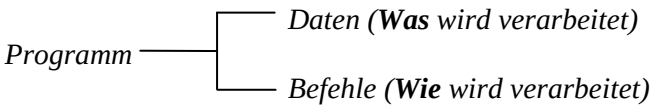
DELPHI 01

Allgemeine Grundlagen

[1] GRUNDLAGEN DES PROGRAMMIERENS

[1.01] Was ist ein Programm?

Am Anfang steht immer ein Problem. Dieses wird vom Programmierer analysiert und ein Lösungsverfahren (Algorithmus) entwickelt. Der Algorithmus wird sodann mit Hilfe einer Programmiersprache in einer Folge von entsprechenden Befehlen formuliert. Die Befehle bilden das Programm und steuern den Computer derart, dass die Problemlösung realisiert wird. Ein Programm enthält daher sowohl die relevanten Daten als auch die entsprechenden Befehle zur Verarbeitung dieser Daten.



Der Vorteil der Programmierung in einer höheren Computersprache ist die symbolische Codierung. Der Zugriff auf die Daten erfolgt mittels einer symbolischen Adressierung. D.h., der Programmierer kann die Daten über Variablenbezeichner (das sind beliebig wählbare Namen) ansprechen und braucht sich nicht um deren effektive Speicheradressen kümmern. Auch die Befehle werden durch symbolische Schlüsselworte dargestellt und entsprechen meist sehr umfangreichen internen Maschinencodes. Einfache symbolische Adressierung der Daten und mächtige symbolische Befehlsworte kennzeichnen somit eine Hochsprache, die dadurch relativ maschinenunabhängig und komfortabel wird. Damit zwischen Daten und Befehlen keine Verwechslungen auftreten, werden sie in getrennten Bereichen des Hauptspeichers abgelegt.

Vor der Verwendung von Daten in einem Programm muss deren Speicherformat festgelegt werden, damit der Computer den Daten entsprechende Bytes des Speichers zuordnen kann. Will man zum Beispiel eine Berechnung durchführen und das Ergebnis in der Variablen *Resultat* abspeichern, so muss dem Computer am Beginn des Programms mitgeteilt werden, dass die Variable *Resultat* eine reelle Zahl enthalten soll. Dies geschieht im **Definitionsteil** des Programms.

Variable und Konstante (Daten, deren Werte sich während der Laufzeit des Programmes nicht ändern) werden erzeugt, um damit Berechnungen durchzuführen. Zu diesem Zweck muss eine Menge von zulässigen Operationen zur Verfügung stehen (z.B. die Addition "+"). Aus primitiven Operationen werden mächtige Operationsabläufe zusammengesetzt. Dies erfolgt im **Ablaufteil** des Programmes. Zwei grundlegende Programmbefehle sind Wertzuweisung und Wertevergleich:

Wertzuweisung: Einer Variablen (***X***) wird der Wert einer anderen Variablen oder Konstanten (***Y***) oder ein Operationsergebnis zugewiesen (z.B. ***X := Y*** oder ***X := Y + Z***).

Wertevergleich: Test auf Gleichheit zweier Variablen (z.B. ***X = Y*** oder ***X = Y + Z***). Dabei wird immer der linke Wert mit dem rechten Wert auf Übereinstimmung verglichen.

Das Schreiben der Programmbefehle (Quelltext) erfolgt mit einem komfortablen **Editor**. Die Umwandlung des symbolischen Codes in den maschinenverständlichen Binärcode übernimmt ein eigenes Übersetzungsmodul, der **Compiler**. Mit Hilfe eines **Linkers** werden den symbolischen Adressen Speicherplätze zugeordnet. Dann erst ist ein lauffähiges Programm entstanden.

- **Ein Schubladen-Modell des Speichers**

Einer Variablen sind folgende drei Bestimmungen zugeordnet:

Name: Durch diesen wird der Speicherplatz adressiert (d.h. wo die Variable gespeichert ist).
Typ: Dieser definiert die Struktur des Speicherplatzes (d.h. wie die Bits angeordnet sind).
Wert: Der Inhalt des Speicherplatzes (d.h. Auswertung der Bits, z.B. als Zahl oder Zeichen).

Vereinfacht kann eine solche Variable als eine mit einem Namensschild versehene Schublade im Speicherkasten des Computers aufgefasst werden. Damit laufen in vereinfachter Weise bei einer **Wertzuweisung** ***X := Y + Z*** folgende Arbeitsschritte im Computer ab:

(1) Suche im Speicherkasten die Schublade ***Y***.
(2) Transportiere ihren Inhalt in ein Register in der Zentrale.
(3) Suche im Speicherkasten die Schublade ***Z***.
(4) Transportiere deren Inhalt in ein Register in der Zentrale.
(5) Transportiere die Registerinhalte ins Rechenwerk und führe dort den Additionsbefehl (+) aus.
(6) Stelle dann das Ergebnis in ein zentrales Register zurück.
(7) Transportiere das Rechenergebnis aus dem zentralen Register in die Speicher-Schublade ***X***.

Ein **Wertevergleich** ***X = Y*** kann in vereinfachter Weise folgendermaßen dargestellt werden:

(1) Suche im Speicherkasten die Schublade ***Y***.
(2) Transportiere ihren Inhalt in ein zentrales Speicherregister.
(3) Suche im Speicherkasten die Schublade ***X***.
(4) Transportiere ihren Inhalt in ein zentrales Speicherregister.
(5) Transportiere diese zwei Registerinhalte in das Rechenwerk und vergleiche sie dort. Setze, je nach Ausgang des Vergleiches (gleich, ungleich), ein Statusbit (1 = true, 0 = false) im Statusregister der Zentrale.
(6) Das Vergleichsergebnis kann dann vom Programm im zentralen Statusregister abgelesen und zur Steuerung des weiteren Programmablaufes verwendet werden.

Die symbolischen Programmbefehle werden im Hauptspeicher im maschinenverständlichen Binärcode abgelegt und von der Zentrale des Computers (CPU) hintereinander abgearbeitet. Dabei erfolgt die Durchführung eines einzelnen Maschinenbefehls in einem dreiteiligen Zyklus: Befehl holen (vom Hauptspeicher in die Zentrale), Befehl decodieren und Befehl ausführen. Die Arbeitsgeschwindigkeit dieses Maschinenzyklus hängt weitgehend von der Taktrate der CPU ab.

- **Ein Programmbeispiel**

Eine Liste von Personennamen wird über die Tastatur in einen dafür reservierten Bereich des Hauptspeichers eingegeben. Dort erfolgt eine alphabetische Sortierung. Zum Schluss wird die sortierte Liste am Drucker ausgegeben.

Der eigentliche Kern des Programmes liegt in der Entwicklung eines geeigneten Sortierverfahrens. Dazu gibt es verschiedene Möglichkeiten: Beispielsweise durchläuft man mehrmals schrittweise die eingespeicherte Liste und vergleicht jedes Listenelement mit seinem Nachfolger. Steht das Element im Alphabet hinter seinem Nachfolger, dann müssen die beiden Elemente in der Liste ihre Plätze tauschen. Verglichen werden dabei die ANSI-Codes der einzelnen Textzeichen der Listenelemente. Am Ende des Verfahrens erhält man eine sortierte Liste.

Das Programm zur Lösung des Problems könnte in folgende Teilschritte (Module) zerlegt werden:

→ Programmbeginn.
→ Reservierung eines Speicherbereichs von Textvariablen (Strings) für die Personennamen.
→ Tastatureingabe der Personennamen und deren Abspeicherung auf die vorher reservierten Textvariablen.
→ Sortierung der Variablenliste im Hauptspeicher mit einem entsprechenden Sortierverfahren. Dieses wird als Unterprogramm im eigentlichen Hauptprogramm erstellt.
→ Ausgabe der Textvariablen am Drucker.
→ Programmende.

• Das Problem und seine Lösung

Wir wollen nun ganz allgemein untersuchen, was man unter einem Problem und seiner Lösung versteht. Zur Illustration ein einfaches Beispiel: Eine sortierte Liste von ganzen Zahlen soll vorliegen. Eine neue Zahl soll in diese sortierte Liste an der richtigen Stelle eingefügt werden. Zur Erledigung dieser Aufgabe stellen wir zunächst die Zahl an das Ende der Liste. Dann vergleichen wir – beginnend mit dem letzten Listenelement – jedes Element mit seinem Vorgänger. Ist das Element kleiner als sein direkter Vorgänger, so tauschen die beiden Platz. Dadurch ist das Element um eine Stelle nach vor gerückt. Dieses Verfahren wird dann abgebrochen, wenn das Element größer oder gleich seinem Vorgänger ist. Dann ist die neue Zahl an der richtigen Stelle eingefügt.

Unser Beispiel demonstriert sehr anschaulich das allgemeine Schema von Problemlösungen:

[1] Es existiert ein unerwünschter Anfangszustand ***AZ***.

[2] Der Problemlöser hat einen erwünschten Zielzustand ***ZZ*** vor Augen.

[3] Der Problemlöser sucht nach einer Transformation, die aus einer Folge von Operationen besteht, welche den Anfangszustand über Zwischenstufen in den Zielzustand überführen (***AZ → ZZ***).

Meistens ist leider die Problemlösung nicht sofort einsehbar (evident), sondern sie wird durch verschiedene Barrieren erschwert. Beispielsweise könnte der Zielzustand zu grob und zu wenig präzise formuliert sein oder es stehen keine zielführenden Operationen zur Verfügung.

