

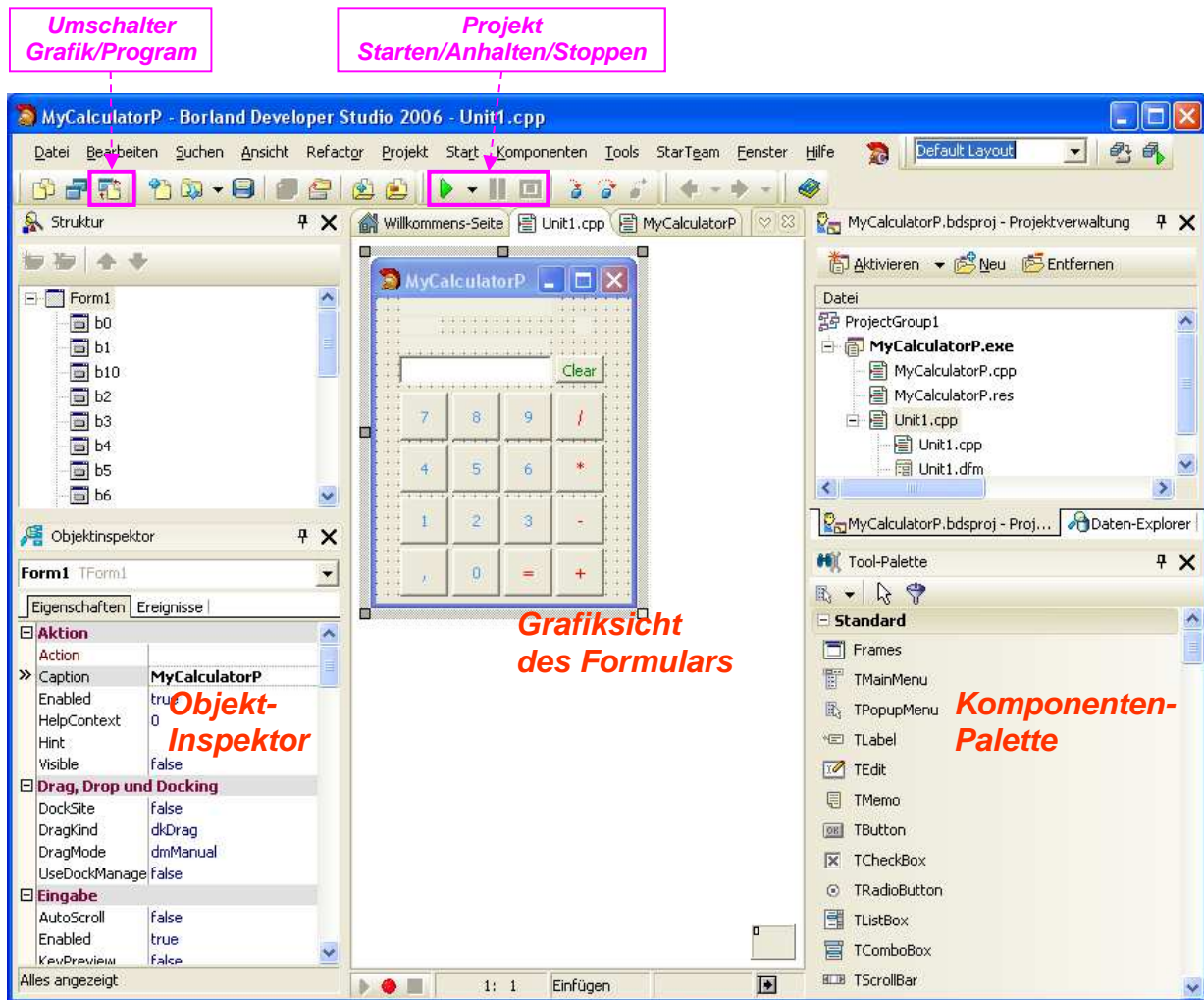
Ziel: Kurzeinführung in die Embarcadero-Entwicklungsumgebung

Inhalt:

1	Übersicht	2
1.1	Die Oberfläche	2
1.2	Tastenkürzel, Menüs→Befehl	2
1.3	Übung „MyProject“	3
1.4	Die Projektdatei und Unitdatei dieses Beispiels	4
1.5	Was ist ein Object?	5
1.6	Elementare Klassen zur Gestaltung von Oberflächen	6
2	Komponenten-Eigenschaften, -Methoden, -Ereignisse	7
2.1	TForm: Eigenschaften, Ereignisse	7
2.2	TButton: Eigenschaften, Ereignisse	7
2.2.1	Übung: TButton-Eigenschaften	7
2.2.2	TButton- Ereignisse	8
2.3	TEdit: Eigenschaften, Methoden, Ereignisse	9
3	Wichtige Datentypen in C++	11
4	Steuerstrukturen	12
4.1	if-else	12
4.2	for	12
4.3	do-while (post-tested)	12
4.4	while (pre-tested)	12
4.5	switch.....	12

1 Übersicht

1.1 Die Oberfläche



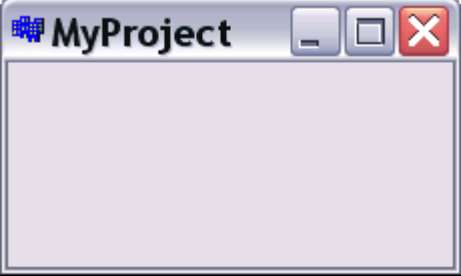
1.2 Befehl, Tastenkürzel, Menü

Befehl	Kürzel	Menü
Fensterliste	ALT+0	Ansicht→Fensterliste
Umschalten: Formular/Unit	F12	Ansicht→Umschalten Formular/Unit
Objektinspektor	F11	Ansicht→Objektinspektor
Start mit Debugger	F9	Start→Start mit Debugger
Start ohne Debugger	UMSCH+STRG+F9	Start→Start ohne Debugger
Hilfe	F1+<ausdruck>	







☀ **Übung:** Legen sie Befehle zur Fehlersuche in die Symbolleiste!

🔍 Ansicht→Symbolleisten→Symbolleisten→Fehlersuche

1.3 Übung „MyProject“

	<p>So soll die Windowsanwendung MyProject zu Beginn aussehen. Wir werden diese Schritt für Schritt erweitern.</p> <p>Beachte: es ist sinnvoll, für jedes Projekt ein neues Verzeichnis zu verwenden, damit man die automatische Bezeichnerfunktion der IDE nutzen kann.</p>
---	--

Schritt-für-Schritt-Anleitung

-  Datei → Neu → VCL-Formularanwendung oder
Datei → Neu → Weitere → C++Builder-Projekte → VCL¹-Formularanwendung
-  Datei → Speichern unter... *user\borlandcpp\MyProject* mit Namen *MyProject*
-  Gehen Sie mit dem Dateieexplorer in den Ordner "MyProjekt" und kontrollieren sie, welche Dateien und Ordner die Entwicklungsumgebung angelegt hat!
-  Das Projekt starten → Compilieren, Linken, Ausführen
-  Ausprobieren: In den Hintergrund, ganze Seite, Schließen
 → Das Objekt Form1 hat Basiseigenschaften eines Windows-Fensters geerbt.
-  Dateieexplorer öffnen und erneut im Ordner "MyProjekt" nachsehen, welche Dateien neu angelegt wurden.

→ Mehrere Dateien werden für jedes Projekt erzeugt:

Datei → Neu → Weitere → C++Builder-Projekte → VCL-Formularanwendung erzeugt folgende Dateien	
MyProject.bdsproj	Borland-Developer Studio Project File: Enthält Quelltext zur Initialisierung des Projektes sowie Informationen zu den verwendeten Formularen und Units.
MyProject.bdsproj.local	LOCAL-Datei mit Aufzeichnung der Nutzeraktionen
MyProject.cpp	CPP-Datei mit Projektquelltext (Hauptprogramm), wird durch die IDE angelegt
MyProject.res	Ressourcendatei, z.B. WAV-Dateien oder Bilder die das Programm benutzt. Ressourcen werden mit an das Programm gelinkt
Unit1.cpp	Formular-Quelltext (das geschriebene Programm)
Unit1.dfm	Delphi-Formulardatei, grafische Eigenschaften des Formulars. Zu jeder *.dfm-Datei gehört eine *.cpp-Datei
Unit1.h	Header-Datei der Unit1, enthält Objektdeklaration
Beim Compilieren und Linken wird neuer Ordner Win32\Debug u.a. mit folgenden Dateien erstellt:	
MyProject.exe	Ausführbares Programm
MyProject.obj	CPP-compiled-unit des Hauptprogrammes
MyProject.tds	Hilfsdatei für den Linker
Unit1.obj	CPP-compiled-unit der Unit1

Überschriebene Dateien haben vor der Erweiterung ein Tilde: *.~cpp, *.~dfm. Solche Dateien kann man löschen.

