Inhalt

Das **objektorientierte Entwerfen und Programmieren** stellt eine neue Sichtweise der Programmentwicklung dar, die im Gegensatz zur **prozeduralen Programmierung** steht. Anstatt den Programmfluß von der ersten bis zur letzten Code-Zeile genau festzulegen, müssen Sie sich Gedanken über das Erstellen von Objekten machen: Dies sind eigenständige Komponenten einer Anwendung, die sowohl eine interne als auch eine für den Benutzer sichtbare Funktionalität besitzen.

Das Objekt - object

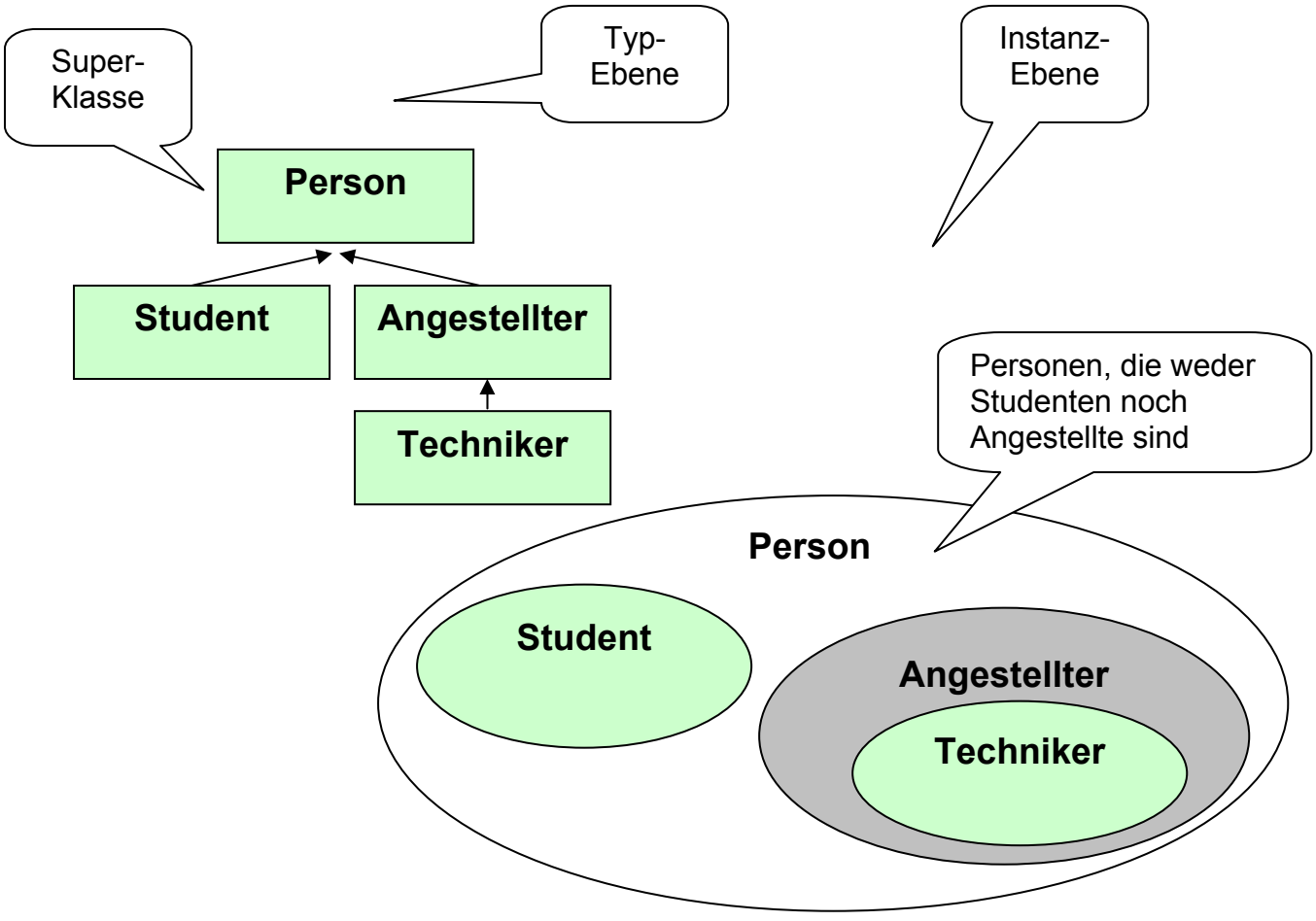
Ein Objekt im Sinne der OOP besteht aus

- Methoden: Operationen zur Manipulation von Daten
- Eigenschaften: Daten

Ein Objekt ist ein abgeschlossener Speicherbereich mit einer **genau definierten Datenstruktur** (den Eigenschaften) und **fest umrissenen Aufgaben**. Die **Kommunikation** mit einem Objekt erfolgt **über Nachrichten**. Jedes Objekt weiß, auf welche **Nachrichten** es mit welchen **Methoden** zu reagieren hat.
Objekte können deklariert werden, indem man ein **Muster für den lokalen Zustand und die Methoden** erzeugt. **Dieses Muster nennt man Klasse.**

Benutzeraktionen wie z.B. ein Mausklick, lösen beim betreffenden Objekt ebenfalls eine Methode über eine vom Betriebsystem gesendete Nachricht aus. In diesem Fall spricht man von einem Ereignis. Ereignisse sind für Visual Basic Programmierer immer das Produkt einer höheren Gewalt und können nicht selbst erzeugt werden.