Die Entwicklung eines Programmes zur Lösung von Problemen erfolgt in bestimmten Schritten:

PROBLEMSTELLUNG

LÖSUNGSENTWURF

PROGRAMMIERUNG

PROGRAMMTESTUNG

Der wesentliche Schritt dabei ist der Lösungsentwurf (Programmentwurf). Er gliedert sich in zwei Abschnitte:

→ **Analyse des Problems**

Analyse der Ausgabedaten (Was will ich erreichen?)
Analyse der Eingabedaten (Was ist vorgegeben?)
Analyse des Lösungsverfahrens (Algorithmus: Wie erreiche ich das Ziel?)

→ **Darstellung des Lösungsverfahrens**

In umgangsprachlicher Form
Als Struktogramm
Als Flussdiagramm

Die wichtigsten Richtlinien für einen strukturierten Programmentwurf sind der ***Top-Down-Entwurf*** und die ***Modularisierung***. Darunter versteht man einerseits die schrittweise Verfeinerung der Problemlösung (vom Groben zum Feinen) und andererseits die Gliederung in verschiedene, wohldefinierte Teilbereiche (Module).

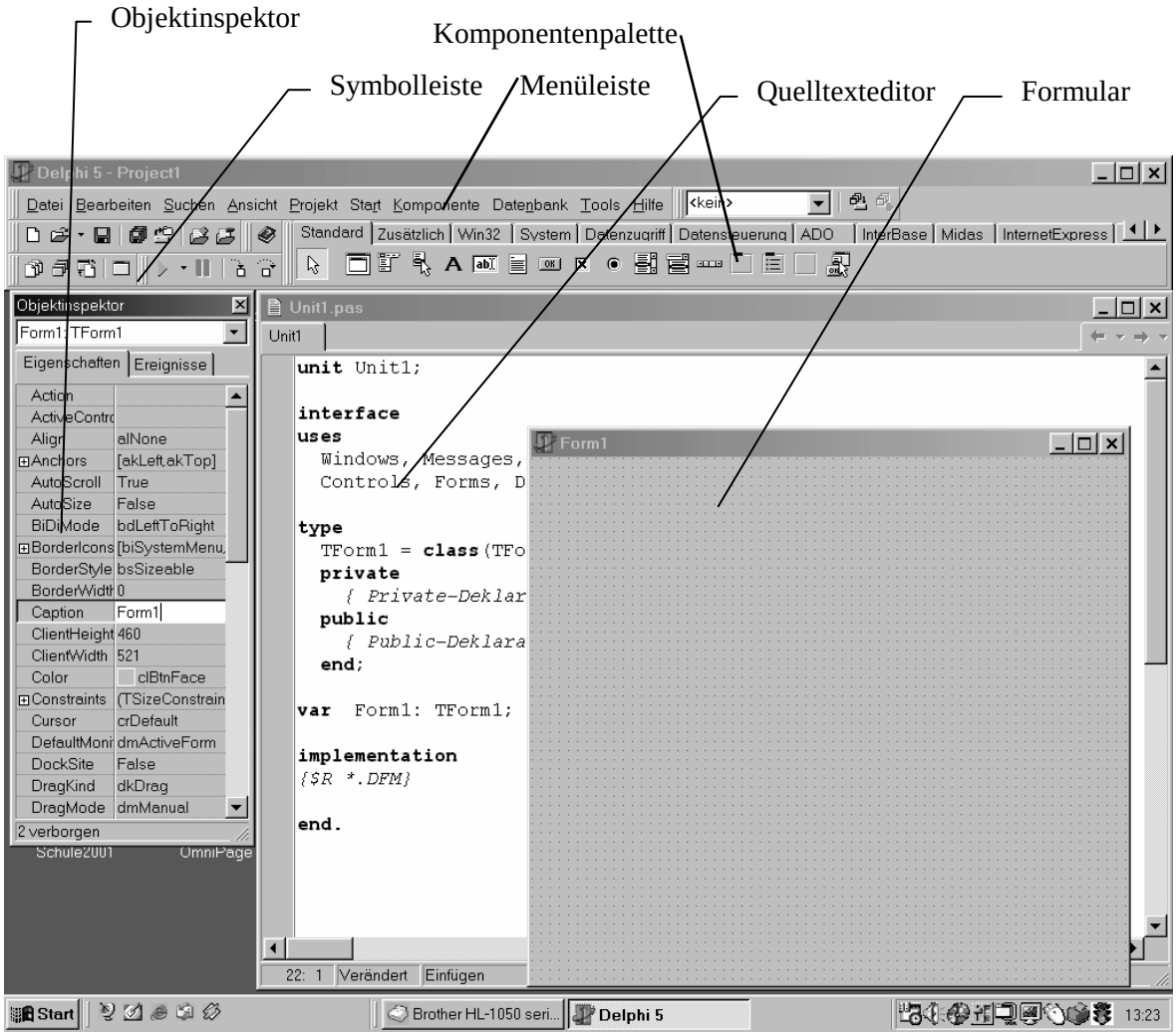
[1.02] Programmieren mit DELPHI

DELPHI ist der Nachfolger der im Betriebssystem MSDOS erfolgreichen Programmiersprache TURBO PASCAL. Das war eine Hochsprache, die sich durch einfache Erlernbarkeit und große Leistungsfähigkeit auszeichnete. Diese Vorteile wurden in DELPHI noch erweitert und führten zu einem sehr leistungsfähigen, aber auch leicht zu handhabenden Programmiersystem. DELPHI präsentiert sich wie viele andere moderne Programmiersprachen in einer so genannten Entwicklungsumgebung (Integrated Development Environment, IDE). In diesem Rahmenprogramm sind alle Werkzeuge zur Programmentwicklung (z.B.: Editor, Compiler, Linker) integriert, die zur Erstellung von zeitgemäßer Software notwendig sind. Die Entwicklungsumgebung von DELPHI setzt sich aus verschiedenen Modulen zusammen. Es handelt sich um ein so genanntes visuelles Programmiersystem, mit dem unter WINDOWS Programme erstellt werden. Es arbeitet dabei mit den üblichen Fenster-Techniken. Der vorliegende Programmierkurs bezieht sich auf die DELPHI-Versionen 5 und höher.

- **Die Entwicklungsumgebung**

Nach dem Start präsentiert sich eine Bedienungsoberfläche mit vier frei schwebenden Fenstern:

(1) Das DELPHI-Hauptfenster mit Menüleiste, Symbolleiste und Komponentenpalette.
(2) Der Objektinspektor mit Objektauswahlfeld (Selektor), Eigenschafts- und Ereignis-Registern.
(4) Das Formular, in welchem die Bedienungselemente des Programmes platziert sind.
(3) Der Quelltexteditor zum Editieren des Programmes (hinter dem Formularfenster).



- **Das Startformular**

Bereits unmittelbar nach Programmstart wird ein leeres Formular namens *Form1* angeboten, das gewissermaßen als Behälter (Container) für die darauf zu platzierenden Komponenten dient. Dieses Formular ist standardmäßig auf der oberen Randleiste mit den WINDOWS-typischen Checkboxen zum Verkleinern auf Symbolgröße, Vergrößern/Verkleinern und Schließen ausgestattet. Änderungen des Aussehens können mit Hilfe des Objektinspektors vorgenommen werden.

- **Die Komponentenpalette**

Komponenten (manchmal auch als Dialog- bzw. Steuerelemente bezeichnet) sind die Grundbausteine eines Anwendungsprogrammes (Applikation) und entstammen der **VCL** (Visual Component Library) von DELPHI. Das umfangreiche Sortiment ist in verschiedene Kategorien aufgeteilt, die getrennt in Registerseiten untergebracht sind. Die gewünschte Komponente wird mit Hilfe der Maus in der Komponentenpalette ausgewählt und auf dem Formular platziert. Dadurch erzeugt man für ein Programm auf einfachste Weise sein sichtbares Design. Einige Beispiele für Standard-Komponenten sind: ***MainMenu*** (Menüleiste), ***Label*** (Textanzeige), ***Edit*** (einzeilige Texteingabe und Ausgabe), ***Memo*** (mehrzeilige Texteingabe und Ausgabe), ***Button*** (Schaltknopf), ***RadioButton*** (Ja/Nein-Schaltknopf), ***ListBox*** (Auswahlliste), ***ComboBox*** (editierbare Auswahlliste), ***Scrollbar*** (Bildlaufleiste) usw.

DELPHI fügt automatisch für alle im Formular platzierten Komponenten deren entsprechende Objektdefinitionen in den Quelltext des Programmes ein. Um die Deklaration dieser Objekte braucht sich der Programmierer nicht zu kümmern. Diese Arbeit nimmt ihm DELPHI ab.

- **Der Objektinspektor**

Mit dem Objektinspektor werden die **Eigenschaften** (Properties) der Komponenten verändert und die **Ereignisse** (Events), auf welche die Komponenten reagieren sollen, ausgewählt.

In der obersten Zeile des Inspektors (Objektselektor) erscheinen Name und Typ der gerade angeklickten Komponente (bzw. des Formulars). Man kann aber auch mit einer Rollbox die gewünschte Auswahl treffen. Jedem ausgewählten Objekt werden bestimmte Eigenschaften und Ereignisse zugeordnet. Dafür besitzt der Objektinspektor zwei getrennte Registerseiten.

