Workshop Delphi 5 ISBN 3-8273-1665-0

3 Formulare und Komponenten

Delphi ist ein visuelles Programmiersystem, bei dem große Teile der Anwendung nicht programmiert, sondern über die Verwendung von Komponenten zusammengestellt werden. Diese Komponenten besitzen bereits eine Grundfunktionalität, so dass der Anwender sich z. B. nicht mehr um die Anzeige eines Textes auf dem Bildschirm oder das Handling der Eingabe kümmern muss.

3.1 Grundlagen

3.1.1 Formulare

Delphi als visuelles Programmiersystem nimmt dem Programmierer eine Menge Arbeit ab. Kern der Programmierung sind stets die Formulare, denen durch die unterschiedlichen Komponenten Leben eingehaucht wird. Delphi speichert jedes Formular in einer eigenen Datei mit der Endung dfm. Diese Datei enthält die Definitionen des Formulars. Der auszuführende Programmcode steht in der Unit des Formulars, einer Datei mit der Endung *.pas*. Das Hauptprogramm des Projekts, in dem die Initialisierung der Anwendung vorgenommen wird, wird in einer Datei mit der Endung *.dpr* gespeichert.

Das Hauptprogramm hat die Aufgabe, die Anwendung zu initialisieren. Auch werden die verschiedenen Formulare der Anwendung im Speicher erzeugt, so dass sie später leicht angezeigt werden können. Diese automatische Erzeugung der Formulare kann in den Projektoptionen abgeschaltet werden.

Das Hauptformular

Eine Ausnahme stellt das Hauptformular einer Anwendung dar, dessen Erzeugung Sie immer Delphi überlassen sollten. Dieses Formular, das Hauptfenster Ihrer Anwendung, können Sie ebenfalls frei festlegen. Standardmäßig nimmt Delphi das erste erzeugte Formular. Bei der Initialisierung der Anwendung ist das Hauptformular das einzige Formular, das nicht nur erzeugt, sondern auch gleich beim Programmstart angezeigt wird.

Am Hauptprogramm der Anwendung müssen Sie in der Regel keine Änderungen vornehmen. Die Arbeit des Programmierers beschränkt sich auf die Programmierung der Funktionalität innerhalb der Formulare.

Weitere Formulare

Sie können leicht über den Menüpunkt DATEI/NEUES FORMULAR neue Formulare zu Ihrer Anwendung hinzufügen. Standardmäßig übernimmt Delphi die Erzeugung des jeweiligen Formulars in Speicher, Sie müssen die Formulare später nur noch anzeigen. Dazu gibt es zwei Möglichkeiten:

ShowModal

Ein Formular kann entweder *modal* oder *nicht modal* angezeigt werden. Bei der modalen Anzeige bleibt das Programm stehen, solange das Formular angezeigt wird, der Fokus wird auf das angezeigte Formular übertragen. Erst wenn Sie dieses Formular geschlossen haben, wird die Anwendung weiter ausgeführt. Sie können ein Formular mit der Methode *ShowModal* modal anzeigen.

Show

Die andere Anzeigeform bringt das Formular zwar auf den Bildschirm, die Anwendung läuft aber weiter und Ihnen steht die komplette Funktionalität des Hauptfensters zur Verfügung. Diese Art der Anzeige wird als nicht-modale Anzeige bezeichnet, d.h. das Formular wird zwar auf dem Bildschirm angezeigt und seine Funktionen können ausgeführt werden, aber der Rest der Anwendung läuft weiter. Sicherlich kennen Sie dieses Verhalten bereits, ein Beispiel dafür ist die IDE von Delphi. Nicht-modale Formulare werden über die Methode *Show* des Formulars angezeigt.

Formulare erzeugen

Um Ressourcen zu sparen – jedes erzeugte Formular verbraucht natürlich Ressourcen – sollten Sie nicht häufig benutzte Formulare selbst erzeugen. Dazu müssen Sie Delphi zunächst mitteilen, dass das Formular nicht erzeugt, sondern nur verfügbar gemacht werden soll. Die entsprechende Einstellung finden Sie unter dem Menüpunkt PROJEKT/OPTIONEN.

3.1.2 Komponenten

Zusammen mit den Formularen bilden die Komponenten die Basis eines jeden Delphi-Programms. Eigentlich jedoch sind Formulare ebenfalls Komponenten, mit eigenen Eigenschaften und Ereignissen.

Das Konzept der Komponenten

Die Komponenten der Komponentenpalette stellen die unter Delphi verfügbaren Windows-Steuerelemente dar. Sie können also die Funktionalität eines Steuerelements in Ihr Programm einfügen, indem Sie einfach die entsprechende Komponente auf einem Formular platzieren. Dabei handelt es sich allerdings nur um die Funktionalität des Steuerelements selbst. So können Sie z.B. in ein Eingabefeld Daten eingeben oder enthaltene Daten auslesen. Was mit diesen Daten aber innerhalb des Programms geschieht, müssen Sie selbst programmieren. Lediglich die Darstellung und Anzeige des Steuerelements oder die Eingabe selbst erledigen die fertig implementierten Methoden der Komponente.

Fensterelemente

Komponenten sind Klassen, die von einer Basisklasse abgeleitet sind. (Falls Sie nicht wissen, was das bedeutet, es wird in Kapitel 10 genauer erklärt). In Delphi existieren verschiedene Basisklassen mit unterschiedlichen Möglichkeiten. So gibt es spezielle Klassen für die unsichtbaren Komponenten, weiterhin Klassen für die sichtbaren Steuerelemente und auch eine Klasse für so genannte Fensterelemente. Auf solche Fensterkomponenten (sie sind abgeleitet von der Klasse *TWinControl*) können Sie Komponenten ablegen. Diese Fensterelemente sind nicht auf die Formulare beschränkt, auch andere Komponenten werden, obwohl es sich dabei nicht um Fenster im eigentlichen Sinn handelt, von Windows wie Fenster behandelt. Unter anderem handelt es sich dabei um die Komponenten *TPanel* oder *TButton*. Das bedeutet unter anderem auch, dass diese

Elemente Ressourcen belegen, da Windows für sie ein Handle bereitstellen muss. Sie sollten im Interesse Ihrer eigenen Anwendung die Formulare nicht mit solchen Fensterelementen überfrachten.

Von *TWinControl* abgeleitete Komponenten haben noch eine weitere Eigenschaft, sie können nämlich andere Steuerelemente aufnehmen. Ebenso, wie Sie z.B. eine Schaltfläche auf ein Formular platzieren können, können Sie es auch auf ein anderes, von *TWinControl* abgeleitetes Steuerelement platzieren. Zur Laufzeit müssen Sie dazu nur die Eigenschaft *Parent* des Steuerelements ändern, in der das übergeordnete Fenster angegeben ist.

Visuelle Komponenten

Alle Komponenten, die zur Laufzeit sichtbar sein können, bezeichnet man als visuelle Komponenten. Einige davon beinhalten relevante Funktionen für das Programm (z.B. Eingabefelder), einige dienen der Abgrenzung vom Rest des Fensters, z.B. die Komponente *TGroupBox*.

Unsichtbare Komponenten

Delphi enthält auch einige Komponenten, die zwar Funktionalität beinhalten, aber dennoch zur Laufzeit nicht sichtbar sind. Ein gutes Beispiel für solche Komponenten sind z.B. diejenigen zum Datenzugriff oder für die DDE-Funktionalität.

3.1.3 Eigenschaften

Komponenten in Delphi besitzen so genannte Eigenschaften, mit denen ihr Verhalten zur Laufzeit und auch das Aussehen beeinflusst werden können. Viele dieser Eigenschaften können bereits zur Entwurfszeit über den Objektinspektor eingestellt werden, die Veränderung wird in den meisten Fällen auch direkt sichtbar. In diesem Buch können die umfangreichen Eigenschaften selbstverständlich nicht aufgezählt werden. Die meisten sind aber allein durch ihren Namen schon selbsterklärend, für die anderen bemühen Sie bitte die Online-Hilfe.

Andere Eigenschaften sind nur zur Laufzeit verfügbar. Um eine Übersicht über alle Eigenschaften einer Komponente zu erlangen, bemühen Sie am besten die Online-Hilfe, da die Anzahl der Komponenten (und auch ihrer Eigenschaften) mit jeder Version von Delphi zunimmt. Die Leistungsfähigkeit erhöht sich dadurch natürlich, aber die Übersicht leidet.

Den meisten Eigenschaften, ob sie bereits im Objektinspektor zur Verfügung stehen oder erst zur Laufzeit, können Sie jederzeit neue Werte zuordnen oder Werte auslesen, d.h. sie verhalten sich wie Variablen. Bei manchen Eigenschaften ist es aber nur möglich, die Werte auszulesen, diese sind in der Online-Hilfe als *Read-Only* gekennzeichnet.