Personenverwaltungsmodell



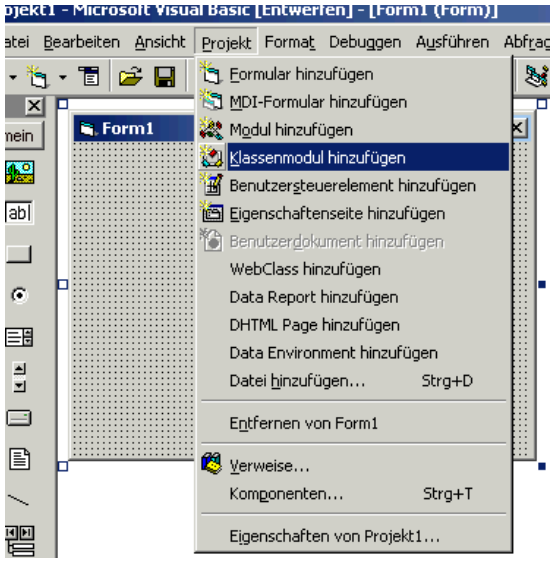
Eine VB Klasse erstellen

Der Oberbegriff für Klasse ist das Objekt. Aber was ist ein Objekt? Visual Basic Programme bestehen fast ausschließlich aus Objekten, so ist beispielsweise jedes Steuerelement (Commandbutton, Textbox, Formular, etc.) ein Objekt. Um dieses theoretische Gebilde des Objekts zu verstehen, verlassen wir kurz Visual Basic und verwenden zum besseren Verständnis ein reales Objekt: Die (Schul-) Klassen.
Wir stellen und vor, es gäbe keinerlei Schulsysteme. Die Eltern aller Kinder unterrichteten diese selber. Das Chaos wäre perfekt, denn keines der Kinder würde gezielt unterrichtet. Jedes Kind hätte einen anderen Wissensstand. Es gäbe keine Zertifikate, die eine bestimmte Bildungsstufe nachweisen, usw. Irgendwann würde jemand versuchen, dieses Chaos zu beseitigen. Er schafft zunächst nur theoretisch ein Gedankengebäude – ein Konzept – in dem Kinder gemäß ihres Alters in Klassen zusammengefasst werden. Diese Klassen haben z.B. die Eigenschaft nur Kinder eines bestimmten Alters aufzunehmen. Gleichzeitig werden die Kinder dieser Klassen durch bestimmte Lehrer und damit durch unterschiedliche Methoden unterrichtet. Natürlich gibt es nicht nur eine Klasse, sondern mehrere, die sich durch verschiedene Eigenschaften und die Anwendung verschiedener Methoden von einander unterscheiden.
Kommen wir nun aber zurück zu Visual Basic: Dort finden wir, wenn wir die Werkzeugsammlung betrachten, eine Reihe von Symbolen, hinter denen sich in dem Moment, in dem wir die Werkzeugsammlung betrachten, nichts weiter als ein theoretisches Gebäude mit Vorschriften verbirgt. Diese Vorschriften definieren, was geschehen soll, wenn eines dieser Konzepte in der Praxis umgesetzt wird. Auch in der objektorientierten Programmierung sprechen wir bei einem solchen theoretischen „Gebäude“ (siehe Schule) von einer Klasse.
Es gibt Verfahren – nennen wir es passend zum dem Beispiel mit der Schule „Einschulung“ – das aus der Theorie Praxis werden lässt. Unsere theoretische Klasse wird zu einem mehr oder weniger greifbaren Objekt. Bei der Programmierung heißt dieser Vorgang natürlich nicht Einschulung, sondern Instanzierung. Wenn man also das Symbol eines Commandbuttons von der Werkzeugsammlung auf einem Formular platziert, wird die Klasse CommandButton instanziert und man erhält ein CommandButton-Objekt. Dieses Objekt kann, wie die jetzt eingeschulte Klasse, durch bestimmte Eigenschaften klassifiziert werden. Sie bestimmen also beispielsweise die MaximaleStundenProTag-Eigenschaft (Schulklasse) bzw. die Caption-Eigenschaft (CommandButton).
Das Konzept einer Klasse gibt es nur ein einziges Mal. Auch die Rahmenbedingungen, in denen sich eine Klasse bewegen kann, werden nur ein einziges Mal vorgeschrieben. Sämtliche Aufgaben einer Klasse werden nur ein einziges Mal bestimmt. Aber dennoch lässt sich eine Klasse beliebig oft zu einem realen Objekt instanzieren, und mehrere Instanzen einer Klasse (Objekte) können völlig unabhängig von einander agieren.
Beim Programmieren werden die Konzepte und Regeln (die Rahmenbedingungen) einerseits von bestimmten Variablen definiert, andererseits durch Programmcode. Wenn wir eine Klasse instanzieren, schafft Visual Basic ein Objekt vom Typ dieser Klasse. Das Objekt enthält den Code der zuvor durch die Klasse definiert wurde. Der Code wiederum bestimmt, was geschieht, wenn man eine bestimmte Methode auf das Objekt anwendet oder eine Eigenschaft des Objekts verändert.
In VB-Code könnte ein derartiges Objektmodell einer Schule in etwa so dargestellt werden (Achtung: dieser Code kann nicht ausgeführt werden!):

```
Option Explicit

Dim Schule1 As New Schule
Dim Klasse1a As New Klasse
Dim Klasse1b As New Klasse
Dim Klasse2a As New Klasse
Dim Klasse2b As New Klasse

Private Sub Schule1_Einschulung()

      Klasse1a.Raumnummer = 1
      Klasse1b.Raumnummer = 2
      Klasse2a.Raumnummer = 3
      Klasse2b.Raumnummer = 4

      Klasse1a.StarteUnterricht „Mathe“
      Klasse1b.StarteUnterricht „Deutsch“
      Klasse2a.StarteUnterricht „Sport“
      Klasse2b.StarteUnterricht „Englisch“

End Sub

Private Sub Schule1_EsSchellt(ByVal Pausennummer As
Integer)

      Klasse1a.InPause
      Klasse1b.InPause

End Sub
```

Die erste Klasse in VB

Nun wollen wir die erste Klasse in Visual Basic realisieren. Dazu erstellen wir zunächst ein neues Projekt vom Typ „Standard EXE“ und fügen diesem mit einem Klick auf den Menüpunkt Klassenmodul hinzufügen im Menü Projekt eine neue Klasse hinzu.



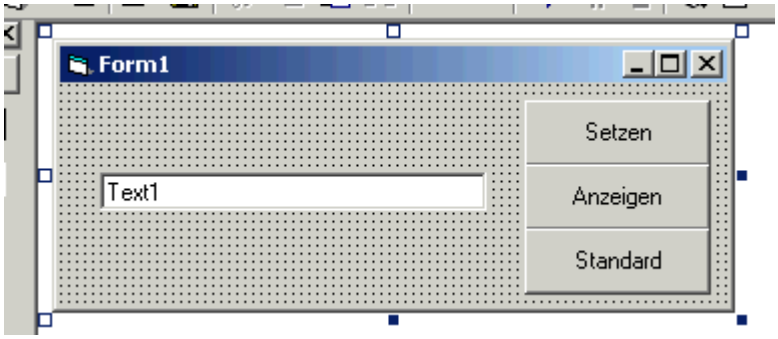
Im nun erscheinenden Dialogfeld wählen wir „Klassenmodul“ aus und geben unsere Klasse im Eigenschaftsfenster noch einen Namen, z.B. „TestKlasse“. Als nächstes müssen wir die oben genannten „Rahmenbedingungen“ für die Klasse schaffen. Dazu fügen wir der Klasse eine Eigenschaft hinzu (der Name der Eigenschaft soll einfach „Eigenschaft“ sein). Im Klassenmodul sollte nun folgender Code enthalten sein:

```
Option Explicit

Private m_strEigenschaft As String


Public Property Get Eigenschaft() As String
  Eigenschaft = m_strEigenschaft
End Property

Public Property Let Eigenschaft(ByVal
NewValue As String)
  m_strEigenschaft = NewValue
End Property
```