In der linken Spalte der **Eigenschaften**-Seite findet man die Eigenschaften und in der rechten die zugehörigen Standardeinstellungen, die wahlweise veränderbar sind. Meistens wird dafür eine Auswahl angeboten, z.B. für alle möglichen Farbeinstellungen (*Color*). Manchmal muss man aber auch direkt etwas eintasten, z.B. für die Beschriftung (*Caption*). Das einigen Eigenschaften vorangestellte Plus (+) weist darauf hin, dass es sich hier um einen Objekttyp handelt, der sich erst eine (oder auch mehrere) Ebenen tiefer in einzelne Eigenschaften aufsplitten lässt. Durch Doppelklicken öffnet man diese "Sub-Properties" (Untereigenschaften). Einige Beispiele für Eigenschaften sind: ***Name*** (Objektname), ***Align*** (Ausrichtung des Objektes in seinem Fenster: alTop, alBottom, alLeft, alRight), ***Caption*** (Beschriftung), ***Color*** (Farbe), ***+Font*** (Schriftart), ***Visible*** (Sichtbarkeit des Objektes), ***ReadOnly*** (Schreibschutz), ***Enabled*** (Reaktion auf Tastatur und Maus) usw.

Die zweite Registerseite des Objektinspektors zeigt die **Ereignisse**, auf welche ein ausgewähltes Objekt reagieren kann. Das *OnClick*-Ereignis tritt beispielsweise dann ein, wenn mit der Maus auf die Komponente geklickt wird. Für die verschiedenen Ereignisse kann der Programmierer so genannte Ereignisbehandlungsroutinen (Event Handler) schreiben. Diese müssen mit dem Quelltexteditor erstellt werden. Durch einen Doppelklick in das Wertefeld neben einem Ereignis wird im Quelltext eine Programmschablone für die Ereignisbehandlungsroutine angeboten. In diesen von DELPHI automatisch bereitgestellten Coderahmen braucht der Programmierer nur noch die gewünschten Programmbefehle zu schreiben. Einige Beispiele für Events sind Mausereignisse und Tastaturereignisse: ***OnClick*** (einfacher Mausklick), ***OnDblClick*** (doppelter Klick), ***OnMouseUp*** (Loslassen der Maustaste), ***OnKeyPress*** (Niederdrücken einer Tastaturtaste), ***OnKeyUp*** (Loslassen einer Tastaturtaste) usw.

- **Menüfunktionen und die Symbolleiste**

Häufig benötigte Menüaufrufe liegen auf der Symbolleiste. Das Hauptmenü enthält die wichtigen Aufrufe zum Öffnen, Speichern, Bearbeiten, Editieren, Compilieren, Linken und Ausführen der Programmdateien. Wie bei allen WINDOWS-Applikationen lässt sich jeder Menüpunkt auch ohne Maus durch Eintippen von <*Alt*> und des jeweils unterstrichenen Buchstabens auswählen.

- **Der Quelltexteditor**

Der Aufruf dieses Fensters erfolgt über das Menü <*Ansicht/Code-Explorer*>. Durch dieses Fenster gelangt man in die Welt der Objekt-Pascal-Programmierung. Hier steht ein Texteditor zum Editieren des Programmcodes zur Verfügung. Über das Menü <*Ansicht/Projektverwaltung*> kann man sich einen Überblick über die vorhandenen Module verschaffen. Bei der Programmedition ist zu beachten, dass die im System reservierten Wörter (d.h. Anweisungen mit einer vordefinierten Bedeutung) fett gedruckt erscheinen.

- **Die Online-Hilfe**

DELPHI bietet, wie jede größere und professionelle WINDOWS-Applikation, ein Online-Hilfe-System. Im Menü <*?*> findet man zu sämtlichen Sprachelementen eine ausführliche Beschreibung. Wenn die <*F1*>-Taste gedrückt wird, erhält man eine dem Kontext entsprechende Hilfestellung.

[1.03] Objektorientiertes Programmieren

Prinzipiell kann man die ganze Welt objektorientiert betrachten. Objekte sind zum Beispiel ein *Fernseher* oder ein *Tisch*. Objekte besitzen verschiedene Eigenschaften. Einen *Fernseher* kann man beispielsweise durch seine "Farbe" (z.B. schwarz) und seine "Bilddiagonale" (z.B. 82 cm) beschreiben. Außerdem gibt es für Objekte bestimmte Ereignisse, auf die dann eine bestimmte Aktion folgt. Beim Fernseher ist so ein Ereignis "auf den Einschaltknopf drücken". Durch dieses Ereignis wird immer dieselbe Aktion ausgelöst, nämlich der Fernseher wird eingeschaltet, und es erscheint ein Bild. Einen *Tisch* hingegen kann man durch die Eigenschaften "Tischplatte" (z.B. rund) und "Tischbeine" (z.B. drei) charakterisieren. Zulässige Aktionen sind beispielsweise "den Tisch verschieben" oder "den Tisch decken".

Unter WINDOWS laufen alle Aktionen objekt- und ereignisorientiert ab. Für jede Aktivität ist ein eigener Programmteil zuständig, der weitgehend unabhängig von anderen Programmteilen agieren kann und muss. Ein WINDOWS-Objekt ist zum Beispiel eine Datei. Diese hat die Eigenschaften Name und Größe. Ein Ereignis wäre der Doppelklick mit der Maus auf das Dateisymbol, wodurch diese geöffnet, d.h. ein aktiver Zugriff auf die Datei ermöglicht wird.

Das Programmiersystem DELPHI basiert ebenfalls auf dieser Philosophie. Eines der wichtigsten Objekte von DELPHI ist das Formular. Darunter versteht man das WINDOWS-Fenster, in dem eine DELPHI-Anwendung ausgeführt wird, d.h. etwaige Daten eingegeben, verarbeitet und ausgegeben werden. In einem Formular können weitere untergeordnete Formulare, Komponenten (z.B. ein Menü oder ein Objektsymbol), Texte und Grafiken enthalten sein.

Unter den **Eigenschaften** (Properties) versteht man die Attribute von Objekten, wie zum Beispiel die *Höhe* (Height) und die *Breite* (Width) eines Formulars oder die *Farbe* (Color) eines Textfeldes. Jedes Objekt verfügt über einen eigenen Satz von Eigenschaften.

Methoden sind die in einem Objekt definierten Funktionen und Prozeduren, welche die Aktion bzw. das Verhalten beim Eintreffen einer Nachricht über ein Ereignis bestimmen. Eine Methode bestimmt zum Beispiel das Verhalten eines Objekts bei einem Mausklick oder einer Tastatureingabe.

Im Unterschied zu den Eigenschaften, die eine "statische" Beschreibung liefern (deren Ergebnis ist sofort sichtbar), bestimmen die Methoden die "dynamischen" Fähigkeiten des Objekts.

Ereignisse (Events), die meist an externen Geräten stattfinden, erzeugen Nachrichten (Messages). Diese werden von WINDOWS in einer eigenen Warteschlange (Queue) zwischengespeichert, von wo sie das DELPHI-Programm übernimmt. Ereignisse stellen die eigentliche Verbindung zwischen Anwender und Programm dar. So ruft zum Beispiel das Anklicken eines Schaltknopfes mit der Maustaste ein *OnClick*-Ereignis hervor. Aufgabe eines DELPHI-Programms ist es nun, auf solche Ereignisse gemäß dem Wunsch des Anwenders zu reagieren. Dies geschieht in so genannten Ereignisbehandlungsroutinen (Event Handler).

- **Vererbung und Polymorphie**

Betrachten wir zunächst als Beispiel die beiden Objekte *Fahrrad* und *Auto*. So unterschiedlich diese auch sein mögen, so haben sie auch gewisse Eigenschaften und Methoden (Aktionen) gemeinsam, etwa die Eigenschaften "Geschwindigkeit", "Lenkung", "Räder" und die Methoden "Signalgebung", "Beschleunigen" und "Bremsen". Diese gleich lautenden Merkmale können zu der neuen und allgemeineren Objektklasse *Fahrzeug* zusammengefasst werden. *Fahrzeug* ist somit ein Vorfahre von *Fahrrad* und *Auto*, welche ihrerseits als Nachfahren bezeichnet werden.

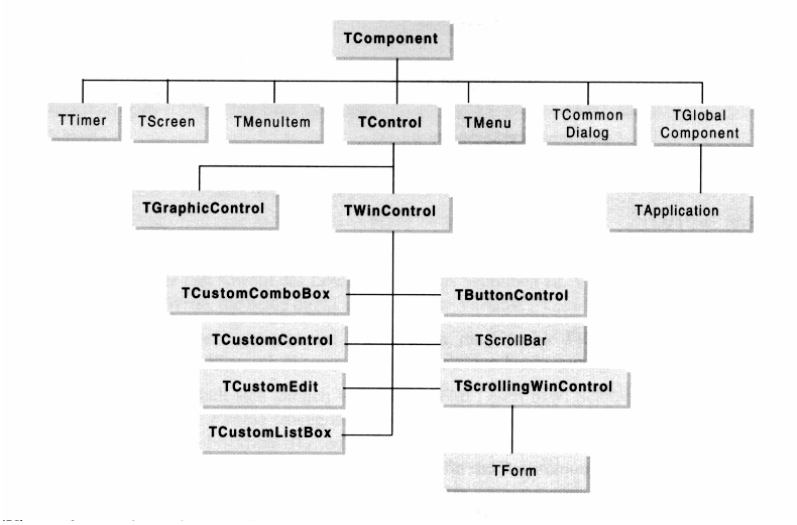
Die Nachfahren erben von ihren Vorfahren deren Merkmale. Diese können bei der **Vererbung** ungeändert übernommen werden. Eine solche statische Eigenschaft ist in unserem Fall die "Geschwindigkeit". Oft aber sind gleich lautende Methoden für jede nachfolgende Objektklasse unterschiedlich zu realisieren. Die Methode "Beschleunigen" wird, trotz gleicher Bedeutung und gleichen Namens, offenkundig für *Fahrrad* und *Auto* sehr unterschiedlich durchgeführt. Beim *Fahrrad* wird das "Beschleunigen" durch ein intensiveres Strampeln bewirkt, beim *Auto* durch einen stärkeren Druck auf das Gaspedal. In beiden Fällen erfolgt derselbe Effekt der Beschleunigung, aber durchaus in verschiedener Form. In diesem Sinne ist auch die Methode "Signalgebung" zu verstehen, als "Klingeln" oder als "Hupen". Diesen Sachverhalt bezeichnet man als **Polymorphie** und die entsprechenden Methoden heißen virtuell. Durch das Prinzip der Vererbung von statischen und virtuellen Merkmalen ist es möglich, den Objektbestand eines Weltbereiches übersichtlich und hierarchisch (baumartig) anzuordnen und zu beschreiben. Klarerweise werden in den Nachfahren zu den vererbten Merkmalen oftmals noch zusätzliche spezifische Merkmale hinzugefügt. Beispielsweise enthält das Objekt *Auto* auch noch "Sitze" und ein "Dach", das *Fahrrad* hingegen einen "Sattel". Dies wird als Objekt-**Erweiterung** bezeichnet.