Die Eigenschaft Parent

Die Eigenschaft *Parent* bezeichnet die übergeordnete Komponente eines Steuerelements. Dabei handelt es sich immer um eine Komponente, die vom Typ *TWinControl* abgeleitet ist. Alle anderen Eigenschaften, die mit *Parent* beginnen, beziehen sich immer auf diese übergeordnete Komponente (*ParentFont*,

ParentColor ...). Sie können also leicht die Farbe aller Steuerelemente und Komponenten ändern, indem Sie einfach die Farbe der übergeordneten Komponente ändern.

Die Eigenschaft Tag

Eine Eigenschaft besitzt eine Besonderheit. Es handelt sich um die Eigenschaft *Tag*, die nur dazu da ist, von Ihnen benutzt zu werden. Jede Komponente in Delphi, ob nun standardmäßig von Borland oder von einem Fremdhersteller, besitzt die Eigenschaft *Tag*. *Tag* ist ein Integerwert, der intern von Delphi nicht benutzt wird (damit ist gemeint: nicht benutzt wird und nie benutzt werden wird). Sie können diese Eigenschaft daher für eigene Zwecke nutzen.

Caption und Text

Weitere häufig benutzte Eigenschaften sind die Eigenschaften *Caption* und *Text*. *Caption* stellt die Beschriftung einer Komponente dar, z. B. bei den Komponenten *TPanel* oder *TGroupbox*, *Text* entspricht der Eingabe in eine Komponente, z. B. bei *TEdit*. Beide Eigenschaften sind vom Datentyp *String*. Zwar werden Strings erst später behandelt, aber zur Anzeige von Daten in einer *TEdit*-Komponente bzw. zum Auslesen einer Eingabe und der Weiterverwendung benötigen Sie einige Umwandlungsfunktionen. Delphis Konzept behandelt die Datentypen sehr streng, so dass Werte nicht automatisch von einem Datentyp in einen anderen konvertiert werden, Sie müssen das selbst tun. Delphi stellt aber eine umfassende Auswahl von Umwandlungsroutinen zur Verfügung. Tabelle 3.1 zeigt die am häufigsten benötigten Umwandlungsfunktionen für Strings.

Tab. 3.1: Umwandlungsfunktionen für Strings

Umwandlungsfunktion	Auswirkung
s := IntToStr(i);	Umwandlung von Integer nach String
i := StrToInt(s);	Umwandlung von String nach Integer
r := StrToFloat(s);	Umwandlung von String nach Gleitkomma
s := FloatToStr(r);	Umwandlung von Gleitkomma nach String
d := StrToDate(s);	Umwandlung von String nach Datum
s := DateToStr(d);	Umwandlung von Datum nach String
d := StrToTime(s);	Umwandlung von String nach Zeit
s := TimeToStr(d);	Umwandlung von Zeit nach String

Natürlich sind diese Funktionen nicht auf die Eigenschaften *Text* oder *Caption* beschränkt, sondern gelten für alle Variablen vom Typ *String*.

3.1.4 Ereignisse und Ereignisbehandlung

Windows ist ein ereignisorientiertes Betriebssystem, Delphi eine ereignisorientierte Programmiersprache. Alle Aktionen, die ein Benutzer vornimmt, sind im Prinzip Ereignisse, die an die aktuelle Anwendung weitergeleitet werden. Die Anwendung kümmert sich dann darum, diesen Ereignissen eine oder mehrere Aktionen zuzuweisen.

Ereignisbehandlung

In Delphi besitzen alle Komponenten eine Liste von Ereignissen, auf die sie reagieren können. Sie können für alle diese Ereignisse so genannte Ereignisbehandlungsroutinen schreiben, um darauf zu reagieren. Das ist auch die Standard-Vorgehensweise bei der Programmierung der Funktionalität einer Applikation, nämlich die Reaktion auf die Handlungen des Anwenders.

Windows-Ereignisse

Ereignisse müssen aber nicht zwangsläufig vom Anwender ausgehen. Auch Windows sendet verschiedene Ereignisse, z.B. wenn ein Programm aktiviert wird oder wenn es geschlossen werden soll. Auch diese Ereignisse können Sie abfangen und darauf reagieren. Aber es geht auch anders herum, Sie können nämlich aus Ihrem Programm heraus ein Ereignis an Windows senden, worauf dann eine entsprechende Reaktion seitens des Betriebssystems folgt.

Der Parameter Sender

Jedem Ereignis, das die Komponenten bereitstellen, wird ein Parameter namens *Sender* übergeben. Dieser Parameter enthält immer die Komponente, die das Ereignis ausgelöst hat. Der Parameter *Sender* ist vom Typ *TObject*, dem Urvater aller Klassen und Komponenten in Delphi. Daher kann dieser Parameter auch jede Komponente aufnehmen (zu diesem Prinzip namens Polymorphie mehr im Kapitel 10). Über die Möglichkeit der Typumwandlung ist es möglich, zu erfahren, welche Komponente das Ereignis ausgelöst hat, und auch auf deren Eigenschaften zuzugreifen.

Der Is-Operator

Über den *Is*-Operator können Sie kontrollieren, von welchem Typ der Sender ist. Sie können also kontrollieren, welche Komponente die Ereignisbehandlungsroutine ausgelöst hat. Da gleiche Ereignisbehandlungsroutinen mehreren Komponenten zugeordnet werden können, ist diese Auswertung oft nützlich. Somit können z.B. mehrere Schaltflächen eine Ereignisbehandlungsroutine *OnClick* nutzen, in der dann unterschiedliche Operationen durchgeführt werden, abhängig davon, welche Schaltfläche das Ereignis ausgelöst hat.

Der As-Operator

Der *As*-Operator ist ein Umwandlungsoperator, mit dem Sie über den Parameter *Sender* auf das entsprechende Element zugreifen können. So können Sie z.B. auf der *Sender*-Parameter wie auf eine Komponente vom Typ *TButton* oder *TSpeedButton* zugreifen. Das folgende Beispiel zeigt eine solche Auswertung unter Einbeziehung des *Is*-Operators:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  IF (Sender IS TSpeedButton) THEN
    TSpeedButton(Sender).Caption := 'Ok';  //1.
Möglichkeit
  ELSE
    (Sender AS TButton).Caption := 'Ok';   //2.
Möglichkeit
END;
```

Wenn ohnehin sichergestellt ist, dass das entsprechende Ereignis nur von einer bestimmten Komponentenart aufgerufen werden kann, kann die Abfrage über den *Is*-Operator entfallen:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  WITH (Sender AS TButton) DO
  BEGIN;
    Caption := 'Wurde geklickt';
    Tag     := 1;
    Width   := 150;
  END;
END;
```

Die *With*-Anweisung und auch die *If*-Anweisung aus den Beispielen werden später noch genauer erklärt.

3.1.5 Methoden

Delphis Komponenten besitzen Methoden, die gewisse Aktionen ausführen. Diese Methoden sind Prozeduren und Funktionen, die in die jeweilige Komponentenklasse implementiert sind und die Grundfunktionalität der Klasse beinhalten. Viele dieser Methoden sind veröffentlicht, so dass Sie sie zur Laufzeit aufrufen können. Eine Liste der Methoden einer Komponente finden Sie ebenfalls in der Online-Hilfe, eine Auflistung aller Methoden ist aufgrund des Umfangs nicht möglich.

Zu den Methoden eines Eingabefelds gehören unter anderem Methoden zur Verwendung der Zwischenablage, durch die der Inhalt des Eingabefeldes in die Zwischenablage kopiert oder daraus eingefügt wird. Andere Komponenten besitzen andere sinnvolle Methoden. In jedem Fall sollten Sie bei der Arbeit mit Delphi normalerweise darauf achten, einen wichtigen Grundsatz zu beachten: nie etwas zu programmieren, was es ohnehin schon gibt. Nehmen Sie sich lieber die Zeit und schauen Sie ab und zu mal durch, was eine Komponente denn von Haus aus bereits kann, oft sparen Sie sich damit eine Menge Arbeit. Ein kleiner Nebeneffekt ist, dass Sie auch einen besseren Überblick bekommen und die Online-Hilfe nicht mehr ganz so oft benötigen.

Eigene Methoden

Sie können selbst geschriebene Prozeduren bzw. Funktionen ebenfalls zu einem Bestandteil des Formulars machen. Delphi stellt im Rumpf der Formulardefinition Platz für eigene Prozeduren und Funktionen zur Verfügung. Der Grund dafür ist, dass eines der Konzepte für die objektorientierte Programmierung lautet, dass Prozeduren, die sich auf ein bestimmtes Formular beziehen, auch in diesem programmiert sein sollen.