Nun möchten wir aber, dass unsere Eigenschaft einen Standardwert bekommen kann, also fügen wir noch eine Funktion (welche in Klassen übrigens Methoden genannt werden) hinzu:

```
Public Function SetzeStandard()
  Eigenschaft = "Standardwert"
End Function
```

Jetzt müssen wir noch den Code für das Formular schreiben. Zunächst platzieren wir auf dem Formular aber 3 CommandButtons und eine TextBox. Das ganze sollte dann in etwa so aussehen:



Der Quellcode sollte nun folgender sein (natürlich ohne die Zeilennummern!):

```
1:    Option Explicit
2:
3:    Dim Objekt As New TestKlasse
4:
5:
6:    Private Sub Command1_Click()
7:
8:          Objekt.Eigenschaft = Text1.Text
9:
10:   End Sub
11:
12:   Private Sub Command2_Click()
13:
14:         Objekt.SetzeStandard
15:
16:   End Sub
17:
18:   Private Sub Command3_Click()
19:
20:         MsgBox Objekt.Eigenschaft
21:
22:   End Sub
```

Zunächst wird hier in Zeile 3 ein Objekt aus der grade definierten Klasse erstellt. Das Schüsselwort New sorgt dafür, dass unser Objekt auch wirklich instanziert wird, und somit verwendet werden kann. In Zeile 8 sind schon die Parallelen zu den Objekten aus der Werkzeugsammlung (Commandbuttons etc.) ersichtlich. Alle Methoden und Eigenschaften einer Klasse werden (genau wie bei den Controls aus der Werkzeugsammlung) durch den Namen des Objektes + Punkt + Name der Eigenschaft/Methode angesprochen.

Was nun noch etwas störend ist an diesem Beispiel, ist dass unsere Eigenschaft nicht zu Beginn ihren Standardwert hat und dieser erst durch einen Aufruf der Methode SetzeStandard() hergestellt wird. Um dieses Problem zu lösen besitzen alle Objekte 2 vordefinierte Methoden, nämlich den Konstruktor und den Destruktor. Der Konstruktor wird ausgeführt sobald eine Instanz der Klasse geschaffen wurde (also in Zeile 3). Der Destruktor wird beim Beenden des Programms ausgeführt, in ihm könnte beispielsweise belegter Arbeitsspeicher der Klasse wieder freigegeben werden. Wir benötigen allerdings zunächst nur den Konstruktor. Wir können ihn der Klasse ganz einfach hinzufügen, indem wir ihr die Methode Class_Initialize() implementieren:

```
Private Sub Class_Initialize()
  SetzeStandard      'Standardwert      der
Eigenschaft setzen
End Sub
```

Und siehe da, wenn wir die Anwendung nun starten und auf „Anzeigen“ klicken, wird tatsächlich der Standardwert ausgegeben.

Als letztes wollen wir der Klasse noch ein Ereignis hinzufügen. Jedes Mal, wenn die Eigenschaft den Standardwert bekommt, soll das Ereignis „StandardGesetzt“ ausgelöst werden. Dazu ändern wir zunächst den Code der Klasse wie folgt ab:

```
1:    Option Explicit
2:
```

```
3:    Public Event StandardGesetzt()
4:
5:    Private m_strEigenschaft As String
6:    Private Const STANDARD_STRING = "Standardwert"
7:
8:
9:    Public Property Get Eigenschaft() As String
10:         Eigenschaft = m_strEigenschaft
11:   End Property
12:
13:   Public Property Let Eigenschaft(ByVal NewValue
As String)
14:
15:         m_strEigenschaft = NewValue
16:
17:         If Eigenschaft = STANDARD_STRING Then
18:               RaiseEvent StandardGesetzt
19:         End If
20:   End Property
21:
22:   Public Function SetzeStandard()
23:         Eigenschaft = STANDARD_STRING
24:   End Function
25:
26:   Private Sub Class_Initialize()
27:         SetzeStandard
28:   End Sub
```

Zu den Änderungen:

- In Zeile 3 teilen wir Visual Basic mit, dass diese Klasse ein Event (=Ereignis) auslösen soll, diesen wird mit der RaiseEvent – Funktion in Zeile 18 getan
- Der Standardwert unserer Eigenschaft wurde in einer Konstanten abgelegt

Jetzt muss der Code des Formulars nur noch ein wenig abgewandelt werden, damit uns das Ereignis der Klasse dort zur Verfügung steht:

```
1
:     Option Explicit
2:
3:    Dim WithEvents Objekt As TestKlasse
4:
5:
6:    Private Sub Form_Load()
7:
8:          Set Objekt = New TestKlasse
9:
10:   End Sub
11:
12:   Private Sub Command1_Click()
13:
14:         Objekt.Eigenschaft = Text1.Text
15:
16:   End Sub
17:
18:   Private Sub Command2_Click()
19:
20:         Objekt.SetzeStandard
21:
22:   End Sub
23:
24:   Private Sub Command3_Click()
25:
26:         MsgBox Objekt.Eigenschaft
27:
```

```
28:   End Sub
29:
30:   Private Sub Objekt_StandardGesetzt()
31:
32:         MsgBox "Es wurde der Standardwert gesetzt"
33:
34:   End Sub
```

Zu den Änderungen:

- Das Schlüsselwort, um an die Ereignisse eines Objektes heranzukommen, lautet in VB WithEvents (siehe Zeile 3)
- Das Erstellen des eigentlichen Objektes wurde in die Form_Load() – Funktion verlagert, da VB ein gleichzeitiges WithEvents und New nicht unterstützt
- In Zeile 30 wird dann das Event definiert, wie man es auch von CommandButtons, TextBoxes, etc kennt.

Polymorphismus in Visual Basic

Mit Polymorphismus meint man die Eigenschaft, Objekte aus anderen Objekten zu erstellen, auch Vererbung genannt. Dies wird von Visual Basic leider erst ab .NET richtig unterstützt. Die einzigste Möglichkeit der Vererbung in VB 6 sind die sogenannten Interfaces. Dieses sind abstrakte Klassen, dh. Klassen, die nur die Funktionsrümpfe enthalten. Diese Interfaces können dann von beliebigen Klassen implementiert werden. Dadurch „erbt“ die Klasse alle Funktionen des Interfaces. Dies hat den Vorteil, dass sich alle Klassentypen die ein bestimmtes Interface implementiert haben, untereinander zuweisen lassen. Hier ein Beispiel dazu:

Als ersten wollen wir unser Interface definieren, dazu fügen wir einem neuen Projekt eine Klasse hinzu, die wir „ITier“ nennen (das „I“ steht dabei für Interface). Sie soll folgenden Code enthalten:

```
Option Explicit

Public Function Essen()
End Function

Public Function Schlafen()
End Function
```