Die Sprachobjekte von DELPHI sind in dieser Art und Weise hierarchisch konzipiert. Beginnend mit einer sehr allgemeinen Objektklasse werden immer speziellere untergeordnete Objekttypen definiert. Ein konkretes Objekt, welches einem bestimmten Objekttyp entspricht, wird dann als Vertreter (Instanz) des Objekttyps bezeichnet. Die bekannten Komponenten von WINDOWS sind solche Objekte mit bestimmten Eigenschaften und Ereignissen bzw. Methoden. Der allgemeine Vorfahre aller Komponenten ist *TComponent*. Von diesem leiten sich alle anderen Komponententypen ab, wie beispielsweise die Anzeigefelder *TLabel*, die Texteingabefelder *TEdit*, die Schaltknöpfe *TButton*, die Menükomponenten *TMenu*, die Formulare *TForm*, das allgemeine Programmobjekt *TApplication* und viele mehr. Das vorangestellte "T" bezeichnet immer den Typ (Klasse) eines Objektes.

Die Deklaration beispielsweise eines Objekttyps ***TForm1*** als Nachfahre von ***TForm*** erfolgt durch die Anweisung ***type TForm1 = class(TForm)***, wobei ***class*** die Bezeichnung für eine allgemeinere Objektklasse ist.

Die eigentliche Definition einer Objektvariablen ***Form1*** vom Typ ***TForm1*** bewirkt die Deklaration ***var Form1 : TForm1.*** Diese Schreibweise entspricht den alltäglichen Sprachgepflogenheiten. Viele Menschen weisen den sie umgebenden Objekten bestimmte Namen zu. Beispielsweise heißt ein Auto "Herbi". Dieses konkrete Auto ist damit ein Vertreter (Instanz) des allgemeinen Begriffes *Auto*. Objekttyp und Objektinstanz spiegeln den Unterschied zwischen einer allgemeinen Begriffskategorie und dem konkret gemeinten Gegenstand wieder.

Die Grafik zeigt den Stammbaum der vordefinierten Komponententypen des Systems. Über ***TComponent*** befinden sich die beiden Klassen ***TPersistent*** und ***Exception***. Darüber liegt schließlich die allgemeinste Objektklasse ***TObjekt*** (hier nicht eingezeichnet).



[1.04] Stufen der Programmentwicklung

DELPHI erlaubt bei der Erstellung eines Programmes eine systematische Vorgehensweise in folgenden drei Phasen:

> ***Visueller Entwurf der Bedienungsoberfläche***
> ***Zuweisen der Objekteigenschaften***
> ***Verknüpfen der Objekte mit Ereignissen***

In der ersten Phase ist die Ausgangsbasis das vom System bereitgestellte Startformular, das mit verschiedenen Komponenten, wie Editierfenster (*Edits*) oder Schaltknöpfen (*Buttons*), ausgestattet werden soll. Unter DELPHI findet man ein Angebot vieler WINDOWS-typischer Komponenten. Diese werden von einer Komponentenpalette ausgewählt, mittels Maus an ihre endgültige Position platziert und falls nötig in ihrer Größe verändert.

In der zweiten Phase braucht man sich nur mehr um jene Objekteigenschaften zu kümmern, die von den Voreinstellungen (Defaults) abweichen. DELPHI schreibt automatisch für jede neu eingefügte Komponente die entsprechende Objektdefinition in den Quelltext und die Objekteigenschaften in eine spezifische Formulardatei, deren Namenserweiterung immer ***dfm*** ist.

In der dritten Phase wird festgelegt, wie die vorhandenen Objekte auf bestimmte Ereignisse zu reagieren haben. DELPHI stellt auch hier im Quelltext eine vorgefertigte Programmschablone für jede zum jeweiligen Objekt passende Ereignisbehandlungsroutine zur Verfügung. Der Programmierer füllt diese Schablone mit seinen Programmbefehlen aus.

[1.05] Ein Einführungsbeispiel (prog1)

Wir wollen nun ein kleines Anwendungsprogramm (Applikation) erstellen, um die wichtigsten Grundlagen an einem praktischen Beispiel zu demonstrieren. Auf einem Formular mit dem Titel 'Begrüßung' wird ein Knopf mit der Beschriftung 'Click' platziert. Wird dieser Knopf mit der Maus angeklickt, dann soll der Text 'Hallo Welt' im Formular erscheinen. Bei einem neuerlichen Anklicken verschwindet der Text wieder. Der Knopf dient so als Wechselschalter (Toggle).

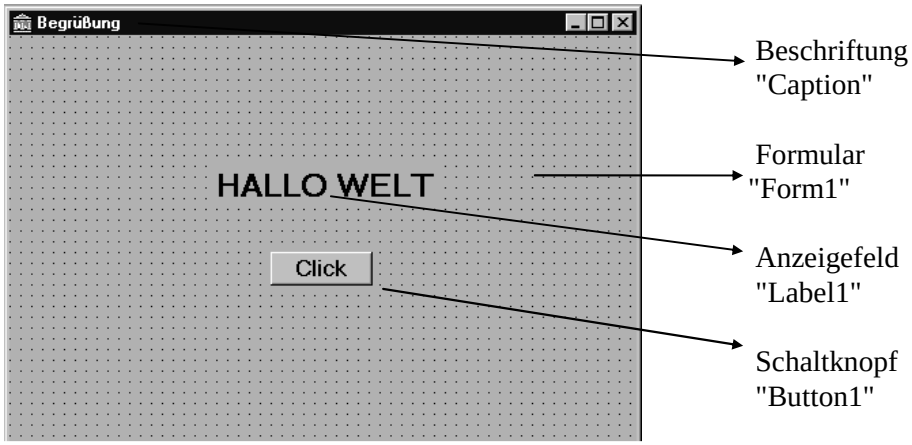
In der ersten Phase erfolgt der visuelle Entwurf der Bedienungsoberfläche für das Programm. Nach dem Start von DELPHI wird vom System standardmäßig ein Startformular (*Form1*) zur Verfügung gestellt, welches in der Bildschirmmitte zu sehen ist. Dieses Formular stellt die sichtbare Bedienungsoberfläche für unser Programm dar. Außer diesem Startformular brauchen wir einen Schaltknopf (*Button*), der auf dem Formular platziert ist.

Alle Elemente, die wir in Formulare einbauen können, finden wir in der Komponentenpalette. Die Komponentenpalette besteht aus mehreren Registern, zwischen denen wir durch Anklicken des jeweiligen Eintrages am oberen Rand wechseln können.

Das Symbol für die Komponente <*Button*> befindet sich im Register <*Standard*>. Zuerst muss durch Anklicken das Register <*Standard*> aufgeschlagen werden. Um den Namen bzw. die Funktion eines Symbols zu erfahren, bewegen wir die Maus auf dieses Symbol und verweilen eine Weile auf diesem. Dann erscheint der Name. In unserem Fall brauchen wir das Symbol mit dem Namen <*Button*>. Um eine Komponente auszuwählen, muss diese angeklickt werden. Wir wählen so durch Anklicken die Komponente <*Button*> aus. Um diese Komponente auf dem Formular zu platzieren, bewegen wir die Maus an die gewünschte Stelle am Formular und klicken wieder die linke Maustaste.

Der Vorgang der Größenänderung einer Komponente ist analog zur Größenänderung von WINDOWS-Fenstern. Zuerst müssen wir die zu bearbeitende Komponente auswählen, indem wir diese anklicken. Um die Größe des Schaltknopfes zu ändern, bewegen wir den Mauszeiger auf eines der kleinen schwarzen Quadrate (Anfasser) am Rand, drücken die linke Maustaste und ziehen den Button bei gedrückter Maustaste auf die gewünschte Größe. Um die Position des Buttons zu verändern, klicken wir mit der linken Maustaste in die graue Fläche des Schaltknopfes und verschieben diesen bei festgehaltener Maustaste. Am Zielort angelangt, lassen wir die Maustaste wieder los (Drag and Drop). Um eine platzierte Komponente wieder zu entfernen, klicken wir einfach auf diese und drücken dann die <Entf>-Taste. Dann werden automatisch auch im Quelltext des Programmes die entsprechenden Objektdefinitionen gelöscht.

