C++Builder (5.0), kurze OOP-Einführung

(C++Builder: "Visuelle" C++ Entwicklungsumgebung von Borland, OOP: objektorientierte Programmierung)

1. Subroutinen

Am Anfang (60er Jahre) war die Hardware und der "Spaghetticode"...
Neu war, mit Hilfe von **Subroutinen** Code einzusparen. (Fast) alles war erlaubt, nur Speicher durfte nicht belegt werden, da Hardwareteile und insbesondere Speicherbausteine teuer waren. Programmaufbau darstellen mit Hilfe eines **Flussdiagramms**.

BASIC alt (oh Schreck...)

```
010: REM Kommentar, init
020: limit=100
030: count=0
040: REM Hauptprogramm
050: anweisungen
060: count=count+1
070: IF count<limit GOTO 50
080: GOSUB 200
090: anweisungen
200: REM Subroutine meldung
210: anweisungen
220: RETURN
```

im Vergleich:
C strukturiert

```
//
//File Size, Date in Preview
//==========================
TSearchRec sr;
String Path = LsFilelistView281->SelectedItem;
DateSeparator = '-';
ShortDateFormat = "d/m/yyyy ";
if (FindFirst(Path, faAnyFile, sr) == 0)
{
   do
   {
      if ((sr.Attr & faAnyFile) == sr.Attr)
      {
         Label7->Caption = FormatFloat("#,# Byte", (sr.Size)) +
            "   / " + IntToStr(sr.Size);
      }
   } while (FindNext(sr) == 0);
   FindClose(sr);
}
```

Listing - "Zeilen-Basic", eine frühe Hochsprache (70er Jahre), im Vergleich zu einer C-Sequenz aus dem C++Builder.

Programme wurden immer umfangreicher (und unübersichtlicher), darum mussten neue Werkzeuge und Methoden zur Softwareerstellung her. So entstand in den 70er Jahren die

2. Strukturierte Programmierung

mit Sprachen, die **Prozeduren und Funktionen** verwenden, wie Pascal (siehe Niklaus Wirth, ETHZ), C, strukturierte BASIC-Dialekte u.a. Die Regeln der strukturierten Programmierung zwingen den Programmierer in gut lesbarer und nachvollziehbarer Form zu schreiben (eben kein Spaghetticode!). Merkmale der strukturierten Programmierung:

- **Daten**: werden hierarchisch definiert: Dateien, Datensätze, Felder.
- **Strukturblöcke**: Beschränkung der Ablaufmuster auf die folgenden:

Sequenz	`{...; ...;}`
Selektion	`if () {...;} else {...;} switch () {...;}`
Iteration	`for() {...;}, while () {...;}`

Darstellung durch Begin-End-Klammerung `{...;}` und Einrücken, Beispiel:

```
//BEGIN_BLOCK
{
   IF (...) {
       ...;
   }
}
//END_BLOCK
```

(strukturierte Programmierung)

- **Zweipoligkeit**: Jeder Block darf nur einen Eingang und einen Ausgang aufweisen, d.h. keine "wilden" JUMPs, GOTOs etc. im Programm.

- **Assemblerprogramme:** Verzweigungen dürfen nicht beliebig erfolgen, Anweisungsblöcke müssen den Regeln der strukturierten Programmierung genügen.

- **Programmentwurf:** Top-Down-Entwurf (Programm-Verfeinerung von oben nach unten, vom abstrakten Entwurf bis zum Programmcode, "schrittweise Verfeinerung" nach N. Wirth), Darstellung mit dem **Struktogramm** (auch Nassi Shneiderman-Diagramm genannt).

80er Jahre:

3. Modulare Programmierung

Ein Modul in C (im C++Builder "Unit" genannt)

Header, H-Datei = Schnittstelle ("SICHTBAR")
Datei: HEADER.H C++Builder: UNIT.H

Schnittstellenbeschreibung: Deklaration von
Typen, Konstanten, Variablen, Funktionen/Prozeduren,
zum Export nach externen Modulen
(externes Modul importiert Daten und Funktionen)

Code, C-Datei = Implementation ("UNSICHTBAR")
Datei: CODE.C C++Builder: UNIT.CPP

Programmcode / Implementation: Funktionen/Prozeduren und lokale Deklarationen, verborgen und geschützt (information hiding).

Code kann geändert werden, sofern Schnittstelle und Verhalten des Moduls gleich bleiben.

Bild - Ein C-Modul besteht aus der Schnittstelle und dem Implementationsteil,
Vergleich: PC-Steckkarte mit Steckerleiste (Schnittstelle) und Schaltung (Implementation).

Module sind **getrennt übersetzbar**, unterstützen **lokale Namen** und den **Import und Export** von Daten und Funktionen von und nach externen Modulen. Eine gute Modularisierung zeichnet sich dadurch aus, dass zwischen den Modulen eine möglichst geringe Verflechtung besteht (geringes "coupling").

Modul-Architekturen

(1) Eingabe → Verarbeitung → Ausgabe

- **Funktionale Gliederung entlang dem Datenfluss**, Beispiel EVA (1)

- **Schichten-Modell**

  z.B. Windows-OS-Layer (2) oder ISO/OSI-Layer-Modell (Datenübertragung, Schnittstellen)

(2)

Applikation
Windows
DOS
BIOS

- **Kapselung** von Daten: **abstrakte Datentypen (ADT)**

  Bindeglieder zu OO-Sprachen, in der OO-Programmiersprache C++ entspricht die Klassenbeschreibung (Deklaration eines Objekt-Typs im Header) einem ADT.

90er Jahre: OOP (objektorientierte Programmierung)...

4. OOP und Klassen am Beispiel von C++ Builder

Warum OOP?