Da man aus solchen abstrakten Klassen keine Objekte erstellen kann, brauchen wir noch weitere Klassen, die dieses Interface implementieren. Deswegen fügen wir als nächstes die Klasse „Hund“ hinzu. Sie enthält folgenden Code:

```
Option Explicit

Implements ITier


Private Function ITier_Essen() As Variant
  MsgBox "Der Hund isst"
End Function

Private Function ITier_Schlafen() As Variant
  MsgBox "Der Hund schläft"
```

```
End Function

Public Function Beiss()
  MsgBox "Der Hund beisst"
End Function
```

Erklärung:

- In der zweiten Zeile wird durch das Schlüsselwort Implements unser Interface eingebunden. Alle im Interface definierten Funktionsrümpfe müssen von der Hund-Klasse mit Code gefüllt, oder zumindest als leere Funktionen implementiert werden
- Durch die neue Methode Beiss() wird der Hund spezialisiert. Er ist zwar immer noch ein Tier (weil er dieses Interface implementiert), aber er hat im Vergleich zu einem normalen Tier weiter Eigenschaften

Da es wenig Seinn hat Interfaces zu nutzen, wenn es nur eine Klasse gibt, die dieses implementiert, definieren wir noch eine zweite, und zwar die Klasse „Katze“:

```
Option Explicit

Implements ITier

Private Function ITier_Essen() As Variant
  MsgBox "Die Katze isst"
End Function

Private Function ITier_Schlafen() As Variant
  MsgBox "Die Katze schläft"
End Function

Public Function Kratz()
  MsgBox "Die Katze kratzt"
End Function
```

Die Katze unterscheidet sich vom Hund in sofern, dass sie nicht beißen kann, dafür kann sie aber kratzen. Trotzdem bleibt sie, genau wie der Hund, aber ein Tier, weil sie ebenfalls das Interface implementiert.
Jetzt müssen wir nur noch den Code unseres Formulars aufsetzten. Zunächst bekommt unser Formular 4 CommandButtons (Captions: Essen; Schlafen; Beißen; Kratzen). Dazu kommen noch zwei OptionButtons mit den Aufschriften Hund + Katze. Das ganze sieht dann in etwa so aus:

Der Code sollte folgender sein:

```
Option Explicit

Dim Tier As ITier

Private Sub Form_Load()
  Set Tier = New Hund
End Sub

Private Sub Form_QueryUnload(Cancel As Integer,
UnloadMode As Integer)
  Set Tier = Nothing
End Sub

Private Sub Command1_Click()

  Tier.Essen

End Sub

Private Sub Command2_Click()

  Tier.Schlafen

End Sub

Private Sub Command3_Click()

  Dim objTempHund As Hund

    If Not TypeOf Tier Is Hund Then
      MsgBox "Das Tier ist kein Hund und kann
nicht beißen"
    Else
      Set objTempHund = Tier
      objTempHund.Beiss
    End If

End Sub

Private Sub Command4_Click()

  Dim objTempKatze As Katze
    If Not TypeOf Tier Is Katze Then
      MsgBox "Das Tier ist keine Katze und kann
nicht kratzen"
    Else
      Set objTempKatze = Tier
      objTempKatze.Kratz
    End If

End Sub
Private Sub Option1_Click()

  Set Tier = New Hund

End Sub

Private Sub Option2_Click()

  Set Tier = New Katze

End Sub
```

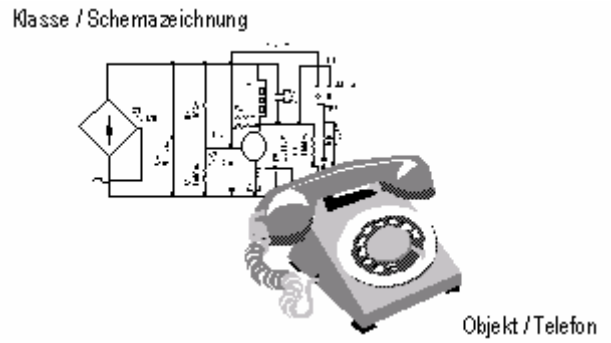
Erklärungen:
Zunächst definieren wir die Variable Tier vom Typ ITier. Da dies ein Interface ist und aus diesem kein Objekt erstellt werden kann, fällt das Wort New hier weg. Stattdessen wird im Form_Load() – Ereignis ein neuer Hund instanziert. Dies ist nur dadurch möglich, dass der Hund das Interface ITier implementiert hat. Mit den beiden OptionButtons kann man zwischen einem Hund und einer Katze wählen. Mit einem Klick auf „Essen“ oder „Schlafen“ wird die entsprechende Funktion aus dem Interface aufgerufen, da diese dort nur deklariert, jedoch nicht definiert ist, wird es an die Klasse, die das Interface implementiert hat weitergeleitet. Also entweder an den Hund oder die Katze, je nachdem als was das Tier momentan definiert ist. Bei einem Klick auf „Beißen“ wird zunächst kontrolliert, ob das Tier ein Hund oder eine Katze ist. Dies geschieht mit der VB-Funktion TypeOf. Wenn es wirklich ein Hund ist, wird unser Tier in eine spezialisierte Klasse vom Typ Hund übertragen, es bleibt jedoch die selbe Klasse, es wird dabei keine neue Klasseninstanz erstellt. Das ist nötig, da das Objekt Tier vom Typ ITier ist und die Methode Beiss() nicht kennt. Der Hund kennt diese aber, da sie in ihm definiert wurde. Der selbe Vorgang wird dann auch mit der Katze wiederholt.

Klassen und Objekte:

Die Bausteine einer Anwendung

Klassen und Objekte stehen in einer engen Beziehung zueinander, sollten jedoch nicht verwechselt werden. Eine Klasse enthält Informationen darüber, wie ein Objekt aussehen und sich verhalten soll. Eine Klasse kann mit einer Blaupause oder der Schemazeichnung eines Objektes verglichen werden. Das elektrische Schema und das Entwurfslayout eines Telefons würden beispielsweise einer Klasse entsprechen. Das Telefon selbst wäre ein Objekt oder eine Instanz dieser Klasse.

Die Klasse legt die Merkmale des Objekts fest.



Objekte besitzen Eigenschaften

Ein Objekt besitzt bestimmte Eigenschaften oder Attribute. Ein Telefon besitzt z. B. eine bestimmte Farbe und Größe. Wenn Sie Ihr Büro mit einem Telefon ausstatten, dann nimmt es einen bestimmten Platz auf Ihrem Schreibtisch ein. Der Hörer kann auf der Telefongabel liegen oder nicht.
Auch die in Visual Basicerstellten Objekte besitzen Eigenschaften, die durch die Klasse festgelegt werden, zu der das Objekt gehört. Diese Eigenschaften können zur Entwurfszeit oder zur Laufzeit bestimmt werden.
In der nachstehenden Tabelle finden Sie als Beispiel einige der Eigenschaften, die ein Kontrollkästchen aufweisen kann.

