PROGRAMMIEREN MIT LAZARUS

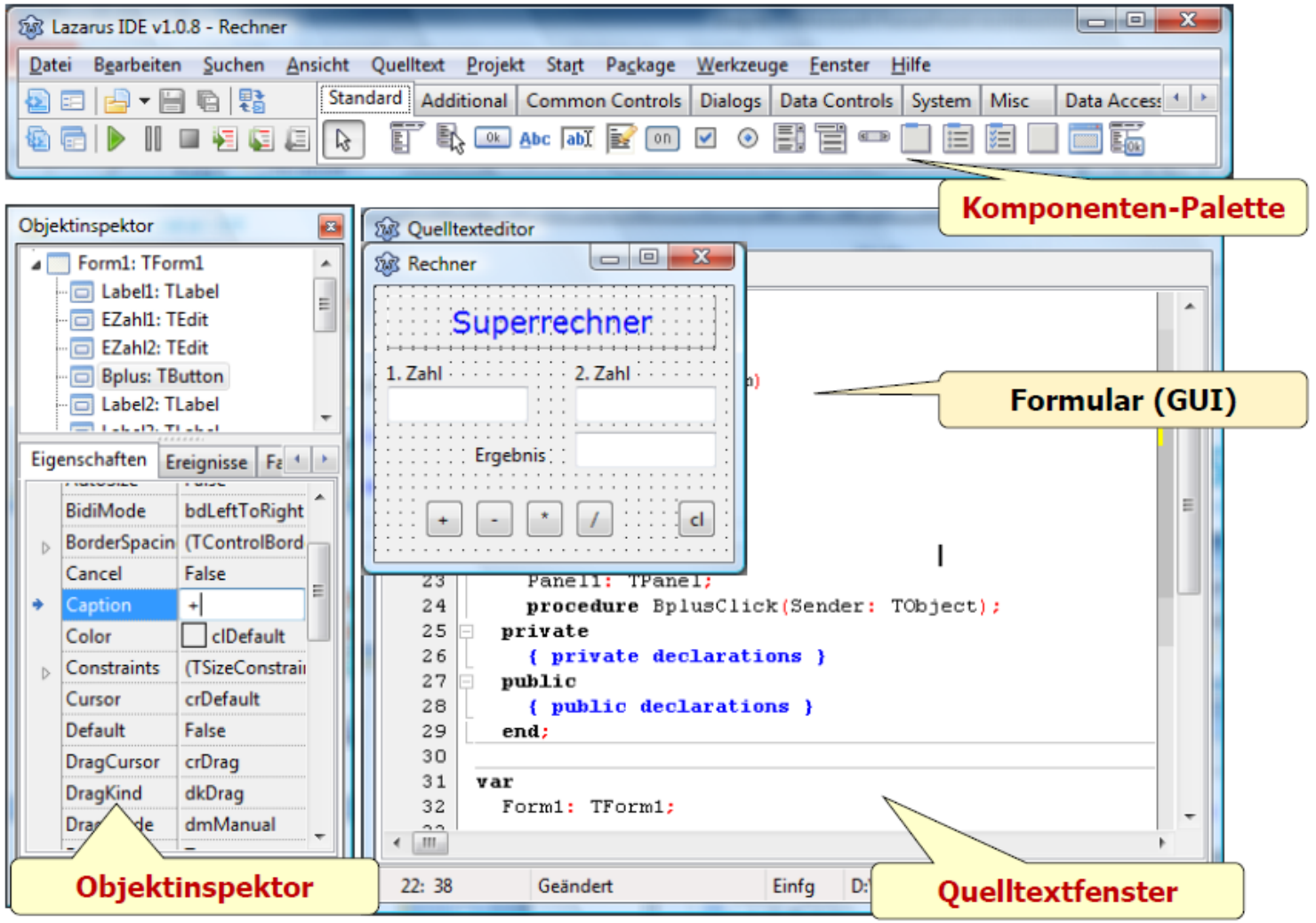
Einführung in die Lazarus-Entwicklungsumgebung

1

WAS IST LAZARUS?

- **Lazarus** ist eine plattformunabhängige freie **Entwicklungsumgebung** für die Programmiersprache **Objekt Pascal** - eine Bezeichnung für mehrere Programmiersprachen-Varianten, die Pascal um Objektorientierte Programmierung erweitern.
- baut auf dem Free Pascal Compiler (FPC) auf.
- Lazarus-Projekt: http://www.lazarus-ide.org/

ENTWICKLUNGSUMGEBUNG (IDE)



LAZARUS-PROJEKT

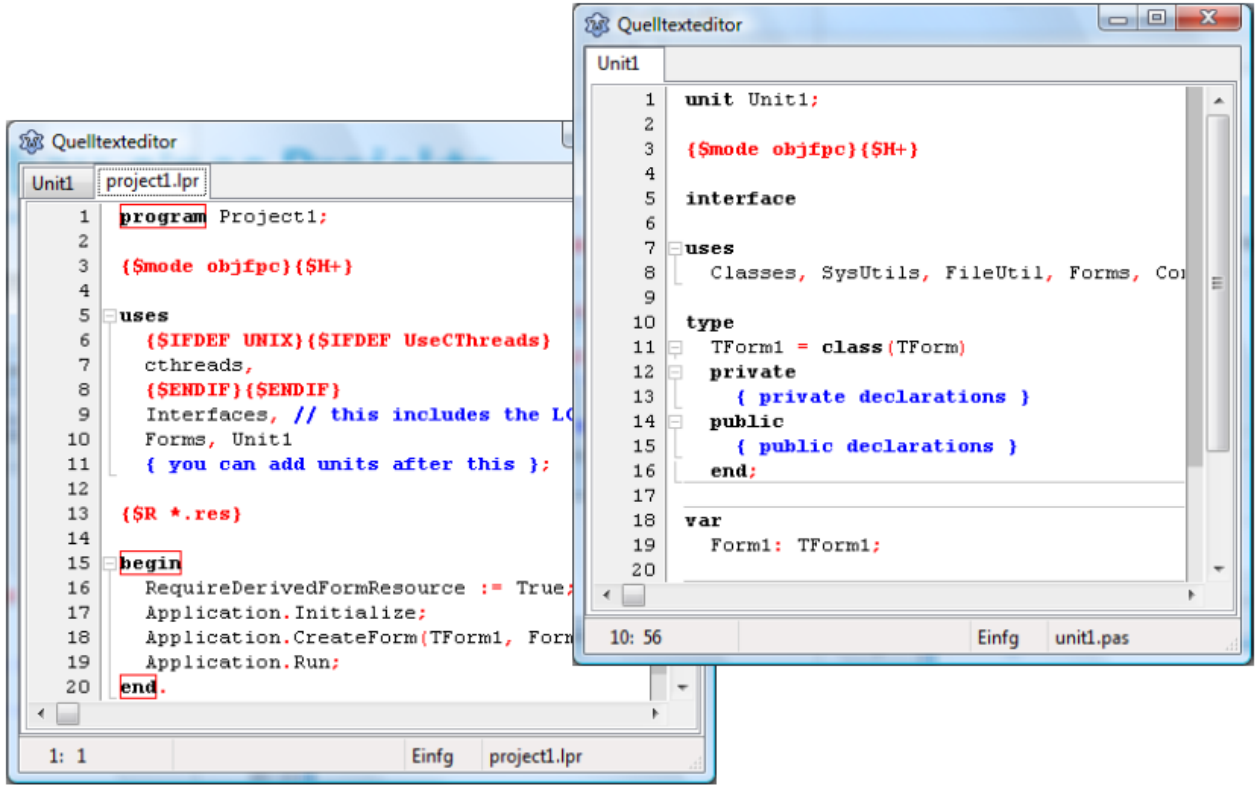
- Lazarus erzeugt eine ganze Reihe von Dateien zur **Verwaltung** eines Programmierprojektes. Diese Dateien sollte man **immer in einem eigenen Ordner speichern**. (Quellcodedateien werden als Units bezeichnet.)
- **Vorbereitende Schritte:**
  - 1. Ordner für das neue Projekt anlegen
  - 2. Neues Projekt anlegen: Datei → Neu → Anwendung
  - 3. Projektdateien sofort speichern:

    (1)Datei - Alles speichern

    (2)die vom System vorgeschlagen Namen unit1.pas und project1.lpr umbenennen (z. B.: kreisberechnung_u.pas und Rechner.lpr): u – für Unit (Quelltextdatei)
- Neben den oben genannten beiden Dateien werden automatisch im Projektordner viele weitere Dateien angelegt und laufend von Lazarus verwaltet und aktualisiert. Beim Kompilieren wird aus den Projektdateien ein **lauffähiges** Programm z.B. **Rechner.exe** erzeugt, das anschließend ausgeführt werden kann.

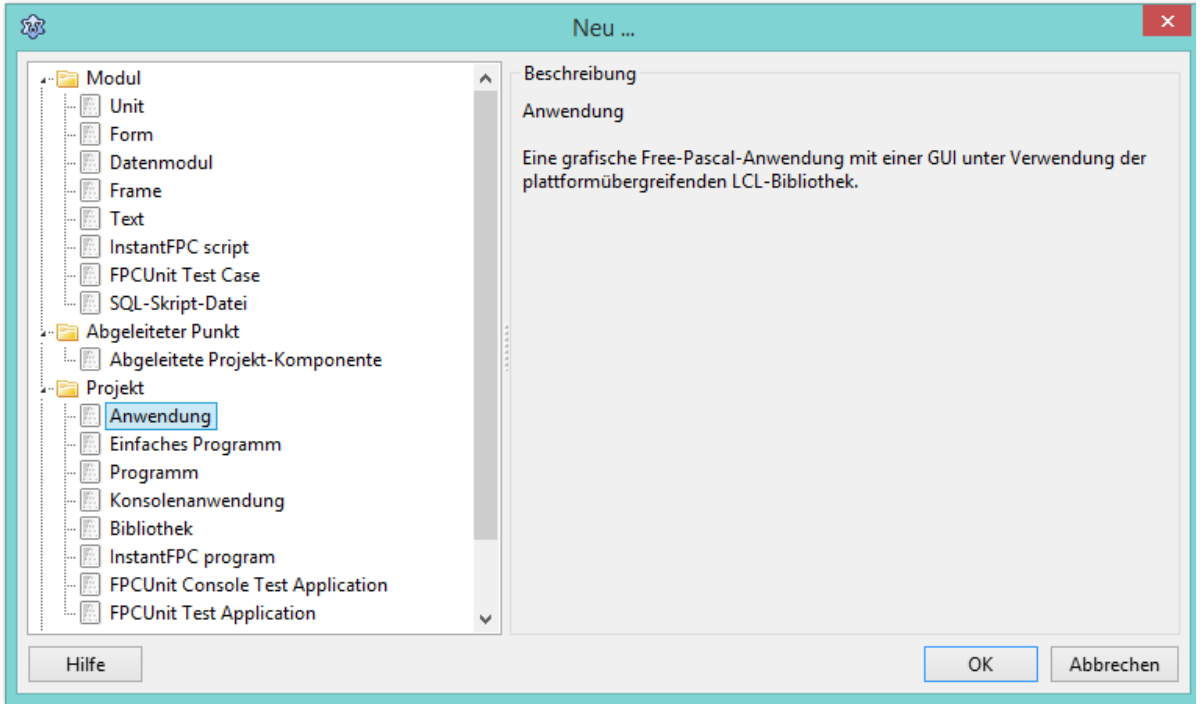
PROJEKTAUFBAU

- Ein Projekt besteht einzelnen Units, die einer Projektdatei verwaltet werden.