- Programmierung und Wartung von Software soll **realitätsnah** sein.
- Einfache Erweiterung der Software (nur inkrementelle Ergänzungen bei Subklassen).
- Kapselung: Zugriffe auf Daten und Operationen eines Objekts regeln.
- Vererbung: erweiterte Wiederverwendbarkeit von Softwarekomponenten, Klassenbibliotheken.
- Polymorphie: Verschiedene Klassen können auf die gleiche Nachricht unterschiedlich reagieren.

Objekte, Klassen und Kapselung

Aus der Beobachtung der realen Welt, in der die Eigenschaften eines Objekts und dessen Verhalten (meistens) untrennbar verbunden sind, entstand der **Datentyp Klasse (class)**. Im Datentyp Klasse werden Elementdaten und Elementfunktionen zusammengefasst und **gekapselt**. Der Datentyp Klasse bildet die grundlegende Stütze der OOP.

Elementdaten einer Klasse sind die Variablen und Konstanten. Sie beschreiben die Eigenschaften resp. Attribute (properties) dieser Klasse.

Elementfunktionen (auch Methoden oder Operationen) sind die Funktionen und Prozeduren, mit deren Hilfe man auf die Elementdaten eines Objekts einer Klasse zugreifen und diese verändern kann. Sie bilden die Schnittstelle einer Klasse und beschreiben das zugehörige Verhalten.

Eine Klasse ist eine Beschreibung einer Gruppe von Objekten, welche gleiche Struktur und gleiches Verhalten besitzen. Eine Klasse kann als Vorlage oder Muster eines Objekts aufgefasst werden.

Mit den Zugriffsmodifizierern **public** und **private** (sowie protected) spezifiziert eine Klasse die Zugriffsrechte auf ihre Elemente (information hiding = verbergen und abkapseln von Informationen). Zugriffe von aussen (durch Objekte von fremden Klassen) auf Elemente dieser Klasse sind nur möglich, wenn die Zugriffsrechte dies erlauben (public). Ebenso können Zugriffe von aussen auf Elemente verhindert werden (private, protected).

```
            // Der Vererbungszugriffs- Modifizierer "public" bezeichnet den
            // Status der geerbten Elemente:
class TForm1 : public TForm   // public: geerbte Elemente von TForm1 gleich wie
                              //                Elemente der Oberklasse TForm
                              // private:
                              // geerbte Elemente der Basisklasse werden privat
{
// Zugriffsmodifizierer bezeichnen Zugriffsrechte auf Elemente:
__published:      // im Objektinspektor aufgeführt, sonst wie "public"
                  Variablen (Eigenschaften)
                  Funktionen (Methoden)

private:          // Zugriff nur von eigener Klasse (oder friend) möglich
                  Variablen (Eigenschaften)
                  Funktionen (Methoden)

public:           // Zugriff von aussen möglich (öffentlicher Zugriff)
                  Variablen (Eigenschaften)
                  Funktionen (Methoden)

                  // protected: Zugriff auch von abgeleiteten Klassen
};
```

Listing - C++Builder: Klassen-Deklaration im Header (z.B. Unit1.h), die neue Klasse TForm1 erbt von TForm alle Elemente mit gleichbleibenden Zugriffsrechten

Was ist ein Objekt im Sinne der OOP?

Eine Klasse kann ihre "Aufgabe" erst erfüllen (ist lauffähig), wenn ein Objekt dieser Klasse "instanziert" (d.h. erzeugt) wird. Aus einer Klasse können beliebig viele Objekte erzeugt werden.

Ein Objekt ist eine Instanz (erzeugtes Exemplar) der Klasse. Die Klasse beschreibt die Eigenschaften und das Verhalten des Objekts. Bei der Instanzierung wird dem Objekt Speicherplatz zugewiesen und die Instanzvariablen (Eigenschaften, Daten) initialisiert.

Jedes Objekt verfügt im Speicher über eigene Instanzvariablen. Eigene und geerbte Methoden dieser Klasse sind jedoch nur einmal im Arbeitsspeicher vorhanden und werden von den Objekten (gemäss ihren Zugriffsrechten) gemeinsam genutzt.

Beispiele:
- *Felix Muster ist eine Instanz der Klasse "Mensch"*
- *Kreis1 und Kreis2 sind Instanzen der Klasse TKreis1*
- *ini ist eine Instanz der Klasse TIniFile (siehe unten)*

Objekte können in C++ auf folgende Arten erzeugt und gelöscht werden:

- **Statisch**: Objekt wird erzeugt und gelöscht beim Start, resp. Verlassen des Programms.
- **Dynamisch**: Objekt wird zur Laufzeit erzeugt und gelöscht, mit den Speicherverwaltungs-Operatoren **new** und **delete**.
- **Automatisch**: Objekt wird jedes Mal beim Durchlauf eines Blocks erzeugt und gelöscht entsprechend der jeweiligen Deklaration.
- **Als Element-Objekt**, falls es Mitglied einer andern Klasse oder Arrays ist.

Beim Erzeugen eines Objekts wird jedes Mal automatisch die **Konstruktor-Methode** der Klasse (richtet Speicherplatz ein und initialisiert Datenelemente) aufgerufen, beim Löschen der Instanz eine **Destruktor-Methode** (Speicher freigeben). Existiert für die Klasse keine Konstruktormethode, so weist der Compiler einen Standard-Konstruktor zu, gleiches gilt sinngemäss für den Destruktor.