Eigenschaft	**Beschreibung**
Caption (Beschriftung)	Der beschreibende Text neben dem Kontrollkästchen.
Enabled (Aktiviert)	Legt fest, ob das Kontrollkästchen vom Benutzer aktiviert oder deaktiviert werden kann.
ForeColor (Vordergrundfarbe)	Die Textfarbe der Beschriftung.
Left (Links)	Die Position der linken Seite des Kontrollkästchens.
MousePointer (Mauszeiger)	Das Aussehen des Mauszeigers, wenn er sich auf dem Kontrollkästchen befindet.
Top (Oben)	Die Position der Oberseite des Kontrollkästchens.
Visible (Sichtbar)	Legt fest, ob das Kontrollkästchen sichtbar ist.

Objekten sind Ereignisse und Methoden zugeordnet

Jedes Objekt ist in der Lage, bestimmte Aktionen, sogenannte Ereignisse, zu erkennen und auf sie zu reagieren. Ein Ereignis ist eine spezielle vorbestimmte Aktivität, die entweder vom Benutzer oder vom System ausgelöst wird. Ereignisse werden in den meisten Fällen durch Aktionen der Benutzer erstellt. Beim Telefon wird z. B. ein Ereignis ausgelöst, wenn der Benutzer den Hörer von der Gabel nimmt. Ereignisse werden auch ausgelöst, wenn der Benutzer auf die Telefontasten drückt, um jemanden anzurufen.
Ereignisauslösende Aktionen der Benutzer umfassen in Visual Basic**Mausklicks**, **Mausbewegungen** und **Tastenbetätigungen**. Das Initialisieren eines Objekts und das Ausführen einer Code-Zeile, die einen Fehler verursacht, sind Ereignisse, die durch das System ausgelöst werden.
Methoden sind Prozeduren, die mit einem Objekt verknüpft sind. Methoden unterscheiden sich von normalen Visual C++-Prozeduren: Methoden sind unauflösbar mit einem Objekt verbunden und werden anders aufgerufen als die gewöhnlichen Prozeduren.
Ereignisse können mit ihnen verbundene Methoden besitzen. Wenn Sie z. B. eine Methode für das Ereignis **Click** schreiben, dann wird der Code dieser Methode ausgeführt, sobald das Ereignis **Click** auftritt. Methoden können auch unabhängig von Ereignissen existieren. Diese Methoden müssen im Code explizit aufgerufen werden.
Der Ereignissatz ist fest, wenn auch groß. Sie können keine neuen Ereignisse erstellen. Der Methodensatz jedoch ist unendlich erweiterbar.
Die folgende Tabelle enthält einige Ereignisse, die zu einem Kontrollkästchen gehören.

Ereignis	**Beschreibung**
Klick	Der Benutzer klickt auf das Kontrollkästchen.
GotFocus	Der Anwender wählt das Kontrollkästchen, indem er darauf klickt oder es durch Drücken von TAB aktiviert bzw. deaktiviert.
Lostfokus	Der Benutzer wählt ein anderes Steuerelement.

Die folgende Tabelle enthält einige Methoden, die zu einem Kontrollkästchen gehören:

Methode	**Beschreibung**
Rehfrech	Der Wert des Kontrollkästchens wird aktualisiert, um alle Änderungen widerzuspiegeln, die in der zugrundeliegenden Datenquelle aufgetreten sind.
SetFocus	Der Fokus wird so auf das Kontrollkästchen gesetzt, als habe der Benutzer TAB so oft gedrückt, bis das Kontrollkästchen gewählt ist.

Klassen in Visual C++

Alle Eigenschaften, Ereignisse und Methoden eines Objekts werden in der Klassendefinition angegeben. Außerdem besitzen Klassen folgende Merkmale, wodurch sie sich besonders gut zum Erstellen von wiederverwendbarem, leicht zu wartenden Code eignen:

- Kapselung
- Unterklassen
- Vererbung

Verstecken nicht notwendiger Komplexität

Wenn Sie Ihr Büro mit einem Telefon ausstatten, ist es von zweitrangigem Interesse zu wissen, wie das Telefon intern einen Anruf erhält, Verbindungen zu elektronischen Schalttafeln aufbaut, beendet oder die gedrückten Zifferntasten in elektronische Signale umwandelt. Sie können den Hörer abnehmen und die entsprechenden Nummern wählen, um mit der angewählten Person sprechen zu können. Die Komplexität des Verbindungsaufbaus bleibt unsichtbar. Der Vorteil, die inneren Einzelheiten eines Objekts ignorieren zu können und Ihr Hauptaugenmerk auf die von Ihnen benötigten Aspekte des Objekts zu lenken, wird Abstraktion genannt.

Interne Komplexität kann verdeckt werden.

Kapselung

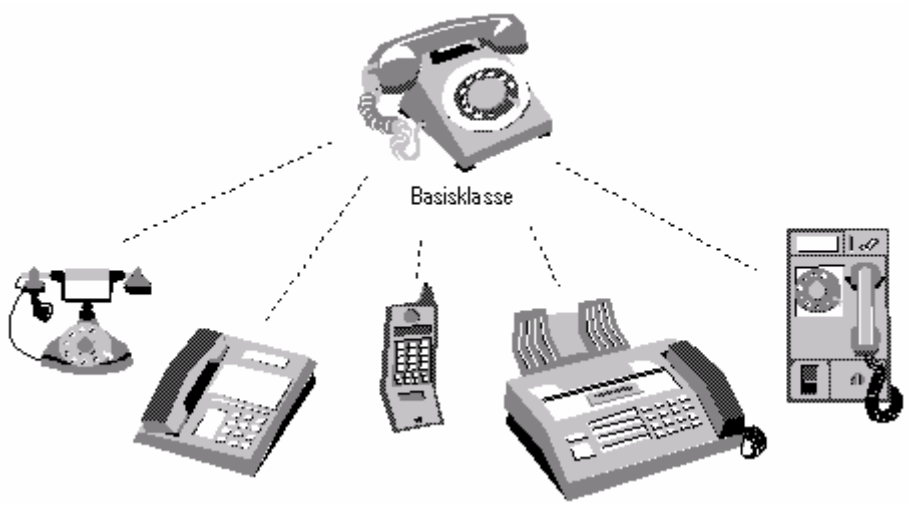
Die Kapselung, die die Zusammenfassung des Codes von Methoden und Eigenschaften in einem Objekt mit sich bringt, trägt zur Abstraktion bei. Zum Beispiel können sowohl die Eigenschaften, durch die Elemente innerhalb eines Listenfeldes festgelegt werden, als auch der Code, der ausgeführt wird, sobald ein Element aus dieser Liste ausgewählt wird, in ein einziges Steuerelement eingekapselt werden. Dieses können Sie dann in ein Formular einfügen.