Private und Public

Sie können diese Prozeduren entweder im *Private*-Teil des Formulars oder im *Public*-Teil deklarieren. Ist eine Prozedur im *Private*-Teil deklariert, können andere Units nicht darauf zugreifen. Ist sie im *Public*-Teil deklariert, ist der Zugriff von außerhalb der Unit möglich. Woran Sie allerdings bei der Implementation, also der Programmierung der eigentlichen Funktionalität, denken müssen, ist, den Prozedurkopf zu qualifizieren. Hierzu am besten ein Beispiel:

Deklaration

```
TYPE
  TForm1 = CLASS(TForm)
  PRIVATE
    { Private-Deklarationen }
    PROCEDURE Test(VAR Value: integer);  //Eigene Prozedur
  PUBLIC
    { Public-Deklarationen }
  END;
```

Implementation

```
PROCEDURE TForm1.Test(var Value: integer);
BEGIN
   Value := 5;
END;
```

Sie bemerken, dass bei der Implementation der Prozedur der Bezeichner *TForm1* mit angegeben werden muss.

Klassenvervollständigung

Delphi hilft Ihnen hierbei mit einer der umfangreichen Hilfsfunktionen, der Klassenvervollständigung. Bei dieser Art der Deklaration einer Prozedur haben Sie diese zu einem Bestandteil der Klasse *TForm1* gemacht. Über die Tastenkombination Strg + Shift + C können Sie Delphi nun anweisen, diese Klasse zu vervollständigen. Delphi wird dann für jede Prozedur der Klasse, für die noch kein Rumpf programmiert ist, einen solchen erstellen. D.h. Sie müssen lediglich die Prozedur als Bestandteil des Formulars deklarieren und dann die Klassenvervollständigung benutzen. Danach müssen Sie lediglich noch die Funktionalität programmieren.

3.2 Drag&Drop

Im Deutschen wird Drag&Drop auch als »Ziehen und Ablegen« bezeichnet. Diese unter Windows weit verbreitete Technik kann auch mit Delphis Komponenten sehr leicht in eigene Projekte implementiert werden. Viele Komponenten in Delphi ermöglichen die Verwendung von Drag&Drop.

DragMode

Über die Eigenschaft *DragMode* legen Sie fest, wie die Drag&Drop-Operation gestartet werden soll. Wenn *DragMode* auf *dmAutomatic* steht, wird die Operation sofort gestartet, andernfalls müssen Sie die Methode *BeginDrag* selbst aufrufen. Am sinnvollsten geschieht das im Ereignis *OnMouseDown* des Steuerelements. Die manuelle Vorgehensweise hat den Vorteil, dass der Mauszeiger sich nicht bei jedem Klick auf das Steuerelement ändert, sondern erst nachdem die Maus mit gedrückter linker Maustaste ein Stück gezogen wird.

OnDragOver

Das Ereignis *OnDragOver* tritt auf, wenn das Element über die Zielkomponente gezogen wird. Über den Parameter *Source* können Sie abfragen, von wo das gezogene Element stammt, und dann entsprechend den Wert des Parameters *Accept* auf *true* oder *false* setzen. *Source* ist vom Typ *TObject* und kann wie der Parameter *Sender* behandelt werden.

OnDragDrop In diesem Ereignis programmieren Sie den Code, der beim Ablegen des Elements (also beim Loslassen der Maustaste) ausgeführt werden soll. Auch dieses Ereignis verfügt über den Parameter *Source*.

OnEndDrag Das Ereignis *OnEndDrag* für die Quellkomponente wird nach dem Ablegen auf dem Zielelement ausgeführt. So können Sie z. B. beim Verschieben eines Eintrags von einer Listbox in eine andere in diesem Ereignis das gezogene Element aus der Listbox entfernen. *OnEndDrag* verfügt über einen Parameter *Target*, der das Zielobjekt beinhaltet.

3.3 Schutzblöcke

Bei der Arbeit mit den Komponenten werden oftmals Umwandlungen vorgenommen. So kann ein Eingabefeld z. B. nur Text anzeigen, der im Datenformat *String* vorliegt. Um nun einen Zahlenwert in einen String zu verwandeln, wird eine der Umwandlungsfunktionen aufgerufen. Dabei gibt es auch kein Problem.

Exceptions Soll jedoch ein String in einen Zahlenwert umgewandelt werden, so kann es zu einem Laufzeitfehler kommen. Hat der Anwender z. B. eine Buchstabenfolge eingegeben und das Programm versucht, diese in einen Zahlenwert umzuwandeln, kommt es zu einem solchen Fehler. Diese Exceptions – es gibt eine Menge davon – können durch so genannte Schutzblöcke abgefangen werden. Exceptions existieren in Delphi in allen Bereichen, in denen ein Fehler auftreten kann. Über die Schutzblöcke können Sie diese Fehler, gleich welcher Art, abfangen und auch eine eigene Fehlerbehandlungsroutine dafür schreiben. Da wir in den folgenden Übungen von diesen Schutzblöcken Gebrauch machen wollen, werden sie hier erklärt.

Try...Except Der *Try..Except*-Schutzblock versucht, die Anweisungen hinter Try auszuführen. Tritt eine Exception auf, wird der *Try*-Block verlassen und die Anweisungen hinter *Except* werden ausgeführt. Danach wird die Routine verlassen. Sie können also im *Except*-Block Prozeduren zur Fehlerbehandlung aufrufen, es ist auch möglich, festzustellen, welche Exception aufgetreten ist, und dementsprechend zu verzweigen.

Try...Finally Der *Try...Finally*-Schutzblock arbeitet prinzipiell genauso, allerdings werden die Anweisungen, die hinter *Finally* stehen, in jedem Fall ausgeführt – ob eine Exception auftritt oder nicht. In der Regel wird diese Art eines Schutzblocks dazu verwendet, belegte Ressourcen freizugeben, ob nun eine Fehler auftritt oder nicht. Die so genannten Speicherleichen werden so vermieden.

Schutzblöcke verschachteln Schutzblöcke können ineinander verschachtelt werden. Somit können auch *Try...Except*- und *Try...Finally*-Schutzblöcke miteinander gemischt werden. Teilweise kann dieses Vorgehen sinnvoll sein, in den meisten Fällen wird jedoch eine der Konstruktionen ausreichen.

Fehlerbehandlung Innerhalb des *Except*-Abschnitts könne Sie die Exception auch genauer bestimmen. In dem beschriebenen Fall, also wenn ein alphanummerisches Zeichen in

eine Zahl umgewandelt werden soll, tritt die Exception *EConvertError* auf. Für die Behandlung einer exakt definierten Exception stellt Delphi die Konstruktion *On (Exceptionname) do* zur Verfügung. Mit dieser Konstruktion können Sie kontrollieren, ob die von Ihnen erwartete Exception aufgetreten ist, und falls ja in eine eigene Fehlerbehandlungsroutine verzweigen. Ein Beispiel, das in unserem Fall Anwendung finden könnte:

```
PROCEDURE HandleOtherErrors;
BEGIN;
   ShowMessage('Ein Fehler ist aufgetreten');
END;

PROCEDURE HandleConvertError;
BEGIN;
   ShowMessage('Konvertierungsfehler');
END;

PROCEDURE TForm1.Button1Click(Sender);
BEGIN;
   TRY
      MyValue := StrToInt(Edit1.Text);
   EXCEPT
      ON EConvertError DO
         HandleConvertError
      ELSE
         HandleOtherErrors;
   END;
   ...
END;
```

In diesem Fall wird unsere eigene Fehlerbehandlung aufgerufen, wenn die Exception *EConvertError* auftritt. Falls irgendeine andere Exception auftritt, wird die Standard-Routine *HandleOtherErrors* aufgerufen, die ebenfalls von uns definiert ist.

Der Fehler wird durch den Aufruf bzw. durch das Ausführen des *Except*-Blocks gelöscht. Falls Sie also keine Meldung an den Anwender Ihres Programms weitergeben möchten, berauchen Sie den *Except*-Block nur leer zu lassen. Im Falle eines Fehlers wird die Prozedur bzw. Funktion, in der er auftrat, ohnehin verlassen.

Exceptions erzeugen

Sie können eine Exception auch selbst erzeugen. Dazu dient das Schlüsselwort *Raise*, gefolgt vom Constructor der Exception, die Sie aufrufen möchten. So können Sie z. B. eine Exception vom Typ *EMathError* folgendermaßen aufrufen:

```
RAISE EMathError.Create('Ungültiger Vorgang');
```

Innerhalb eines *Except*-Abschnitts, wenn der Fehler bereits behandelt ist und Sie die aufgetretene Exception erneut aufrufen möchten (z. B. in verschachtel-

ten Schutzblöcken), können Sie das reservierte Wort *Raise* auch alleine verwenden. Die behandelte Exception wird dann nochmals erzeugt.

3.4 Übungen

Übung 1

Erstellen Sie eine neue Anwendung. Auf das Formular platzieren Sie eine Komponente vom Typ *TMemo*. Nun stellen Sie die Werte für die Breite und Höhe des Memofeldes ein, es soll 500 Pixel breit und 300 Pixel hoch sein.

