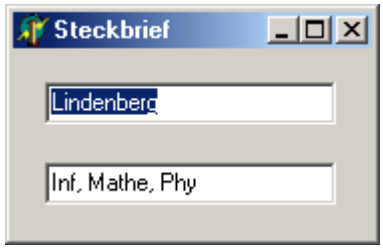
Mehrere Formulare



Um in einem Programm mit mehreren **Formblättern** zu arbeiten, erstellt man nur ein einziges Projekt. Die einzelnen Objekte auf den Formblättern werden wir teils dynamisch, teils mit Hilfe der Delphi-***IDE*** *(Integrated Development Environment = Programmierumgebung)* erzeugen. Da die Programmierung von Ereignismethoden (z.B. Mausklick) zunächst etwas schwierig ist, werden wir obiges **Hauptformular** komplett mit der Delphi-IDE erzeugen. Die Editfelder im Steckbriefformular werden hingegen schon dynamisch erzeugt. Zunächst befassen wir uns mit dem Steckbriefformular.

Gib als erstes dem neuen, leeren Formblatt den Namen *Steckbriefformular* (damit wird eine neue ***Klasse*** namens *TSteckbriefformular* erzeugt) und die Überschrift „Steckbrief“, und speichere **sofort** dessen zugehörige **Unit** (auch **Modu**l genannt) unter dem Namen *mSteckbrief.pas* . Speichere anschließend das (noch leere) **Projekt** unter dem Namen *pMultiFormular.dpr* .

```
unit mSteckbrief;
interface
uses …… StdCtrls; // sonst wäre TEdit unbekannt
TYPE TSteckbriefformular = class(TForm)
       private
        EdName, EdFach: TEdit;
       public
        constructor create(AOwner: TComponent); override;
        procedure SchreibName(pName: String);
        function LiesName: String;
        procedure SchreibFach(pFach: String);
        function LiesFach: String;
     end;
```

```
var  Steckbriefformular: TSteckbriefformular;
```
{ Diese Variablenzeile muss gelöscht werden, *weil die Variable für dieses Formblatt im später noch zu erstellenden Hauptprogramm deklariert wird. }*

……

```
implementation
{$R *.dfm}

constructor TSteckbriefformular.Create(AOwner: TComponent);
                        // AOwner ist der spätere Besitzer dieses Formulars
begin
 inherited create(AOwner);
 self.width := 200;
 self.height := 100;
 EdName := TEdit.Create(Self);
 With EdName DO BEGIN
  Parent := self;
  // Editfelder benötigen ein Formblatt, auf dem sie dargestellt werden.
  Left := 10;
  Top := 10;
  Width := self.Width - 30;
  Height := 15;
  ReadOnly := TRUE
 END;
 EdFach := TEdit.Create(Self);
 With EdFach DO BEGIN
  Parent := self;
  Left := 10;
  Top := 40;
  Width := self.Width - 30;
  Height := 15;
  ReadOnly := TRUE
 END
end;

procedure TSteckbriefformular.SchreibName(pName: String);
  BEGIN
    EdName.Text := pName
  END;

function TSteckbriefformular.LiesName: String;
  BEGIN
    Result := EdName.Text
  END;
```

```
procedure TSteckbriefformular.SchreibFach(pFach: String);
  BEGIN
    EdFach.Text := pFach
  END;

function TSteckbriefformular.LiesFach: String;
  BEGIN
    Result := EdFach.Text
  END;

end.
```

Erzeuge nun ein weiteres, neues Formblatt, nenne es *Main* , gib ihm die Überschrift *Haupt-Programm*, bringe mithilfe der *IDE* die drei Buttons an (es soll so aussehen wie in der Abbildung oben links), und speichere die zugehörige Unit unter dem Namen *mMain.pas* .

Damit man nun in diesem Hauptprogramm auch Steckbriefformulare benutzen kann, muß hier zusätzlich unter *uses* der Name der Unit *mSteckbrief* eingetragen werden. Außerdem wird in diesem Hauptmodul eine Variable namens *S1* vom Typ *TSteckbriefformular* deklariert.

```
unit mMain;
interface
uses  Windows, ……..mSteckbrief, StdCtrls;
type  TMain = class(TForm)
          ..............................
      end;

var   Main: TMain;
      S1 : TSteckbriefformular;
```

{ Da dieses Formblatt das Hauptprogramm beinhaltet, und somit von keinem anderen Modul aus aufgerufen wird, ist es eigentlich egal, ob z.B. die Variable S1 unter private, public oder hinter VAR deklariert wird. }