Leistungsfähigkeit bestehender Klassen ausnutzen

Unterklasse

Eine Unterklasse kann die gesamte Funktionalität einer vorhandenen Klasse und darüber hinaus zusätzliche Steuerelemente oder sonstige Funktionalität besitzen, die Sie ihr verleihen möchten. Ist Ihre Klasse ein einfaches Telefon, dann können Sie Unterklassen erstellen, die die gesamte Funktionalität des Originaltelefons besitzen. Sie können die Unterklassen auch mit beliebigen von Ihnen gewünschten, speziellen Merkmalen ausstatten.

Unterklassen ermöglichen Ihnen die erneute Verwendung von Code.



Das Bilden von Unterklassen ist eine Möglichkeit zum Verringern des von Ihnen zu schreibenden Codes. Sie können mit der Definition eines Objekts beginnen, das Ihren Wünschen annähernd entspricht, und dieses später anpassen.

Rationalisierte Pflege von Code

Vererbung

Beispiel.: Ein Kreuzfahrtschiff ist ein Motorschiff, ein Motorschiff ist ein Schiff

Man kann bestimmte Eigenschaften und Methoden suchen, die alle Schiffe gemeinsam haben und diese zu einer Klasse Schiff zusammenfassen. Für die Klasse Motorschiff muß man nun diese in der Klasse Schiff definierten Eigenschaften und Methoden nicht erneut entwickeln, sondern man leitet die Klasse Motorschiff von Schiff ab. Damit „erbt" die Klasse Motorschiff die Eigenschaften und Methoden von Schiff. Wenn man nun auf die Idee kommt ein Segelschiff zu bauen, kann man die Klasse Schiff wiederverwenden und spart sich damit enormen Entwicklungsaufwand. Aus Motorschiff und Segelschiff könnte man nun einen Motorsegler bauen. Dabei kann es zu Konflikten kommen, weil Motorschiff und Segelschiff einige Methoden und Eigenschaften gemeinsam haben.

Durch **Vererbung** spiegeln sich alle Änderungen, die Sie in eine Klasse einarbeiten, in allen auf dieser Klasse basierenden Unterklassen wider. Diese automatische Aktualisierung erspart Ihnen Zeit und Mühe. Wenn beispielsweise ein Telefonhersteller die Telefone von Wahlscheiben auf Wahltasten umrüsten möchte, dann würde es ihm viel Arbeit ersparen, wenn er lediglich das zugrundeliegende Schema verändern müßte und alle zuvor nach diesem Schema gefertigten Telefone automatisch das neue Merkmal besitzen würden.

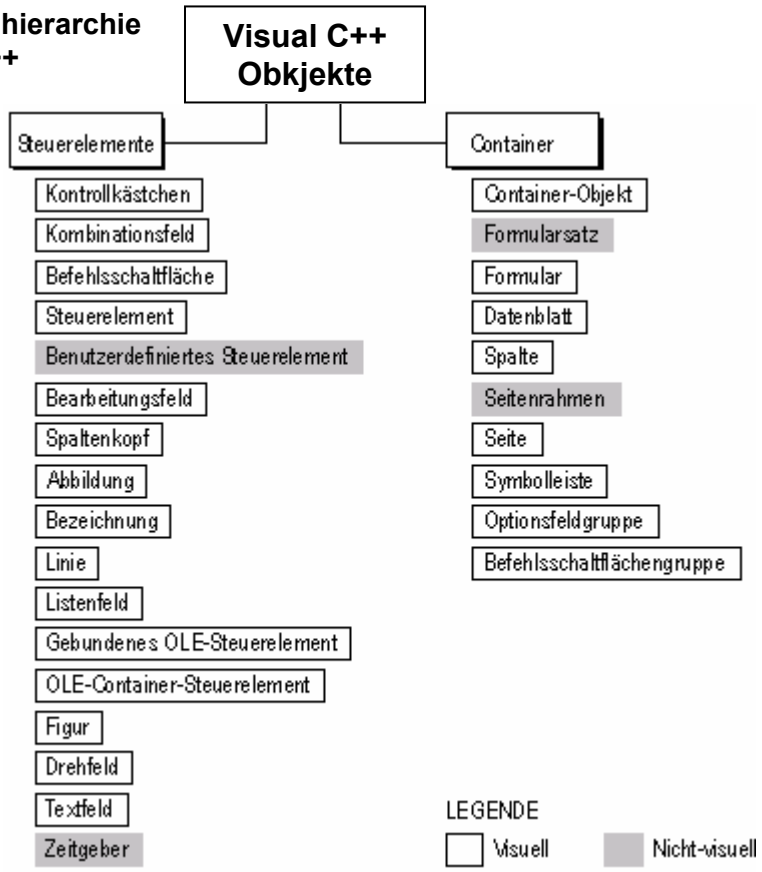
Die Vererbung erleichtert die Pflege Ihres Codes

Eine Vererbung kann nicht bei Hardware erfolgen, jedoch steht Ihnen diese Fähigkeit bei Software zur Verfügung. Wenn Sie in einer Klasse einen Fehler feststellen, müssen Sie den Code nicht in allen Unterklassen korrigieren. Sie können den Fehler in der Klasse beheben, und diese Änderung wird dann in allen Unterklassen dieser Klasse durchgeführt.

Die Klassenhierarchie in Visual C++

Wenn Sie benutzerdefinierte Klassen erstellen möchten, dann ist es hilfreich, die Klassenhierarchie von Visual Basiczu kennen.

Die Klassenhierarchie in Visual C++

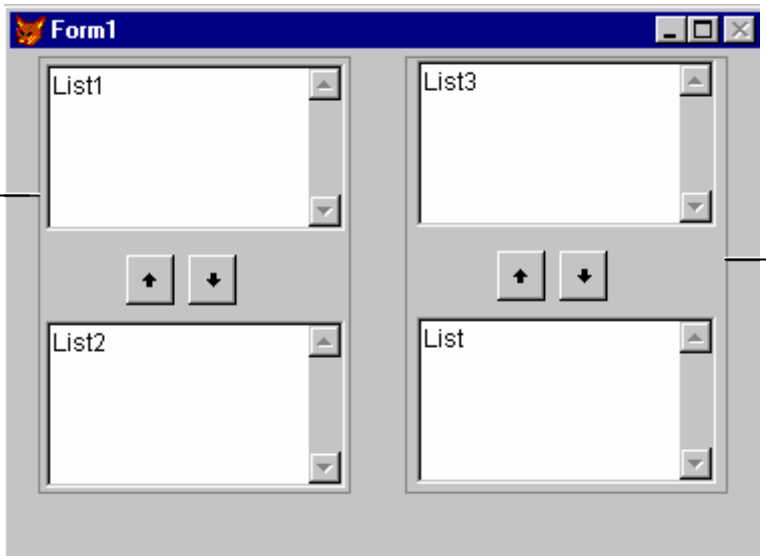


Polymorphismus wird in der Literatur oft auch als dynamisches oder spätes Binden (late binding) bezeichnet.
Polymorphie bedeutet, daß ein Objekt auf eine Nachricht richtig reagiert, sofern es die Nachricht kennt, wobei verschiedene Objekte unterschiedlich auf die Nachricht reagieren können. Weil erst zur Laufzeit bekannt ist, welche Methode (von welchem Objekt) ausgeführt werden soll, kann beim Linken (Binden) noch keine Zuordnung erfolgen, sondern erst, wenn die Nachricht ankommt. Deshalb nennt man den Vorgang auch spätes oder dynamisches Binden.