Die zweite Komponente, die wir für unser Beispiel brauchen, ist ein Anzeigefeld (Label), welches schließlich die Zeichenkette (String) 'Hallo Welt' aufnehmen soll. Diese Komponente <*Label*> befindet sich ebenfalls im Register <*Standard*>. Wir wählen und positionieren das Anzeigefeld mit dem gleichen Verfahren, welches wir beim Schaltknopf angewendet haben. Nun sind alle benötigten Komponenten im Formular platziert und somit die Benutzeroberfläche des Programms fertig gestellt. Die Abbildung zeigt das Programmformular:



Grundsätzlich muss erwähnt werden, dass man zu den einzelnen Komponenten in einem Formular entweder mit der Maus oder mit der Tabulator-Taste (*Tab* bzw. *Shift Tab*) gelangen kann. Die Reihenfolge wird durch die Eigenschaft *TabOrder* geregelt. Das wichtige Merkmal *Focus* lenkt den Zugriff auf ein bestimmtes Objekt. Die Methode *Button1.SetFocus* beispielsweise setzt den Mauscursor auf den Schaltknopf *Button1*.

In der zweiten Phase werden den Komponenten ihre Eigenschaften zugewiesen. Das erfolgt mit Hilfe des Objektinspektors. Dieser befindet sich in einem eigenen Fenster auf der linken Bildschirmseite. Zuerst wollen wir die von den vordefinierten Standardeigenschaften abweichenden Eigenschaften des Formulars verändern.

Hierzu müssen wir zuerst das Formular mit dem Namen 'Form1' im Objektselektorfeld auswählen. Das geschieht durch Anklicken des Formulars. Falls wir uns auf der Ereignis-Seite befinden, klicken wir auf den Eintrag *<Eigenschaften>* links oben. Auf der Eigenschaften-Seite des Objektinspektors sind verschiedene Eigenschaften aufgelistet. Zuerst wollen wir die Farbe ändern. Hierzu muss der Eintrag *<Color>* angeklickt werden und aus der Rollbox die graugelbe Farbe *<clInfoBk>* ausgewählt werden. Nun wollen wir den Titel des Formulars ändern. Hierzu müssen wir im Eintrag *<Caption>* (Beschriftung) den gewünschten Titel *'Begrüßung'* eintippen. Für die Schrift im Formular wählen wir mittels *<Font>* die Schrift *<SansSerif, normal, 10>*. Schließlich zentrieren wir mit *<Position>* das Formular am Desktop (*<poDesktopCenter>*). Als nächstes ändern wir den Text auf dem Schaltknopf. Wiederum muss die Komponente ausgewählt (einfach angeklickt) und der entsprechende Eintrag auf der Eigenschaften-Seite des Objektinspektors verändert werden (*<Caption> = 'Click'*). In der gleichen Art und Weise beschriften wir das Anzeigefeld *<Label1>* mit dem Text *'Hallo Welt'*. Außerdem machen wir dieses Anzeigefeld noch unsichtbar, indem wir die Eigenschaft *<Visible>* auf *False* setzen. Damit sind den Objekten ihre Eigenschaften zugeordnet.

In der dritten Phase sollen nun die Objekte mit Ereignissen verknüpft werden. Der Programmablauf wird mittels Anweisungen geregelt, die sich in Untereinheiten des Programms, den so genannten Units (Einheiten), befinden. Wenn der Button mit der Maus angeklickt wird, soll im Anzeigefeld *<Label1>* der Text *'Hallo Welt'* abwechselnd erscheinen und wieder verschwinden. Um dies zu programmieren, klicken wir **einfach** im Formular auf die Fläche des Schaltknopfes *<Button1>*. Dadurch wird dieser dem Objektinspektor übermittelt. Wir gehen dort auf die Ereignis-Seite und wählen das Ereignis *<OnClick>* aus. Durch **Doppelklick** auf das Wertefeld daneben gelangen wir in den Quelltext der Programmeinheit *Unit1*. Dort wird eine vorgefertigte Programmschablone für die ausgewählte Routine zur Verfügung gestellt. Wir brauchen in diese Schablone nur mehr zwischen *begin* und *end* die Zeile *Label1.Visible := Not Label1.Visible* einfügen. So wird das Anzeigefeld abwechselnd sichtbar und unsichtbar. Eigenschaften einer Komponete werden durch Voranstellen des Komponentennamens und eines Punktes geschrieben. Beim Entwurf ist es hilfreich, Tabellen für die verwendeten Objekte und Ereignisse anzulegen.

Objekttabelle

Objektname	Eigenschaft	Wert
Form1	Caption	'Begrüßung'
	Color	clInfoBk
	Font	SansSerif, normal,10
	Position	poDesktopCenter
Button1	Caption	'Click'
Label1	Caption	'Hallo Welt'
	Visible	False

Ereignistabelle

Benutzer-Aktion	WINDOWS-Ereignis	Programm-Routine	Programm-Reaktion
Button1 anklicken	OnClick	Button1Click	'Hallo Welt' anzeigen

Unser kleines Programm ist somit fertig. Bevor wir es nun compilieren und starten, sollte es noch gespeichert werden. Wie in WINDOWS-Applikationen üblich, wird das Speichern über das Menü *<Datei>* gesteuert. In DELPHI werden alle zu einem Programm gehörenden Dateien (z.B. Formulare, Units usw.) in einem Projekt organisiert. Um nun das Projekt mit all seinen Bestandteilen zu speichern, müssen wir uns einen passenden Dateinamen überlegen. Standardmäßig heißt die aktuelle Programm-Unit immer *unit1.pas* und das Programm-Projekt (Hauptprogramm) immer *project1.dpr*. Das alles wird standardmäßig in einem bestimmten Systemordner abgespeichert. Unsere Delphiprogramme sollen aber alle im eigens dafür angelegten Ordner *progdata* gespeichert werden und unser erstes Projekt soll *prog1.dpr* und die entsprechende Unit soll *prog1_u.pas* heißen. Dazu wird zuerst die Unit mit dem Menü *<Datei/Speichern unter>* und dann das Projekt mittels *<Datei/Projekt speichern>* im Ordner *progdata* abgespeichert.

Nachdem das Projekt gespeichert ist, steht einem Testlauf nichts mehr im Weg. Hierzu befindet sich im Menü <*Start*> der Befehl <*Start*>. Das gleiche Ergebnis wird durch Anklicken der entsprechenden Schaltfläche in der Symbolleiste erreicht. Um eine ausführbare Programmdatei *prog1.exe* ohne Testlauf zu erhalten, wird das Projekt mittels <*Project/Compile*> compiliert. Tritt dabei ein Fehler auf (der Compiler kann einen Befehl nicht in den Binärcode übersetzen), erscheint eine Fehlermeldung im unteren Teil des Quelltexteditors. Wir berichtigen die fehlerhafte Zeile und speichern danach erneut mittels <*Datei/Alles speichern*>. Wir starten noch einmal und es öffnet sich das Programmformular. Wenn wir den Schaltknopf mit der Aufschrift 'Click' betätigen, erscheint der Schriftzug 'Hallo Welt' am Bildschirm. Ein neuerliches Anklicken bringt den Text wieder zum Verschwinden. Beenden können wir diese Anwendung durch eine der unter WINDOWS üblichen Methoden. (Über das Fenstermenü, das durch Anklicken des Icons im linken oberen Eck des Fensters erscheint oder mit Hilfe des X-Symbols rechts oben).

Ein komplettes DELPHI-Projekt besteht aus mehreren Dateien: *project1.dpr* (Quelltext des Hauptmoduls), *unit1.pas* (Quelltext der Unit), *unit1.dfm* (Formularkomponenten), *unit1.dcu* (Maschinencode der Unit), *project1.res* (Windows-Ressourcendatei), *project1.dof* (Compiler-Einstellungen), *project1.cfg* (Konfigurationsdatei) und schließlich *project1.exe* (Kompiliertes Programm).

[1.06] Was passiert wirklich?

DELPHI bietet ein visuelles Programmiersystem an. Ohne tieferes Hintergrundwissen ist es auch dem Anfänger möglich, recht ansehnliche Programme zu erstellen. Um eine moderne Programmoberfläche zu gestalten, ist es nicht notwendig, den dahinter stehenden Programmcode genau zu kennen. Diesen stellt DELPHI automatisch zur Verfügung.

Wir wollen uns nun anschauen, was bei unserem kleinen Beispiel im Hintergrund wirklich alles passiert. Bereits nach dem Start von DELPHI wird ein einsatzfähiges Programm angeboten, das ein Leerformular (Form1) erzeugt. Der gesamte Quelltext besteht aus dem Hauptprogramm (Project1) und einer Unit (Unit1). Der Entwickler braucht die Programmschablone nur mehr durch seine Befehle zu ergänzen. Durch das Unit-Konzept wird eine modulare Programmierung ermöglicht. Das heißt, dass nicht der ganze Programmcode eines Programms in einer Datei untergebracht, sondern dass er auf Untereinheiten (hier Units genannt) verteilt wird. So kann man den Programmcode nach logischer Zusammengehörigkeit in Module aufteilen. Die einzelnen Units können in voneinander verschiedene Anwendungsprogramme eingebunden werden. Der Anwender darf eigene Units entwickeln. Es gibt aber auch eine ganze Reihe vordefinierter Units.