C++: Zugriff auf Klassenelemente die den Status "public" besitzen

Public-Elemente können von Methoden der eigenen, abgeleiteten und fremden Klassen benutzt werden. Der Zugriff von aussen erfolgt über den Namen einer Instanz und einem der Zugriffsoperatoren "**->**" oder "**.**"

`ZeigerAufInstanz->ElementName`	Aufruf eines Elements mittels Pointer auf die Instanz
`InstanzName.ElementName`	Aufruf eines Elements einer Klassenvariablen (siehe ADT)

```
// Beispiel Wert übernehmen: die Methode HoleWert() ist public
InstanzKlasse1->VerarbeiteWert(InstanzKlasse2->HoleWert(Par) );
```

Public-Elemente (üblicherweise Methoden) bilden die Schnittstelle einer Klasse und entscheiden somit darüber, wie die Klasse benutzt werden kann und ob sie zur Weiterverwendung und Vererbung (z.B. innerhalb einer Klassenbibliothek) geeignet ist. Public-Methoden sollten - zu Gunsten des Benutzers der Klasse - in ihrer Anwendung einfach gehalten werden. Sie dürfen nicht ohne weiteres modifiziert oder erweitert werden, weil damit die Schnittstelle (= Benutzung) der Klasse verändert wird.

Klassenelemente die den Status "private" besitzen

Andererseits dürfen **private-Daten und -Methoden ohne Problem geändert werden**, da diese Elemente dem Benutzer der Klasse ja nicht bekannt sind und auch nicht darauf zugegriffen werden kann. Klassen-interne Vorgänge und Datenelemente, die nicht von öffentlichem Interesse sind, sollten sinnvollerweise "private" deklariert sein (→ Informationen verbergen, kapseln).

```
void __fastcall TForm1::IniReadSection(TObject *Sender)
{
// Instanz von TIniFile dynamisch erzeugen und Sektion der INI-Datei lesen

      // Variablen-Name der Instanz von TIniFile als Pointer deklarieren
      TIniFile *ini;

      // Instanz von TIniFile dynamisch erzeugen (Speicher reservieren)
      ini = new TIniFile(IniFileName);

      try {
         // Sektion lesen und in ListBox eintragen
         ini->ReadSectionValues("COMBO1", CheckListBox1->Items);
      }
      __finally {
         // Instanz löschen und Speicher wieder freigeben
         delete ini;
      }
}
```

*Listing - Objekt erzeugen: Instanz der Klasse TIniFile dynamisch erzeugen (und löschen) mit den Operatoren **new** und **delete** (CPP-Datei).*

Vererbung

Mit dem Konzept der Vererbung können von bestehenden Klassen (Basisklassen oder Oberklassen) neue Klassen (Sub- oder Unterklassen) abgeleitet werden (Spezialisierung). Sie **erben** die Eigenschaften und Methoden der Basisklassen.

Vererbung erlaubt den Klassen, gemeinsame Charakteristika zu teilen. Die Subklassen erben alle Eigenschaften und Methoden von der Oberklasse. Methoden können in den Subklassen **überschrieben** (siehe Polymorphie) oder neu dazugefügt werden.

Eine Klasse, die als Basis für Unterklassen herhalten soll, muss demnach so universell beschaffen sein (Generalisierung), dass ihre Eigenschaften und Methoden auch sinnvoll an Unterklassen weitervererbt werden können.
Auf diese Weise können hierarchische **Klassenbibliotheken** aufgebaut werden.

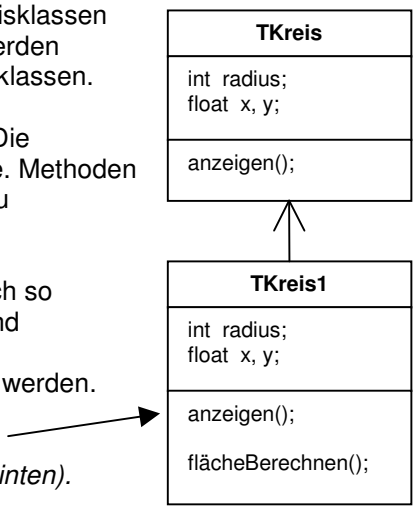


Bild - Neue Klasse TKreis1 ist Subklasse der Oberklasse TKreis.
*Klassen und Vererbung sind in **UML-Notation** dargestellt (UML: siehe hinten).*

```
                  // Der Vererbungszugriffs-Modifizierer "public" bezeichnet den
                  // Status der geerbten Elemente:
class TForm1 : public TForm   // public: geerbte Elemente von TForm1 gleich wie
                              // Elemente der Oberklasse TForm
...
```

Listing - Vererbung: die neue Klasse TForm1 erbt alle Elemente der Oberklasse TForm, die geerbten Elemente besitzen den gleichen Zugriff-Status wie in der Oberklasse (public, private oder protected), Header-Datei.

C++Builder, Vererbung und die Klassenhierarchie

Die Basisklasse TObjekt

ist der Ausgangspunkt der **VCL-Klassenhierarchie** im C++Builder, sozusagen der Urahn aller VCL-Objekte und Komponenten.

Basisklassen (z.B. TObject, TComponent) vererben ihre Eigenschaften und Methoden (TObject besitzt nur Methoden) über die Klassen-Hierarchie an Unterklassen.