Zur Laufzeit soll das Memofeld immer in der Mitte des Formulars angeordnet sein, gleich, welche Größe der Anwender dem Formular gibt. Die Größe des Memofeldes soll sich dabei nicht ändern. Allerdings soll das Formular immer mindestens 600 Pixel breit und 400 Pixel hoch sein.

Suchen Sie die entsprechenden Ereignisse und schreiben Sie eine Ereignisbehandlungsroutine, die die verlangte Funktion sicherstellt. Denken Sie daran, dass Eigenschaften sich zur Laufzeit wie Variablen verhalten, d.h. Sie können Werte auslesen und neue Werte zuweisen.

Übung 2

Für diese Übung können Sie entweder eine neue Anwendung erstellen oder aber die bestehende Anwendung nutzen. In diesem Fall wird das Memofeld nicht mehr benötigt.

Platzieren Sie zwei Eingabefelder vom Typ *TEdit* auf dem Formular. Diese Felder sollen als Eingabemöglichkeit für einen Namen und einen Vornamen dienen. Platzieren Sie auch zwei Komponenten vom Typ *TLabel* auf dem Formular, die der Anzeige dienen sollen. Die Labels müssen nebeneinander angeordnet werden.

Wenn in die Eingabefelder etwas eingegeben wird, sollen die zwei Label-Komponenten den Namen und den Vornamen beinhalten, also jeweils den Text eines Eingabefeldes. Es soll aber immer der Vorname direkt hinter dem Namen stehen, unabhängig von der Länger des Namens. Finden Sie heraus, in welcher Ereignisbehandlungsroutine Sie die Funktionalität programmieren müssen, und programmieren Sie die entsprechende Ereignisbehandlungsroutine.

Übung 3

Üblicherweise wird unter Windows mit Hilfe der Tabulatortaste zwischen den Eingabefeldern gewechselt (bzw. zwischen all den Steuerelementen, die den Eingabefokus erhalten können, z.B. auch Buttons). Die Aufgabe besteht darin, dem Benutzer zu ermöglichen, mit der Eingabetaste zum nächsten Feld zu wechseln.

Vorweg sei gesagt, dass eine allgemeingültige Lösung den Einsatz des Windows API voraussetzt. Sie sollen lediglich einen Weg finden, wie der Anwender von einem Feld ins nächste mit der Return-Taste wechseln kann. Wenn Sie diese Möglichkeit für ein Eingabefeld programmieren, genügt das. In den Lösungen wird auch eine allgemeingültige Form vorgestellt.

Die Eingabetaste hat den Ascii-Wert 13. Finden Sie heraus, welche Methode einem Steuerelement den Fokus gibt und in welcher Ereignisbehandlungsroutine Sie die Funktion programmieren müssen.

Übung 4

Starten Sie ein neues Projekt. Ziel soll es sein, ein Formular zu erstellen, das über einen Mausklick »aufgeklappt« werden kann, so dass vorher versteckte Komponente sichtbar werden.

Sie benötigen lediglich die zwei Schaltflächen zum Aufklappen und Zuklappen auf dem Formular, wobei der Button zum Zuklappen erst dann sichtbar werden soll, wenn das Formular aufgeklappt wird. Beim Programmstart soll das Formular in der zugeklappten Version angezeigt werden.

Übung 5

Platzieren Sie fünf Komponenten vom Typ *TButton* und eine Komponente vom Typ *TPanel* auf einem Formular. Beschriften Sie die Button-Komponenten unterschiedlich. Die Aufgabe besteht darin, bei einem Mausklick auf einen Button dessen Beschriftung im Panel anzuzeigen.

Übung 6

Schreiben Sie Übung 5 um. Benutzen Sie nur eine einzige Ereignisbehandlungsroutine, die Sie allen Buttons zuweisen können. Die Funktionalität soll exakt die gleiche sein wie in Übung 5, d. h. der Anwender soll keinen Unterschied bemerken.

Übung 7

Platzieren Sie eine ComboBox auf dem Formular und setzen Sie dessen Eigenschaft Style auf *csDropDownList*. Fügen Sie einige Einträge hinzu. Über die Eigenschaft *ItemIndex* der Komponente *TComboBox* können Sie erfahren, welches Element gerade angezeigt wird. Welchen Wert hat *ItemIndex*, wenn kein Element angezeigt wird? Der wie vielte Eintrag wird angezeigt, wenn *ItemIndex* den Wert 3 hat?

Übung 8

Platzieren Sie eine Komponente vom Typ *TPageControl* auf einem Formular. Fügen Sie drei Seiten hinzu (mit der rechten Maustaste auf die Komponente

klicken und NEUE SEITE auswählen). Fügen Sie dem Formular noch drei *TButton*-Komponenten hinzu.

Die Aufgabe besteht darin, nicht nur über die Tabs der *Pagecontrol*-Komponente die Seite zu wechseln, sondern auch über die Schaltflächen. Finden Sie heraus, welche Eigenschaft Sie dazu benutzen müssen, und programmieren Sie die Funktionalität. Es gibt zwei mögliche Lösungen, die erste ist eigentlich einfach herauszufinden, die andere ist zwar auch einfach zu programmieren, aber nicht so leicht herauszufinden.

Übung 9

Delphi stellt zur Laufzeit eine Klasse namens *TScreen* zur Verfügung, die einige Eigenschaften des Systems zum Lesen bereitstellt. Benutzen Sie *TScreen*, um ein Formular zur Laufzeit unabhängig von seiner Größe auf der Mitte des Bildschirms zu platzieren. Suchen Sie auch noch eine andere Möglichkeit, dieses Verhalten zu erreichen.

Übung 10

Erstellen Sie ein Programm mit zwei Formularen. Vom Hauptfenster aus soll das zweite Formular angezeigt werden können. Zeigen Sie das Formular nicht nur an, sondern erzeugen Sie es auch.

Übung 11

Basierend auf Übung 10 soll nun das zweite Formular ein Eingabefeld und einen Button enthalten. Wenn im Eingabefeld ein Eintrag vorgenommen wird, soll dieser nach einem Klick auf den Button des zweiten Formulars in der Titelleiste des ersten Formulars erscheinen.

Übung 12

Ein Spaßprogramm. Erstellen Sie ein Programm, das ein Meldungsfenster erzeugt. Das Meldungsfenster soll einen Button enthalten. Wenn der Anwender versucht, den Button anzuklicken, soll dieser der Maus »davonlaufen«, also immer Koordinaten relativ zur Maus erhalten.

Natürlich soll es möglich sein, das Fenster zu verlassen, nämlich über die Eingabetaste.

Übung 13

Platzieren Sie zwei Komponenten vom Typ *TListBox* auf einem Formular. Die erste Listbox soll bereits Einträge enthalten (Sie können diese zur Entwurfszeit hinzufügen), die zweite Listbox wollen wir über Drag&Drop füllen.

Die Vorgehensweise beim eigentlichen Drag&Drop-Vorgang sollte kein Problem darstellen. Den markierten Eintrag der Listbox erreichen Sie über *ListBox1.Item-*

Index. Aus der ersten Listbox soll dabei nichts gelöscht, aber immer zur zweiten Listbox hinzugefügt werden. Das Hinzufügen können Sie mit dem Befehl *Listbox2.Items.Add* erledigen.

Übung 14

Ich habe diese Übung als schwer bezeichnet, so unbedingt schwer ist sie aber eigentlich gar nicht.

Platzieren Sie zwei *TPanel*-Komponenten auf einem Formular und eine Schaltfläche vom Typ *TButton* auf einem der Panels. Das Ziel dieser Übung ist, die Schaltfläche verschieben zu können, und zwar nicht nur auf dem Panel selbst, sondern auch auf das zweite Panel oder auf das Formular. Die Schaltfläche soll dabei dort zu liegen kommen, wo die Maus losgelassen wird. Die Schwierigkeit bei dieser Aufgabe soll in der Ausführung liegen, Sie dürfen nämlich nur drei Ereignisbehandlungsroutinen verwenden. Diese müssen Sie so programmieren, dass sie für alle Komponenten gültig sind, auf denen der Button zu liegen kommen kann, so das Sie anderen Komponenten einfach durch Zuweisung der Ereignisbehandlungsroutinen ermöglichen können, die Schaltfläche aufzunehmen.

Der Sinn dahinter liegt darin, das Programm beliebig erweitern zu können, ohne eine weitere Routine schreiben zu müssen. Die Schaltfläche soll aber immer die gleiche sein. Dabei soll sie auch noch funktionieren, falls man eine Funktionalität für ihr *OnClick*-Ereignis programmiert.

Verwenden Sie für diese Übung für *DragMode* den Wert *dmManual*.

Übung 15

Beantworten Sie folgende Fragen zu den Komponenten von Delphi.

