3 Computergrafik

Im folgenden Kapitel wollen wir einige wesentliche Sprachelemente von *Object-Pascal* einführen. Wir wählen dafür Grafikprogramme, weil sich damit leicht recht hübsche Effekte erzielen lassen und – vor allem –, weil viele Fehler, die man fast immer bei der Programmentwicklung macht, direkt am Bildschirm sichtbar werden. Eingeführt werden

- *Pascal-Namen* und deren Gültigkeitsbereiche.
- *Delphi-Konstante* und *Variable*
- einfache *Unterprogramme*
- *Fallunterscheidungen* und *Schleifen*

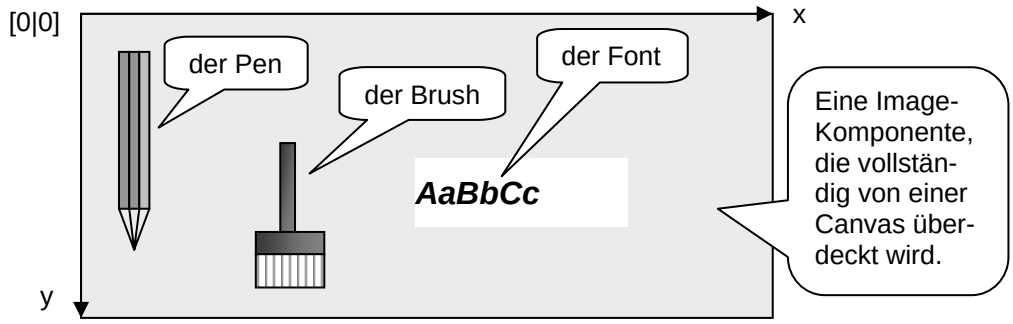
3.1 Eine Leinwand zum Zeichnen – die Canvas

Einige Delphi-Komponenten enthalten eine Leinwand – die *Canvas* –, also eine rechteckige Fläche, auf der gezeichnet werden kann. Zu diesen Komponenten gehören die *Forms* (die späteren Windows-Fenster), der Drucker und die *Paintbox*. Am einfachsten zu benutzen ist aber die *Image*-Komponente, weil man sich bei dieser nicht um das Wiederherstellen der Zeichnung kümmern muss, wenn z. B. die Größe des Fensters verändert oder dieses von einem anderen überlagert wurde. Wir werden sie deshalb durchweg benutzen.

Zum Zeichnen benutzt man geeignete Werkzeuge, und diese werden natürlich auch von Delphi bereitgestellt. Zu jeder Canvas gehören

- ein *Pen* (Stift) zum Linienzeichnen, dessen Farbe, Strichstärke und Strichart
- ein *Brush* (Pinsel) zum Flächenfüllen, dessen Farbe und Füllmuster
- ein *Font* (Schriftsatz) zum Schreiben, dessen Farbe, Größe, Name und Stil

man ändern kann. Zu jeder Canvas gehört ein Koordinatensystem, dessen Ursprung (der Punkt [0|0]) in der linken oberen Ecke der Canvas liegt.



Die Anzahl der zur Verfügung stehenden Punkte in den beiden Richtungen ergibt sich aus der momentanen Größe der Image-Komponenten (wobei durchaus auch außerhalb dieses Bereichs gezeichnet werden kann). Heißt die Image-Komponente z. B. *Bild*, dann laufen die x-Werte von *0* bis *Bild.width*, die y-Werte von *0* bis *Bild.height*.

Zur Erzeugung von Grafiken stehen zahlreiche Befehle zur Verfügung. Wir werden allerdings vorerst nur einige wenige benutzen.

- *moveTo(x,y)*..... bewegt den Stift zur angegebenen Position, ohne dass dabei gezeichnet wird.
- *lineTo(x,y)*..... arbeitet entsprechend, zeichnet aber eine Linie in der momentan gesetzten Farbe, Dicke und Stilart.
- *rectangle(x1,y1,x2,y2)*.... zeichnet ein Rechteck mit den angegebenen Eckpunkten. Das Rechteck wird in der durch die Attribute des Brush angegebenen Art gefüllt.
- *ellipse(x1,y1,x2,y2)*..... zeichnet in das angegebene Rechteck eine Ellipse.

Die *Farben* von Pen und Brush werden durch Delphi-Konstante gesetzt, die durch ein vorangestelltes *cl* (für *color*) gekennzeichnet sind. Beispiele sind *clWhite, clRed, clBlue* usw. Beim Zeichnen von Flächen ist die *Style*-Eigenschaft des Brush wichtig. Hat diese den Wert *bsClear* (*bs* für *brushstyle*), dann ist z.B. das Rechteck durchsichtig: es werden nur die Randlinien gezeichnet. Wurde der Wert auf *bsSolid* gesetzt, dann werden die Flächen mit der Farbe des Brush gefüllt. Weitere Füllmuster sind möglich.