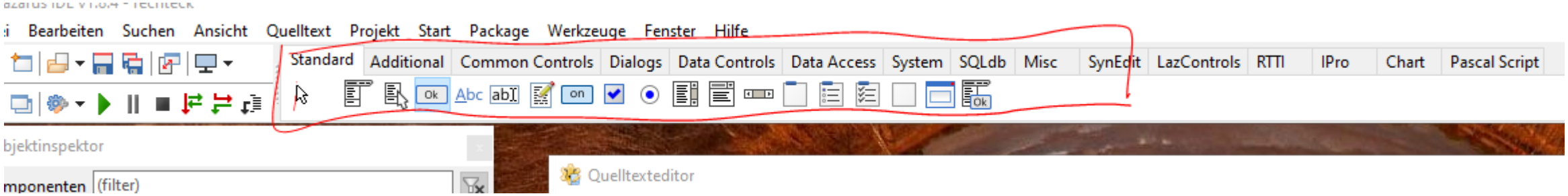
PROJEKT „KREIS“

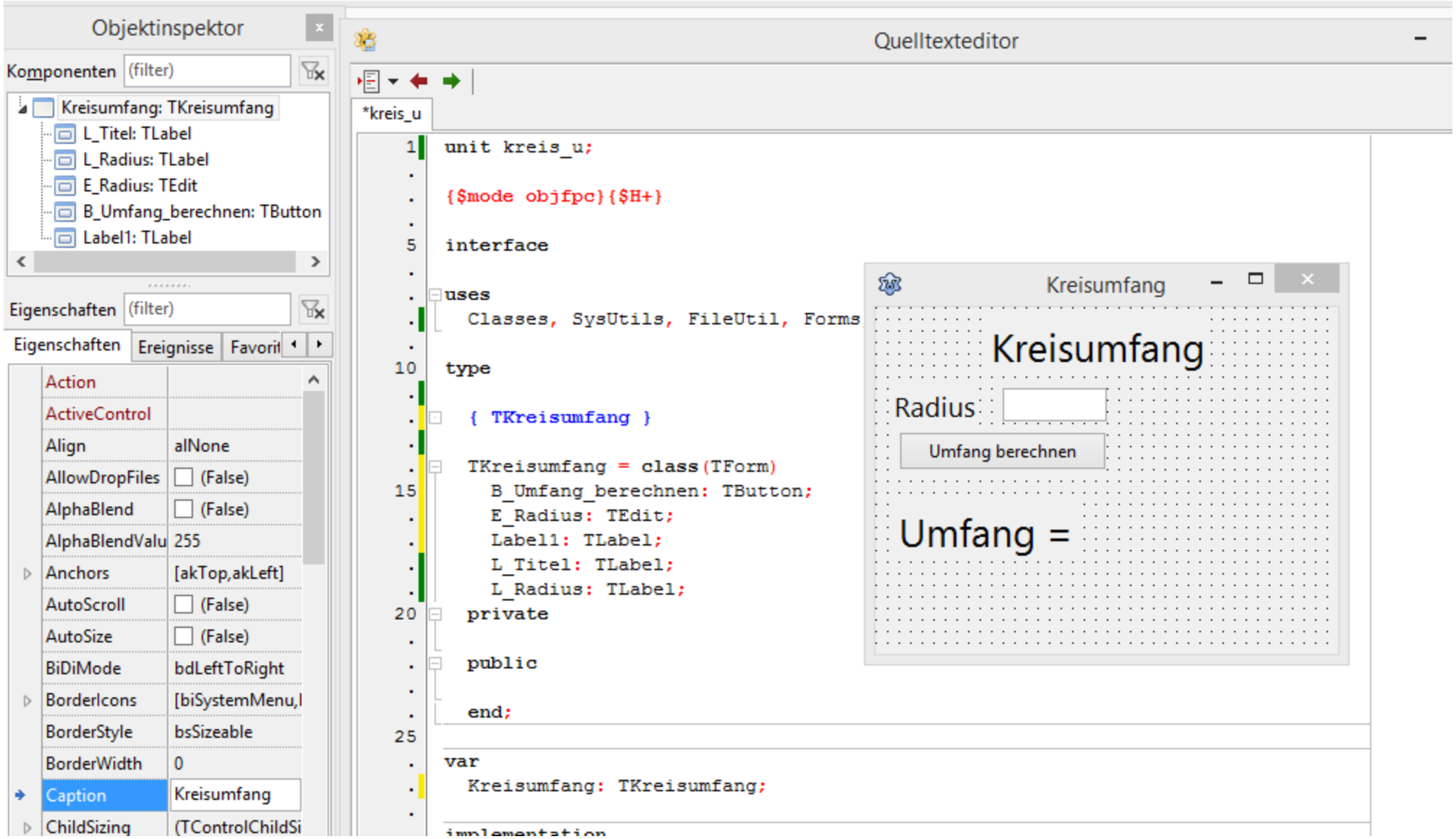
- Ordner „Kreis“ anlegen
- Lazarus starten – Datei | Neu | Anwendung



GUI ENTWICKELN

- GUI-Objekte (Komponenten der Komponentenpalette) werden auf dem **Formular** platziert und mit Hilfe des **Objektinspektors** (Register Eigenschaften) editiert.
- GUI-Objekte haben einen **Namen** und gehören ein **Klasse** an.




Objektinspektor
Komponenten
(filter)
Kreisumfang: TKreisumfang
L_Titel: TLabel
L_Radius: TLabel
E_Radius: TEdit
B_Umfang_berechnen: TButton
Label1: TLabel
Eigenschaften
(filter)
Eigenschaften
Ereignisse
Favorit
Action
ActiveControl
Align
alNone
AllowDropFiles
(False)
AlphaBlend
(False)
AlphaBlendValu
255
Anchors
[akTop,akLeft]
AutoScroll
(False)
AutoSize
(False)
BiDiMode
bdLeftToRight
BorderIcons
[biSystemMenu,
BorderStyle
bsSizeable
BorderWidth
0
Caption
Kreisumfang
ChildSizing
(TControlChildSi
Quelltexteditor
*kreis_u
unit kreis_u;
{$mode objfpc}{$H+}
interface
uses
Classes, SysUtils, FileUtil, Forms
type
{ TKreisumfang }
TKreisumfang = class(TForm)
B_Umfang_berechnen: TButton;
E_Radius: TEdit;
Label1: TLabel;
L_Titel: TLabel;
L_Radius: TLabel;
private
public
end;
var
Kreisumfang: TKreisumfang;
implementation
Kreisumfang
Kreisumfang
Radius
Umfang berechnen
Umfang =

KOMPONENTEN

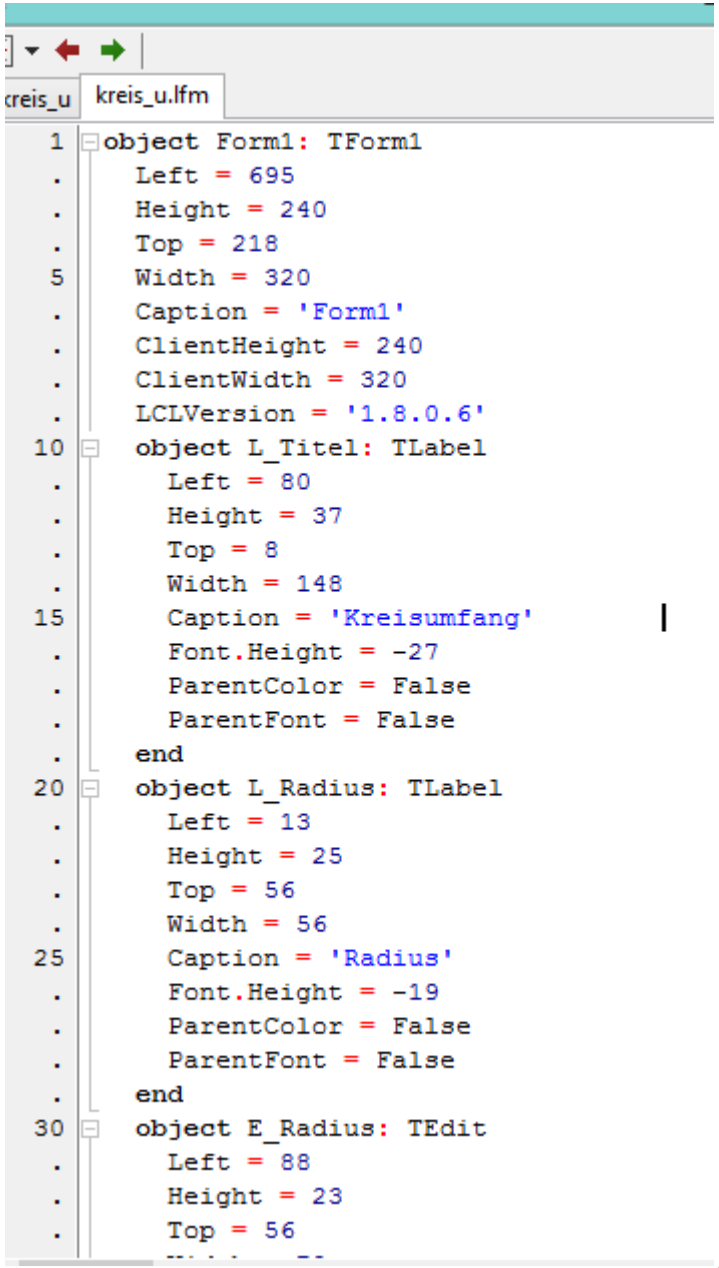
- Das Formular enthält Komponenten (GUI-Objekte) vom Typ:
  TForm, TEdit, TButton, TLabel
- Sinnvolle **Namenskonvention**:
  Der Editor gibt den GUI-Objekten automatisch Namen,
  z. B. TForm1, Label1, Button1, Button2
  Die Objekte sollten im Objektinspektor mit sinnvollen Namen (nach Konventionen) belegt werden;
  z. B.: je nach Typ des Objekts einen Präfix:
  B_Umfang (Button)
  E_Radius (Editkomponente)
  L_Titel (Label)

