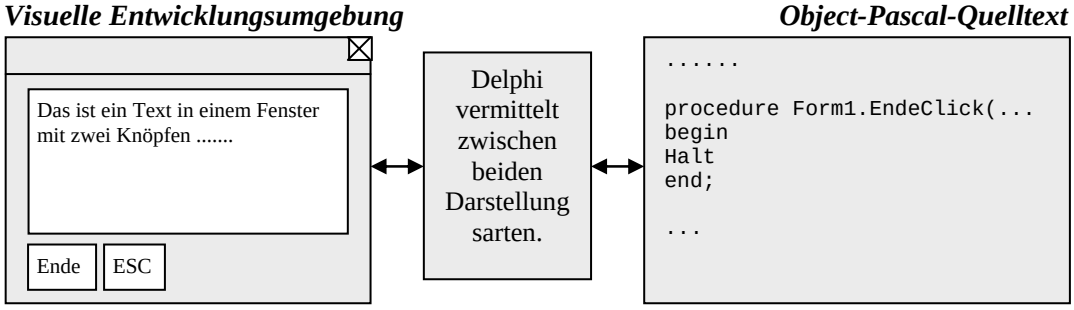
1. **Zusammenstellung der Programmoberfläche in der Entwicklungsumgebung.**
   Dazu werden Instanzen der Klassen erzeugt, die in der Visuellen-Komponentenpalette am oberen Bildschirmrand angeboten werden, z. B. Fenster, Knöpfe, Textfelder, ... Die Eigenschaften dieser Objekte werden mit Hilfe des Objektinspektors geeignet gesetzt. Delphi erzeugt automatisch das dazu gehörende Object-Pascal-Programm.

2. **Ausfüllen der benötigten Ereignisbehandlungsmethoden mit Pascal-Quelltext.**
   Nach Auswahl des gewünschten Ereignisses im Objektinspektor (z. B. *OnMouse-Click*) erzeugt Delphi eine leere Pascal-Routine und springt an die richtige Stelle im Pascal-Programm. Der „wohlüberlegte“ Text wird hier eingegeben.

3. **Übersetzen und Testen des erzeugten Programms**
   Delphi übersetzt das Programm in ausführbaren Code und erzeugt eine ohne weitere Zusätze ausführbare Datei, die sofort gestartet wird. Das so erzeugte Windows-Programm wird normalerweise fehlerhaft sein, zumindest aber unvollständig. Deshalb sucht man die Fehler und ergänzt ggf. die Oberfläche bzw. ändert den Programmcode.

Insgesamt wird mit einem Delphi-System auf zwei verschiedenen Ebenen gearbeitet:

- In der visuellen Entwicklungsumgebung werden Objekte erzeugt und mit Eigenschaften ausgestattet, die die spätere Programmoberfläche bilden.
- Parallel dazu wird von Delphi automatisch Programm-Quelltext erzeugt, der zur Laufzeit genau die in der Entwicklungsumgebung definierte Oberfläche nachbildet. Dieser wird ggf. per Hand verändert oder ergänzt.