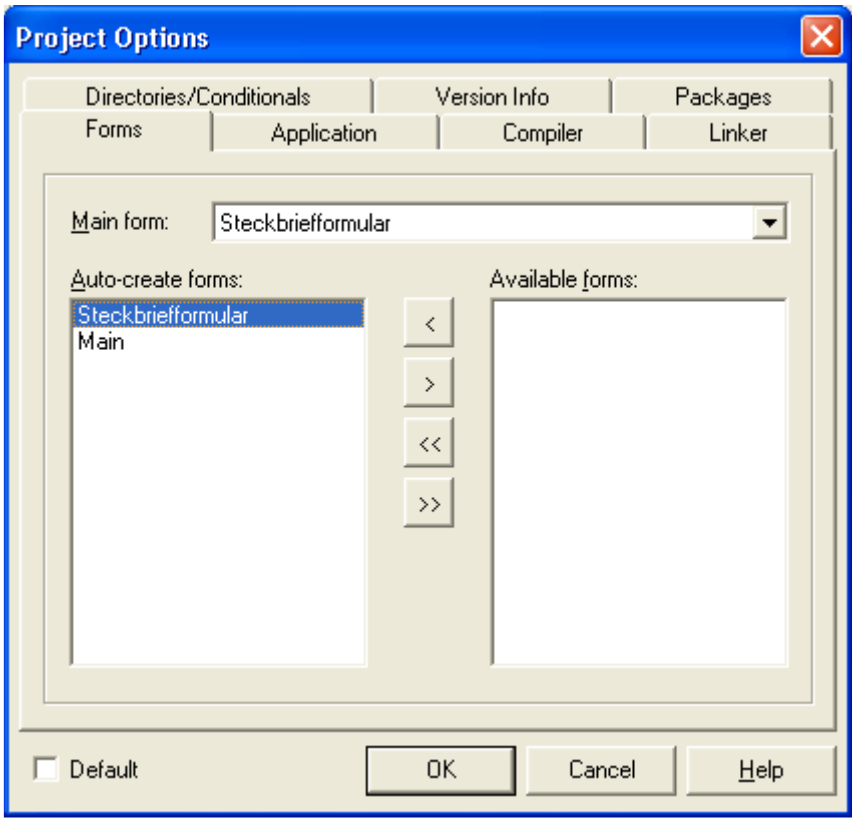
```
implementation
…….
procedure TMain.BtErzeugenClick(Sender: TObject);
begin
 S1:= TSteckbriefformular.Create(Main);
                    // als Parameter muss der sog. Owner übergeben werden.
 S1.Top := 200;
 S1.Left := 390;
 S1.SchreibName('Lindenberg');
 S1.SchreibFach('Inf, Mathe, Phy')end;
```

{ Alle Objekt-Variablennamen enthalten als Wert nicht den Inhalt des Objektes selbst, sondern nur die Anfangsnummer desjenigen Speicherbereiches, an dem der eigentliche Objektinhalt steht. Die Objektvariablen sind damit sog. Zeigervariablen. Falls ein Objekt noch gar nicht existiert, wird die Objektvariable in einen wohldefinierten Zustand versetzt, der besagt, dass der Zeiger auf keinen Inhalt zeigt. Die Variable zeigt ins Leere. Ihr Wert ist dann NIL (= Not In List) }

```
procedure TMain.BtLoeschClick(Sender: TObject);
begin
 IF S1 <> NIL THEN Begin
                     S1.Destroy;
                     S1 := NIL  // sehr wichtig, weil
                                // dies leider nicht
                                // automatisch
                                // geschieht
                   End
end;

procedure TMain.BtAnzeigenClick(Sender: TObject);
begin
 IF S1 <> NIL THEN S1.Show
end;

end.
```

Jedes Formblatt innerhalb eines Projektes (Programms) wird in einer eigenen Delphi-Unit (einem Modul) verwaltet.

Delphi stellt beim Start eines Programms grundsätzlich nur das Hauptformular auf dem Bildschirm dar. Da Delphi nicht von vornherein weiß, welches der Formulare das Hauptformular sein soll, müssen wir dies dem Delphi-System noch mitteilen. (Je nach der verwendeten Delphi-Version kann das Folgende etwas variieren). Dazu wählen wir im Menue *Projekte* die Möglichkeit *Optionen*.

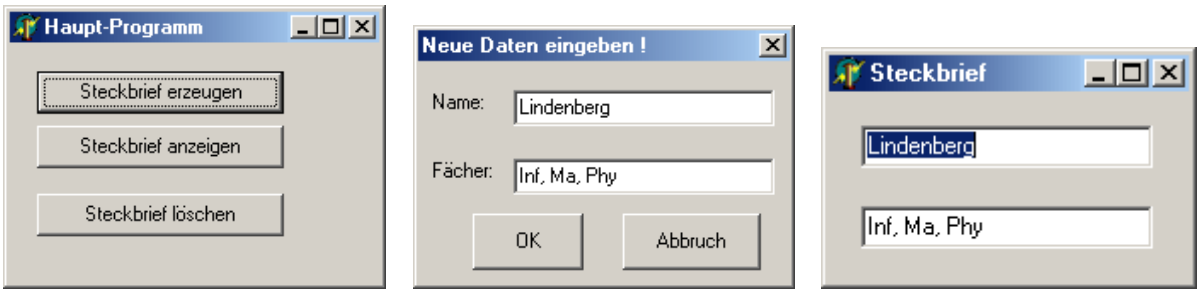


Hier markieren wir dasjenige Formular (hier mit dem Namen Steckbrief-formular), das nicht als Hauptformular fungieren soll und schieben es mit der entsprechenden Pfeiltaste in das rechte Fenster. Damit ist dieses Formular nur verfügbar, wird aber beim Programmstart nicht sofort erzeugt, sondern erst, wenn es im Hauptprogramm entsprechend gefordert wird.

Wenn man das Hauptformular schließt, werden automatisch alle Variablen und Objekte, die zum Hauptformular gehören, vorher gelöscht und aus dem Speicher entfernt. Das Steckbriefformular (Variable *S1*) gehört zum Hauptformular.

Der Aufruf der Methode *Close* (entspricht dem Anklicken des Kreuz-Buttons oben rechts im Fenster) schließt nur das Fenster, löscht aber das entsprechende Formular nicht aus dem Speicher!

Datenfluß zwischen den Formularen



Die Daten für den Steckbrief werden nun mithilfe eines Dialog-Formulars eingegeben. Nur das Hauptprogramm kann auf die beiden anderen Formulare zugreifen. Zu Beginn des Programms erscheint nur das Hauptformular. Nach Betätigung des Buttons „*Steckbrief erzeugen*“ erscheint nur das Dialogformular. Nach Betätigung des Buttons „*OK*“ verschwindet das Dialogformular. Nach Betätigung des Buttons „*Steckbrief anzeigen*“ erscheint das Steckbrieformular.