Standardattribute (Eigenschaften) der **GUI-Objekte**:

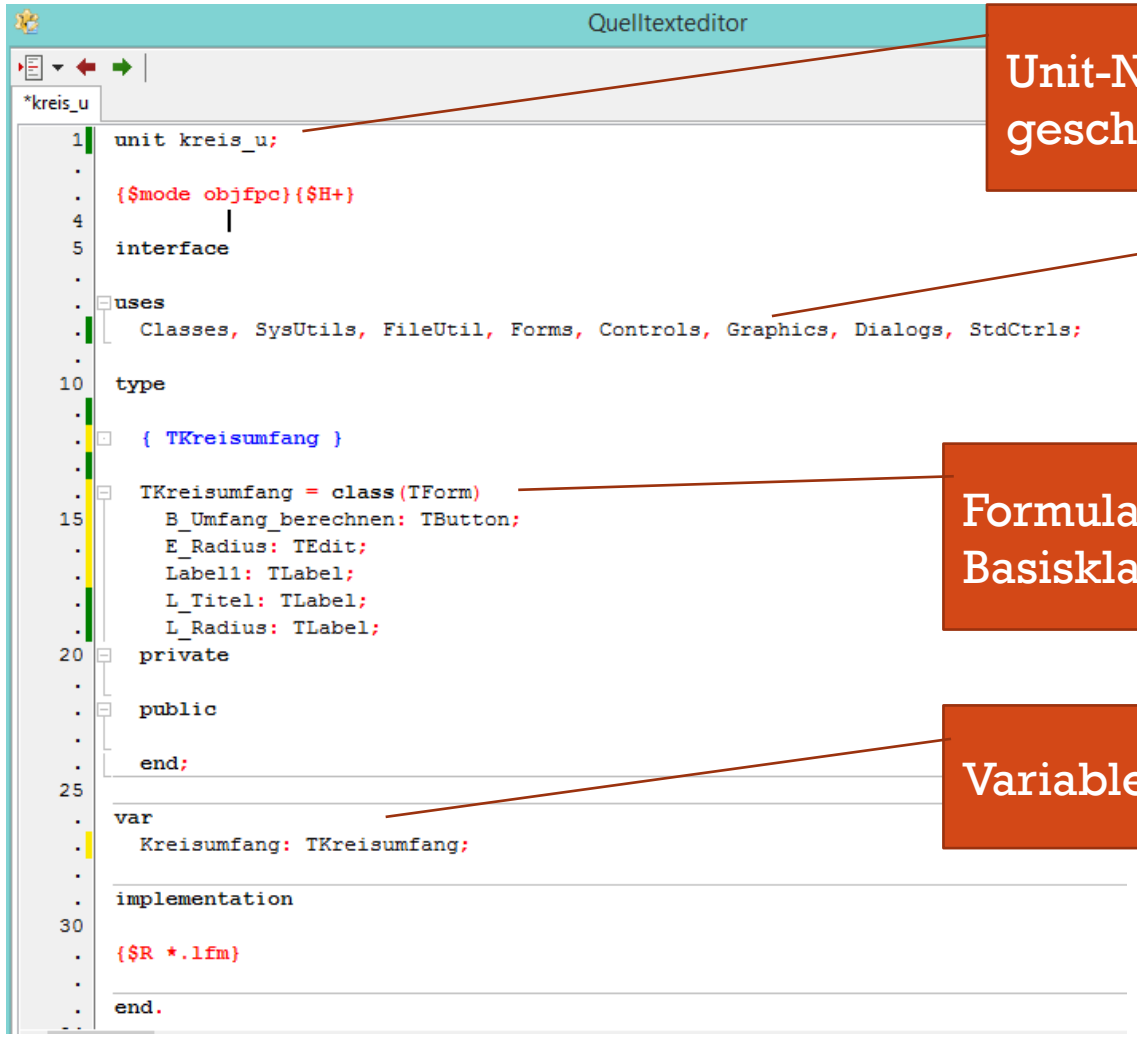
- **Caption**: Beschriftung einer Komponente
  (Aufschrift eines Buttons, Labels, Panels,…)
- **Text**: Inhalt eines Textfeldes (Edit)
- **Name**: Name, mit dem das Objekt im Programm angesprochen wird

FORMULARDATEI *.LFM

- Die zur **Entwurfszeit** definierten Attributwerte werden in der so genannten **Formulardatei** kreis_u.lfm gespeichert.
- Die Formulardatei wird **automatisch** erzeugt und gepflegt und hat immer den gleichen Namen wie die Quelltextdatei kreis_u.pas.
- Das **Formularobjekt Form1** verwaltet alle anderen Objekte als **Unterobjekte.**



QUELLTEXTINTERFACE



Unit-Name - **beim Speichern** vom System geschrieben

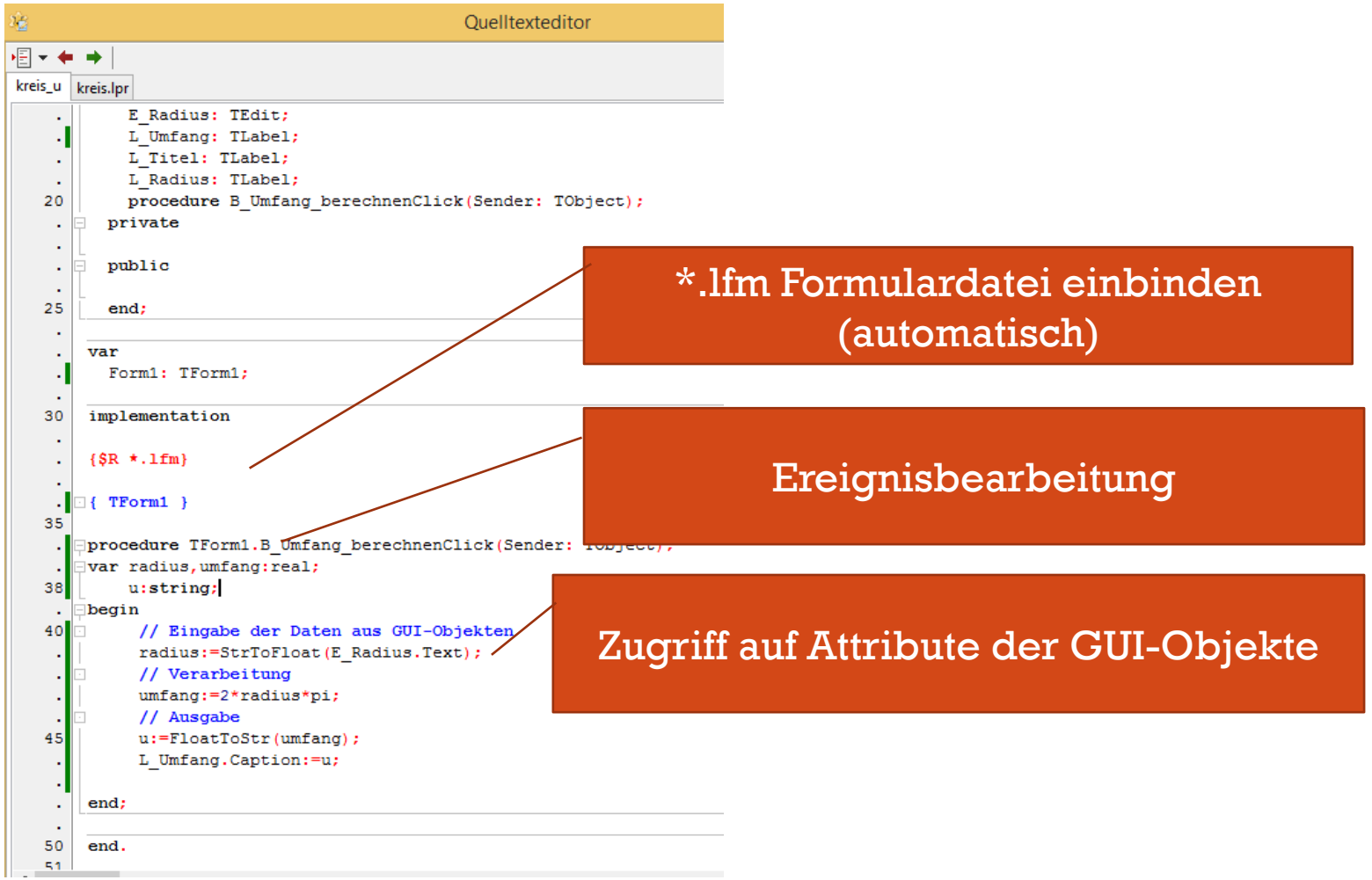
Einbinden von Systemunits (automatisch)