- Kann man ein Eingabefeld auf einer *TPanel*-Komponente platzieren?
- Was ist der Unterschied zwischen den Komponenten *TButton* und *TBitBtn*?
- Was unterscheidet die Komponenten *TEdit* und *TMaskEdit*?
- Welche Eigenschaft hat jede Komponente in Delphi?
- In welchem Ereignis sollte eine Abfrage programmiert werden, die den Anwender entscheiden lässt, ob er eine Anwendung wirklich beenden will?
- Wozu dient die Eigenschaft *KeyPreview* eines Formulars?
- Wozu dient die Eigenschaft *PasswordChar* einer Komponente vom Typ *TEdit*?
- Wie groß ist die Standard-Schrittweite der Komponente *TProgressbar*?

- Mit welcher Methode kann ein Formular (nicht das Hauptformular) angezeigt werden? Es existieren zwei Methoden mit unterschiedlicher Auswirkung dafür.

3.5 Tipps

Tipps zu Übung 1

- Überlegen Sie, wie man aus den vorhandenen Eigenschaften den linken oberen Punkt des Memofeldes ermitteln kann.
- Den oberen Punkt können Sie über (Höhe des Formulars-Höhe des Memofelds) dividiert durch 2 ermitteln. Den linken Punkt entsprechend durch die gleiche Rechnung mit der Breite.
- Sehen Sie sich die Eigenschaften des Formulars an, um herauszufinden, wie Sie die Größe einschränken können.
- Öffnen Sie die Untereigenschaften zur Eigenschaft *Constraints*.

Tipps zu Übung 2

- Die zweite *TLabel*-Komponente muss bereits während der Eingabe in ihrer Position relativ zur ersten *TLabel*-Komponente verändert werden.
- Überlegen Sie, wie Sie die notwendige linke Position der zweiten *TLabel*-Komponente ermitteln können, und weisen Sie sie dann während der Eingabe zu.
- Benutzen Sie zur Zuweisung das Ereignis *OnChange* des Eingabefelds.
- Sie müssen das Ereignis zwar nur einmal programmieren, können es aber beiden Eingabefeldern als Ereignis zuweisen.

Tipps zu Übung 3

- Suchen Sie sich eine Ereignisbehandlungsroutine, die mit einem Tastendruck zu tun hat.
- Kontrollieren Sie, ob die Taste mit dem ASCII-Wert 13 gedrückt wurde. Wenn ja, wechseln Sie den Fokus auf das jeweils andere Eingabefeld.
- Benutzen Sie für beide Eingabefelder das Ereignis *OnKeyPress* und wechseln Sie darin zum entsprechend anderen Eingabefeld.

Tipps zu Übung 4

- Erstellen Sie das Formular in aufgeklappter Form, platzieren Sie die Schaltflächen und verkleinern Sie das Formular beim Programmstart über ein Ereignis des Formulars.

- In den Ereignisbehandlungsroutinen der Schaltflächen müssen Sie lediglich die entsprechende Größe des Formulars ändern, damit andere Steuerelemente sichtbar werden.

Tipps zu Übung 14

- Die Ereignisbehandlungsroutinen, die zu programmieren sind, sind *OnMouseDown* des Buttons zum Starten der Drag&Drop-Operation, eine der Routinen *OnDragOver* und eine der Routinen *OnDragDrop*.
- Die Routine *OnDragOver* allgemein zu halten, ist einfach. Kontrollieren Sie nur, ob das Quellobjekt die Schaltfläche ist, wenn ja, soll die Drag&Drop-Operation akzeptiert werden.
- Der Sender der Routine *OnDragDrop* übergibt die Komponente, auf der die Schaltfläche abgelegt werden soll.
- Überlegen Sie sich, was alle Komponenten gemeinsam haben, auf denen eine andere Komponente oder ein Steuerelement platziert werden kann.
- Versuchen Sie ein Typumwandlung in *TWinControl*.

3.6 Lösungen

Lösung 1

Da die Größe des Memofeldes gleich bleiben soll, müssen wir uns zunächst Gedanken machen, wie wir das Memofeld in der Mitte des Formulars platzieren. Schauen wir uns die Eigenschaften eines Formulars an, so gibt es vier Eigenschaften, die wir dazu benutzen könnten, nämlich *Height*, *Width*, *ClientHeight* und *ClientWidth*. Beide können benutzt werden, für das Beispiel habe ich mich für *ClientHeight* und *ClientWidth* entschieden, da diese Eigenschaften Höhe und Breite des Arbeitsbereiches unseres Formulars darstellen.

Über die Breite und Höhe können wird zusammen mit der Breite unseres Memofeldes die linke obere Ecke bestimmen, an der es platziert werden muss. Die Formel dafür ist

```
Oben  := (ClientHeight-Memo1.height) DIV 2;
Links := (ClientWidth-Memo1.Width) DIV 2;
```

Jetzt müssen wir uns nur noch überlegen, wann wir die Berechnung durchführen müssen. Nämlich immer dann, wenn das Formular in der Größe verändert wird. Das entsprechende Ereignis ist *OnResize* des Formulars. Die berechneten Werte müssen wir in der entsprechenden Ereignisbehandlungsroutine nur noch den Eigenschaften *Top* und *Left* des Memofeldes zuweisen. Die gesamte Ereignisbehandlungsroutine sehen Sie hier:

```
PROCEDURE TForm1.FormResize(Sender: TObject);
BEGIN
  Memo1.Top  := (ClientHeight-Memo1.Height) DIV 2;
  Memo1.Left := (ClientWidth-Memo1.Width) DIV 2;
END;
```

Das zweite Problem ist das Problem der Größenbeschränkung. Dieses ist aber noch einfacher zu lösen. Alles, was Sie tun müssen, ist die Untereigenschaften der Eigenschaft *Constraints* des Formulars zu öffnen. Dort finden Sie die Eigenschaften *MinWidth* und *MinHeight*. Setzen Sie sie auf die verlangten Werte. Fertig, Aufgabe erfüllt.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_1.

Lösung 2

Zunächst müssen wir aufgrund der Aufgabenstellung überlegen, welches Ereignis dafür verwendet werden kann, die Daten sofort bei einer Eingabe zu übernehmen. Das entsprechende Ereignis ist schnell gefunden, es handelt sich um das Ereignis *OnChange* der Eingabefelder. Wir benötigen nur eine Ereignisbehandlungsroutine, wir können sie nach der Programmierung auch dem entsprechenden Ereignis des zweiten Eingabefelds zuweisen.

Die Zuweisung der eingegebenen Inhalte ist trivial:

```
Label1.Caption := Edit1.Text;
Label2.Caption := Edit2.Text;
```

Nun müssen wir im Anschluss an die Zuweisung noch dafür sorgen, dass beide Labels direkt hintereinander stehen, nun, sagen wir in einem Abstand von 5 Punkten (damit es besser aussieht). Die Komponente *TLabel* besitzt eine Eigenschaft namens *AutoSize*, die für unsere Übung auf *true* stehen muss. Damit wird bei einer Zuweisung die Größe des Labels automatisch angepasst. Um nun das zweite Label dahinter zu platzieren, müssen wir nur das rechte Ende des ersten Labels berechnen. Dabei handelt es sich um den linken Punkt des Labels plus seiner Breite (plus unsere 5 Punkte für das bessere Erscheinungsbild):

```
Label2.Left := Label1.Left+Label1.Width+5;
```

Die komplette Ereignisbehandlungsroutine sehen Sie hier:

```
PROCEDURE TForm1.Edit1Change(Sender: TObject);
BEGIN
  Label1.Caption := Edit1.Text;
  Label2.Caption := Edit2.Text;
  Label2.Left    := Label1.Left+Label1.Width+5;
END;
```

Vergessen Sie nicht, die Ereignisbehandlungsroutine auch dem entsprechenden Ereignis des zweiten Eingabefelds zuzuweisen (über den Objektinspektor).

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_2.

Lösung 3

Wir müssen die Lösung für diese Aufgabe nur für ein Eingabefeld programmieren. Der Einsatz des Windows API (Es heißt wirklich *das* Windows-API, nicht *die* Windows-API, wie manche Fachbücher schreiben. Die Übersetzung ist nämlich *das* Application Interface) soll später noch vorgestellt werden, geht aber für eine grundlegende Übung zu weit.

Machen wir uns wieder Gedanken um das richtige Ereignis. Es muss sich um ein Ereignis handeln, mit dem wir einen Tastendruck abfangen können. Da bieten sich zwei Ereignisse an, nämlich *OnKeyPress* oder *OnKeyDown*. Auch *OnKeyUp* wäre eine Alternative. Alle drei können für diese Übung benutzt werden. Ich habe mich für *OnKeyPress* entschieden.