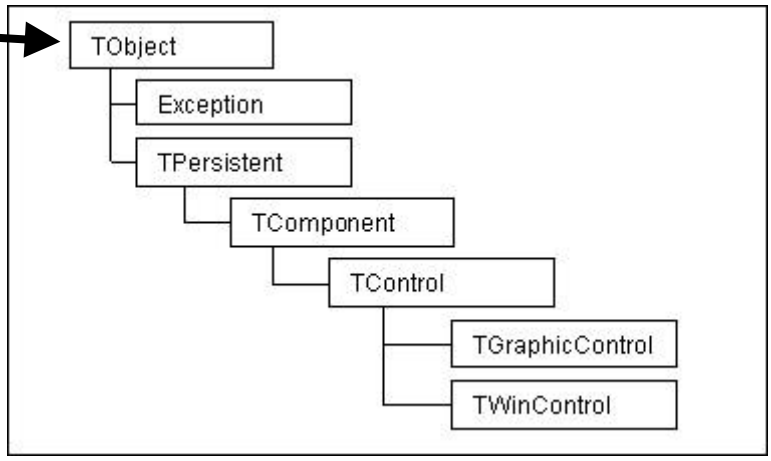
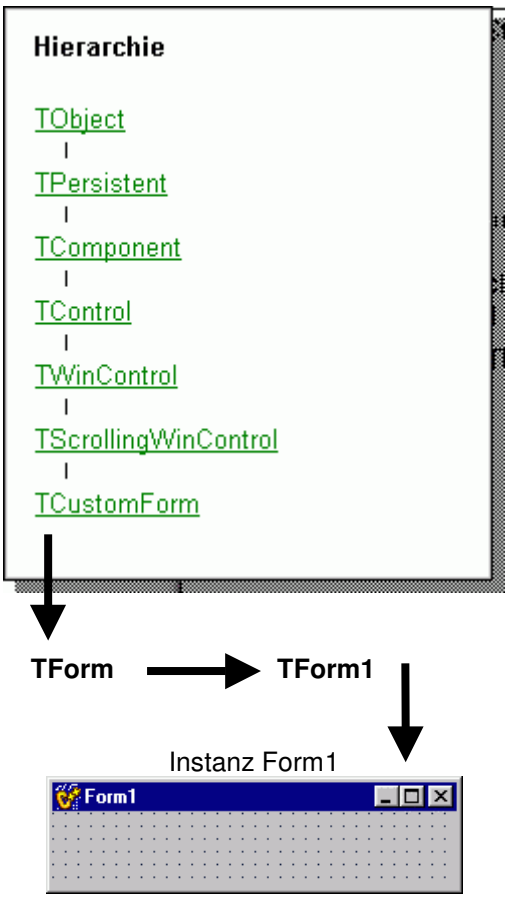


Bild - Klassenhierarchie der VCL-Bibliothek des C++Builder

C++Builder, Vererbung am Beispiel der Klassen TForm und TBitBtn

Klassen-Hierarchie von **TForm**



Beispiel-Klasse TForm:

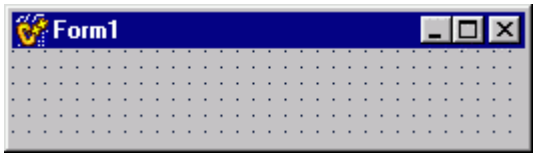


*Bild - Die Klasse **TForm** repräsentiert ein Standard-Anwendungsfenster (Formular).*

Formulare, die im Formular-Designer erstellt werden, werden als **Nachkommen von TForm** (z.B. die neue Klasse TForm1) implementiert. Formulare können das Hauptfenster einer Anwendung, Dialogfelder oder untergeordnete MDI-Fenster repräsentieren. Das Formular kann andere Objekte enthalten, z.B. TButton, TCheckBox und TComboBox.

Beispiel-Klasse TBitBtn:

*Bild - **TBitBtn** ist eine Schaltfläche, die auf ihrer Oberfläche ein Bitmap darstellen kann.*

TBitBtn stammt von **TButton** ab und erbt die Eigenschaften und Methoden von TButton. Neu dazu kommt die Eigenschaft, ein Bitmap darstellen zu können.

Bild - Ein neues Fensterformular erstellen bedeutet:
Eine neue Klasse (TForm1) wird abgeleitet von der Basisklasse TForm. Die neue Klasse TForm1 erbt alle Elemente von TForm. Weiter wird sie um zusätzliche Elemente (nämlich die Event Handler, Komponenten etc., die der Programmierer hinzufügt) erweitert und eine Instanz von ihr (Variable Form1) erzeugt.

Polymorphie

Polymorphie oder Polymorphismus (Vielgestaltigkeit, Vieldeutigkeit) ist die Fähigkeit einer Basisklasse und ihrer Unterklassen, eine von mehreren gleichartigen Methoden (gleicher Name, aber unterschiedliches Verhalten) einzusetzen. Geschieht dies **zur Laufzeit**, wird von "dynamischer Bindung" oder late binding gesprochen. Die entsprechende Methode muss in der Basisklasse als **virtual** vereinbart sein. Eine Basisklasse mit einer oder mehreren virtuellen Methoden nennt man **abstrakte Klasse**.

Beispiel Polymorphie

TFigur: abstrakte Klasse (UML: Darstellung in Kursivschrift)

TFigur ist Basisklasse der Subklassen TKreis und TRechteck. Da TFigur (als Basisklasse) sinnvollerweise auch die Möglichkeit bieten sollte (Schnittstelle vorbereiten!), eine noch unbekannte Figur am Bildschirm anzuzeigen, enthält TFigur eine "anzeigen()"-Funktion. Dies, obwohl noch nicht klar ist, was zukünftige Unterklassen von TFigur unter "anzeigen()" verstehen (ein Kreis wird naturgemäss anders angezeigt als ein Rechteck).