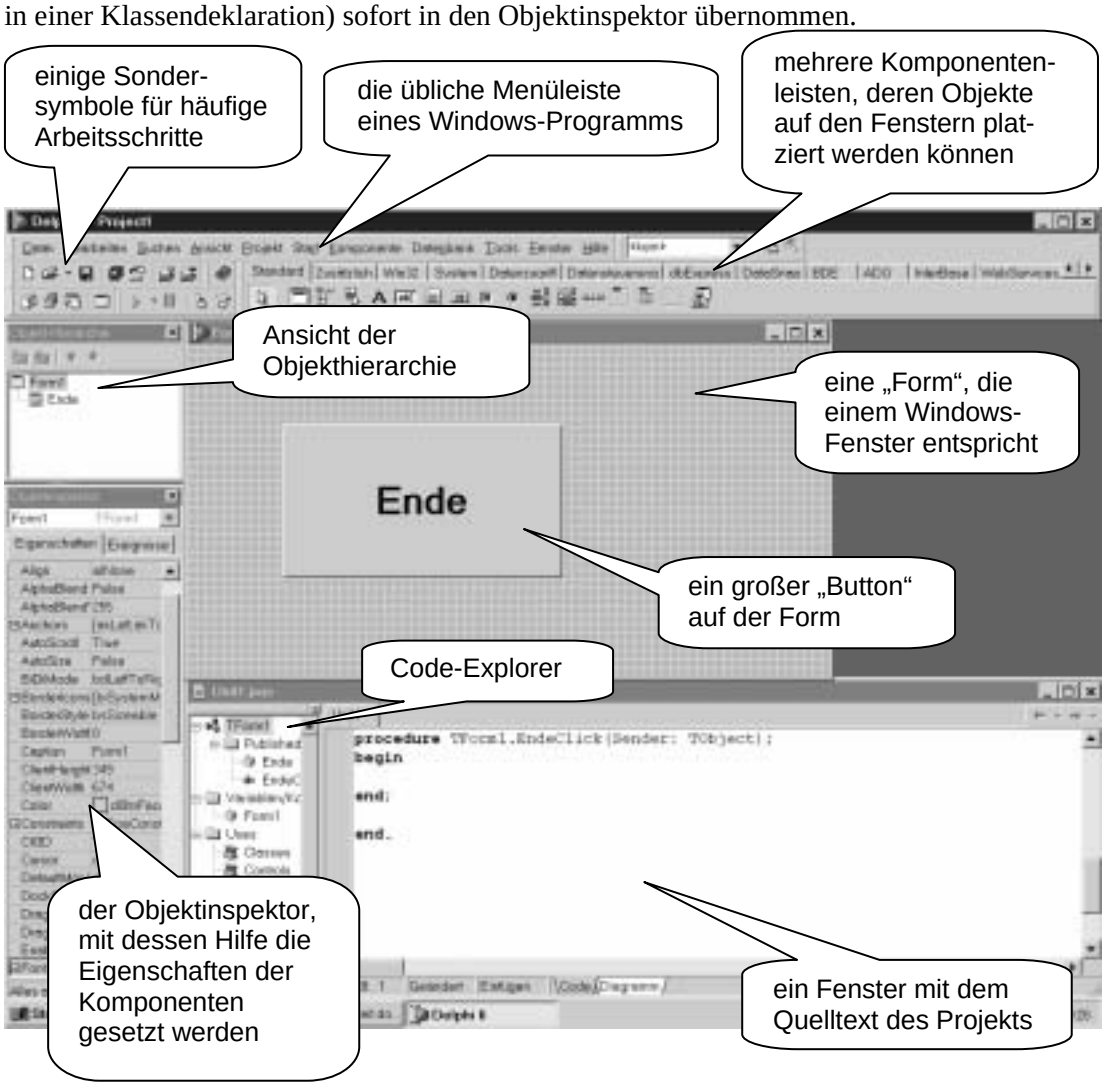
2.2 Die Komponenten eines Delphi-Projekts

Delphi arbeitet mit *Projekten*, die aus allen Bestandteilen bestehen, die erforderlich sind, um vom Compiler zu einem lauffähigen Programm zusammengesetzt zu werden. Dazu gehören neben den Quelltextdateien auch andere, die z. B. die benutzten Komponenten beschreiben. Üblicherweise speichert man alle diese Bestandteile zusammen in einem eigenen Unterverzeichnis.

Startet man ein neues Projekt, dann enthält der Bildschirm eine leere *Form*, die nach der Übersetzung des Programms zu einem (leeren) Windows-Fenster wird. Schon dieses Rahmenprogramm kann man erfolgreich übersetzten, und das leere Fenster verfügt über einige nicht ganz triviale Eigenschaften: es kann z. B. in der Größe verändert, verschoben oder geschlossen werden.

Der Bildschirm der Delphi-Entwicklungsumgebung enthält zur Bearbeitung eines Projekts eine Reihe von Werkzeugen und Anzeigehilfen. Ihre Art und Anzahl hängt von der benutzten Delphi-Version ab. Hier stehen Ihnen Formular-Designer, Objektinspektor, Objekt-Hierarchie, Komponentenpalette, Projektverwaltung, Quelltexteditor und Debugger sowie andere Hilfsmittel zur Verfügung (einige Produktversionen enthalten nicht alle dieser Tools). Sie können jederzeit zwischen dem visuellen Entwerfen des Objekts (Formular-Designer), dem Bearbeiten seines Laufzeit-Anfangsstatus (Objektinspektor) und dem

Erstellen der Ausführungslogik (Quelltexteditor) wechseln. Wenn Sie im Objektinspektor eine quelltextbezogene Eigenschaft ändern (z.B. den Namen einer Ereignisbehandlungsroutine), wird die entsprechende Stelle im Quelltext automatisch aktualisiert. Außerdem werden Änderungen im Quelltext (z.B. Umbenennen einer Methode in einer Klassendeklaration) sofort in den Objektinspektor übernommen.