Im Dialogformular (es hat übrigens die Eigenschaft *BorderStyle = bsDialog*) werden wir nur die beiden Buttons mithilfe der Delphi-IDE erzeugen. Die Label und Editfelder werden dynamisch erzeugt (damit können sie als *private* deklariert werden!).

```
unit mDialog;.........
USES ……..……StdCtrls; // sonst wäre TEdit unbekannt
type
  TDialogFormular = class(TForm)
    BtOK: TButton;
    BtAbbruch: TButton;
    procedure BtOKClick(Sender: TObject);
    procedure BtAbbruchClick(Sender: TObject);
    private
      LabName, LabFach: TLabel;
      EdName,EdFach: TEdit;
    Public
      constructor create(AOwner: TComponent);
      function LiesName: String;
      function LiesFach: String;
  end;

procedure TDialog.BtOKClick(Sender: TObject);
begin
 IF (EdName.Text = '') OR (EdFach.Text = '') THEN
    Showmessage('Eingabe fehlt')
 ELSE close
end;
```

```
procedure TDialog.BtAbbruchClick(Sender: TObject);
begin
 EdName.Text := '';
 EdFach.Text := '';
 close
end;


constructor TDialogFormular.create(AOwner: TComponent);
begin
  inherited create(AOwner);
  LabName := TLabel.Create(self);
  With LabName DO BEGIN
    Parent := self;
    Top := 20;
    Left := 10;
    Height := 15;
    Caption := 'Name:'
  END;
  LabFach := TLabel.Create(self);
  With LabFach DO BEGIN
    Parent := self;
    Top := 60;
    Left := 10;
    Height := 15;
    Caption := 'Fächer:'
  END;
  EdName := TEdit.Create(self);
  With EdName DO BEGIN
    Parent := self;
    Top := 20;
    Left := 70;
    Height := 15;
    Text := ''
  END;
  EdFach := TEdit.Create(self);
  With EdFach DO BEGIN
    Parent := self;
    Top := 60;
    Left := 70;
    Height := 15;
    Text := ''
  END;
end;
```

```
function TDialogformular.LiesName: String;
BEGIN
  Result := EdName.Text
END;

function TDialogformular.LiesFach: String;
BEGIN
  Result := EdFach.Text
END;

end.
```

Achte wieder darauf, dass im Menue Projekte/Optionen das obige Formblatt nicht schon beim Programmstart automatisch erzeugt wird!

In diesem Programm kann das Hauptformular nicht **direkt** auf die Editfelder des Dialogformulars oder des Steckbriefformulars zugreifen, um z.B. den gewünschten Namen zu lesen oder zu schreiben.
Eine derartige Vorgehensweise wäre auch sehr schlecht. Wollte man nämlich z.B. das Dialogformular anders gestalten und den Namen etwa in einem Memofeld einlesen, so müsste man die entsprechenden Änderungen sogar auch noch im Hauptformular (und evtl. in weiteren Formularen) durchführen. Bei größeren Programmen wäre dies keineswegs mehr akzeptabel!

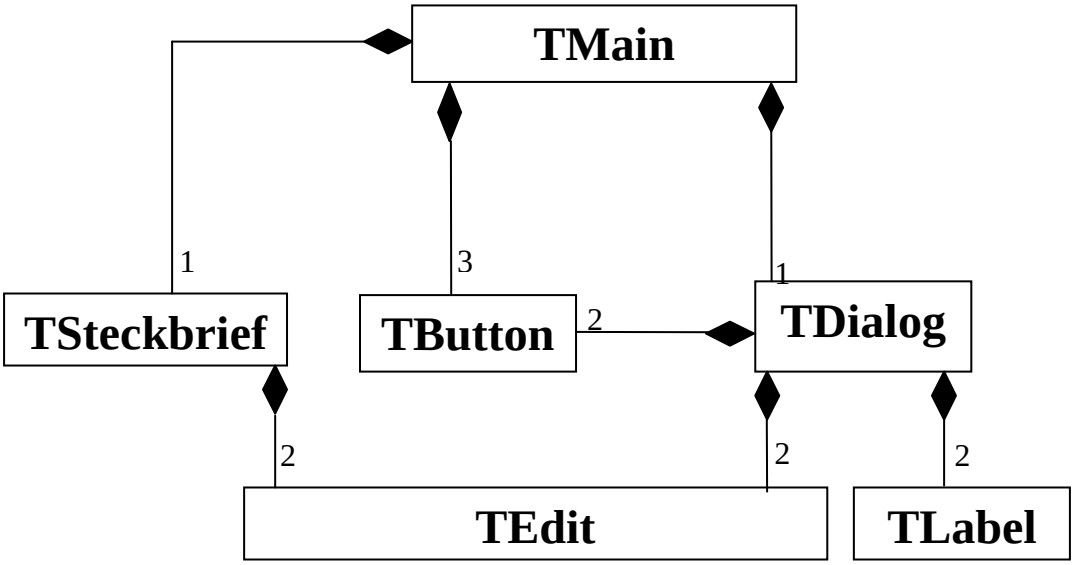
Aus diesem Grunde gilt in der Softwareentwicklung folgender Grundsatz:

Der Datenaustausch zwischen den einzelnen Modulen erfolgt ausschließlich durch geeignete Methoden.