Jedes Programm besteht aus einem Hauptprogramm (hier Projekt genannt), welches das Programm startet, und mehreren Units, wobei sehr viele Units bereits vordefiniert sind und mit Hilfe der ***uses***-Anweisung in das Hauptprogramm oder andere Units eingebunden werden können. Die einzelnen Programmroutinen (Prozeduren und Funktionen) einer Unit werden dann in dem Modul verwendet, in welchem die Unit eingebunden ist. Wird zum Beispiel ein Button auf einem Formular platziert, so erfordert dies bereits umfangreiche Routinen, in denen dieser Button definiert wird. Automatisch wird das Objekt *Button* dann der entsprechenden Unit hinzugefügt. Der Programmierer braucht sich darum nicht mehr zu kümmern.

Wir wollen uns nun zuerst das Hauptprogramm ansehen. An diesem brauchen normalerweise keine händischen Änderungen vorgenommen werden. Man kann sich das Hauptprogramm über das Menü <*Ansicht / Code-Explorer*> anschauen. In unserem Fall gibt es zwei Einträge, zwischen denen zu wählen ist: dem Hauptprogramm und einer Unit. Im Folgenden sollen die einzelnen Anweisungen nur kurz beschrieben werden. Eine genauere Erklärung der verschiedenen Programmelemente wird in den nachfolgenden Buchkapiteln gegeben. Der prinzipielle Aufbau eines Hauptprogrammes ist immer derselbe. Das Zeichen // leitet einen Kommentar für den Rest der Zeile ein. Kommentare werden vom Compiler überlesen. Sie dienen einzig und allein zur Information und näheren Beschreibung des Quelltextes. Solche Kommentare können auch in geschweiften Klammern stehen. Die fett geschriebenen Bezeichner sind in DELPHI vordefiniert, die nicht fett geschriebenen Bezeichner sind vom Programmierer definierte Elemente. Jede Anweisungseinheit wird von einem Strichpunkt begrenzt!

Quellcode des Hauptprogrammes:

```
program prog1;                                  // [1]  Kopfzeile des Programmes

uses                                            // [2]  Unit-Einbindung.
  Forms,                                        //      Das zur Unit gehörige
  prog1_u in 'prog1_u.pas' {Form1};             //      Formular steht in Klammern.

{$R *.RES}                                      // [3]  Compiler-Anweisung

begin                                           // [4]  Eigentlicher Befehlsblock
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Quellcode der Unit:

```
unit prog1_u;                                   // [5]  Kopfzeile der Unit

interface                                       // [6]  Interface-Teil

  uses                                          // [7]  Unit-Einbindung
    Windows, Messages, SysUtils, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

  type                                          // [8]  Objekt-Typisierung
    TForm1 = class(TForm)
      Button1: TButton;
      Label1:  TLabel;
      procedure Button1Click(Sender: TObject);
    private
      { Private-Deklarationen }
    public
      { Public-Deklarationen }
  end;

  var
    Form1: TForm1;                              // [9]  Variablen-Definition

implementation                                  // [10] Implementations-Teil

{$R *.DFM}                                      // [11] Compiler-Anweisung

  procedure TForm1.Button1Click(Sender: TObject);   // [12] Prozeduren-Code
  begin
    Label1.Visible := Not Label1.Visible;
  end;

end.                                            // [13] Programmende.
```

[1] Kopfzeile des Hauptprogrammes. Sie beginnt immer mit dem reservierten Wort *program*, gefolgt von dem Programmnamen (z.B. *prog1*).

[2] Die *uses*-Anweisung ermöglicht die Einbindung von Units, das sind die vordefinierte Unit *Forms* und die selbst definierte Unit *prog1_u*, in welcher das Objekt *Form1* definiert ist.

[3] Anweisung an den Compiler zum Zugriff auf alle Ressourcen-Dateien (*.RES), welche DELPHI dem Programmierer zur Verfügung stellt.

[4] Anfang eines Befehlsblockes, der immer durch die reservierten Worte *begin* und *end* eingegrenzt sein muss. Der Punkt hinter *end* signalisiert dem Compiler das effektive Ende des Programmcodes. Der Befehlsblock selbst besteht aus drei Methodenaufrufen des vordefinierten Objektes *Application*. Dabei folgt der Name der Methode, durch einen Punkt getrennt, dem Namen des Objektes, welches die Methode enthält. Zuerst wird initialisiert, dann das Formular erzeugt und schließlich das Programm ausgeführt.

[5] Kopfzeile der Unit. Sie beginnt mit dem reservierten Wort *unit*, gefolgt von ihrem Namen.

[6] Beginn des Interface-Teiles (Routinen-Schnittstelle) der Unit. Hier werden alle Sprachelemente angeführt, auf welche von außerhalb der Unit zugegriffen werden kann. Diese heißen **global**; es sind Objekttypisierungen, Variablendefinitionen und die Kopfzeilen von Programm-Routinen (so genannten Prozeduren oder Funktionen).

[7] Einbindung von vordefinierten Standard-Units. Die Basisunit *System*, die automatisch eingebunden ist, enthält die wichtigsten Sprachelemente von DELPHI (Laufzeitbibliothek). In der Unit *Forms* beispielsweise befinden sich die Hilfsmittel für die Arbeit mit Formularen.

[8] Definition eines Objekttyps *TForm1* als Nachfahre des vordefinierten Typs *TForm*. Zuerst werden die Komponenten *Button1* und *Label1* festgesetzt, welche Instanzen von dem Schaltknopftyp *TButton* und dem Anzeigefeldtyp *TLabel* sind. Dann folgt der Prozedurenkopf jener Ereignisbehandlungsroutine, welche durch Doppelklick im Objektinspektor erzeugt worden ist. Der Programmcode dieser Routine muss im Implementations-Teil der Unit stehen. Im Prozedurenkopf wird zusätzlich der Parameter *Sender* vom vordefinierten Typ *TObjekt* angegeben. Über diesen Parameter kann jene Komponente abgefragt werden, an der ein Ereignis stattgefunden hat (Nachrichten-Sender). Außerdem ist es möglich, Objektmerkmale zu sperren, sodass sie zwar von außen gelesen, aber nicht geändert werden können. Diese werden mit Read-only (*privat*) bezeichnet. Andernfalls sind sie öffentlich (*public*). Beim objektorientierten Programmieren ist es üblich, zum Lesen oder Ändern von Objektmerkmalen eigene zuständige Methoden (Routinen) zu erstellen.

[9] Definition der Objektvariablen *Form1* als Vertreter (Instanz) des weiter oben definierten Objekttyps *TForm1*.

[10] Beginn des Implementations-Teiles (Routinen-Codierung) der Unit. Hier wird der eigentliche Befehlscode jener Routinen geschrieben, deren Kopfzeilen im Interface-Teil angeführt worden sind. Auch können hier zusätzliche **lokale Sprachelemente** definiert werden, die dann von außerhalb nicht verwendbar sind, sondern nur innerhalb der Unit gelten.

[11] Anweisung an den Compiler zum Zugriff auf alle Formularinformations-Dateien (*.DFM), welche DELPHI dem Programmierer zur Verfügung stellt.

[12] Die Routine zur Ereignisbeantwortung eines Mausklicks auf den Schaltknopf *Button1*. Dabei wird der Text 'Hallo Welt' am Bildschirm abwechselnd sichtbar und unsichtbar. Der Parameter Sender übermittelt jene Komponente, wo ein Ereignis stattgefunden hat. Eine Abfrage des Senders ist hier nicht nötig, weil *Button1* selbst der Sender ist.

[13] Endgültiges Ende der Unit. Wichtig ist dabei der Punkt hinter der *end*-Anweisung!

Hinter den Kulissen

Wie bereits erklärt, enthält DELPHI eine Hierarchie von vordefinierten Objekttypen, welche mit der allgemeinsten Klasse ***TObjekt*** beginnt. Wie jedes Objekt enthält ***TObjekt*** einerseits Objektattribute und andererseits Objektmethoden (siehe dazu auch Listingnummer [8]). Diese Merkmale werden auf alle abgeleiteten Objekte (Nachfahren) vererbt. Beispielsweise ist ***TApplication*** über ***TComponent*** und ***TPersistent*** von ***TObjekt*** abgeleitet. Diese wichtige Objektklasse enthält Attribute und Methoden, welche das Grundgerüst zur Verwaltung eines Programmprojektes sind (siehe dazu auch Listingnummer [4]). Die globale Objektvariable ***Application*** ist ein Vertreter (Instanz) vom Typ ***TApplication*** und steht jedem Programm automatisch zur Verfügung.

Die Methode ***Application.Initialize*** kann zur Belegung von wichtigen globalen Variablen mit Anfangswerten herangezogen werden und ***Application.Terminate*** beendet ein Programm. Die Methode ***Application.CreateForm(TForm1, Form1)*** erzeugt zur Laufzeit des Programmes jenes Formular ***Form1***, welches bei der Programmerstellung entwickelt wurde. Das Formular ***Form1*** ist ein Vertreter des selbst definierten Formulartyps ***TForm1*** in der Unit ***prog1_u***. Dieser Objekttyp seinerseits ist ein Nachfahre des allgemeineren Typs ***TForm*** (siehe dazu auch Listingnummer [8]). ***TForm*** wiederum ist weitschichtig abgeleitet von ***TControl*** und ***TControl*** schließlich ist ein Nachfahre von ***TObjekt***. Eine übersichtliche Darstellung der internen Objekthierarchie von DELPHI findet man im Buchabschnitt [1.03].

Der Browser im Menü *<Ansicht / Browser>* erlaubt es, die in einem Programm verwendeten Objektklassen grafisch darzustellen. Zusätzlich können alle Units, Prozeduren, Funktionen und globalen Variblenbezeichner des Programmes aufgelistet werden.

Um hinter die Kulissen eines entwickelten Formulares zu blicken, muss das Menü *<Datei/ Öffnen>* aufgerufen werden. Dort wählt man die entsprechende Formulardatei (z.B. ***Unit1.dfm***) aus. Die Erweiterung des Dateinamens ***dfm*** steht für **D**elphi**F**or**M**ular. Mit Hilfe dieser Menüauswahl wird der binär verschlüsselte Code des Objektes ***Form1*** am Bildschirm lesbar angezeigt, d.h., alle eingestellten Eigenschaften und Methoden der verwendeten Komponenten sind sichtbar.