ACHTUNG: Folgende Einstellungen sind vorzunehmen, damit die EXE-Files auf jedem Rechner laufen:

- (1) Menü Project > Optionen > Packages > Mit Laufzeitpackages linken → abwählen!
- (2) Menü Project > Optionen > C++-Linker > Mit der Delphi-Laufzeitbibliothek linken → abwählen!

¹ VCL - Visual Component Library, Satz visueller Komponenten für die beschleunigte Erstellung von Windows-Anwendungen.

1.4 Die Projektdatei und Unitdatei dieses Beispiels

📁 Projekt → Quelltext → MyProject

➔ Inhalt dieser Datei wird automatisch erzeugt, als Anfänger → Hände weg!!!

```
//-----  
#include <vcl.h> //vcl - Visual Component Library  
#pragma hdrstop //Compilerdirektive: Abschluss der Header-Dateien-Liste  
//-----  
USEFORM("Unit1.cpp", Form1); //nutze Unit1.cpp  
//-----  
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) //Erzeugung einer Applikation  
{  
//Das try-catch-Statement bietet eine Möglichkeit, einige oder alle möglichen Fehler zu  
behandeln, die in dem durch try gekennzeichneten Codeblock auftreten können, während dieser  
Code noch ausgeführt wird. Die Ausnahmebehandlung steht nach catch.  
    try  
    {  
        Application->Initialize(); //Appl. wird initialisiert  
        Application->CreateForm(__classid(TForm1), &Form1);  
        Application->Run(); //Appl. wird gestartet  
    }  
    catch (Exception &exception)  
    {  
        Application->ShowException(&exception);  
    }  
    catch (...)  
    {  
        try  
        {  
            throw Exception("");  
        }  
        catch (Exception &exception)  
        {  
            Application->ShowException(&exception);  
        }  
    }  
    return 0;  
}
```

📁 In Unit1.cpp → RMT auf "Unit1.h" → Quelltext/Headerdatei öffnen

```
#ifndef Unit1H  
#define Unit1H  
//-----  
#include <System.Classes.hpp>  
#include <Vcl.Controls.hpp>  
#include <Vcl.StdCtrls.hpp>  
#include <Vcl.Forms.hpp>  
//-----  
class TForm1 : public TForm //neue Klasse TForm1, erbt Öffentliches von TForm  
{  
    __published: // Von der IDE verwaltete Komponenten  
    private: // Anwender-Deklarationen  
    public: // Anwender-Deklarationen  
    __fastcall TForm1(TComponent* Owner);  
};  
//-----  
extern PACKAGE TForm1 *Form1;  
#endif
```

📁 Projekt → Quelltext → Unit1

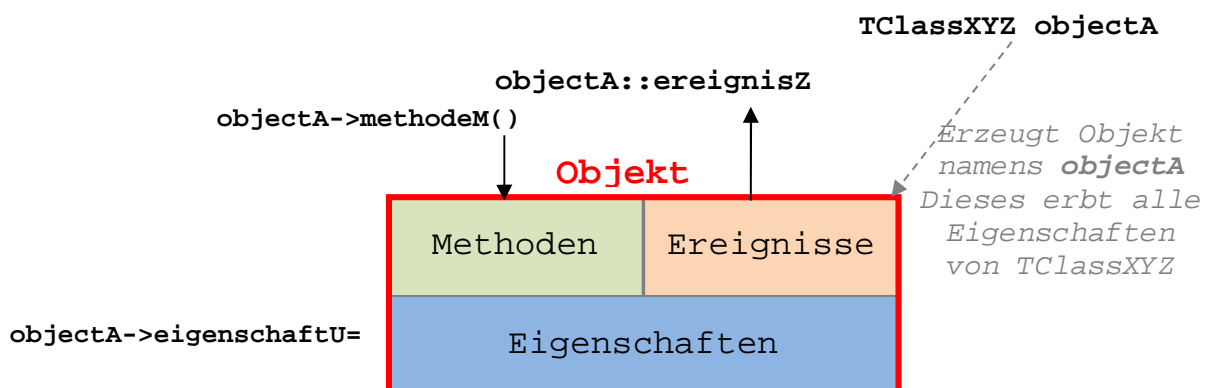
```
#include <vcl.h>  
#pragma hdrstop  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1; //Zeiger auf Variable Form1 der Klasse TForm1  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner) //Konstruktor der Klasse TForm1  
: TForm(Owner)  
{  
}
```