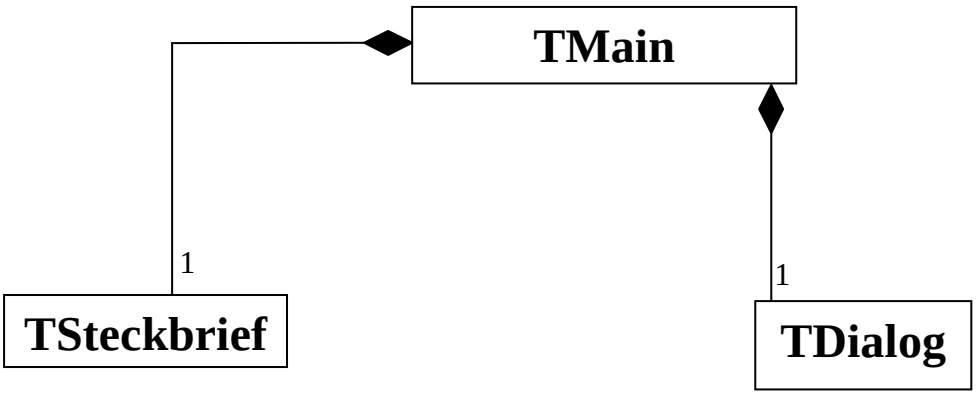
Das leicht veränderte Hauptformular sieht nun so aus:

```
unit mMain;
interface  ….uses  ....., mSteckbrief, mDialog;

type  TMain = class(TForm)
        BErzeugen: TButton;
        BLoesch: TButton;
        BAnzeigen: TButton;
        procedure BErzeugenClick(Sender: TObject);
        procedure BLoeschClick(Sender: TObject);
        procedure BAnzeigenClick(Sender: TObject);
      end;

var  Main: TMain;
     S1 : TSteckbriefformular;
     D1 : TDialogformular;

Implementation……..


procedure TMain.BErzeugenClick(Sender: TObject);
Var Name, Fach: STRING;
begin
 D1 := TDialogformular.Create(Main);
 D1.Top := 200;
 D1.Left := 450;
 D1.ShowModal; // Das restliche Programm muss warten
 Name := D1.LiesName;
 Fach := D1.LiesFach;
 IF Name <> '' THEN Begin
   S1:= TSteckbriefformular.Create(self);
   S1.Top := 200;
   S1.Left := 390;
   S1.SchreibName(Name);
   S1.SchreibFach(Fach);
 End; // of IF
 D1.Destroy;
 D1 := NIL
end;

……
end.
```

Im letzten Programm wurden verschiedene **Klassen** bzw. **Objekte** von verschiedenen **Klassen** eingesetzt. Unser Hauptprogramm besaß z.B. drei Buttons, genauer: drei **Objekte** der **Klasse** *TButton*. Außerdem besitzt bzw. hat unser Hauptprogramm ein **Objekt** der **Klasse** *TSteckbrief* und auch ein **Objekt** der **Klasse** *TDialog*. Das folgende ***Klassen-Beziehungsdiagramm*** zeigt die sog. ***Hat-Beziehung*** zwischen den Klassen (genauer: zwischen Objekten der verschiedenen Klassen):



Man kann sich leicht vorstellen, dass ein vollständiges Klassen-Beziehungs-diagramm sehr schnell unübersichtlich wird. Deshalb lässt man oft *"relativ unwichtige"* Informationen weg:



Für jede Klasse kann man noch ein eigenes sog. ***Klassen-Strukturdiagramm*** angeben, welches alle Eigenschaften und Methoden eines Objektes dieser Klasse angibt. Ein Objekt der Klasse *TButton* hat zum Beispiel die Eigenschaften *Left, Top, Visible, Width, Height* usw. und die Methoden *OnClick, OnDoubleClick, OnMouseOver* usw.

Üblicherweise gibt man in einem Klassen-Strukturdiagramm nur die wichtigsten Eigenschaften und Methoden an. Was „*wichtig*“ ist, hängt oft auch davon ab, für wen dieses Klassendiagramm zur Information bestimmt ist. Bloße Anwender von Objekten einer Klasse brauchen üblicherweise weniger Informationen als ein Programmierer, der eventuell die Möglichkeiten einer Klasse verändern will.

Ein mögliches Strukturdiagramm der Klasse *TSteckbriefformular* könnte folgendermaßen aussehen:

TSteckbriefformular
– EdName, EdFach: TEdit
+! Create(AOwner: TComponent) **+! SchreibName(pName: String)** **+! SchreibFach(pFach: String)** **+? LiesName** **+? LiesFach**

Dieses Strukturdiagramm entspricht eigentlich nur einer kurzen „Inhaltsangabe“ dessen, was ein Objekt der entsprechenden Klasse kann. Die Methodennamen sollten so gewählt sein, dass man sich einigermaßen vorstellen kann, was diese Methode bewirkt. Natürlich sind noch präzise Angaben erforderlich. Diese werden geliefert durch die sog. ***Dokumentation*** einer Klasse.

Eine mögliche Dokumentation der Klasse *TSteckbriefformular* könnte folgendermaßen aussehen:

Dokumentation der Klasse TSteckbriefformular

Objekte der Klasse *TSteckbriefformular* zeigen den Namen und die Fächer eines Lehrers an.

Konstruktor	**create(AOwner: TComponent)**
nachher	Ein Steckbrief ist angelegt, welcher zwei leere Edit-Felder enthält. Der Besitzer dieses Steckbriefes ist das Objekt AOwner, zu dessen Komponentenliste der Steckbrief nun gehört.
Auftrag	**SchreibName(pName: String)**
nachher	Der Text *pName* erscheint auf dem Steckbrief.
Auftrag	**SchreibFach(pFach: String)**
nachher	Der Text *pFach* erscheint auf dem Steckbrief.
Anfrage	**LiesName**
nachher	Die Anfrage liefert den Namen des Lehrers.
Anfrage	**LiesFach**
nachher	Die Anfrage liefert die Fächer des Lehrers.