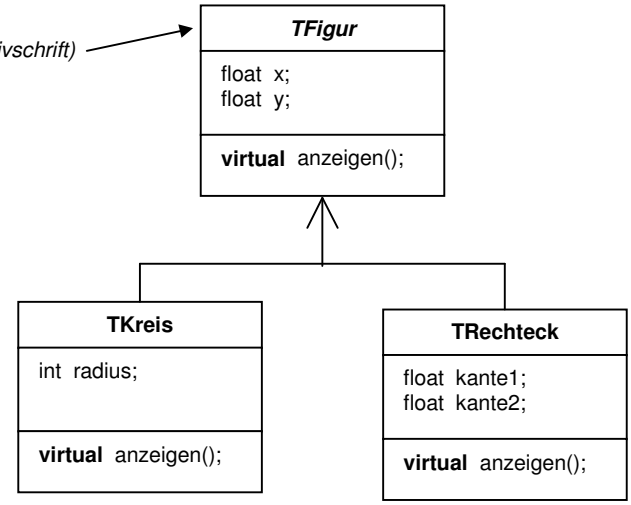


Bild - Die neuen Unterklassen weisen gleiche Elemente wie die Basisklasse auf (Vererbung), neue Elemente sind dazugekommen.

Im Beispiel ist *TFigur* eine abstrakte Klasse, weil die Methode "anzeigen()" in der Oberklasse nur deklariert, aber noch nicht definiert ist. Darum wird die Funktion anzeigen() in der Basisklasse mit der Vereinbarung **virtual** deklariert. Im Beispiel werden auch die entsprechenden Funktionen in den Unterklassen als virtual deklariert, dies zu Gunsten einer besseren Uebersicht. In den Unterklassen wird die Funktion anzeigen() jeweils passend definiert (codiert) und die geerbte, "leere" Funktion wird **überschrieben** (Ueberschreibung, overriding).

Mit der Vereinbarung virtual wird dem Compiler mitgeteilt, eine virtuelle Tabelle mit den Adressen aller Funktionen "anzeigen()" zu erstellen. Beim Aufruf einer anzeigen()-Funktion, wird **zur Laufzeit** zum jeweils passenden Funktionscode (mittels der Tabelle) gesprungen.

```
class TFigur      // abstrakte Basisklasse mit virtueller Methode
{
public:
      virtual void anzeigen(void);  // virtuelle (polymorphe) Methode
};
//---------------------------------------------------------------------

class TKreis : public TFigur        // Vererbung von TFigur an Unterklasse
{
public:
      virtual void anzeigen(void);  // virtuelle (polymorphe) Methode
};
//---------------------------------------------------------------------

class TRechteck : public TFigur     // Vererbung von TFigur an Unterklasse
{
public:
      virtual void anzeigen(void);  // virtuelle (polymorphe) Methode
};
//---------------------------------------------------------------------
```

Listing - Deklaration von virtuellen Methoden im Header-File (vereinfacht)

Objektorientierte Programmierung und Programmentwicklung

Was heisst visuelle Programmierung, GUI und RAD?

Sprachen mit dem Zusatz "visual" (z.B. C++Builder, Visual-C, Visual-Basic, Delphi) enthalten einen GUI-Builder (GUI = graphical user interface), der visuelle Komponenten (z.B. Forms, Panels, Buttons etc.) als Teil einer Klassenbibliothek zur Verfügung stellt. Zur Programmerstellung werden diese auf einem Fenster-Formular platziert, so dass die entstehende Benutzeroberfläche sofort sichtbar wird. Auf diese Weise können Windows-Anwendungen mit GUI in kurzer Zeit erstellt werden (→ RAD = rapid application development, schnelle Anwendungsentwicklung).

Jedoch: C++Builder starten, Komponenten platzieren und Event Handler mit Code "füllen", ist noch lange keine OO-Programmierung!

Die OO-Programmierung

geschieht unter **Zuhilfenahme und Benutzung von Klassen und Objekten** (Gegenstände, Aktionen), die miteinander kommunizieren und kooperieren.

OOP ist nicht an eine bestimmte Programmiersprache gebunden, wird aber von objektorientierten Sprachen wie Java, C++, Objekt-Pascal (Delphi) u.a. stark unterstützt.

Die OO-Programmentwicklung

besteht darin, das zu erfassende System in Objekte und ihre Beziehungen untereinander zu zerlegen. Objektorientierte Programmierer identifizieren Klassen und Objekte mit Eigenschaften und Methoden. Meldungen, die den Objekten geschickt werden, sorgen dafür, dass die entsprechenden Methoden auf die Daten des Objekts zugreifen und diese modifizieren.

Vereinfacht (nach Booch) kann die OO-Programmentwicklung in folgende Teile gegliedert werden:

OOA, Objekt orientierte Analyse
Während der OO-Analyse ist der Entwickler mit der Frage beschäftigt, wie er passende Objekte und Klassen für das zu lösende Problem gewinnen und entwickeln kann.
Typische Fragestellungen lauten:
- "Welche Objekte und Klassen sind in meinem Programm nützlich?"
- "Was soll mein Programm und was sollen die Objekte meiner Klassen tun?"
- "Welche Beziehungen haben die Objekte untereinander?"

OOD, Objekt orientiertes Design
Während der Phase des OO-Designs ist der Entwickler mit der Frage beschäftigt, wie er die gefundenen Klassen implementieren (umsetzen) kann, damit deren Eigenschaften und Methoden die Anforderungen an die Klasse erfüllen.

Für die OO Entwicklung stehen Methoden wie Entwurfsmuster (Design Patterns), grafische Modellierungssprachen wie UML, u.a. zur Verfügung.

Unified Modeling Language (UML) ist eine grafische Sprache zur Visualisierung, Spezifikation, Konstruktion und Dokumentation von Softwaresystemen. Es ist eine standardisierte Modellierungs-Sprache, die jedoch keinen bestimmten Software-Entwicklungsprozess vorschreibt. UML ist losgelöst von einer konkreten Programmiersprache, basiert aber auf OOP und Klassendiagrammen.
UML ermöglicht die Modellierung eines Systems, d.h. die Abstraktion der Realität mit Konzentration auf das Wesentliche. Modellierung hilft, die gewünschte Struktur und das Verhalten eines Systems zu beschreiben und ein System besser zu verstehen.