Formularklasse TForm1
Basisklasse: Tform (automatisch)

Variablendeklaration (Formularobjekt)

QUELLTEXTIMPLEMENTATION

- Ein Doppelklick auf den Button erzeugt eine **Ereignismethode (procedure)**



PROJEKTVERWALTUNG

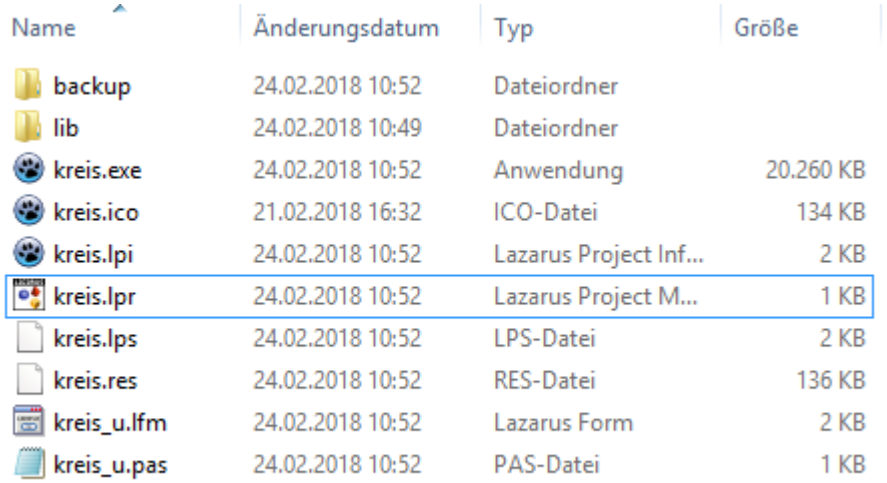
- Ein Lazarus-Projekt besteht aus mehreren Dateien, die im einfachsten Fall alle im gleichen "**Projekt-Ordne**r" liegen.
- Die Verbindung wird durch die Projekt-Datei *.lpr hergestellt.
- Einbindung von Units und Ressourcen
- Erzeugung des Fensterobjektes
- Start des Programms (Hauptprogramm)

- Diese Datei wird **automatisch** vom System gepflegt!

kreis_u | kreis.lpr

```
1  program kreis;
.
.  {$mode objfpc}{$H+}
.
5  uses
.    {$IFDEF UNIX}{$IFDEF UseCThreads}
.    cthreads,
.    {$ENDIF}{$ENDIF}
.    Interfaces, // this includes the LCL widgetset
10   Forms, kreis_u
.    { you can add units after this };
.
.  {$R *.res}
.
15 begin
.    RequireDerivedFormResource:=True;
.    Application.Initialize;
18   Application.CreateForm(TForm1, Form1);
.    Application.Run;
20 end.
21
```

ÜBERBLICK: DATEIEN DES PROJEKTS



Name	Änderungsdatum	Typ	Größe
backup	24.02.2018 10:52	Dateiordner	
lib	24.02.2018 10:49	Dateiordner	
kreis.exe	24.02.2018 10:52	Anwendung	20.260 KB
kreis.ico	21.02.2018 16:32	ICO-Datei	134 KB
kreis.lpi	24.02.2018 10:52	Lazarus Project Inf...	2 KB
kreis.lpr	24.02.2018 10:52	Lazarus Project M...	1 KB
kreis.lps	24.02.2018 10:52	LPS-Datei	2 KB
kreis.res	24.02.2018 10:52	RES-Datei	136 KB
kreis_u.lfm	24.02.2018 10:52	Lazarus Form	2 KB
kreis_u.pas	24.02.2018 10:52	PAS-Datei	1 KB

Datei	Inhalt
***.pas**	Pascal-Quellcode Datei (Unit)
***.lfm**	**L**azarus **Form**ulardatei – wird zu jeder Unit, in der ein Formularobjekt definiert ist, automatisch angelegt, enthält die Formulareinstellungen einschließlich aller Komponenten
***.lpr**	(**L**azarus **Pr**ojektdatei) - das Hauptmodul (Verwaltung) des Projektes
***.exe**	Das lauffähige Programm, zur Benutzung ist nur diese „EXE-Datei" erforderlich
***.lpi**	**L**azarus **P**roject **I**nformation - alle wichtigen Informationen des Lazarus-Projektes im XML-Format
***.lrs**	**Res**ourcendatei des **L**azarus-Projektes
***.ppu**	Compilierter Code der einer Unit, wird mit der Uses-Anweisung im Quellprogramm eingebunden

EREIGNISVERARBEITUNG

- Durch Doppelklick auf den Button B_Umfang_berechnen wird eine **Ereignismethode** erzeugt und dem Methodenzeiger **OnClick** des Buttonobjekts zugewiesen.
- Sie ist eine Methode des Formularobjekts und wird vom Buttonobjekt durch Click ausgelöst.