Auch der Umfang einer Klassen-Dokumentation hängt davon ab, für wen sie bestimmt ist.
Ein Programmierer, der obige Klasse *TSteckbriefformular* nur benutzen will, braucht nicht mehr zu wissen als das, was oben angegeben wurde.
Ein Programmierer, der die obige Klasse *TSteckbriefformular* verbessern will (z.B. im äußeren Erscheinungsbild der Objekte), benötigt eine weitergehende Dokumentation (z.B. Größe, Breite und Hintergrundfarbe des Formulars; enthält es Edit-Felder oder Memo-Felder? usw.).

Wir werden uns in Zukunft damit begnügen, Klassen-Dokumentationen zu schreiben, die für Programmier-Benutzer bestimmt sind.

Verwalten von Komponenten

Es sollte bei den letzten Programmen aufgefallen sein, dass man zwar beliebig viele Steckbriefe erzeugen (und jeden einzelnen an beliebige Bildschirmpositionen verschieben kann), aber immer nur den letzten wieder löschen kann. Das liegt daran, dass alle Steckbriefe mit derselben globalen Variablen ***S1*** erzeugt werden. Diese Variable beinhaltet immer nur den letzten Steckbrief. Einen ersten Hinweis auf eine gute Lösung dieses Problems liefert der Konstruktor ***Create***, der als Parameter den übergeordneten Besitzer (das Hauptformular) angibt.

Jedes Formblatt besitzt eine Komponentenliste aller auf ihm befindlichen und zu ihm gehörenden Komponenten (jede Komponente, die weitere Unterkomponenten besitzen kann, besitzt eine solche Liste).

Die Anzahl der Komponenten in der Komponentenliste wird durch die Eigenschaft ***ComponentCount*** geliefert. Ist ***ComponentCount*** = n, so sind die einzelnen Komponenten von **0** bis **n-1** durchnummeriert. Die Reihenfolge der Einträge hängt von der Reihenfolge ab, in der die einzelnen Komponenten während der Programmierung im Formular erzeugt werden (Objekte vom Typ *TImage* werden allerdings auch nachträglich noch an den Anfang der Liste gestellt).

Jede Komponente, die weitere Unterkomponenten besitzen kann, besitzt ein ARRAY namens ***Components[0..n-1]: TComponent***, welches alle Unterkomponenten enthält. Auf z.B. die fünfte Unterkomponente lässt sich leider nur teilweise durch ***Components[4]*** zugreifen. Der Grund dafür liegt darin, dass der Inhalt ***Components[index: Integer]*** vom ganz allgemeinen Typ ***TComponent*** ist. Diese Oberklasse aller Komponenten besitzt nur wenige Eigenschaften (z.B. *Name, Enabled, Destroy; man kann noch unterscheiden zwischen dem Variablennamen S1 für ein Objekt und dem eigentlichen Namen dieses Objektes*). Falls man jedoch den wirklichen Typ von ***Components[4]*** kennt, so lässt sich eine sog. ***Typumwandlung*** durchführen: Durch Voranstellen des Typnamens kann man auch auf alle Eigenschaften von ***Components[4]***

zugreifen. Siehe dazu auch im Folgenden die Programmierung des Buttons 6 (Bildquelle ändern)!

Möchte man z.B. erfahren, welche Nummer ein bestimmter Button in der Komponentenliste seines ihn besitzenden Formblattes hat, so benutzt man das Attribut ***Button-Name.ComponentIndex: INTEGER***

Möchte man den Namen einer Komponente erfahren, deren Nummer in der Komponentenliste man kennt, so benutzt man das Attribut
Components[nummer: INTEGER].Name: String

Auch die ***function FindComponent(name: String): TComponent*** liefert leider nur den allgemeinen Typ ***TComponent*** als Ergebnis.

```
unit Haupt;
……
procedure TMain.Button1Click(Sender: TObject);
begin
 Edit1.Text := 'Es gibt '+IntToStr(ComponentCount)+
                 'Komponenten'
end;


procedure TMain.Button2Click(Sender: TObject);
begin
  Edit2.Text := IntToStr(Button2.ComponentIndex)
end;

procedure TMain.Button3Click(Sender: TObject);
begin
 Edit3.Text := TComponent(Sender).Name;
{Beachte hierbei die Typumwandlung!
 möglich wären auch folgende Alternativen:
      Edit3.Text := Components[2].Name;
      Dies setzt allerdings voraus, dass man die Nummer des Buttons in der
      Komponentenliste schon kennt.
      Edit3.Text := Button3.Name;
      Eigentlich unsinnig, da man den Namen des Buttons offensichtlich schon
      kennt
      Edit3.Text := Components[Button3.ComponentIndex].Name
 }
end;
```

```
procedure TMain.Button4Click(Sender: TObject);
VAR i: INTEGER;
begin
 i:=0;
 WHILE i <= ComponentCount - 1 DO BEGIN
   IF Components[i].Name = 'Image1' THEN Begin
     Components[i].Name := 'Bild1';
     Label1.Caption := 'Bild1:'
   End;
   i:=i+1
 END
end;



procedure TMain.Button5Click(Sender: TObject);
VAR K: TComponent;
begin
 K := FindComponent('Bild1');
 IF K = NIL THEN ShowMessage('nicht gefunden')
 ELSE Showmessage('gefunden')
end;



procedure TMain.Button6Click(Sender: TObject);
VAR i: INTEGER;
begin
 i:=0;
 WHILE i <= ComponentCount - 1 DO BEGIN
   IF Components[i] is TImage THEN
       TImage(Components[i]).picture.LoadFromFile('dog.ico');
   i:=i+1
 END
end;

end.
```

Aufgaben

1. Ergänze das Formular um eine Listbox und einen Button. In dieser Listbox sollen (nach Betätigung des Buttons) alle Komponenten des Formblattes mit Indexnummer und Namen aufgelistet werden.