Der Code für die gedrückte Taste wird dem Ereignis übermittelt. Wir müssen kontrollieren, um welche Taste es sich handelt, falls es die Taste *Return* ist, müssen wir dem nächsten Steuerelement (*Edit2*) den Fokus geben. Wenn Sie (was Sie immer tun sollten) in der Online-Hilfe nachgeschaut haben, haben Sie sicher die Methode *SetFocus* gefunden, die ein sichtbares Steuerelement zum aktuell aktiven Steuerelement macht. Diese Funktion werden wir nutzen.

```
PROCEDURE TForm1.Edit1KeyPress(Sender: TObject;
                               VAR Key: Char);
BEGIN
  IF Key = #13 THEN Edit2.SetFocus;
END;
```

Die Raute vor der 13 weist Delphi an, diesen Wert als Ascii-Wert (bzw. Ansi-Wert) für ein Zeichen zu betrachten. Auch das finden Sie in der Online-Hilfe. Alles, was wir tun, ist die gedrückte Taste zu kontrollieren und, falls es sich um die Eingabetaste handelt, das nächste Eingabefeld zu aktivieren. Das Gleiche können Sie auch für das zweite Eingabefeld programmieren.

Nun noch zu der versprochenen allgemeingültigen Lösung. Setzen Sie zunächst die Eigenschaft *KeyPreview* des Formulars auf *true*. Damit wird jeder Tastendruck zuerst an das Formular geliefert und danach erst an das aktive Steuerelement. Das ist wichtig, denn nur das Formular enthält die Liste aller darauf befindlichen Steuerelemente und kann dementsprechend das nächste aktivieren.

In die Eigenschaft *OnKeyPress* des Formulars geben Sie dann folgenden Quellcode ein:

```
PROCEDURE TForm1.FormKeyPress(Sender: TObject;
                              VAR Key: Char);
BEGIN
  IF Key = #13 THEN Self.Perform(WM_NEXTDLGCTL,0,0);
END;
```

Dieser Code sendet eine Nachricht an Windows mit dem Auftrag, das nächste Steuerelement zu aktivieren.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_3. UEBUNG3A enthält die erste Lösung, UEBUNG3B die allgemeingültige Lösung.

Lösung 4

Die Lösung für diese Aufgabe ist eigentlich nicht besonders schwierig. Zunächst erstellen wir das Formular in der ganzen Größe, also in aufgeklappter Form, und platzieren die beiden Schaltflächen. Jetzt müssen wir feststellen, welche Größe das Formular in zugeklapptem Zustand haben muss. Diese Größe weisen wir der entsprechenden Eigenschaft des Formulars über die Schaltfläche zum Zuklappen zu:

```
PROCEDURE TForm1.Button2Click(Sender: TObject);
BEGIN
  Form1.Height := 220;
END;
```

Als Nächstes müssen wir dem Formular über die Schaltfläche zum Aufklappen die Größe zuweisen, und zwar genau die Größe, die es aktuell besitzt, denn wir haben es ja in aufgeklapptem Zustand erstellt.

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  Form1.Height := 345;
END;
```

Damit ist die grundsätzliche Funktionalität bereits hergestellt. Nun müssen wir noch sicherstellen, dass das Formular beim Programmstart in der zugeklappten Version erscheint. Die entsprechende Routine haben wir bereits geschrieben, nämlich die Ereignisbehandlungsroutine *Button2Click*. Wir verwenden nun das Eeignis *OnShow* des Formulars, um diese Ereignisbehandlungsroutine aufzurufen. Alternativ können Sie natürlich die Größe des Formulars auch nochmals zuweisen.

```
PROCEDURE TForm1.FormShow(Sender: TObject);
BEGIN
  Button2Click(Sender);
END;
```

In diesem Beispiel wurde das Formular nach unten aufgeklappt. Sie können natürlich das Aufklappen auch nach der Seite programmieren oder auch nach beiden Seiten.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_4. Einen Screenshot jeweils im aufgeklappten und zugeklappten Zustand sehen Sie in den Abbildungen 3.1 und 3.2.

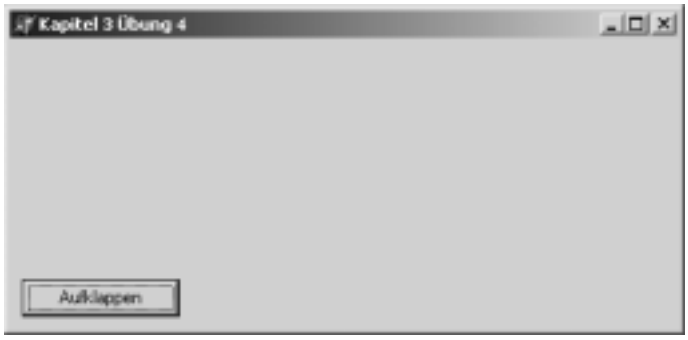

Abbildung 3.1: Das Formular in zugeklapptem Zustand ...

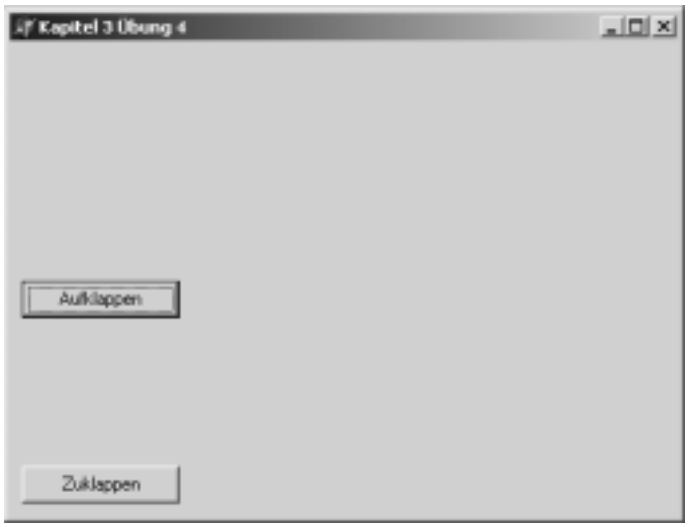

Abbildung 3.2: ... und in aufgeklapptem Zustand

Lösung 5

Die Lösung zu dieser Übung ist trivial. In der Ereignisbehandlungsroutine zum Ereignis *OnClick* einer jeden Schaltfläche weisen Sie einfach deren *Caption* der *Caption* des Panels zu:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  Panel1.Caption := Button1.Caption;
END;
```

Das Ganze wird wiederholt, für alle anderen Schaltflächen. Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_5.

Lösung 6

An dieser Stelle wird es etwas schwieriger, mit dem richtigen Lösungsansatz kommen wir aber weiter. Der Tipp für diese Übung war die Verwendung der Typumwandlung. Diese funktioniert selbstverständlich nicht nur bei den vordefinierten Datentypen, sondern auch bei Klassen. Aufgrund des Konzepts der Polymorphie kann der Parameter *Sender*, der jeder Ereignisbehandlungsroutine übergeben wird, jede Komponente darstellen. Also müssen wir logischerweise auch in der Lage sein, einerseits herauszufinden, welche Komponente das ist, und andererseits den Parameter *Sender* in diese Komponente umwandeln können. Genau das geschieht durch die Typumwandlung, ebenso wie bei den anderen Datentypen auch.

In diesem Beispiel kann der Parameter *Sender* aufgrund unserer Vorgaben nur eine Komponente vom Typ *TButton* sein, wir müssen also keine Abfrage durchführen, sondern können sofort die Typumwandlung vornehmen:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  Panel1.Caption := TButton(Sender).Caption;
END;
```

Im Gegensatz zu Übung 5, bei der wir für das gleiche Resultat fünf verschiedene Ereignisbehandlungsroutinen schreiben mussten, kommen wir hier mit einer allgemeingültigen Routine aus, die wir den anderen Schaltflächen zuweisen können.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_6.

Lösung 7

Wenn in einer ComboBox mit dem Stil *csDropDownList* kein Eintrag angezeigt wird, ist der Wert von *ItemIndex* -1. Das erste Element hat den Index 0, das zweite den Index 1 usw. Wenn *ItemIndex* den Wert 3 hat, wird also das vierte Element der Liste angezeigt.

Einen Quelltext, mit dem Sie die Werte herausfinden können, finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_7.

Lösung 8

Es gibt natürlich nicht nur zwei Lösungen, aber zumindest zwei Lösungsprinzipien, die bereits in den Übungen vorher vorgestellt wurden.

Das erste Prinzip ändert bei einem Klick auf eine Schaltfläche entweder den Wert von *ActivePageIndex* oder den Wert von *ActivePage*. Beide Werte korrespondieren miteinander, eine Änderung eines Wertes hat automatisch auch die Änderung des anderen Wertes zur Folge. Im Beispiel habe ich mich für die Lösung mit *ActivePageIndex* entschieden. Für jede Schaltfläche wird eine Ereignis-

behandlungsroutine erstellt und der Wert der Eigenschaft *ActivePageIndex* der *TPageControl*-Komponente geändert.

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  PageControl1.ActivePageIndex := 0;
END;

PROCEDURE TForm1.Button2Click(Sender: TObject);
BEGIN
  PageControl1.ActivePageIndex := 1;
END;

PROCEDURE TForm1.Button3Click(Sender: TObject);
BEGIN
  PageControl1.ActivePageIndex := 2;
END;
```

Auch die Komponente *TPageControl* beginnt, wie wir sehen, mit der Zählung bei 0 und nicht bei 1. Allgemein ist das fast immer so, wenn Sie in Delphi oder einer anderen Programmiersprache programmieren. In den Programmiersprachen wird bei 0 mit der Zählung begonnen, nicht bei 1.