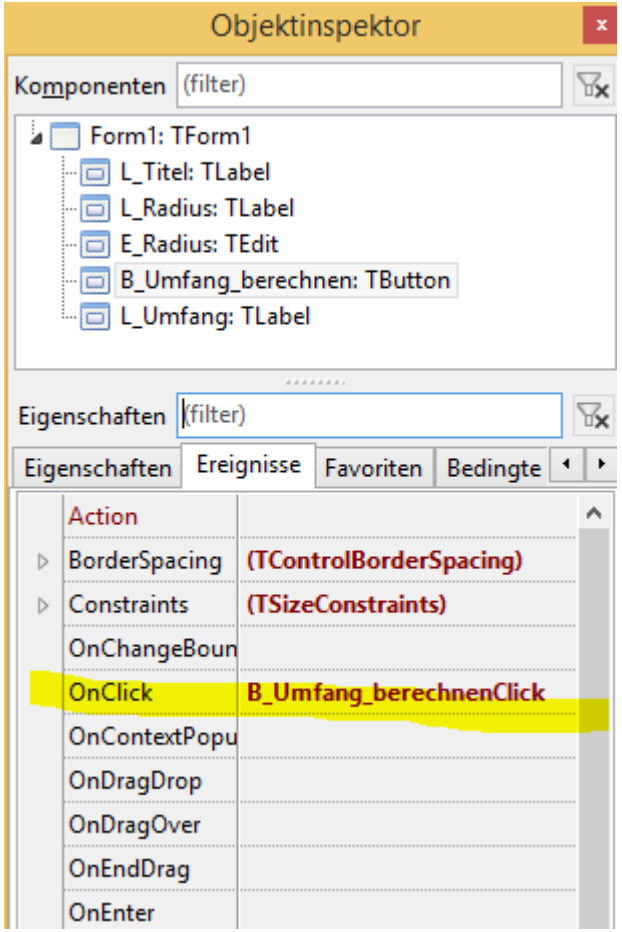
```
procedure TForm1.B_Umfang_berechnenClick(Sender: TObject);
```

- Der vom System gebildete Name enthält die Namen der auslösenden Komponente und des Methodenzeigers als Bestandteile.
- Der vorangestellte Klassenname TForm1 bindet die Prozedur an das Formularobjekt.

- Die Ereignisse des **Button-Objekts** sind Methodenzeiger, die mit Ereignismethoden verknüpft werden können.
- **Implementierung** der Ereignismethode:

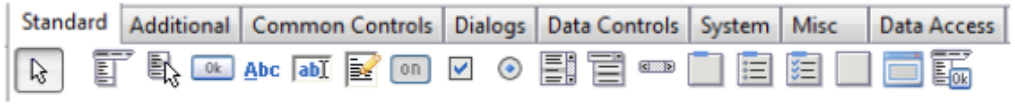
```
procedure TForm1.B_Umfang_berechnenCl
var radius,umfang:real;
    u:string;
begin
     // Eingabe der Daten aus GUI-Objekten
     radius:=StrToFloat(E_Radius.Text);
     // Verarbeitung
     umfang:=2*radius*pi;
     // Ausgabe
     u:=FloatToStr(umfang);
     L_Umfang.Caption:='Fläche = ' + u;

end;
```

Deklaration der lokalen Variablen

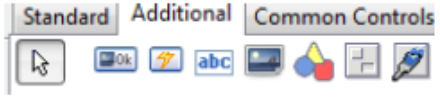


- Ein-/ Ausgabesystem mit **lokalen Variablen** als **temporären** Zwischenspeicher!

STANDARDKOMPONENTEN



Klasse	Beschreibung	Wichtige Eigenschaften/ Methoden
TButton	Schaltfläche	Caption, Color, Font, Enabled
FLabel	Anzeige von Text	Caption, Color, Font
Edit	Einzeiliges Eingabefeld	Text, Color, Font, MaxLength, Clear
TMemo	Mehrzeiliges Eingabefeld	Text, Lines, Append, Clear
TListBox	Auswahlliste mit mehreren Textzeilen ein	Items, Append, Clear
TGroupBox	Rahmen zur Anordnung von Objekten mit Gruppenbezeichnung.	Caption, Color, Font
TPanel	Rahmen zur Anordnung von Objekten	Caption, Color, Font

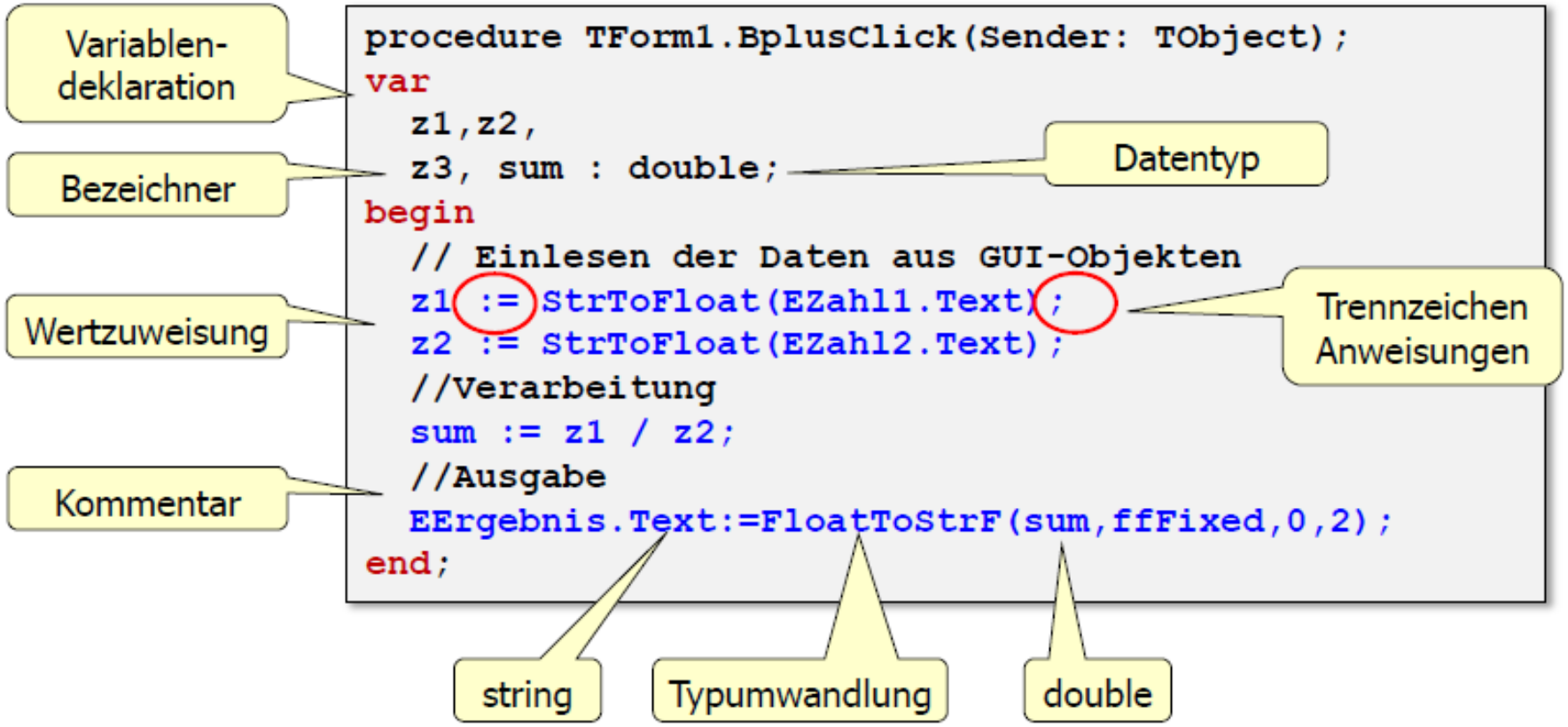


TImage	Anzeige von Grafiken	Picture, Canvas, loadfromfile

GRUNDELEMENTE OBJECT-PASCAL

Teil 2

19



Variablendeklaration

var Bezeichner : Datentyp;

Wertzuweisung

a := b ;

Die Variablen müssen zuweisungskompatibel sein!