Unterschiede zwischen OOP und prozeduraler Programmierung

- OOP: der Programmierer kümmert sich zuerst um die Daten und in einem zweiten Schritt um die Operationen (Methoden), die er für die Manipulation dieser Daten braucht.
- Prozedurale Sprache: der Programmierer denkt zuerst über die Prozeduren nach und dann erst über die Daten, die von den Prozeduren bearbeitet werden. "Prozedurale" Programmierer schreiben Funktionen und übergeben ihnen Daten.

5. C++Builder: Was ist die API und die VCL?

API

Application Programming Interface
Anwendungs-Programmier-Interface von MS-Windows. Dies ist die Programmierschnittstelle (vor allem System-Funktionen und -Konstanten), die MS-Windows dem Softwareentwickler zur Verfügung stellt.

Beispiele für Funktionen der API:
- Die Funktion **MessageBox**() zeigt ein konfigurierbares Windows-Meldungsfenster mit Buttons an.
- Die Funktion **ShellExecute**() öffnet Dateien, mit der über die Dateiendung zugeordneten EXE-Anwendung oder führt EXE-Dateien aus.
- Die Funktion **SHFileOperation**() kopiert, verschiebt, löscht, benennt Dateien und Ordner, in Windows-Manier, mit Grafik-Animation "fliegende Dateien", löschen in den Papierkorb etc.

Die API ist in der **Windows-SDK-Hilfe-Datei** dokumentiert (SDK = Software Development Kit = Werkzeuge und Hilfestellung von Microsoft zur Softwareentwicklung).

Beispiel: Suche in der Hilfe-Datei WIN32_API_SDK.hlp oder in der SDK-Hilfe im C++Builder die obigen Funktionen.

VCL

Visual **C**omponent **L**ibrary, visuelle Komponentenbibliothek für Borland-Delphi und C++ Builder, ist eine Klassenbibliothek mit visuellen Komponenten. Sie kapselt u.a. Windows-API-Funktionen und stellt sie dem Programmierer als Klassen und (visuelle) Komponenten zur Verfügung.

Dies vereinfacht die Anwendung der Windows-API-Funktionen.

Die VCL basiert auf einem Objektmodell mit Eigenschaften, Methoden und Ereignissen. Dieses Modell definiert die Datenelemente (Eigenschaften), die Funktionen, die mit den Daten durchgeführt werden (Methoden) und eine Möglichkeit der Interaktion zwischen den Objekten (Ereignisse). Die Bibliothek besteht aus einer Hierarchie von in Object Pascal geschriebenen Objekten und ist nahtlos in die IDE integriert. Mit Hilfe der Komponentenpalette und des Objektinspektors können beliebige VCL-Komponenten in ein Formular eingefügt und deren Eigenschaften bearbeitet werden, ohne dass auch nur eine Quelltextzeile zu schreiben ist.

Eigenschaften
Eigenschaften definieren die Merkmale der Objekte und können zur Entwurfszeit angezeigt und bearbeitet werden. Bei Wertänderungen ist sofort ersichtlich, wie sich diese auf die Komponente auswirken. Mit Hilfe einer durchdachten Eigenschaftsimplementierung können die Komponenten von den Benutzern einfacher eingesetzt und besser gepflegt werden.

Methoden
Methoden sind Funktionen, die zu einer bestimmten Klasse gehören. Mit ihnen kann auf alle Eigenschaften (public, protected und private) und Datenelemente der Klasse zugegriffen werden. Sie werden oft auch Elementfunktionen genannt.

Ereignisse
Ereignisse werden durch bestimmte Operationen in einem Objekt ausgelöst. Dieser Mechanismus ermöglicht ein Konzept von Aktion und Reaktion, bei dem der Ausführungsfluss nicht zwangsläufig sequentiell erfolgen muss, die ereignisgesteuerte Programmierung. Hier bestimmt der Benutzer und nicht die Anwendung, welche Funktionen in welcher Reihenfolge ausgeführt werden. Der Programmierer einer Windows-Anwendung kann dann aber natürlich nicht wissen, welche Aktion als nächste vorgenommen wird. Die Benutzer können einen Menübefehl wählen, auf eine Schaltfläche klicken oder einen bestimmten Textblock markieren. Somit bedeutet ereignisgesteuert, dass die entsprechende Behandlungsroutine auf die Ereignisse reagiert, die für die Logik der Anwendung von Belang sind.

(Angepasster Auszug aus der C++Builder-Help)

Beispiel zum Thema API, VCL

C++Builder stellt Teile der API - "verpackt" in der Klassenbibliothek VCL - zur Verfügung, zum Teil in Form visueller Komponenten. Als Beispiel die API-Funktion MessageBox().

API: Funktion MessageBox(Parameter)

Doku: Windows-SDK-Help-Datei, C++Builder

Win32 Developer's References

Datei Bearbeiten Textmarke Optionen ?