Die Elemente der Komponentenleisten der Delphiumgebung entsprechen Objektklassen, die über vorgegebene Eigenschaften und Methoden verfügen. Diese können (teilweise) mit Hilfe des *Objektinspektors* angezeigt bzw. verändert werden, aber auch zur Laufzeit des Programms.

- Die wichtigste Eigenschaft ist der *Name* der Komponente, der standardmäßig aus dem Typ und einer laufenden Nummer gebildet wird. Die erste Form hat damit den Namen *Form1*, der erste Button (Knopf) den Namen *Button1* usw. Hier sollte man Namen vergeben, die der Funktion des Objekts entsprechen.
- Beschriftete Komponenten enthalten eine Aufschrift *Caption*, die zuerst dem Namen der Komponenten entspricht. Die Caption kann aber getrennt verändert werden, so dass Name und Aufschrift nicht übereinstimmen müssen – z. B. wenn die Aufschrift aus mehreren Worten besteht oder Sonderzeichen enthält, was in Namen nicht erlaubt ist.
- Die Farbe der Objekte wird in der Eigenschaft *Color* gespeichert. Die Werte können aus einer Auswahlliste gewählt werden, wobei die vorgegebenen Konstanten in der in Delphi üblichen Form durch vorangestellte Buchstaben („cl“ für „color“) bezeichnet sind: *clWhite*, *clRed*, *clButtonface*, ...
- Weitere Eigenschaften hängen sehr von der Funktion der Komponenten ab: Position und Größe werden durch Anordnen der Symbole mit der Maus gesetzt, Schriftart und –größe (Eigenschaft *Font*) können aus einem Auswahlmenü gebildet werden, enthaltener Text kann aus einer Zeile (Eigenschaft *Text*) oder aus mehreren Zeilen (Eigenschaft *Lines*) bestehen, grafische Elemente werden auf der Leinwand (Eigenschaft *Canvas*) erzeugt. Details hierzu folgen in den späteren Kapiteln.
- Im Objektinspektor werden auch die Standardereignisse angegeben, auf die eine Komponente reagieren kann. Jedem Ereignis (z. B. *OnClick*) ist eine (leere) Ereignisbehandlungsmethode zugeordnet, die durch eine eigene ersetzt werden kann (aber nicht ersetzt werden muss). Die Bezeichnung der Methode ergibt sich aus dem Namen des Objekts und dem Namen des Ereignisses. Reagiert also ein ENDE-Knopf auf „OnClick“, dann heißt die entsprechende Methode *EndeClick*. Dieser Name kann aber auch geändert werden, z. B um unterschiedlichen Elementen die gleichen Methoden zuzuordnen.
- Am häufigsten wird die Reaktion auf einen Mausklick benutzt. Aber auch die Erzeugung eines Objekts (*OnCreate*) oder das Neuzeichnen (*OnPaint*) lösen Ereignisse aus
- Der Programmcode dieser Methoden bildet den Hauptteil eines Delphiprogramms. Hier werden Aktionen ausgelöst, die den eigentlichen Sinn des Programms enthalten.

Wenn Änderungen an den in der Form dargestellten Objekten oder im Objektinspektor vorgenommen werden, dann ändert sich gleichzeitig der Programm-Quelltext, den Delphi erzeugt. Dieser Text besteht aus mindestens zwei Dateien:

- Einer Bibliothek (*Unit*), die alle Elemente der Form (des Fensters) enthält, in dem das Programm läuft. Diese Datei wird mit der Endung *PAS* (für PASCAL) gespeichert.
- Einem Hauptprogramm (*Program*), mit dessen Hilfe das Hauptfenster erzeugt und das Programm gestartet wird. Dieses Hauptprogramm wird immer automatisch erzeugt und muss praktisch nie per Hand bearbeitet werden. Die Datei wird mit der Endung *DPR* (für Delphi-Projekt) gespeichert.