1.5 Was ist ein Objekt?

- (1) Ein Objekt wird (wie eine Variable) von einem Typ abgeleitet.
Diesen Typ nennt man im Unterschied zu Datentypen → Klasse (class) oder → ObjektTyp.
Bei Embarcadero-CPP beginnen alle Klassen mit einem großen "T", z.B. TButton, TMemmo, TEdit, TForm.
- (2) Alle Objekte sind von dem Urtyp TObject abgeleitet, d.h. sie "erben" alle Merkmale von TObject.
- (3) Objekten bestehen aus:
 - **Daten**, bzw. **Datenfelder**: dies sind lokale Daten des Objektes und nur zugreifbar, durch eigene Methoden des Objektes.
 - **Methoden**, dies sind Unterprogramme des Objektes. Mittels Methoden kann man auf Daten und Eigenschaften des Objektes zugreifen, Objekte erzeugen und beenden. Man unterscheidet deshalb:
 - **function**, Methode zum Zugriff auf Daten und Eigenschaften,
 - **constructor**, Methode zum Erzeugen eines Objektes und ev. der Initialisierung,
 - **destructor** Methode zum Freigeben der allokierten Ressourcen eines Objektes.
 - **Eigenschaften (properties)**, sind Merkmale von Objekten wie: Aussehen, Position, Sichtbarkeit usw.
 - **Ereignisse**, die ein Objekt in einer grafischen Oberfläche auslösen kann: OnClick, OnEnter, OnExit, OnCreate usw.

Zusammengefasst:

- Ein Objekt ist eine "Variable" mit den Merkmalen von "TClassXYZ", die Deklaration lautet:
`TClassXYZ objectA.`
Der einmalige Name macht Objekte unterscheidbar.
- **Objektmethoden** sind Funktionen ohne oder mit Rückgabe:
`objectA->methodeM(); retWertK = objectA->methodeN();`
- **Objekteigenschaften** kann man durch Zuweisungsoperationen ändern:
`objectA->eigenschaftU = wert, bzw.
wert = objectA->eigenschaftV.`
- **Objektereignisse** werden durch "Ereignisbedienroutinen" verarbeitet (ähnlich einer Interruptbedienroutine).
`objectA::ereignisZ(Sender:TObject)`



☀ **BEISPIELE:**

```
Form1->Visible=True;
```

```
//der Objekt-Eigenschaft "Sichtbar", des Objektes Form1, wird Wert "True" zugewiesen.
```

```
Form1->Edit1->Text ="Hallo";
```

```
//der Objekt-Eigenschaft Text, des Objektes Edit1 im Form1, wird Wert "Hallo" zugewiesen.
```