Inhalt Index Zurück << >>

MessageBox Quick Info Overview Group

The **MessageBox** function creates, displays, and operates a message box. The message box cor
predefined icons and push buttons.

```
int MessageBox(
  HWND hWnd,          // handle of owner window
  LPCTSTR lpText,     // address of text in message box
  LPCTSTR lpCaption,  // address of title of message box
  UINT uType          // style of message box
);
```

Parameters

hWnd
Identifies the owner window of the message box to be created. If this parameter is NULL, the m

lpText
Points to a null-terminated string containing the message to be displayed.

lpCaption
Points to a null-terminated string used for the dialog box title. If this parameter is NULL, the de

uType
Specifies a set of bit flags that determine the contents and behavior of the dialog box. This par
Specify one of the following flags to indicate the buttons contained in the message box:

Flag	Meaning
MB_ABORTRETRYIGNORE	The message box contai
MB_OK	The message box contai
MB_OKCANCEL	The message box contai
MB_RETRYCANCEL	The message box contai
MB_YESNO	The message box contai
MB_YESNOCANCEL	The message box contai

VCL: Application->MessageBox(Parameter);

Die VCL kapselt die API-Funktion, welche jetzt eine Methode der Klasse TApplication ist.
Doku: VCL-Help-Datei, C++Builder

Borland C++Builder - Hilfe

Datei Bearbeiten Textmarke Optionen ?

Hilfethemen Zurück Drucken << >>

TApplication::MessageBox

TApplication Siehe auch Beispiel

Mit der Methode MessageBox können Sie eine bestimmte Meldung für den Benutzer anzeigen.

```
int __fastcall MessageBox(const char * Text, const char * Caption, int
```

Beschreibung

Mit MessageBox können Sie ein generisches Dialogfeld mit einer Meldung und einer oder mehrere
Parameter Caption ist der Titel des Dialogfeldes und ist optional.

MessageBox ist eine Kapselung der Funktion MessageBox der Windows-API. Das für die API ber
automatisch bereitgestellt.

Der Wert des Parameters Text ist die Meldung, die gegebenenfalls länger als 255 Zeichen sein ka
Meldungsfeld automatisch umbrochen. Der Wert der Eigenschaft Caption ist der Titel, der in der T
wird. Titel können länger als 255 Zeichen sein. Sie werden jedoch nicht umbrochen. Ein langer Tit

Die Werte des Parameters Flags bestimmen die Schaltflächen, die im Meldungsfeld angezeigt we
Fensters (mögliche Rückgabewerte). Die folgende Tabelle führt die zulässigen Werte auf. Um den
können die Werte auch kombiniert werden.

Wert	Bedeutung
MB_ABORTRETRYIGNORE	Das Meldungsfeld enthält die drei Schaltflächen: Abbruch, Wied
MB_OK	Das Meldungsfeld enthält eine Schaltfläche: OK. Dies ist die Sta
MB_OKCANCEL	Das Meldungsfeld enthält two Schaltflächen: OK und Abbrechen
MB_RETRYCANCEL	Das Meldungsfeld enthält two Schaltflächen: Wiederholen und A
MB_YESNO	Das Meldungsfeld enthält two Schaltflächen: Ja und Nein.
MB_YESNOCANCEL	Das Meldungsfeld enthält three Schaltflächen: Ya, Nein und Abb

Der Rückgabewert der Methode MessageBox ist 0, wenn nicht genügend Speicher zur Erstellung
Andernfalls wird einer der folgenden Werte zurückgegeben:

Wert	Numerischer Wert	Bedeutung

Bild - Die VCL kapselt die API-Funktion "MessageBox"

6. C++Builder: Ereignisgesteuerter Programmablauf und visuelle Komponenten

C++Builder und Windows: was ist ein Ereignis (Event) und was läuft ab?

Windows arbeitet mit **Ereignissen (Events)**: die Maus wird bewegt, ein Button geklickt, ein Menüpunkt ausgewählt, eine Taste gedrückt etc. Hier ein Beispiel-Ablauf im Falle eines Mausklick-Events:

- Der Mausklick (z.B. auf den OK-Button einer Anwendung) erzeugt über den Mouse-Port einen Hardware-Interrupt im Prozessor.
- Der Prozessor beantwortet den Interrupt, indem er den laufenden Job unterbricht und die passende Interruptroutine (Mouse-Interrupt-Handler) abzuarbeiten beginnt. Der Mouse-Interrupt-Handler ist in diesem Fall ein Teil der Maustreiber-Software.
- Als Resultat der Abarbeitung des Interrupt-Handlers wird das Betriebssystem (hier Windows) über die Art des eingetretenen Ereignisses informiert.
- Windows stellt fest, auf welcher Applikation (welchem Fenster) und auf welcher GUI-Komponente (hier der OK-Button) der Mausklick stattgefunden hat (wo der Mauszeiger gerade ist...).
- Windows ruft jetzt eine vordefinierte Methode dieser Anwendung auf und veranlasst, dass diese aktiviert wird.
- Die Anwendung führt nun den zugehörigen Event Handler (z.B. TForm1::ButtonClick1(...)) aus, der vom Anwendungsprogrammierer vordefiniert wurde.

C++Builder, Komponenten und Ereignisse

Dank den visuellen Komponenten der VCL kann eine Anwendung mit einem grafischen Benutzer-Interface (GUI) das von Ereignissen gesteuert wird, sehr schnell und einfach erstellt werden (RAD).

Beispiel: **Komponente der Klasse TButton** auf dem Formular (z.B. Form1) platzieren und die Eigenschaften der Komponente - eigentlich Instanz von TButton - im Objektinspektor festlegen. Die Variable Button1 ist jetzt ein Datenelement der Klasse TForm1.

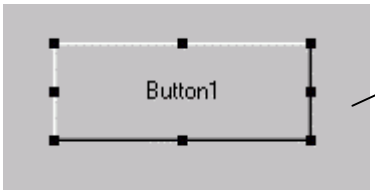


Bild - Instanz von TButton

Bild - Einige Eigenschaften von TButton

Da C++Builder zur Programmierung von Windows-Anwendungen erschaffen wurde, stellt die VCL-Klassenbibliothek dem Programmierer Windows-Ereignisse (siehe oben) zur Verfügung. Einem Ereignis kann eine Funktion - ein **event handler** resp. eine **Ereignisbehandlungsroutine** - zugeordnet werden, die beim Eintreten des Ereignisses (z.B. OnClick) ausgeführt wird.