GRUNDELEMENTE

- Konventionen für Bezeichner
  - Es dürfen nur folgende Zeichen verwendet werden:
    - Buchstaben (aber keine Umlaute)
    - Zahlen, diese aber erst nach mindestens einem Buchstaben
    - Unterstrich (auch am Anfang)
    - ansonsten keine Sonderzeichen wie „,“, „;“ …
- Blöcke: Anweisungsblöcke *begin … end*;
- Kommentare: //…. Bis Zeilenende { … } mehrzeilig
- Wertzuweisungen := *Beispiel: x := x + 1;*
- Vergleich: =

VARIABLE UND KONSTANTEN

- Variablen sind **Platzhalter** (Container) für **Daten.** Diese werden im Computer durch einen Speicherbereich repräsentiert und mit Hilfe eines **Namens** angesprochen.
- Variablen haben einen **Namen**, einen **Datentyp**, einen **Gültigkeitsbereich** und eine **Lebensdauer.**
- Bevor eine Variable benutzt werden kann, muss sie **deklariert** werden.
  Dabei werden der **Name** und der **Datentyp** festgelegt.
  Gleichzeitig wird ein Speicherbereich für den speziellen Datentyp reserviert.
- Bei **Wertzuweisungen** muss die **Zuweisungs-Kompatibilität** gewährleistet sein.

Variablendeklaration

Programmabschnitt: var

Syntax: **var** Bezeichner : Datentyp;

Beispiel: **var**
anzahl : integer;
z1, z1 : double;

Konstanten

Syntax: const Bezeichner= wert;

Beispiel: const pi = 3.14;

Konstanten haben einen unveränderlichen Wert.

ELEMENTARE DATENTYPEN

- **Einfache Datentypen**
  - integer : Ganze Zahlen {17, -12}
  - double : Fließkommazahlen {2.675. 1.5e-10}
  - boolean : Wahrheitswerte {true, false}
  - string : Zeichenkette {‘Hallo‘}
  - Char : ASCII-Zeichen {‘A‘, #65, chr(65)}
- **Arrays / Reihungen (statisch)**
  - var
    Bezeichner : array[StartInd..EndInd] of Datentyp;
  - So definierte Array ist statisch, die Anzahl der Elemente kann nicht mehr verändert werden.
  - Start- und Endindex vom Typ Integer oder Aufzählungstyp
  - Datentyp muss ein einfacher Typ sein!

TYPUMWANDLUNGEN

- Bei der **Ein-** und **Ausgabe** von Daten mit Hilfe von GUI-Objekten müssen häufig **Datentypen konvertiert** werden.

Funktion : Ergebnistyp	**Beispiel**
StrToInt(Para) : integer	groesse := StrToInt(EGroesse.Text);
IntToStr(Para) : string	PAusgabe.Caption := FloatToStr(BMI);
StrToFloat(Para) : real	
FloatToStr(Para) : string	
Formatierte Ausgabe:	**Siehe Kontexthilfe**
FloatToStrF() : string	PAusgabe.Caption := FloatToStrF(BMI, ffnumber, 8, 2);
Char – ASCII-Zeichen	
Chr(0..255) : Char	Ch := Chr(65);
Ord(ch) : integer	code := Ord('A');

OPERATOREN

- Operieren mit **Zeichenketten**
  - Zeichenkettentexte werden in einfachen Anführungszeichen geschrieben.
  - Beispiel: var Ausgabe : string; Ausgabe := ‘Du hast ‘;
  - Zeichenketten kann man mit ‚+‘ aneinander hängen.
  - Ausgabe := Ausgabe + ‘leider verloren‘;
  - Die Länge einer Zeichenkette bestimmt man mit der System-Funktion „length()“.
  - Zugriff auf die einzelnen Zeichen einer Zeichenkette über den Index
    Ausgabe[2] → ‘u‘

- Arithmetische Operatoren

integer / real	
+, -, *, /	Division (liefert einen Realtyp bei Anwendung auf Integer z. B. 12/4)
integer	
div	Ganzzahldivision Beispiel: 7 div 5 → 1
mod	Modulo Beispiel: 4 mod 3 → 1

- Logische Operatoren

AND	true : wenn beide Operanden wahr
OR	true : wenn einer oder beide Operanden wahr
NOT	Negation des Operanden (**NOT** a)

- Vergleichsoperatoren

=	gleich
<>	ungleich
<	Kleiner
<=	kleiner gleich
>	Größer
>=	größer gleich

FALLUNTERSCHEIDUNGEN

- If .. then .. Else

- Case .. of

Einseitig

```
If Ausdruck then
begin
   Anweisung;
end;
```

zweiseitig

```
If Bedingung then
begin
   Anweisung;
end
else
   begin
      Anweisung;
   end;
```

kein Semikolon vor **else**

Mehrfachverzweigung

```
case OrdWert of
 1 : ... ;
 2 : ... ;
 3 : ... ;
else ... ;
end;
```

SCHLEIFEN

- Anfangsbedingung
- Endbedingung
- Zählschleife
- Schleifen über arrays

while Schleife

```
while Bedingung do
begin
   Anweisung;
end;
```

Repeat – Schleife

```
repeat
   Anweisung;
until Bedingung;
```

For –Schleife

```
for i := 1 to n do
begin
   Anweisung;
end;
```

```
var
  augen : array[1..6] of integer;
for i := 1 to 6 do
  augen[i] := i;
```

PROZEDUREN UND FUNKTIONEN

Prozeduren

```
procedure Bezeichner(Parameter);
var lokale Variablen;
begin
   Anweisungen;
end;
```

Deklarationsteil

Anweisungsteil

Die Parameterliste kann auch fehlen

Funktionen

- Das Ergebnis muss einen einfachen Datentyp besitzen
- Das Funktionsergebnis wird der Funktionsvariablen **result** zugewiesen.

```
function Bezeichner(Parameter): Datentyp;
var lokale Variablen;
begin
   Anweisung;
   result := Ergebnis;
end;
```

Rückgabentyp

Rückgabewert

STRUKTUR EINER UNIT

```
unit name;
{$mode objfpc}{$H+}
interface
  uses ...
  Type ...
  var ...
implementation
  uses ...
  type ...
  var ...
  procedure …
end.
```

Kopf Schlüsselwort unit

Das System schriebt den Namen beim **Speichern**!