Die zweite Möglichkeit des Lösungsansatzes bedeutet wieder eine Arbeitserleichterung, weil nur eine Ereignisbehandlungsroutine programmiert werden muss. Wir wollen die Eigenschaft *ActivePageIndex* in Abhängigkeit der angeklickten Schaltfläche ändern. Dazu benötigen wir einen Wert, der Button-spezifisch ist und mit dessen Hilfe wir *ActivePageIndex* ändern können.

Im Vorfeld wurde bereits gesagt, dass jede Komponente eine Eigenschaft besitzt, nämlich die Eigenschaft *Tag*, die von Delphi nicht verwendet wird. Dabei handelt es sich um einen *Integer*-Wert, wir können diese Eigenschaft also verwenden. Wir weisen der Eigenschaft *Tag* der Schaltflächen den Wert zu, den *ActivePageIndex* annehmen soll, wenn auf die Schaltfläche geklickt wird.

Jetzt müssen wir nur noch eine Ereignisbehandlungsroutine für alle drei Schaltflächen schreiben. Wir benutzen wieder die Typumwandlung, mit deren Hilfe wir den Parameter *Sender* wieder in den Typ *TButton* ändern.

```
PROCEDURE TForm1.Button4Click(Sender: TObject);
BEGIN
  PageControl1.ActivePageIndex := TButton(Sender).Tag;
END;
```

Damit benötigen wir nur eine Ereignisbehandlungsroutine für alle drei Schaltflächen. Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_8.

Die oberen drei Schaltflächen sind nach dem ersten Prinzip programmiert, jede Schaltfläche hat eine eigene Ereignisbehandlungsroutine. Die unteren drei

Schaltflächen sind nach dem zweiten Prinzip programmiert, die Werte ihrer Eigenschaft *Tag* sind angepasst und es wird nur eine Ereignisbehandlungsroutine benutzt um die Seite der *TPageControl*-Komponente zu wechseln. Einen Screenshot des laufenden Programms sehen Sie in Abbildung 3.3.

Abbildung 3.3: Das fertige Programm zu Übung 8

Lösung 9

Die Klasse *TScreen*, die von Delphi zur Verfügung gestellt wird, besitzt einige Eigenschaften, die das aktuelle System betreffen. Unter anderem findet man in der Online-Hilfe die Eigenschaften *Screen.width* und *Screen.Height*, die die Größe des verfügbaren Bildschirms angeben.

Um nun ein Formular beim Programmstart auf der Mitte des Bildschirms zu platzieren, können wir die gleiche Berechnung anwenden wie in Übung 1, wir berechnen mit Hilfe der Höhe und Breite des Bildschirms den oberen linken Punkt, an dem das Formular angezeigt werden soll, und setzen entsprechend die Eigenschaften *Top* und *Left*.

Als Ereignis nehmen wir diesmal das Ereignis *OnCreate* des Formulars. Dieses Ereignis tritt auf, wenn das Formular erstellt wird, also gerade der richtige Zeitpunkt, um die Koordinaten festzulegen.

```
PROCEDURE TForm1.FormCreate(Sender: TObject);
BEGIN
  Top  := (Screen.Height-Height) DIV 2;
  Left := (Screen.width-width) DIV 2;
END;
```

Da es sich bei dieser Ereignisbehandlungsroutine um eine Prozedur handelt, die innerhalb der Formularklasse definiert ist, müssen wir die Eigenschaften *Left*, *Top*, *Width* und *Heigth* nicht qualifizieren, wenn es sich dabei um die Eigen-

schaften des Formulars handeln soll. Delphi nimmt sich automatisch die Eigenschaften der Klasse, in der die jeweilige Prozedur deklariert ist. Die Eigenschaften *Width* und *Height* der Klasse *TScreen* müssen wir hingegen qualifizieren.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_9.

Die einfachere zweite Möglichkeit ist, einfach die Eigenschaft *Position* des Formulars auf *poScreenCenter* zu setzen. Delphi übernimmt die Ausrichtung dann automatisch.

Lösung 10

Die Lösung des Problems ist eigentlich trivial. Wichtig ist aber zunächst, dass Sie das Formular aus der Liste der automatisch erzeugten Formulare entfernen. Das können Sie im Dialog PROJEKT/OPTIONEN tun. Nun müssen wir die Unit, in der das zweite Formular deklariert ist, noch in die Unit des ersten Formulars einbinden. Sie können das manuell erledigen oder auch Delphi die Arbeit überlassen. Der entsprechende Menüpunkt ist DATEI/UNIT VERWENDEN, Sie können dazu auch die Tastenkombination Alt + F11 verwenden.

Nach dieser Vorarbeit sind die Erzeugung und die Anzeige des zweiten Formulars trivial:

```
PROCEDURE TForm1.Button1Click(Sender: TObject);
BEGIN
  Form2 := TForm2.Create(Application);
  Form2.ShowModal;
  Form2.Release;
END;
```

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_10.

Lösung 11

Das Eingabefeld und den Button einzufügen ist kein großes Problem. Viel größer ist das Problem, dass bei einem Klick auf die Schaltfläche die *Caption* des aufrufenden Formulars geändert werden soll. Wir müssen also eine kreuzweise Einbindung der Units vornehmen. Stellen Sie dazu zunächst sicher, dass Unit2 im Implementation-Teil von Unit1 eingebunden ist. Falls Sie Delphi die Einbindung überlassen haben, ist das automatisch der Fall.

Nun müssen wir noch Unit1 in Unit2 einbinden, ebenfalls im Implementation-Teil. Auch hier können Sie dies entweder manuell tun oder es Delphi überlassen (Menüpunkt DATEI/UNIT VERWENDEN oder Alt + F11).

Nachdem das Eingabefeld und die Schaltfläche eingefügt sind, genügt eine simple Zuweisung, um den Text der Titelzeile zu ändern. Benutzt wird die Ereignisbehandlungsroutine der Schaltfläche auf dem zweiten Formular. Bei meinem

Programm hat diese den Namen *Button2*, weil *Button1* zum Schließen des zweiten Formulars verwendet wurde.

```
PROCEDURE TForm2.Button2Click(Sender: TObject);
BEGIN
  Form1.Caption := Edit1.Text;
END;
```

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_11.

Lösung 12

Es gibt natürlich viele Möglichkeiten, eine derartige Funktion zu programmieren. Wir beschränken uns in diesem Fall einfach darauf, die linke obere Ecke dann anzupassen, wenn die Maus sich über dem Formular befindet. Damit können wir die Schaltfläche stets vor der Maus platzieren, der Anwender kann sie niemals treffen.

Natürlich müssen wir zunächst herausfinden, welches Ereignis dafür am geeignetsten ist. Ich habe das Ereignis *OnMouseMove* des Formulars gewählt. Das wird immer dann ausgelöst, wenn die Maus über dem Formular bewegt wird. Dabei werden die aktuellen Mauskoordinaten relativ zum Formular übergeben, so dass es ein Leichtes ist, die Koordinaten der Schaltfläche anzupassen.

Wir müssen nur noch darauf achten, dass der Button nicht verschwindet. Dazu kontrollieren wir einfach, ob er über den Rand des Formulars hinausgeschoben wird. Ist dies der Fall, setzen wir seine Position einfach auf den gegenüberliegenden Rand des Formulars.

Und hier die Ereignisbehandlungsroutine:

```
PROCEDURE TForm2.FormMouseMove(Sender: TObject;
                Shift: TShiftState; X,Y: Integer);
BEGIN
  Button1.Top := y+10;
  Button1.Left := x+10;
  IF Button1.Top>ClientHeight THEN
    Button1.Top := 0;
  IF Button1.Left>ClientWidth THEN
    Button1.Left := 5
END;
```

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_12.