Container und Nicht-Container

Zwei Haupttypen der Visual C++-Klassen und dementsprechend der Visual C++-Objekte sind Container-Klassen und Steuerelementklassen.

Container- und Steuerelementklassen



Steuerelementklasse: Komponenten werden als zusammenhängend Einheit verändert

Containerklasse: Komponenten können einzeln verändert

Container-Klassen

Container können andere Objekte enthalten und ermöglichen den Zugriff auf die in ihnen enthaltenen Objekte. Wenn Sie beispielsweise eine Container-Klasse erstellen, die aus zwei Listenfeldern und zwei Befehlsschaltflächen besteht, und dann ein auf dieser Klasse basierendes Objekt einem Formular hinzufügen, kann jedes einzelne Objekt sowohl zur Laufzeit als auch während des Entwurfs bearbeitet werden. Sie können die Position der Listenfelder oder die Beschriftungen der Befehlsschaltflächen leicht ändern. Sie können dem Steuerelement auch während des Entwurfs Objekte hinzufügen; so können Sie beispielsweise Beschriftungen hinzufügen, die die Listenfelder näher kennzeichnen.

In der folgenden Tabelle finden Sie eine Liste aller Komponenten, die eine Container-Klasse enthalten kann.

Container	**Enthalten**
Befehlsschaltflächengruppen	Befehlsschaltflächen
Container	Beliebige Steuerelemente
Steuerelement	Beliebige Steuerelemente
Benutzerdefiniert	Beliebige Steuerelemente, Seitenrahmen, Container, benutzerdefinierte Objekte
Formularsätze	Formulare, Symbolleisten
Formulare	Seitenrahmen, beliebige Steuerelemente, Container, benutzerdefinierte Objekte
Datenblattspalten	Kopfzeilen und beliebige Objekte außer Formularsätzen, Symbolleisten, Zeitgeber und anderen Spalten
Datenblatt-Steuerelement	Datenblattspalten
Optionsfeldgruppen	Optionsfelder
Seitenrahmen	Seiten
Seiten	Beliebige Steuerelemente, Container, benutzerdefinierte Objekte
Projekt	Dateien, Server
Symbolleisten	Beliebige Steuerelemente, Seitenrahmen, Container

Steuerelementklassen

Steuerelementklassen sind vollständiger eingekapselt als Container-Klassen, sie sind aus diesem Grund jedoch auch weniger flexibel. Steuerelementklassen verfügen nicht über die Methode AddObject.

Anpassen einer Klasse an eine Aufgabe

Klassen möchten Sie evtl. in vielen verschiedenen Zusammenhängen verwenden können. Eine intelligente Planung hilft Ihnen bei der Entscheidung, welche Klassen zu entwerfen sind und welche Funktionen in eine Klasse einbezogen werden sollen.

Entscheiden, wann Klassen zu erstellen sind

Theoretisch könnten Sie für jedes einzelne jemals zu verwendende Steuerelement und Formular eine eigene Klasse erstellen, doch dies wäre nicht das rationellste Verfahren zum Entwerfen Ihrer Anwendungen. Wahrscheinlich besäßen Sie am Ende viele verschiedene Klassen, die zwar ähnliche Funktionen erfüllen, jedoch separat verwaltet werden müßten.

Zusammenfassen generischer Funktionalität

Sie können eine Steuerelementklasse für generische Funktionalität erstellen. Beispielsweise lassen sich Befehlsschaltflächen, mit denen ein Benutzer den Datensatzzeiger in einer Tabelle verschieben kann, eine Schaltfläche zum Schließen eines Formulars sowie eine Hilfeschaltfläche als Klassen speichern und Formularen immer dann hinzufügen, wenn diese die vorgesehenen Funktionen bieten sollen.
Sie können Eigenschaften und Methoden in einer Klasse zusammenfassen, damit der Benutzer sie in die spezielle Datenumgebung eines Formulars oder Formularsatzes integrieren kann.

Schaffen eines einheitlichen Erscheinungsbildes und intuitiven Zugriffs für die Anwendung

Sie können Formularsatz-, Formular-, und Steuerelementklassen mit einem charakteristischen Erscheinungsbild erstellen, so daß alle Komponenten Ihrer Anwendung gleich aussehen. So könnten Sie z. B. einer Formularklasse Graphiken und bestimmte Farbmuster hinzufügen und diese Klasse dann als Vorlage für sämtliche zu erstellenden Formulare verwenden. Sie könnten auch eine Textfeldklasse mit einem besonderen Erscheinungsbild, wie z. B. einer Schattierung, erstellen und diese Klasse in Ihrer gesamten Anwendung immer dann verwenden, wenn Sie ein Textfeld hinzufügen möchten.

Entscheiden, welcher Klassentyp zu erstellen ist

Visual Basicermöglicht es Ihnen, mehrere Arten von Klassen, jede mit individuellen Eigenschaften, zu erstellen. Sie geben den gewünschten Klassentyp entweder im Dialogfeld Neue Klasse oder mit der Klausel AS des Befehls CREATE CLASS an.

Die Visual C++-Basisklassen

Im Klassen-Designer können Sie Unterklassen zu den meisten Visual C++-Basisklassen erstellen:

Die Visual C++-Basisklassen

ActiveDoc	Custom	Label	PageFrame
CheckBox	EditBox	Line	ProjectHook
Column*	Form	ListBox	Separator
CommandButton	Formularsatz	OLEBoundControl	Shape
CommandGroup	Datenblatt	OLEContainerControl	Spinner
ComboBox	Header*	OptionButton*	TextBox
Container	Hyperlink Object	OptionGroup	Timer
Control	Image	Page*	ToolBar

* Da diese Klassen ein integraler Bestandteil eines übergeordneten Containers sind, lassen sich im Klassen-Designer keine Unterklassen dazu bilden.