2. Ergänze das Formular um einen Button. Bei Betätigung dieses Buttons sollen mit Hilfe einer FOR-Schleife die letzten drei Komponenten (außer diesem Button selbst!) gelöscht werden. Würde man den Button selbst mit löschen, so würde auch dessen zugehörige *procedure ButtonClick(Sender: TObject)* gelöscht und eben diese Prozedur, die gerade ausgeführt wird, könnte nicht mehr beendet werden. Das gesamte Programm würde abstürzen. Welches Problem ergibt sich bei der Benutzung der FOR-Schleife?

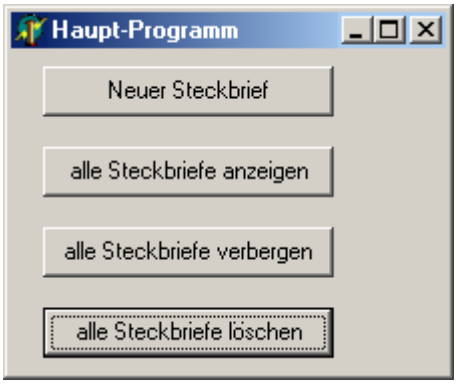
3. Untersuche, wie sich die Reihenfolge der Komponenten ändert, wenn man während des Programmablaufes eine Komponente löscht. Rücken alle folgenden Indexnummern um eine Stelle tiefer oder springt nur die letzte Komponente in die Lücke? Ergänze dafür das Formular um einen weiteren Button und ein Editfeld. In dieses Editfeld schreibt man die Index-Nummer der zu löschenden Komponente.

4. Ergänze das Formular um einen weiteren Lösch-Button und ein Editfeld. In dieses Editfeld schreibt man den Namen der zu löschenden Komponente.

5. Ergänze das Formular um einen weiteren Button. Bei Betätigung dieses Buttons wird in der schon vorhandenen Listbox ausgegeben, wie viele Buttons, Label, Bilder und Listboxen das Formular enthält.

6. Was geschieht, wenn man während des Programmablaufes das Formblatt selbst löscht?

Im Folgenden soll nun das Steckbriefprogramm so verbessert werden, dass alle Steckbriefe (zunächst nur gemeinsam, später auch einzeln) angezeigt, verborgen und gelöscht werden können.

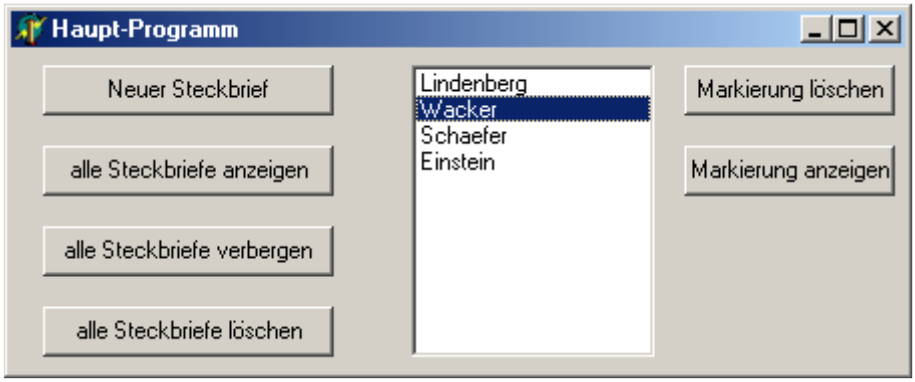



```
unit mMain;
..................
var  Main: TMain;
     S1 : TSteckbriefformular;
     D1 : TDialogformular;
..................
procedure TMain.BtAlleLoeschenClick(Sender: TObject);
VAR i:INTEGER;
begin
 i:= ComponentCount - 1;
 // Diese Schleife muss runter laufen, weil entstehende
 // Lücken vom Ende her wieder gefüllt werden.
 WHILE i > 0   DO BEGIN
   IF Components[i] is TSteckbriefformular THEN
        Components[i].Destroy;
   i := i-1
 END
end;

procedure TMain.BtAlleAnzeigenClick(Sender: TObject);
VAR i:INTEGER;
begin
 i:= 0;
 WHILE i < ComponentCount  DO BEGIN
   IF Components[i] is TSteckbriefformular THEN BEGIN
     TSteckbriefformular(Components[i]).Top := i*40;
     TSteckbriefformular(Components[i]).Left := 0;
     TSteckbriefformular(Components[i]).Show
   END;
   i := i+1
 END
end;

……..
```

Im Folgenden werden alle neuen Steckbriefformulare bei der Erzeugung mit einem Namen versehen, so dass man sie mit diesem Namen in der Komponentenliste des Hauptformulars finden und anschließend bearbeiten kann.

Hinweise: Namen für Objekte, Variablen, Komponenten usw. dürfen in Delphi keine Umlaute enthalten!
Es dürfen keine zwei Komponenten mit dem gleichen Namen erzeugt werden.

Das Hauptformular wird um eine Listbox ergänzt, welche die Namen aller Steckbriefe enthält. Den Steckbrief, dessen Name in der Listbox markiert ist, soll man anzeigen oder löschen können.




```
unit mMain;
..................
procedure TMain.BtNeuerSteckbriefClick.......
..................
   S1.SchreibName(Name);  //betrifft nur das Editfeld
   S1.Name := Name;       // = Name in Komponentenliste
   S1.Caption := Name;
   LBox.Items.Add(Name)
……..……
end;

procedure TMain.BtMarkAnzeigClick(Sender……);
VAR Name: STRING;
begin
 IF LBox.ItemIndex >= 0
    THEN Name:= LBox.Items[LBox.ItemIndex];
 S1 := FindComponent(Name) as TSteckbriefformular;
 IF S1 <> NIL THEN S1.Show
end;
```

```
procedure TMain.BtMarkLoeschClick(Sender……..);
VAR Name: STRING;
begin
 IF LBox.ItemIndex >= 0
    THEN Name := LBox.Items[LBox.ItemIndex];
 S1 := FindComponent(Name) as TSteckbriefformular;
    // weil S1 global vom Typ TSteckbriefformular ist
 IF S1 <> NIL THEN Begin
                    S1.Destroy;
                    LBox.DeleteSelected
                   End
end;
..................
end.
```

Aufgaben

1. Wenn man mehrere Steckbriefe erzeugt, ohne diesen Objekten einen Namen zu geben, so besorgt die Delphi-IDE dies automatisch. Überprüfe dies, indem du die (automatisch erzeugten) Namen der erzeugten Steckbriefe in der Listbox ausgibst!