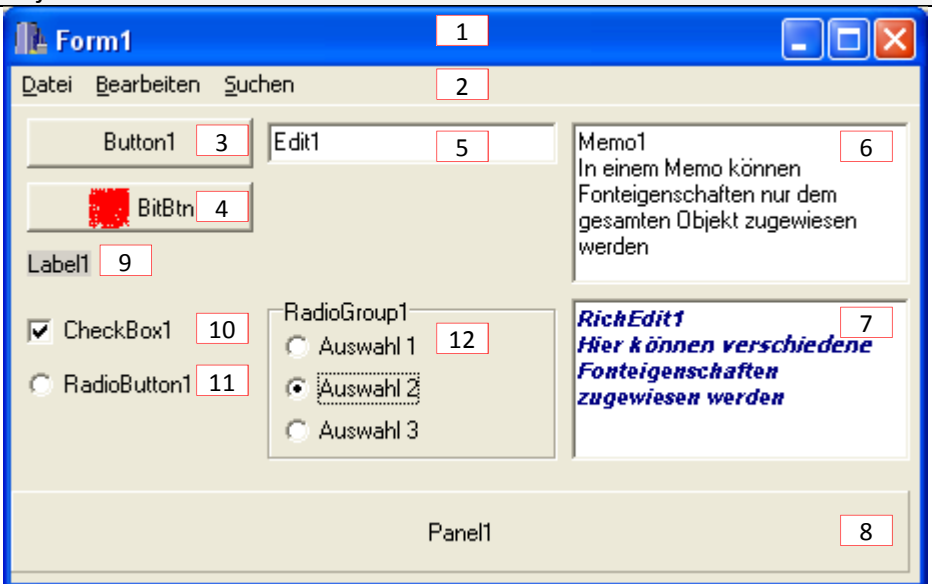
```
Form1->Memo1->Lines->Add("Hallo");
```

```
//auf die Objekt-Eigenschaft Lines des Objektes Memo1 im Form1 wird die Methode Add angewendet.
```

1.6 **Elementare Klassen zur Gestaltung von Oberflächen**

Die IDE "Borland-CPP" stellt viele Klassen in der **Komponentenpalette** bereit.

→ Elementare Bedeutung haben z.B. folgende Klassen:

Klasse	Objekte
1 TForm	
2 TMainMenu	
3 TButton	
4 TBitBtn	
5 TEdit	
6 TMemo	
7 TRichEdit	
8 TPanel	
9 TLabel	
10 TCheckBox	
11 TRadioButton	
12 TRadioGroup	

2 Komponenten-Eigenschaften, -Methoden, -Ereignisse

2.1 TForm: Eigenschaften, Ereignisse

Ein Formular ist das grundlegende Objekt einer Anwendung mit Oberfläche.

→ TForm hat viele Eigenschaften, beispielsweise:

Eigenschaften (properties)	Funktion
Caption	Aufschrift
Constraints	Zwänge: Max. Höhe, Breite; min. Höhe, Breite
Cursor	Kursor-Art bei Mouse over
Enabled	Für Ereignisauslösung freigegeben
Font	Schrift für Caption
Height	Höhe
Left	Position von links
Name	Objektname
Top	Position von oben
Visible	Sichtbarkeit des Objekts
Width	Breite

→ Wichtige Ereignisse im Lebenszyklus sind:

Ereignisse (events)	Funktion
OnCreate	Erzeugen des Formulars, Ressourcenzuweisung
OnShow	Das Formular wird sichtbar
OnPaint	Neuzeichnen eines bisher verdeckten Formulars
OnActivate	Das Formular erhält den Focus, und erhält alle Eingaben (Maus, Tastatur)
OnResize	Formulargröße verändert
OnClose	Formular wird geschlossen
OnHide	Wird unsichtbar
OnDestroy	Formular wird zerstört, Freigabe allozierter Ressourcen

2.2 TButton: Eigenschaften, Ereignisse


Mit TButton wird eine Standardschaltfläche in einem Formular platziert. TButton führt mehrere Eigenschaften zur Steuerung des Verhaltens von Schaltflächen in Dialogfeldern ein. Die Benutzer können durch Auswahl eines Schaltflächen-Steurelements eine Aktion auslösen.

Wenn eine Schaltfläche Bild und Textes anzeigen soll, verwenden Sie TBitBtn. Soll eine Schaltfläche den Status "gedrückt" beibehalten, setzen Sie TSpeedButton ein.

2.2.1 Übung: TButton-Eigenschaften

 Weiterbearbeitung des Projektes "MyProject"

 Standard-Objekt-Bibliothek → TButton auswählen → in Formular platzieren.