```
object Form1: TForm1                    // Die Hauptkomponente Form1
  Left = 288                            // X-Koordinate der linken, oberen Ecke
  Top = 141                             // Y-Koordinate der linken, oberen Ecke
  Width = 533                           // Breite der Komponente in Pixeln
  Height = 417                          // Höhe der Komponente in Pixeln
  Caption = 'Begrüßung'                 // Beschriftung der Komponente
  Font.Charset = DEFAULT_CHARSET        // Eigenschaften der verwendeten Schrift
  Font.Color = clInfoBk
  Font.Height = -17
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 120
  TextHeight = 13
  position = poDesktopCenter
  object Label1: TLabel                 // Die Komponente Label1
    Left = 184
    Top = 120
    Width = 5
    Height = 24
    Caption = 'Hallo Welt'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -40
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    Visible = False
end
```

```
  object Button1: TButton                    // Die Komponente Button1
    Left = 208
    Top = 192
    Width = 105
    Height = 41
    Caption = 'Click'
    TabOrder = 0
    OnClick = Button1Click                   // Behandlungsroutine des OnClick-Ereignisses
  end
end
```

Die einzelnen Komponenten in der Formulardatei sind in folgender Notation dargestellt:

```
Object  Komponentenname : Klassenname (Objekttyp)
   Objekt-Eigenschaft  =  .....
   Ereignis-Methode    =  .....
End
```

Die Einbettung einer Komponente in eine andere entspricht den jeweiligen Besitzverhältnissen. So ist *Form1* das Behälterobjekt (Container) für *Button1*.

[1.07] Eigenschaften und Ereignisse

Zum Abschluss der allgemeinen Betrachtungen über DELPHI sollen die wichtigsten Eigenschaften (*Properties*) und Ereignisse (*Events*) kurz näher erläutert werden. Jedes Objekt des Programmiersystems besitzt eine Reihe von Eigenschaften und Methoden. Die Eigenschaften bestimmen Form und Funktionalität der Objekte, und die Methoden bestimmen die Reaktionen der Objekte auf die einzelnen Ereignisse. Diese Ereignisse können nach der Quelle ihres Entstehens in zwei Gruppen eingeteilt werden: Vom Anwender ausgelöste und vom Programm gesteuerte Ereignisse. Zur ersten Gruppe gehören Aktionen, welche der Anwender mit der Maus oder mit der Tastatur setzt. Zur zweiten Gruppe zählen Ereignisse, die beispielsweise beim Programmstart das Formular auf den Bildschirm bringen (*OnCreate*) und aktivieren (*OnActivate*). Für jedes Ereignis stellt DELPHI die Programmschablone einer Ereignisbehandlungsroutine (*Event Handler*) zur Verfügung, die zunächst nur aus einer Kopfzeile (*Header*) und den Befehlsworten *begin* und *end* besteht. Dazwischen kann der Programmierer dann seine eigenen Befehle schreiben, welche den Prozessor anweisen, wie auf das jeweilige Ereignis reagiert werden soll. In dieser Sichtweise bedeutet Programmieren nichts anderes als die Erstellung von sinnvollen Ereignisbehandlungsroutinen.

- **Wichtige Eigenschaften**

Die Objekte des Programmiersystems können in **visuelle** und **nicht visuelle** eingeteilt werden, das sind einerseits die sichtbaren Komponenten und andererseits die nicht sichtbaren Programmobjekte. Alle Komponenten besitzen gewisse Haupteigenschaften, welche auszugsweise am Beispiel eines Editierfeldes (*TEdit*) innerhalb eines Formulares (*TForm*) erklärt werden. Die meisten Eigenschaften können mit Hilfe interner Methoden sowohl gelesen (*read*) als auch geschrieben (*write*) werden. Diese internen Methoden sind im privaten Definitionsteil der Objekte geschützt.

Name	Eindeutiger Bezeichner der Komponente (z.B. *Edit1*).
Text	Editierbarer Textinhalt (z.B. 'Ich heiße Herbert').
Font	Schrifteigenschaften des Textes (Stil, Größe, Farbe ...).
Color	Farbe der Komponente (vom Datentyp *TColor*).
Top	Entfernung des linken, oberen Eckpunktes der Komponente vom oberen Formularrand, gemessen in Bildpunkten (Pixeln).
Left	Entfernung des linken, oberen Eckpunktes der Komponente vom linken Formularrand, gemessen in Bildpunkten (Pixeln).
Width	Breite der Komponente, gemessen in Bildpunkten (Pixeln).
Height	Höhe der Komponente, gemessen in Bildpunkten (Pixeln).

ReadOnly	Zugriff für "Nur Lesen" oder "Lesen und Schreiben" (*True* oder *False*).
Enabled	Sperre für Maus- und Tastaturereignisse (*True* oder *False*).
Visible	Sichtbarkeit der Komponente (*True* oder *False*).
TabOrder	Nummer der Reihenfolge des Zugriffes in der Komponentenmenge.
Tag	Beliebig nutzbare Zahl.

- **Wichte Ereignisse und ihre Behandlung**

Die häufigsten Ereignisse, welche ein Anwender auslöst, rühren von der Bedienung der Maus oder der Tastatur her. Es sollen hier die wichtigsten exemplarisch beschrieben werden.

(a) Mausereignisse

Bei der Maus müssen die **Aktion** (*Event: OnClick, OnDblClick, OnMouseDown, OnMouseUp, OnMouseMove*), die **Tasten** (*Button: mbLeft, mbRight*), der **Status** (*Shift: ssLeft = mbLeft pressed, ssRigth = mbRight pressed*) und die **Position** (*X, Y*) unterschieden werden. Die Programmierschablonen der zu den einzelnen Aktionen zugehörigen Ereignisbehandlungsroutinen haben alle eine ähnliche Form. Sie werden dann ausgelöst, wenn sich der Mauscursor (Positionszeiger am Bildschirm) über der entsprechenden Komponente befindet und dort eine Aktion durchgeführt wird. In den nachfolgenden Beispielen sind die Kopfzeilen der Routinen aufgelistet. Voraussetzung dabei ist, dass die Mausereignisse durch den Objektinspektor beim Entwurf dem Formular *Form1* zugeordnet worden sind:

procedure TForm1.FormClick(Sender: TObject);
Behandelt einen einfachen Klick. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat.

procedure TForm1.FormDblClick(Sender: TObject);
Behandelt einen doppelten Klick. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat.

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X,Y: Integer);
Behandelt das Niederdrücken einer Maustaste. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat. *Button* bestimmt die Maustaste, *Shift* ermittelt den Tastenstatus und *X, Y* liefert die Position.

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X,Y: Integer);
Behandelt das Loslassen einer Maustaste. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat. *Button* bestimmt die Maustaste, *Shift* ermittelt den Tastenstatus und *X, Y* liefert die Position.

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,Y: Integer);
Behandelt eine Bewegung der Maus. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat. *Button* fehlt hier. *Shift* ermittelt den Tastenstatus und *X, Y* liefert die Position.

(b) Tastaturereignisse

Wenn auf eine Komponente mit der Maus oder mit der TAB-Taste zugegriffen wird (d.h., sie hat den Focus), dann können für dieses Objekt drei Tastatur-Ereignisse (Keyboard-Events) unterschieden werden (OnKeyPress, OnKeyDown und OnKeyUp).

Im nachfolgenden Beispiel soll die fokussierte Komponente das Editierfeld *Edit1* innerhalb von *Form1* sein. Die entsprechenden Ereignisbehandlungsroutinen haben dann folgendes Aussehen:

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
Behandelt das Niederdrücken einer Taste. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat. *Key* übermittelt das gedrückte Tastenzeichen, sofern es ein ANSI-Zeichen ist. Sondertasten (*Shift, Strg, Alt, F1 ... F10*) werden nicht registriert.

procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
Behandelt das Niederdrücken einer Taste. Der Parameter *Sender* liefert die Komponente, an welcher das Ereignis stattgefunden hat. *Key* ist jetzt eine Zahl, die den Code von allen möglichen Tastenkombinationen übermittelt. *Shift* liefert den Tastaturstatus, d.h. ob die Shift-Taste *[ssShift]*, die Alt-Taste *[ssAlt]* oder die Strg-Taste *[ssCtrl]* gedrückt ist.