Wir nehmen einmal an, dass unser Programm P1 ein Fenster enthält, dessen Eigenschaften in einer Unit U1 gespeichert wurden. Die immer gleiche Aufgabe des Hauptprogramms besteht darin, dieses Fenster zu erzeugen und danach dem Fenster die Aufgabe zu übertragen, auf Tastatur- oder Mauserereignisse zu reagieren. Der Quelltext lautet dann:

```
program P1;
uses Forms,u1 in 'u1.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

benutzte Bibliotheken für „Formen“

Anfangswerte setzen

Fenster erzeugen

warten auf Ereignisse

(Die Compilerdirektive $R bewirkt die Einbindung von *Ressourcendateien*, in denen sich Informationen z. B. über die benutzten Buttons (in *.RES) oder Bilder (in *.DFM) befinden können.)

Die eigentliche Arbeit erfolgt dann in der Bibliothek, die die benötigten Objekte und Ereignisbehandlungsroutinen enthält. Hier ist auch der geeignete Ort, um Hilfsprogramme, weitere Daten etc. unterzubringen. In unserem Fall enthält das Fenster einen Button namens *Ende*, der bei Mausklick das Programm beenden soll (Befehl *Halt*). Im Programmtext wurden einige überflüssige Zeilen gelöscht und dafür einige sinnvolle (hier kursiv gedruckte) eingefügt.

```
unit u1;

interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
     Forms, Dialogs, StdCtrls;

type TForm1 = class(TForm)
        Ende: TButton;
        procedure EndeClick(Sender: TObject);
        end;

var Form1: TForm1;

implementation
{$R *.DFM}
```

ab hier: für das Programm bereitgestellte Größen

die Klasse tForm1 als Tochter von tForm enthält einen Endeknopf und eine neue Ereignisbe-handlungsmethode für diesen Knopf

eine „Instanz“ der Klasse tForm1: das Hauptfenster „Form1“

ab hier: die genaue Definition der bereitgestellten Größen

```
procedure TForm1.EndeClick(Sender: TObject);
   begin
   halt
   end;