 Objektinspektor (F11) → Eigenschaften ansehen

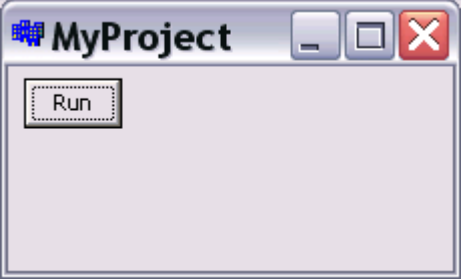
→ TButton hat viele Eigenschaften, beispielsweise

Eigenschaften (properties)	Funktion
Caption	Aufschrift
Constraints	Zwänge: Max. Höhe, Breite; min. Höhe, Breite
Cursor	Kursor-Art bei Mouse over
Enabled	Für Ereignisauslösung freigegeben
Font	Schrift für Caption
Height	Höhe
Hint	Hinweistext
Left	Position von links
Name	Objektname
ShowHint	Zeige Hinweistext
Top	Position von oben
Visible	Sichtbarkeit des Objekts
Width	Breite

Einige der Eigenschaften (z.B. Name) müssen einen Wert haben, andere nicht.
Eigenschaften können verändert, abgefragt werden:

- zur Entwurfszeit durch den so genannten "Objektinspektor",
- zur Laufzeit durch Zuweisungen → kommt später.

Übung:

	<p>Erweitern Sie "MyProject"</p> <ul style="list-style-type: none"> o mit Objekt vom Typ (der Klasse) TButton, o mit dem Namen "Button1", o der Aufschrift "Run", o Größe BxH=50x25 Pixel, o Position: Top/Left=6/8, o Hinweistext "Run, hier klicken".
---	---

2.2.2 TButton- Ereignisse

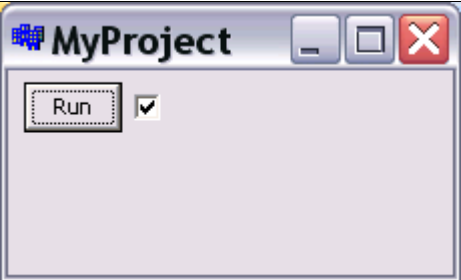
- 🔧 Objekt „Start“ auswählen
- 🔧 Objektinspektor → Ereignisse

→ Ereignisliste

Ereignisse (events)	Quelle	Ereignis tritt ein, wenn
OnClick	Maus	auf Objekt geklickt wurde
OnDragDrop	Maus	Ein gezogenes Objekt über der Komponente abgelegt wird
OnEnter	Fokus	Komponente den Fokus erhält
OnExit	Fokus	Komponente den Fokus verliert
OnKeyDown	Tastatur	Taste gedrückt wird
OnKeyPress	Tastatur	Taste gedrückt ist
OnKeyUp	Tastatur	Taste losgelassen wird
OnMouseDown	Maus	Eine Maustaste wird gedrückt
OnMouseMove	Maus	Maus bewegt wird
OnMouseUp	Maus	Maustaste losgelassen wird

Einige Ereignisse sind bei vielen Objekten identisch, es gibt aber auch spezielle.

Übung:

	<p>Erweitern Sie "MyProject"</p> <ul style="list-style-type: none"> o mit Objekt vom der Klasse TCheckBox, o mit dem Namen "CheckBox1". <p>Erzeugen Sie eine OnClick-Ereignis-Bedienfunktion² für Button1. In dieser sollte folgender Code stehen:</p> <pre>if (CheckBox1->Checked) {CheckBox1->Checked=False;} else {CheckBox1->Checked=True;}</pre>
---	---

🔧 Projekt starten und ausprobieren

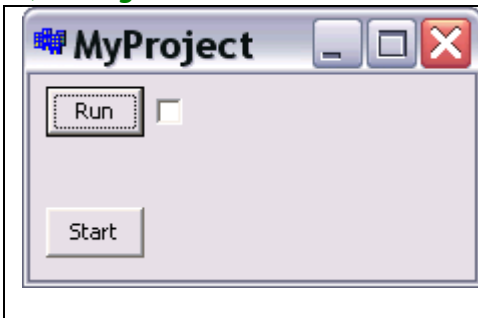
² 📖 Vorgehen beim Erzeugen einer Ereignis-Bedienfunktion:

- (1) In der IDE Button1 auswählen,
- (2) Im Objektinspektor → Ereignisse auswählen
- (3) Rechts neben OnClick 2x ins Fenster klicken,

Folgender Methodenrumpf wird erzeugt:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
}
```


☀ Übung:



Erzeugen sie ein weiteres TButton-Objekt mit folgenden Eigenschaften und Verhalten:

- Name: startstop,
- Caption: Start,
- Hint: Start-/Stop-Button,
- Funktionsweise: Wenn auf den Button geklickt wird, soll Caption von Start auf Stop und umgekehrt gehen.

🔧 Hinweis: Erzeugen sie eine OnClick-Ereignis-Bedienfunktion.