Ein C++Builder-Programm ist (oft nur) eine Sammlung von Event-Behandlungsroutinen, die bei Bedarf (nicht sequenziell!) abgearbeitet werden.

Beispiel: **Komponente TButton** mit zugehörigen Ereignissen. Dem Ereignis OnClick (Mouseklick) eines Buttons kann auf zwei verschiedene Arten ein Event handler (Funktion) zugeordnet werden:

(1) Der Name einer **neuen Funktion** kann eingegeben werden (oder Doppelklick auf das Feld). Die Funktion wird automatisch im Header deklariert (**__published:**) und ein Funktionsrumpf wird im Quelltext platziert.

(2) Ebenso kann eine **bestehende Funktion** aus der DropDown-Liste ausgewählt und dem Ereignis zugeordnet werden. Die Funktion muss bereits im Header unter __published: deklariert sein.

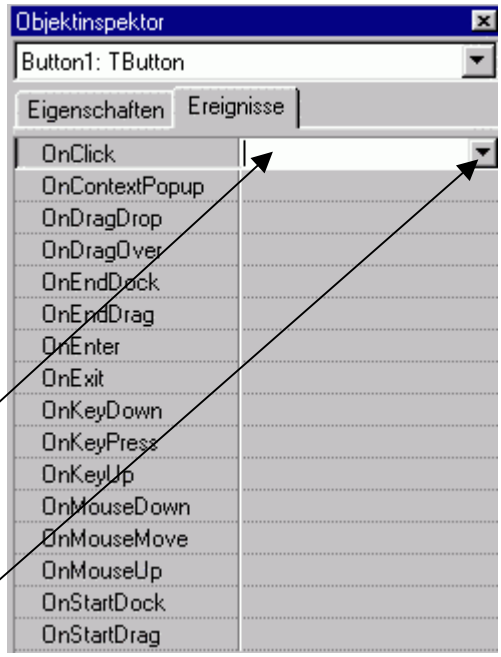


Bild - Ereignisse von TButton

```
//---------------------------------------------------------------------------
                    Klasse      Funktionsname mit Parametern
void __fastcall TForm1::Button1Click(TObject *Sender)
{
// Panel2 ein- und ausschalten

     Panel2->Visible = ! Panel2->Visible;
     Memo1->Lines->Add("Panel2 aus- und einschalten");
}
//---------------------------------------------------------------------------
```

Listing - Event handler "OnClick" von Button1: der Code im Funktionsrumpf wird abgearbeitet beim Eintreten dieses Ereignisses. Button1Click() ist eine Methode der Klasse TForm1, CPP-Datei.

7. Ein C++Builder-Projekt und seine Dateien

	Header-, Quelltext- und Projekt-Dateien
Unit1.H **Unit1.CPP**	Unit1, C-**Header-Datei**, Deklarationen, Schnittstelle Unit1, C-Quelltext, **Implementation**
Projekt.CPP	Main()-Funktion C-Quelltext, von C++Builder vorgegeben
Unit1.DFM	Formularaufbau in Textform zur Unit1.CPP. Als Text darstellen: Eigenschaften der Objekte werden angezeigt
Projekt.RES	Grafik-Ressourcen (Icon-Datei), Binärdatei für Projekt
Projekt.BPR	Textdatei für die Compilierung eines Projekt (make file), alle Dateien, die zum Projekt gehören, sowie Pfadnamen, Compilerinfos etc. sind hier vermerkt.
Projekt.DSK	IDE-Desk, User-Konfiguration der IDE
Dateien mit Endungen **obj, tds, ils, ilc, u.a.**	Compiler-Hilfsdateien Mit der Batch-Datei ***CLEAN.BAT*** kann der Projektordner nach der Arbeit gesäubert werden, da die Hilfsdateien des Compilers sehr viel Speicherplatz benötigen.
	Dateien für die Komponenten-Handhabung
BPK	Textdatei wie BPR, jedoch für die Komponenten-Package-Erzeugung (make file)
BPL	Kompilierte Binärdatei für die Komponenten-Erzeugung, Binär-Package
DCR	Binärdatei, Ressource-Icon-Datei für Komponenten-Package (wie RES)

Tabelle - Wichtigste Datei-Endungen von C++-Builder-Dateien

8. Uebungen C++Builder (ohne OOP)

(1) C++Builder und visuelle Komponenten

Studiere das Beispiel-Programm "Formular-Aufbau im C++Builder" und kompiliere die Anwendung.

(2) C++Builder und Windows-API

Erstelle eine Windows-Anwendung, mittels derer in einer Eingabezeile (TEdit) ein EXE-Befehl (z.B. notepad, regedit, wordpad, www.google.com) oder ein Pfadname (z.B. C:) eingegeben und ausgeführt werden kann. Benutze dafür die API-Funktion ShellExecute(). Erstelle eine mustergültige Dokumentation dazu.

Literatur, Quellen

- C++Builder-Online-Help
- WIN32-API-SDK, Help-Datei zur Windows-API, Microsoft
- C-Programmierung lernen, André Willms, Addison-Wesley, 1998
- C++Builder 5.0, Dirk Abels, Addison-Wesley, 2000
- C++ für Kids, Hans-Georg Schuhmann, MITP-Verlag, 1999
- Delphi- und C++-Schnellübersicht, Dirk Louis, Markt&Technik, 1996
- Software Engineering, div. Vorlesungsunterlagen von Martin Jud, St. Gallen, Luzern
- Diverse Internet-Artikel

■