Lösung 13

Im Prinzip ist die Lösung trivial. Die Eigenschaft *DragMode* wurde im Beispiel auf *dmManual* gesetzt, Sie können sie aber auch auf *dmAutomatic* stehen las-

sen. Die eigentliche Funktionalität beschränkt sich auf die Ereignisbehandlungsroutinen *OnDragOver* und *OnDragDrop* der zweiten Listbox:

```
PROCEDURE TForm1.ListBox2DragOver(Sender, Source: TObject;
X, Y: Integer; State: TDragState; VAR Accept: Boolean);
BEGIN
  Accept := (Source IS TListBox);
END;

PROCEDURE TForm1.ListBox2DragDrop(Sender, Source: TObject;
                                  X, Y: Integer);
BEGIN
  ListBox2.Items.Add(ListBox1.Items[ListBox1.ItemIndex]);
END;
```

Zunächst wird das Ereignis *OnDragOver* ausgelöst, in dem wir nur kontrollieren, wer der Urheber der Drag&Drop-Operation ist. Wenn es sich dabei um eine Listbox handelt (es kann sich ja dann nur um Listbox1 handeln), lassen wir die Drag&Drop-Operation zu, ansonsten nicht. Festgelegt wird das über den Parameter *Accept*.

Das nächste ausgelöste Ereignis ist *OnDragDrop*, nämlich wenn wir die Maustaste loslassen. In der entsprechenden Ereignisbehandlungsroutine fügen wir einfach den in der ersten Listbox ausgewählten Eintrag der zweiten Listbox hinzu.

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_13. Abbildung 3.4 zeigt einen Screenshot des fertigen Programms.

Abbildung 3.4: Das fertige Programm zu Übung 13

Lösung 14

Diese Übung ist ein wenig schwieriger als die vorangegangene. Machen wir uns also zunächst ein wenig Gedanken, wie wir die Funktionalität programmieren wollen.

Die Schaltfläche, die wir verschieben wollen, soll nach wie vor angeklickt werden können. Damit scheidet die Möglichkeit des Ziehens mit der linken Maustaste aus, da wir ja beim Verschieben der Schaltfläche an ihre neue Position kein *OnClick*-Ereignis auslösen wollen. Wir müssen also eine Möglichkeit finden, die rechte Maustaste zu benutzen.

Weiterhin dürfen wir nur drei Ereignisbehandlungsroutinen benutzen (das Ereignis *OnClick* des Buttons nicht mitgerechnet). Da eine weitere Voraussetzung ist, dass die Eigenschaft *DragMode* des Buttons auf *dmManual* stehen soll, muss in einem der Ereignisse die Drag&Drop-Operation gestartet werden. Dafür nehmen wir das Ereignis *OnMouseDown* der Schaltfläche, das außerdem noch die vorteilhafte Eigenschaft hat, einen Parameter für die betätigte Maustaste zu übergeben. Wir können also die Maustaste ermitteln und nur dann die Drag&Drop-Operation starten, wenn es sich um die rechte Maustaste handelt:

```
PROCEDURE TForm1.Button1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
BEGIN
  IF Button=mbRight THEN
    Button1.BeginDrag(false);
END;
```

Vergessen Sie auf keinen Fall die Qualifizierung der Methode *BeginDrag*. Sie müssen die Methode des Buttons aufrufen, nicht die des Formulars, denn Formulare können nicht mittels Drag&Drop verschoben werden. Das Resultat wäre eine entsprechende Fehlermeldung, die Sie sicher schon erhalten haben.

Nun kommen wir zum Ereignis *OnDragOver*. Dieses Ereignis und das Ereignis *OnDragDrop* müssen wir ebenfalls programmieren, allerdings so, dass es sowohl für die beiden Panels als auch für das Formular gültig ist. Im Falle von *OnDragOver* ist das nicht weiter schwer. Wir legen fest, dass jede Komponente die Drag&Drop-Operation dann erlauben soll, wenn der Sender vom Typ *TButton* ist:

```
PROCEDURE TForm1.FormDragOver(Sender, Source: TObject; X,
Y: Integer; State: TDragState; VAR Accept: Boolean);
BEGIN
  Accept := (Source IS TButton);
END;
```

Bleibt nur noch das Ereignis *OnDragDrop*. In diesem wird der Wert der Eigenschaft *Parent* der Schaltfläche neu zugewiesen. Wir wollen aber eine allgemeingültige Funktion schreiben, d.h. es soll vollkommen egal sein, auf welcher Komponente der Button abgelegt wird.

Eine Gemeinsamkeit haben aber alle diese Komponenten: Sie müssen vom Typ *TWinControl* sein, denn nur solche Komponenten können andere Komponenten und Steuerelemente aufnehmen. Der Parameter *Sender* muss also gezwungenermaßen vom Typ *TWinControl* abgeleitet sein. Damit funktioniert auch die Zuweisung über folgende einfache Typumwandlung:

```
PROCEDURE TForm1.FormDragDrop(Sender, Source: TObject;
                              X, Y: Integer);
BEGIN
  Button1.Left := x;
  Button1.Top  := y;
  Button1.Parent := TWinControl(Sender);
END;
```

Welches der drei möglichen *OnDragDrop*-Ereignisse nun wirklich programmiert wird, ist weitgehend egal. Durch die Zuweisung der Ereignisbehandlungsroutinen erhalten auch die Komponenten, für die das Ereignis nicht explizit programmiert wurde, die notwendige Funktionalität.

Das obige Programm ist allgemeingültig für alle Komponenten, die eine andere Komponente oder ein Steuerelement aufnehmen können. Das komplette Programm im Zusammenhang gibt's hier nochmal:

```
PROCEDURE TForm1.Button1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
BEGIN
  IF Button=mbRight THEN
    Button1.BeginDrag(false);
END;

PROCEDURE TForm1.FormDragOver(Sender, Source: TObject;
X, Y: Integer; State: TDragState; VAR Accept: Boolean);
BEGIN
  Accept := (Source IS TButton);
END;

PROCEDURE TForm1.FormDragDrop(Sender, Source: TObject;
                              X, Y: Integer);
BEGIN
  Button1.Left := x;
  Button1.Top  := y;
  Button1.Parent := TWinControl(Sender);
END;
```

Den Quelltext zu dieser Übung finden Sie auf der CD im Verzeichnis ÜBUNGEN\KAPITEL_3\ÜBUNG_14.

Lösung 15

- Kann man ein Eingabefeld auf einer Panel-Komponente platzieren?

  Das ist natürlich möglich. Bei der Komponente *TPanel* handelt es sich um eine Komponente, die andere Komponenten enthalten kann und auch zur Abgrenzung auf einem Formular benutzt wird.

- Was ist der Unterschied zwischen den Komponenten TButton und TBitBtn?

  Der Unterschied liegt einzig und allein darin, dass *TBitBtn* noch ein Symbol anzeigen kann, ein so genanntes *Glyph*, das neben der *Caption* der Schaltfläche steht. In der Funktionsweise sind beide exakt gleich (auch vom Quelltext her).

- Was unterscheidet die Komponenten TEdit und TMaskEdit?

  TMaskEdit ist ein spezielles Eingabefeld, bei dem Sie dem Anwender Ihrer Applikation eine Eingabemaske vorgeben können. Sie können mit dieser Eingabemaske auch die Art der Eingabe fest vorschreiben (z. B. für Telefonnummern). *TEdit* besitzt dieses Verhalten nicht.

- Welche Eigenschaft hat jede Komponente in Delphi?

  Es handelt sich um die Eigenschaft *Tag*, die wirklich jede Komponente in Delphi besitzt. Von Delphi selbst wird diese Eigenschaft aber nicht benutzt.

- In welchem Ereignis sollte eine Abfrage programmiert werden, die den Anwender entscheiden lässt, ob er eine Anwendung wirklich beenden will?

  Das entsprechende Ereignis ist *OnCloseQuery.* Über den Parameter *CanClose*, der der entsprechenden Ereignisbehandlungsroutine übergeben wird, kann die Anwendung entweder beendet werden (*CanClose* auf *true* setzen) oder nicht (*CanClose* auf *false* setzen).

- Wozu dient die Eigenschaft KeyPreview eines Formulars?

  Wenn die Eigenschaft *KeyPreview* eines Formulars auf *true* gesetzt ist, wird ein Tastendruck zuerst an das Formular und dann an die aktive Komponente, z. B. ein Eingabefeld, gesendet. Ist *KeyPreview false*, verhält es sich umgekehrt.

- Wozu dient die Eigenschaft PasswordChar einer Komponente vom Typ TEdit?

  In der Eigenschaft *PasswordChar* können Sie ein Zeichen angeben, das an Stelle der Eingabe erscheinen soll. Die eigentliche Eingabe erscheint nicht. Dieses Verhalten ist sinnvoll bei der Eingabe eines Passworts.

- Wie groß ist die Standard-Schrittweite der Komponente TProgressbar?

  Die Standard-Schrittweite ist 10.

- Mit welcher Methode kann ein Formular (nicht das Hauptformular) angezeigt werden? Es existieren zwei Methoden mit unterschiedlicher Auswirkung dafür.

  Die erste Methode ist *Show.* Das Formular wird angezeigt, das Programm aber läuft im Hintergrund weiter und der Anwender kann weitere Menüpunkte anwählen. Die zweite Möglichkeit ist die Methode *ShowModal*, bei der das Programm so lange stoppt, bis der Anwender das Fenster wieder verlassen hat.