Alle Visual C++-Basisklassen erkennen zumindest die folgenden Ereignisse:

Ereignis	**Beschreibung**
Init	Tritt auf, wenn das Objekt erstellt wird.
Destroy	Tritt auf, wenn das Objekt aus dem Speicher gelöscht wird.
Error	Tritt immer dann auf, wenn in Ereignis- oder Methodenprozeduren dieser Klasse ein Fehler auftritt.

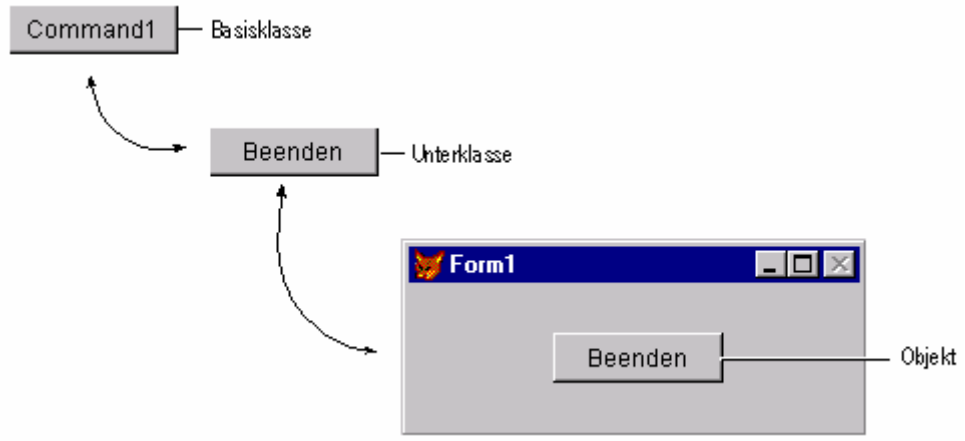
Alle Visual C++-Basisklassen verfügen zumindest über die folgenden Eigenschaften:

Eigenschaft	**Beschreibung**
Class	Legt den Klassentyp fest.
BaseClass	Legt die Basisklasse fest, von der die Klasse abgeleitet wurde, wie z. B. Formular, Befehlsschaltfläche, benutzerdefinierte Klasse usw.
ClassLibrary	Die Klassenbibliothek, in der die Klasse gespeichert ist.
ParentClass	Die Klasse, von der die aktuelle Klasse abgeleitet wurde. Wurde die Klasse direkt von einer Visual C++-Basisklasse abgeleitet, ist die Eigenschaft **ParentClass** die gleiche wie **BaseClass**.

Erweitern der Visual C++-Basisklassen

Zu diesen Klassen können Sie Unterklassen erstellen, um Ihre eigenen Standardeigenschaften für Steuerelemente festzulegen. Wenn z. B. die Standardnamen von Steuerelementen, die Sie Formularen in Ihren Anwendungen hinzufügen, automatisch Ihre

Benennungskonventionen widerspiegeln sollen, könnten Sie hierfür Klassen erstellen, die auf den Basisklassen von Visual Basicbasieren. Sie können Formularklassen mit einem speziell angepaßten Erscheinungsbild oder Verhalten erstellen, die als Vorlagen für alle künftig erstellten Formulare dienen sollen.
Unterklassen zu den Visual C++-Basisklassen können Sie auch erstellen, um Steuerelemente mit zusammengefaßten Funktionen zu erstellen. Wenn beispielsweise eine Schaltfläche beim Klicken Formulare freigeben soll, können Sie eine auf der Befehlsschaltflächen-Klasse basierende Klasse erstellen, deren Eigenschaft **Caption** auf **Beenden** einstellen und in dem Ereignis **Click** den folgenden Befehl hinzufügen:
THISFORM.Release
Diese neue Schaltfläche können Sie jedem Formular in Ihrer Anwendung hinzufügen.
Benutzerdefinierte Befehlsschaltfläche, die einem Formular hinzugefügt wurde



Erstellen von Steuerelementen mit mehreren Komponenten

Ihre Unterklassen sind nicht auf einzelne Basisklassen beschränkt. Sie können mehrere Steuerelemente in einer einzigen Container-Klassendefinition zusammenfassen. Viele der Klassen in der Beispiel-Klassenbibliothek von Visual Basicgehören zu dieser Kategorie. So enthält z. B. die Klasse "VCR" in **Buttons.vcx**, die im Visual Studio-Unterverzeichnis **Visual Studio…\Samples\Vfp98\Classes** zu finden ist, vier Befehlsschaltflächen, mit denen sich ein Benutzer durch die Datensätze einer Tabelle bewegen kann.

Erstellen von nichtvisuellen Klassen

Eine Klasse, die auf der benutzerdefinierten Visual C++-Klasse basiert, enthält kein visuelles Laufzeitelement. Mit Hilfe der Klassen-Designer-Umgebung können Sie Methoden und Eigenschaften für Ihre benutzerdefinierte Klasse erstellen. Beispielsweise könnten Sie die benutzerdefinierte Klasse StrMethods erstellen und darin mehrere Methoden zum Bearbeiten von Zeichenfolgen einbeziehen. Diese Klasse könnten Sie einem Formular mit einem Bearbeitungsfeld hinzufügen und die integrierten Methoden bei Bedarf aufrufen. Eine Methode namens WordCount könnten Sie dann folgendermaßen aufrufen:

```
THISFORM.txtCount.Value = ;
  THISFORM.StrMethods.WordCount(THISFORM.edtText.Value)
```

Nichtvisuelle Klassen (wie benutzerdefinierte Steuerelemente und das Zeitgeber-Steuerelement) werden nur während der Entwurfszeit im Formular-Designer visuell dargestellt. Stellen Sie daher die Eigenschaft **Picture** der benutzerdefinierten Klasse auf die .bmp-Datei ein, die im Formular-Designer angezeigt werden soll, wenn die benutzerdefinierte Klasse einem Formular hinzugefügt wird.

Erstellen von Klassen

Sie können neue Klassen im Klassen-Designer erstellen und während des Entwurfsvorgangs anzeigen, wie jedes Objekt für die Benutzer aussehen wird.
So erstellen Sie eine neue Klasse

- Wählen Sie im Projekt-Manager die Registerkarte **Klassen** und dann die Schaltfläche **Neu**.

  - oder -

- Wählen Sie aus dem Menü **Datei** den Befehl **Neu**, dann die Option **Klasse** und anschließend **Neue Datei**.

  - oder -

- Verwenden Sie den Befehl CREATE CLASS.

Im Dialogfeld **Neue Klasse** können Sie angeben, wie die neue Klasse heißen, auf welcher Klasse sie basieren und in welcher Bibliothek sie gespeichert werden soll.
Erstellen einer neuen Klasse