```
//-----  
void __fastcall TForm1::startstopClick(TObject *Sender)  
{  
    if (startstop->Caption=="Start") {startstop->Caption="Stop";}   
    else {startstop->Caption="Start";}   
}  
//-----
```

2.3 TEdit: Eigenschaften, Methoden, Ereignisse

TEdit ist ein einzeliges Eingabe-/Ausgabefeld für Strings. Man kann Größe, Aussehen, Position festlegen. Schriftart, Schriftfarbe usw. sind nur für das gesamte Objekt festlegbar.

Haupteigenschaften und Methoden:

```
Edit1->Text="irgendein text"; //auf Eigenschaft Text zugreifen  
Edit1->Clear(); //die Methode „Text löschen“ anwenden  
Edit1->SelectAll(); //Die Methode „gesamten Text“ markieren anwenden
```

☀ Übung:



Erzeugen sie in "MyProject" nachfolgende Objekte:

- quelle, senke : TEdit
- transfer : TButton
- Funktionsweise: der in quelle stehende Text soll nach senke transferiert werden, wenn auf dem Button "transfer" das Ereignis "onClick" ausgelöst wird. Das Objekt „quelle“ soll nach dem Transfer inhaltslos sein und den Focus besitzen.

🔧 Hinweis: Erzeugen sie eine OnClick-Ereignis-Bedienfunktion.

```
//-----  
void __fastcall TForm1::transferClick(TObject *Sender)  
{  
    senke->Text=quelle->Text; quelle->Clear();quelle->SetFocus();  
}  
//-----
```

☀ Übung: Der Transfer auch erfolgen, wenn im Objekt „quelle“ die **ENTER-Taste** gedrückt wird.

🔧 Hinweis: Die ENTER-Taste hat den ASCII-Codes (13)_d bzw. (0D)_h; Es soll das TEdit-Ereignis "OnKeyDown" genutzt werden.

```
//-----  
void __fastcall TForm1::quelleKeyDown(TObject *Sender, WORD &Key,  
    TShiftState Shift)  
{  
    if (Key==0x0D) transferClick(Form1);  
}  
//-----
```

→ Quelltext von MyProject (Unit1.cpp)

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
#include "Unit1.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    if (CheckBox1->Checked) {CheckBox1->Checked=False;}  
    else {CheckBox1->Checked=True;}  
}  
//-----  
void __fastcall TForm1::startstopClick(TObject *Sender)  
{  
    if (startstop->Caption=="Start") {startstop->Caption="Stop";}  
    else {startstop->Caption="Start";}  
}  
//-----  
void __fastcall TForm1::transferClick(TObject *Sender)  
{  
    senke->Text=quelle->Text; quelle->Clear();quelle->SetFocus();  
}  
//-----  
void __fastcall TForm1::quelleKeyDown(TObject *Sender, WORD &Key,  
    TShiftState Shift)  
{  
    if (Key==0x0D) transferClick(Form1);  
}  
//-----
```

3 Wichtige Datentypen in C++

Chartypen	Zeichenanzahl bis	Bytes	Anwendung
Char	1	1	ANSI-Zeichen
AnsiChar	1	1	ANSI-Zeichen,
WideChar	1	2	UNI-Code-Zeichen

Stringtypen	Zeichenanzahl bis	Bytes	Anwendung
ShortString	255	2..256	ANSI-Zeichen
AnsiString	$\sim 2^{31}$	4 Byte 2 GB	ANSI-Zeichen,
WideString	$\sim 2^{30}$	4 Byte 2 GB	UNI-Code-Zeichen

Integertypen	Bereich	Bytes
int	-128..127	8 Bit, mit Vorzeichen
Shortint	-128..127	8 Bit, mit Vorzeichen
Smallint	-32768..32767	16 Bit, mit Vorzeichen
Longint	-2147483648..2147483647	32 Bit, mit Vorzeichen
Cardinaltypen	Bereich	Bytes
Byte	0..255	8 Bit, ohne Vorzeichen
Word	0..65535	16 Bit, ohne Vorzeichen
LongWord	0..4294967295	32 Bit, ohne Vorzeichen