Interface (Schnittstelle)

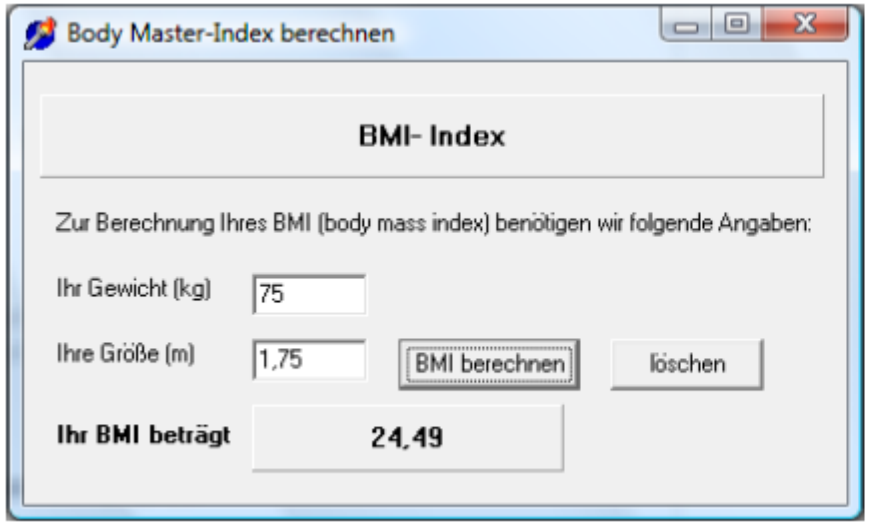
- Auf Alles, was hier deklariert ist, kann von **außen** (anderen Units) zugriffen werden.
- Typ – Vereinbarungen
- gobale Variablen und Konstanten

Implementation

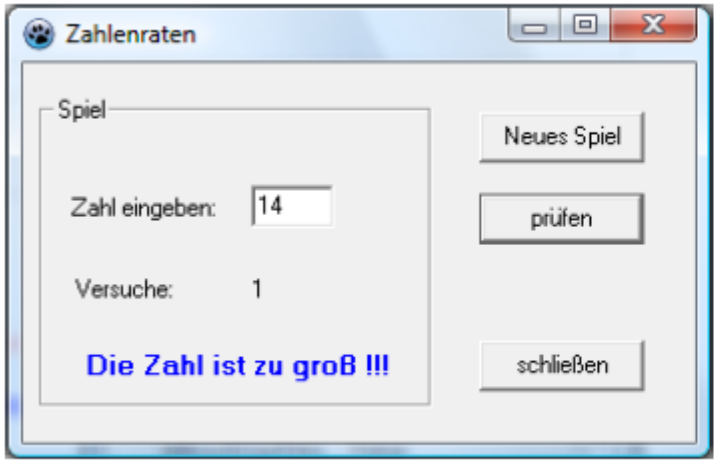
- **Lokale** Deklarationen (nur innerhalb der Unit)
- Implementierung von Prozeduren, Funktionen (Methoden)

PROJEKT BMI

- Mit einem Programm soll der BodyMass-Index BMI berechnet werden.
  - **Eingaben:** Gewicht in kg, Körpergröße in m
  - **Ausgabe:** BMI-Index
  - **Ablauf:** Ereignisgesteuert
- Schreibe ein Programm, das den Body-Mass-Index einer Person ausgibt.



$$BMI = \frac{Gewicht\ in\ Kg}{(Körpergröße\ in\ m)^2}$$



Datenmodell

Anzahl der Versuche: Versuche : integer

Zu erratende Zahl: Ratezahl : integer

Zufallszahlen

Funktion: random

z := random(N); ganzzahlige Zufallszahl z mit $0 \leq z < N$

r := random; Gleitkommazahl $0 \leq r < 1$

GUI UND DATENMODELL

Problem

- Speicherung der Daten in den GUI-Objekten ist **ineffektiv**!!!
- Besser ist eine **Trennung** von **GUI** und **Datenmodell**

Lösung (vorläufig)

- Realisierung eines "internen Datenmodell“ mit **globalen Variablen** als private Attribute des Formulars.
- Diese Variablen sind **innerhalb** des Formularobjekts **global**.

Internes Datenmodell

```
TGUI = class(TForm)
   {Steuerelemente}
 procedure FormCreate(Sender: TObject);
 procedure BtEingabeClick(...);
 private
   { Private-Deklarationen}
  RateZahl : integer;
  Versuche : integer;
 public
   { Public-Deklarationen}
end;
```

Spieldaten als private Attribute des Formulars

Lokale Variablen (temporäre Daten)

```
procedure TGUI.BpruefenClick(...);
var
 Zahl : integer;
 kommentar : string;
begin
  //Eingabe
  Zahl := StrToInt(ETippZahl.text);
  // Verarbeitung
  Versuche := Versuche + 1;
  if Zahl = Ratezahl then
  ...
end;
```

INITIALISIERUNG BEIM PROGRAMMSTART

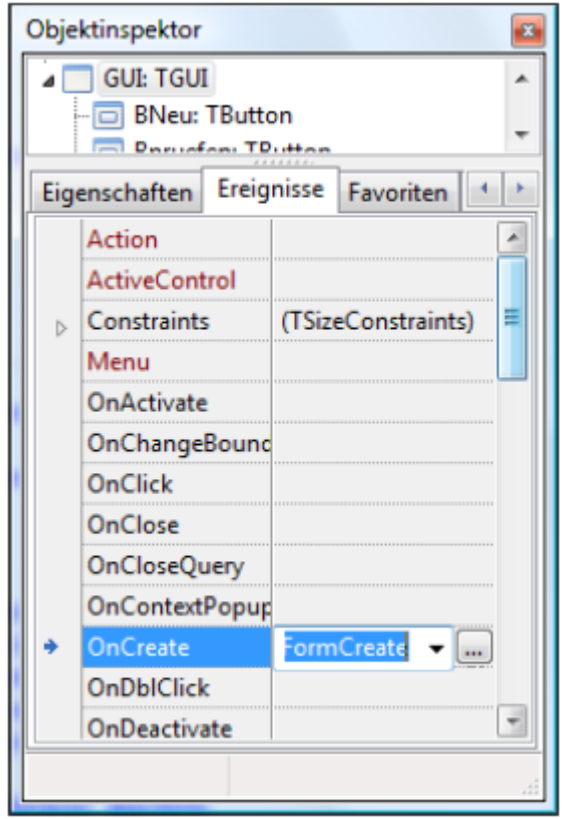
Ereignis **OnCreate** des Formulars

- Die zugehörige Ereignisprozedur **FormCreate** wird automatisch beim Start des Programms aufgerufen.

Anwendungsbeispiel:

- Initialisierung des Zufallszahlengenerators des Systems durch randomize. Dadurch wird bei jedem Programmstart eine andere Zufallszahlenfolge erzeugt.

```
procedure TGUI.FormCreate(Sender: TObject);
begin
  randomize;
end;
```



PROGRAMMIERAUFGABEN

1. ***Kreis***
   Input: Radius
   Output: Umfang und Fläche
2. ***Rechteck***
   Input: Länge und Breite
   Output: Umfang und Fläche
3. ***Quadratische Gleichung***
   Input: Koeffizienten a, b und c ($ax^2 + bx + c = 0$)
   Output: Lösungsfall und Lösung
4. ***1x1-Tainer***
   Programm gibt zwei Faktoren vor – User gibt das Produkt ein
   Output: Richtig oder Falsch
5. ***Toto-Tipp***
   Das Programm erstellt einen Zufalls-Tototipp
6. ***Zahlenrate-Spiel***
   Das Programm „denkt“ sich eine Zahl im Bereich von 1 bis 100.
   Der User soll die Zahl erraten