2. Verdopple die Anzahl der Steckbriefe, indem du die Komponentenliste durchsuchst und, falls du einen Steckbrief findest, einen neuen Steckbrief erzeugst, welcher den Namen „Kopie_von_*Originalname*“ erhält. Allerdings enthält dieser neue Steckbrief noch keinen Inhalt. Wenn der neue Steckbrief auch denselben Inhalt haben soll wie der alte, so muss man an der Klasse *TSteckbriefformular* etwas ändern. Was?
Anschließend sollen alle Steckbriefnamen in der Listbox angezeigt werden.

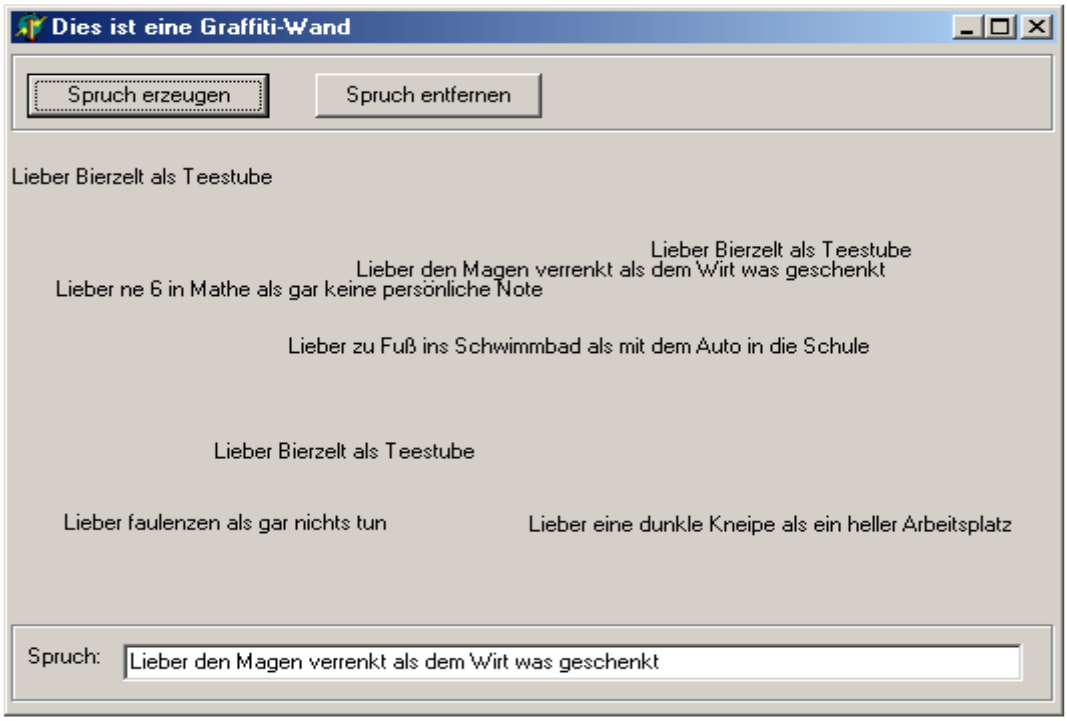
3. Erzeuge mit der Delphi-IDE 5 Objekte vom Typ *TImage* auf dem Hauptformblatt. Diese 5 Objekte heißen Bild1, Bild2, … Bild5. Sie sollen noch keinen Inhalt haben. Aufgrund ihres Namens kann man sie jedoch in der Komponentenliste finden und ihnen allen (in einer For-Schleife) dasselbe Bild zuordnen.
Erstelle einen zusätzlichen Button, der dies bewirkt!

Lösungen

Aufgabe 2

```
procedure TMain.BtSteckbriefeDoppelnClick(Sender:TObject);
VAR S: TSteckbriefformular;
    i: INTEGER;
begin
 i:= ComponentCount - 1;
 WHILE i >=0 DO BEGIN
   IF Components[i] is TSteckbriefformular THEN BEGIN
     S := TSteckbriefformular.Create(self);
     S.Name:='Kopie_von_' +
             TSteckbriefformular(Components[i]).Name;
   END;
   i := i-1
 END;
 LBox.Clear;
 i:= 0;
 While i <= ComponentCount -1  DO BEGIN
   IF Components[i] is TSteckbriefformular THEN
     LBox.Items.add(Components[i].Name);
   i := i+1
 END;
end;
```