Reelle Typen	Bereich	Signifikante Stellen	Bytes
Single	$-1,5 \times 10^{45} \dots 3,4 \times 10^{38}$	7-8	4
Double	$-5,0 \times 10^{324} \dots 1,7 \times 10^{308}$	15-16	8
Extended	$-3,6 \times 10^{4951} \dots 1,1 \times 10^{4932}$	10-20	10
Comp	$-2^{63}+1 \dots 2^{63} - 1$	10-20	8
Currency	-922337203685477.5808.. 922337203685477.5807	10-20	8

Boolsche Typen	Wert	Bytes
Boolean	True False	1
ByteBool		1
WordBool		2
LongBool		4

☀ BEISPIELE:

```

AnsiChar varA='H'; // Variablen varA vom Typ AnsiChar mit Wert 'H'
ShortString varB="kurzer String"; // varB vom Typ AnsiString
AnsiString varC="langer String"; // varC vom Typ AnsiString
int varD=1023; // varD vom Typ Integer mit Dezimalwert 1023
int varD=0x3FF; // varD vom Typ Integer mit Dezimalwert 1023 (0x3FF)
Byte varE=127 // varE vom Typ Byte mit Dezimalwert 127
Byte varE=0x7F // varE vom Typ Byte mit Dezimalwert 127 (0x7F)
Word varF=65535 // varF vom Typ Word mit Dezimalwert 65535
Word varF=0xFFFF // varF vom Typ Word mit Dezimalwert 65535 (0xFFFF)
Double varG=3.14; // varG vom Typ Double mit dem Wert 3.14
Double varG=0.314E+1; // varG vom Typ Double mit dem Wert 3.14
Double varG=31.4E-1; // varG vom Typ Double mit dem Wert 3.14
Boolean varH=True; // varH vom Typ Boolean mit dem Wert True (1)
LongBool varI=False; // varI vom Typ Boolean mit dem Wert False (0)

```

4 Steuerstrukturen

4.1 if-else

Statement 1 wird ausgeführt, wenn `expression` den Wert `True` hat. Der `else`-Zeig ist optional.

Prinzip	Beispiel: Ausgabe von 0
<pre> if (expression) statement1 [else statement2] </pre>	<pre> int i=0; if (i<=3) {RichEdit1->Lines->Add((AnsiString)i);} else {RichEdit1->Lines->Add((AnsiString)i);} </pre>

4.2 for

Die Anweisungen im Schleifenkörper werden solange ausgeführt, bis `cond-expression` erfüllt ist → auch 0-mal!

Prinzip	Beispiel: Ausgabe von 0, 1, 2, 3
<pre> for (init-expression ; cond-expression ; loop-expression) { statement } </pre>	<pre> int i; for (i=0;i<=3;i++) { RichEdit1->Lines->Add((AnsiString)i); } </pre>

4.3 do-while (post-tested)

Die Anweisungen im Schleifenkörper (`statement`) werden solange ausgeführt, bis `cond-expression` erfüllt ist → mindestens aber einmal!

Prinzip	Beispiel: Ausgabe von 0, 1, 2, 3
<pre> do { statement } while (cond-expression) </pre>	<pre> int i = 0; do { RichEdit1->Lines->Add((AnsiString)i); i++; } while (i <= 3); </pre>

4.4 while (pre-tested)

Die Anweisungen im Schleifenkörper werden solange ausgeführt, bis `cond-expression` erfüllt ist → auch 0-mal!

Prinzip	Beispiel: Ausgabe von 0, 1, 2, 3
<pre> while (cond-expression) { statement } </pre>	<pre> int i = 0; while (i<=3) { RichEdit1->Lines->Add((AnsiString)i); i++; } </pre>

4.5 switch

Die Statements werden alle ausgeführt, bis erfüllt ist → auch 0-mal!

Prinzip	Beispiel: Ausgabe von 0
<pre> switch (expression) case constant-expression1: statemt1;break; case constant-expression2: stateme2;break; ... [default : statement] </pre>	<pre> int i=0; switch (i) { case 0: RichEdit1->Lines->Add((AnsiString)i);break; case 1: RichEdit1->Lines->Add((AnsiString)i);break; case 2: RichEdit1->Lines->Add((AnsiString)i);break; default: RichEdit1->Lines->Add((AnsiString)i);} </pre>