initialization

finalization

end.
```

die „OnClick-Methode“

hier ist der geeignete Platz zur Initialisierung von Hilfsgrößen vor dem Start des Programms

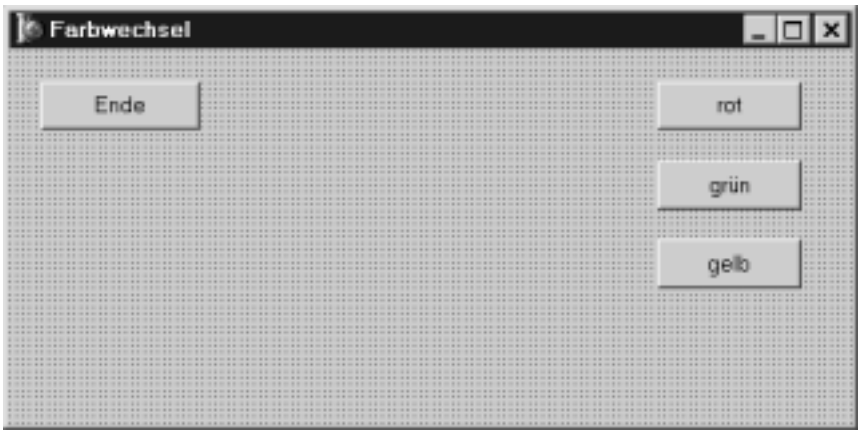
und hier zum „Aufräumen“ nach Beendigung des Programms

Die Anweisungen werden zeitlich in der folgenden Reihenfolge ausgeführt:

1. Bei der Initialisierung der Unit, die das Hauptfenster *Form1* bereitstellt, wird als erstes der *Initialization*-Abschnitt ganz unten ausgeführt. Dieser Abschnitt ist in der automatisch erzeugten Vorlage nicht enthalten und muss „per Hand“ eingefügt werden. In diesem Abschnitt sollten Anweisungen stehen, die sich nicht auf die sichtbaren Komponenten des Fensters beziehen, weil dieses zu diesem Zeitpunkt noch gar nicht existiert.
2. Danach werden die ersten beiden Anweisungen des Hauptprogramms ausgeführt, mit denen das Hauptfenster erzeugt wird. In einer Methode des Fensters, die dem *OnCreate*-Ereignis zugeordnet ist (z. B. *tForm1.FormCreate*), können deshalb Anweisungen stehen, die sich auf Fensterelemente beziehen; z. B. kann die Farbe des Fensters oder die Aufschrift eines seiner Buttons im laufenden Programm gesetzt werden, wenn das noch nicht mit dem Objektinspektor beim Entwurf der Oberfläche geschehen ist.
3. Immer wenn das Fenster neu gezeichnet wird (z. B. auch nach seiner Erzeugung) wird eine Methode ausgeführt, die dem *OnPaint*-Ereignis zugeordnet ist (z. B. *tForm1. FormPaint*). Hier stehen also Anweisungen, die jedes Mal nach dem Neuzeichnen ausgeführt werden sollen.
4. Die weitere Reihenfolge ergibt sich daraus, wie Ereignisse vom Benutzer ausgelöst werden, und natürlich aus dem genauen Programmtext. In einer Schleife werden immer wieder Ereignisse registriert und an die Elemente des Windows-Systems weitergereicht. Das geschieht so lange, bis das Programm beendet wird.
5. Zuletzt werden die Anweisungen des *Finalization*-Abschnitts der Unit ausgeführt, die das Hauptfenster definiert. Hier können z. B. erzeugt Objekte wieder gelöscht werden, um deren Speicherplatz freizugeben.

2.3 Ein einfaches, aber vollständiges Beispielprogramm

Wir wollen mit einem Programm ein Fenster erzeugen, auf dem sich vier Buttons befinden: drei um die Farbe des Fensters zu ändern, und einer, um das Programm zu beenden.



Zu diesem Zweck erzeugen wir ein neues Delphiprojekt (oder starten Delphi neu). Danach gehen wir wie folgt vor:

1. Wir ändern im Objektinspektor den Namen des Fensters (und damit automatisch die Aufschrift der Fensterleiste) in *Farbwechsel*.
2. Wir klicken in der Komponentenleiste *Standard* das Button-Symbol an und erzeugen vier Buttons auf der Form, die automatisch mit *Button1 .. Button4* bezeichnet werden. Wir platzieren diese Knöpfe an den gewünschten Stellen.
3. Wir ändern die Namen der Buttons in *rot, gruen* und *gelb* bzw. *Ende* ab. (Man beachte, dass Umlaute in Namen nicht erlaubt sind!) Beim *gruen*-Button ändern wir deshalb die Caption-Eigenschaft auf *grün*.
4. Wir Doppelklicken auf den *Ende*-Button. Damit wird automatisch eine Methode für das Standardereignis *OnClick* erzeugt. Wir ergänzen den Programmcode um die Zeile *Halt* (Befehl: Programmabbruch).
5. Auf die gleiche Art erzeugen und vervollständigen wir Methoden für die Farbwahl bei den Farb-Buttons. Wir ergänzen die Methoden um die Zeilen *color := clRed* (bzw. *clGreen*, *clYellow*; die Farbkonstanten von Delphi beginnen immer mit „*cl*“).
6. Wir starten das Programm durch Druck auf den Startbutton (▷) oben in der Symbolleiste oder durch Drücken der Taste *F9* oder durch den Befehl *Start* im Startmenü. Dabei sollten die Dateien automatisch gesichert werden, wobei nach den Dateinamen gefragt wird. Wir wählen der Kürze halber die Namen *U2* für die Unit und *P2* für das Projekt.
7. Nach der erfolgreichen Übersetzung erscheint das Fenster so, wie wir es entworfen haben. Durch Drücken der entsprechenden Knöpfe können wir die Farbe des Fensters ändern.

Das Hauptprogramm ist immer gleich – bis auf die jeweils gewählten Namen für Unit und Projekt. Es braucht deshalb nicht abgedruckt zu werden. Die Unit des Hauptfensters lautet:

```
unit U2;

interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
     Forms, Dialogs, StdCtrls;

type TFarbwechsel = class(TForm)
       Ende : TButton;
       rot  : TButton;
       gruen: TButton;
       gelb : TButton;
```

```
        procedure EndeClick(Sender: TObject);
        procedure rotClick(Sender: TObject);
        procedure gruenClick(Sender: TObject);
        procedure gelbClick(Sender: TObject);
        end;

var Farbwechsel: TFarbwechsel;

implementation
{$R *.DFM}

procedure TFarbwechsel.EndeClick(Sender: TObject);
   begin halt end;

procedure TFarbwechsel.rotClick(Sender: TObject);
   begin color := clRed end;

procedure TFarbwechsel.gruenClick(Sender: TObject);
   begin color := clGreen end;

procedure TFarbwechsel.gelbClick(Sender: TObject);
   begin color := clYellow end;
```