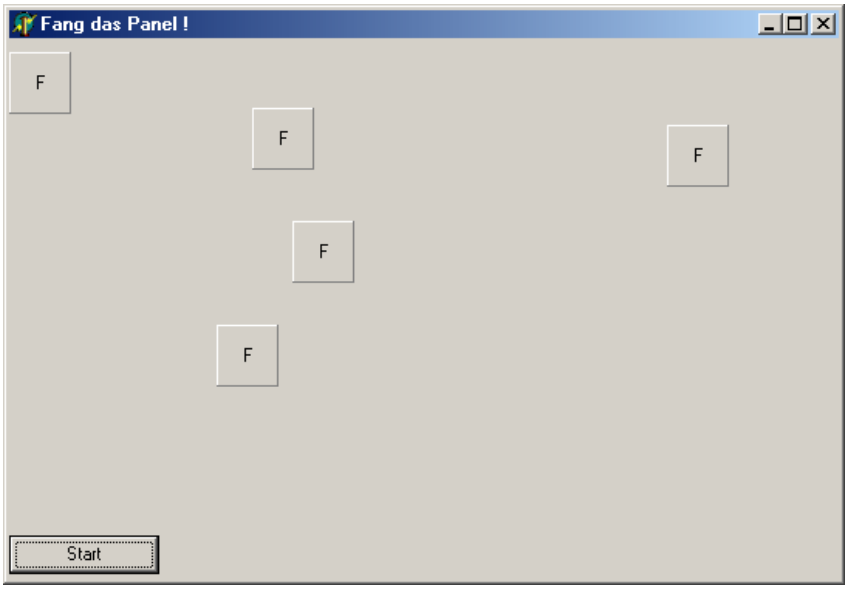
Das folgende Programm erzeugt eine Graffiti-Wand, auf der an zufälligen Stellen Sprüche aufgeschrieben werden.




```
procedure TMain.BtErzeugenClick(Sender: TObject);
VAR LbNeu :TLabel;
begin
 LbNeu := TLabel.Create(HauptForm);
 LbNeu.AutoSize := True;
 LbNeu.Caption := ESpruch.Text;
 LbNeu.Left := Random(Round(0.7*HauptForm.Width));
 LbNeu.Top := 60 + Random(Round(0.6*HauptForm.Height));
 LbNeu.Parent := HauptForm
end;
```

```
procedure TMain.BtEntfernenClick(Sender: TObject);
VAR i: INTEGER;
    K: TComponent;
begin
 FOR i := ComponentCount - 1 DownTo 0 DO BEGIN
   K := Components[i];
   IF K is TLabel THEN
     IF TLabel(K).Caption = ESpruch.Text THEN K.Destroy
 END
end;
```

Aufgabe: Erstelle einen dritten Button, der <u>alle</u> Sprüche entfernt!




```
unit mMain;
USES ......ExtCtrls
{ Das OnClick-Ereignis des Typs TPanel wird erst dann erkannt, wenn (unter
  Windows) ExtCtrls benutzt wird. Tip: Lade ein Panel auf das Formblatt,
  compiliere das Programm und lösche anschließend das Panel wieder.  }

type
  TMain = class(TForm)
……..
var  Main: TMain;
     Panel: TPanel;
     PanelAnzahl: INTEGER;
     AFenster: TAnzeigeForm;
……..


procedure TMain.NeuPositionieren;
VAR i: INTEGER;
begin
 FOR i := 0 TO ComponentCount - 1 DO
   IF Components[i] is TPanel THEN BEGIN
      TPanel(Components[i]).Left :=
                          Random(Main.Width - 50);
      TPanel(Components[i]).Top :=
                         Random(Main.Height - 100)
   END
end;
```

```
procedure TMain.Spielauswertung;
begin
 IF PanelAnzahl = 0 THEN BEGIN
   Timer1.Enabled := FALSE;
   Showmessage('Du hast gewonnen !');
   Main.Close
 END ELSE  IF PanelAnzahl = 10 THEN BEGIN
              Timer1.Enabled := FALSE;
              Showmessage('Du hast verloren !');
              Main.Close
           END
end;
```

```
procedure TMain.PanelWeg(Sender: TObject);
{ Diese Methode wird bei einem OnClick-Ereignis aufgerufen. Deshalb muß sie
  denselben Parameter haben wie jede OnClick-Methode, nämlich die Angabe
  desjenigen Objektes, auf welches geklickt wurde.  }
begin
  TPanel(Sender).Hide;
   { Die Typumwandlung von TObject nach TPanel ist notwendig, weil die
     Oberklasse TObject die Methode Hide nicht kennt.    }
  PanelAnzahl := PanelAnzahl - 1;
  AFenster.SetPunkte(PanelAnzahl);
  Spielauswertung;
  NeuPositionieren
end;
```

```
procedure TMain.LoeschVersteckтePanels;
VAR i: INTEGER;
    K: TComponent;
begin
 FOR i := ComponentCount - 1 DOWNTO 0 DO BEGIN
   K := Components[i];
   IF K is TPanel THEN
      IF TPanel(K).Visible = False THEN K.Destroy
 END
end;
```

```
procedure TMain.PanelErzeugen;
BEGIN
   Panel := TPanel.Create(Main);
   Panel.Caption := 'F';
   Panel.Height := 40;
   Panel.Width := 40;
   Panel.Left := Random(Main.Width - 50);
   Panel.Top := Random(Main.Height - 100);
   Panel.Parent := Main;
   Panel.OnClick := PanelWeg;
   { Bei dieser Zuweisung genügt der Name der zugewiesenen Methode. Dass
     die Übergabeparameter (Sender: TObject) passen müssen, wird
     vorausgesetzt.   }
   PanelAnzahl := PanelAnzahl + 1
 END;


procedure TMain.FormCreate(Sender: TObject);
VAR i: INTEGER;
begin
  PanelAnzahl := 0;
  FOR i := 1 TO 5 DO PanelErzeugen;
  AFenster := TAnzeigeForm.Create(Main);
  AFenster.SetPunkte(PanelAnzahl);
  AFenster.Show
end;


procedure TMain.Timer1Timer(Sender: TObject);
begin
  LoeschVersteckтePanels;
  PanelErzeugen;
  AFenster.SetPunkte(PanelAnzahl);
  Spielauswertung;
  NeuPositionieren
end;


procedure TMain.BStartClick(Sender: TObject);
begin
  Timer1.Enabled := TRUE;
  BStart.Visible := FALSE
end;
end.
```