procedure TForm1.Edit1KeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
Behandelt das Loslassen einer Taste (analog zu *KeyDown*).

Zur bequemen Tastenerkennung stellt DELPHI so genannte *virtuelle Keycodes* für den Parameter *Key* zur Verfügung. So sind beispielsweise *vk_escape* und *vk_return* die Codes für die <Esc>-Taste und die <Enter>-Taste. Das sind die Codes 27 und 13. *vk_f1* ist der Code für <F1>.

Neben den oben beschriebenen Maus- und Tastaturereignissen kennt DELPHI eine Vielzahl weiterer Ereignisse und dazu die entsprechenden Behandlungsroutinen. So tritt *OnChange* ein, wenn eine Komponente verändert wird, und *OnEnter* bzw. *OnExit*, wenn sie den Fokus erhält bzw. verliert.

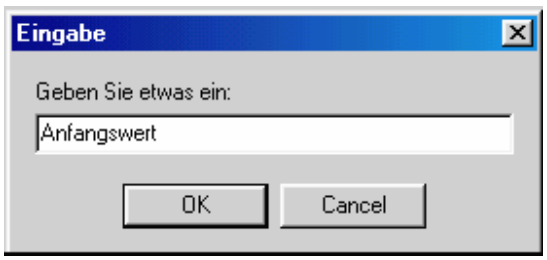
[1.08] Der Dialog zwischen Anwender und Programm

Grundsätzlich dient ein Programm zur Verarbeitung von Daten. Diese müssen vor ihrer Verarbeitung im Computer eingegeben und danach auch ausgegeben werden. Zu diesem Zweck führt der Anwender mit dem Programm einen Dialog, welcher über die verschiedenen Komponenten des Formulares gesteuert wird. Durch Maus oder Tastatur löst der Anwender auf den Formular-Objekten Ereignisse aus, die in den zugehörigen Ereignisbehandlungsroutinen beantwortet werden.

Die einfachste Form eines Dialogs erfolgt über **Inputboxen** und **Messageboxen**. Die einen dienen der Dateneingabe, die anderen der Datenausgabe.

Der Programmbefehl ***S := InputBox(p1,p2,p3)*** öffnet in der Formularmitte ein einzeiliges Eingabefenster. Der erste Parameter ***p1*** ist der Text in der Titelleiste, der zweite ***p2*** eine zusätzliche Information. Der dritte Parameter ***p3*** ist eine mögliche Vorgabe des Eingabetextes (Default), welcher im Fenster editiert werden kann. Beim Verlassen der Inputbox mit der <OK>-Taste wird der eingegebene Text der Zeichenkettenvariablen ***S*** (Datentyp ***String***) zugewiesen. Bei Betätigung von <Cancel> wird der Defaulttext an die Variable ***S*** übergeben.

Das Bild zeigt das Fenster von *S := InputBox('Eingabe','Geben Sie etwas ein!','Anfangswert').*



Der Befehl ***B := InputQuery(p1,p2,p3)*** wirkt ähnlich wie die ***InputBox***-Anweisung. Der Unterschied ist, dass die Rückgabevariable ***B*** vom Datentyp ***Boolean*** ist, d.h., wenn <OK> gedrückt wird, erhält ***B*** den Wert ***True*** zugewiesen. Wird hingegen <Cancel> gedrückt, dann erhält ***B*** den Wert ***False*** zugewiesen.

Die Programmanweisung ***ShowMessage(S)*** öffnet in der Formularmitte ein einfaches Ausgabefenster, welches die Zeichenkettenvariable ***S*** (String) anzeigt und dann auf die Betätigung der <OK>-Taste wartet. Das Bild zeigt das Fenster von *ShowMessage('Ich bin eine einfache Meldung!').*

Weitere Dialogfenster werden mit der Anweisung ***MessageBox*** erzeugt. Der Programmbefehl ***Z := MessageBox(p1,p2,p3,p4)*** öffnet in der Formularmitte ein Meldungsfenster. Der erste Parameter ***p1*** ist die interne Bezugsnummer (Handle) des aktuellen Formulars (z.B. 0 für Form1). Der zweite Parameter ***p2*** ist der eigentliche Meldungstext, der dritte Parameter ***p3*** ist der Text in der Titelleiste und der vierte Parameter ***p4*** ist ein numerischer Kennwert, welcher die visuelle Gestalt der Box bestimmt. Beim Verlassen der *MessageBox* wird ein spezifischer Rückgabewert der ganzzahligen Variablen Z zugewiesen.

Der vierte Parameter ***p4*** setzt sich aus drei Zahlen zusammen, welche addiert werden:

p4 := Symbol + Schaltfläche + Standardeingabe

Symbol bestimmt das angezeigte Symbol in der Box: 0 = ohne Symbol, 16 = Stoppsymbol "X", 32 = Fragesymbol "?", 48 = Warnsymbol "!", 64 = Informationssymbol "i".

Schaltfläche bestimmt die Anzahl und Art der gezeigten Schaltflächen in der Box: 0 = nur "OK", 1 = "OK - Abbrechen", 2 = "Abbrechen - Wiederholen - Ignorieren", 3 = "Ja - Nein - Abbrechen", 4 = "Ja - Nein", 5 = "Wiederholen - Abbrechen".

Standardeingabe gibt an, welche Schaltfläche bei Betätigung der <Enter>-Taste aktiv wird: 0 = erste Schaltfläche, 256 = zweite Schaltfläche, 512 = dritte Schaltfläche.

Die *numerischen Rückgabewerte* geben an, welche Schaltfläche vom Anwender gedrückt wurde: 1 = "OK", 2 oder 3 = "Abbrechen", 4 = "Wiederholen", 5 = "Ignorieren", 6 = "Ja" und 7 = "Nein".

Das Bild zeigt das Fenster von *Z := MessageBox(0,'Programm beenden?' ,'Frage',36).* Für den vierten Parameter 36 gilt: 36 = 32 ("?") + 4 ("Ja - Nein") + 0 ("Ja" bei Enter). Bei Betätigung des "Ja"-Schalters erhält die Variable Z den Rückgabewert 6, andernfalls den Wert 7 zugewiesen. Durch eine Abfrage dieses Wertes kann der weitere Programmverlauf bestimmt werden.

Komplexere Formen der Dateneingabe und Ausgabe erfolgen über die **Standardkomponenten** wie Editierfelder, Memofelder, Listboxen, Dateiauswahlboxen usw. Bei der Programmierung eines Dialogs mit Hilfe entsprechender Komponenten sind zwei Merkmale von besonderer Wichtigkeit: die **Reihenfolge** und der **Fokus.**

Die **Reihenfolge** der Ansteuerung der verschiedenen Komponenten eines Formulares ist ein wichtiges Dialogmerkmal. Die einzelnen Komponenten werden entweder mit der Maus oder der Tabulator-Taste (*Tab* bzw. *Shift Tab*) angewählt. Wird die Eigenschaft *TabStop* auf *False* gesetzt, dann ist die Komponente vorübergehend aus dem Dialog ausgeschlossen. Ähnliches wird auch mit *Enabled := False* bewirkt. Die Reihenfolge der Anwahl ist anfänglich durch die Abfolge der Komponentenerzeugung bei der Formularerstellung festgelegt. Mit der Eigenschaft *TabOrder*, welche automatisch eine Fokusnummer vergibt, kann sie jedoch jederzeit abgeändert werden.

Ein weiterer wesentlicher Aspekt in der Benutzerführung innerhalb des Formulares ist die so genannte **Fokus-Steuerung**. Wird dieser Fokus auf eine Komponente gesetzt, dann wird bei der Programmausführung der Positionszeiger (Mauscursor) automatisch genau auf diese Komponente platziert, welche vom Benutzer sofort bedient werden kann. Beispielsweise stellt die Anweisung *Edit1.SetFocus* im Verlauf eines Programmes den Cursor in das angesprochene Editierfeld, wo sofort mit der Dateneingabe begonnen werden kann.

[1.09] Das Hauptmenü (Kurzübersicht)

Das Hauptmenü der Entwicklungsumgebung gliedert sich in mehrere Untermenüs, wovon die wichtigsten im Folgenden kurz besprochen werden:

<*Datei*> Das Menü enthält Optionen zum Anlegen, Öffnen, Speichern und Schließen von Programmdateien. Wichtig dabei ist die Option <*Speichern unter*>, welche Dateien unter einem neuen Namen abspeichert. Dabei werden in den Quelltexten der Programme und Units automatisch die Namen in den Kopfzeilen entsprechend abgeändert.

<*Ansicht*> In diesem Menü werden einerseits die Hauptfenster der Entwicklungsumgebung ein- oder ausgeblendet. Andererseits können auch die verschiedenen Module und Objekte des aktuellen Programmprojektes betrachtet werden. <*Objektinspektor*> wechselt zum Objektinspektor (auch mit <F11>), <*Umschalten Formular/Unit*> wechselt zwischen Formular und Unit (auch mit <F12>).

<*Projekt*> Das Menü erlaubt zusätzliche Module in ein Programmprojekt einzufügen oder daraus zu entfernen. Außerdem werden hier das Hauptprogramm und seine Units kompiliert (auch mit <Strg><F9>).

<*Start*> Hier wird ein geöffnetes oder gerade entwickeltes Programm ausgeführt (*Start*), was auch mit <F9> erreicht wird. Das Setzen von beliebigen Unterbrechungspunkten (*Breakpoints*) und das Beobachten von Variablenwerten (*Watches*) ist hier zum Zwecke der Programmtestung möglich. Hängt ein Programm durch einen Fehler in einer Endlosschleife, so kann hier mit <Strg><F2> das Programm zurückgesetzt werden.

<*Tools*> Hier sind verschiedene Einstellungen zur Fehlerbehandlung (Exceptionhandling) möglich.

<*Hilfe*> Dieses Menü liefert zu den meisten Sprachelementen von DELPHI eine hilfreiche, ausführliche Beschreibung.