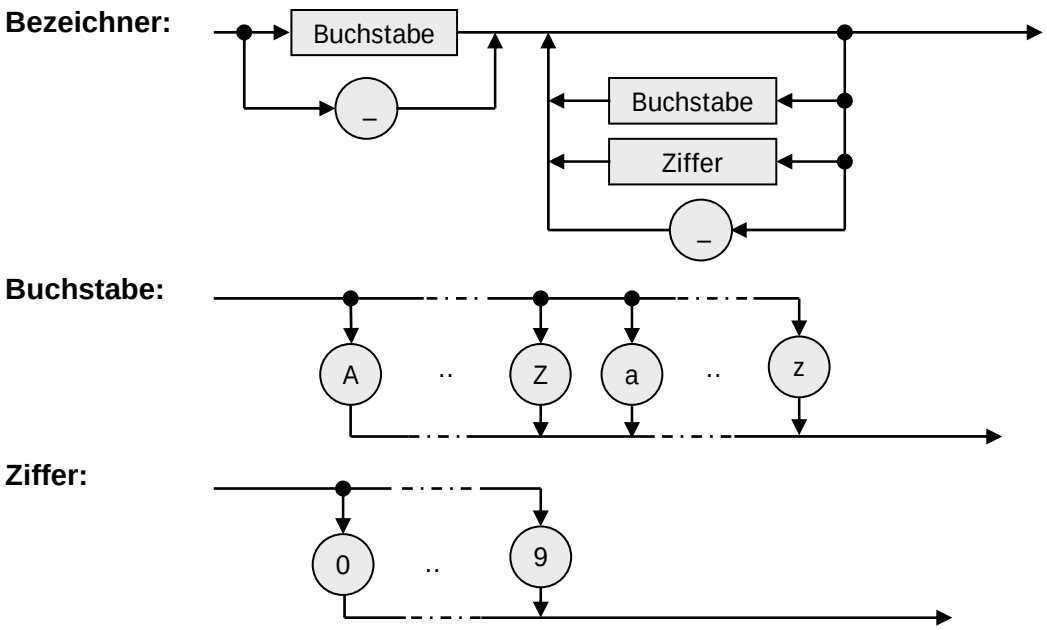
Im Bereich der Computergrafik ist der *Zeichenmodus* von Bedeutung, um bei Bedarf gezeichnete Elemente wieder löschen zu können. Im invertierenden *XOR-Mode* wird die Farbe der veränderten Punkte „umgedreht“, aus weiß wird schwarz usw. Zweimaliges Zeichnen bedeutet dann zweimaliges Umdrehen: der alte Zustand ist wieder hergestellt. In Delphi wird der Zeichenmodus durch die *Mode*-Eigenschaft des Pens festgelegt. Bei *pmCopy* (*pm* für *penmode*) wird in der angegebenen Farbe gezeichnet, bei *pmXOR* (oder – besser noch – bei *pmNotXOR*) wird invertierend gezeichnet.

Bevor wir zu einem konkreten Beispiel kommen, müssen wird noch die Konventionen für Delphi-Namen kennen lernen.

3.2 Bezeichner („Namen“) in Delphi-Programmen

In Delphi erhalten alle wesentlichen Elemente eines Programms eigene Namen.

- Einige Namen, die *reservierten Worte*, sind vorgegeben und dürfen nicht neu vergeben werden: z. B**.** *PROGRAM, UNIT, PROCEDURE, VAR, BEGIN, END***, ...** Aus diesen Worten werden die leeren Programme gebildet, die Delphi automatisch erzeugt. Zusätzlich werden darin Begriffe benutzt, die in den Bibliotheken von Delphi definiert wurden – z. B. Komponenten vom Typ *tForm*. Auch diese Namen sollten auf keinen Fall umdefiniert werden.
- Für neu eingeführte Größen müssen selbst gewählte Namen gefunden werden. Dafür gelten die folgenden Regeln:
  - Nur die ersten 63 Zeichen werden unterschieden.
  - Das erste Zeichen muss ein Buchstabe (oder der Unterstrich „_“) sein.
  - Leer- oder Sonderzeichen (wie Umlaute) sind verboten.
  - Groß- und Kleinschreibung wird nicht unterschieden, sollte aber zur besseren Lesbarkeit benutzt werden.
  - Gültige Bezeichner sind z. B. *A, Test, EinErsterVersuch, Prog12, . . .*
- Erlaubte Namen erhält man, wenn der Name einem möglichen Durchlauf im folgenden *Syntaxdiagramm* entspricht:



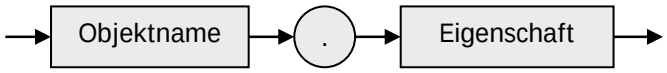
Im selben *Namensbereich* (oder *Gültigkeitsbereich*) darf kein Bezeichner doppelt auftreten. Namensbereiche werden z. B. durch eine Unit, ein Unterprogramm (*procedure*) oder eine Objektklasse definiert. Innerhalb solch eines Bereichs gibt es zwei Arten von Bezeichnern:

- *lokale Namen* werden innerhalb des Bereichs definiert, z. B. für Variable.
- *globale Namen* wurden außerhalb des Bereichs definiert, z. B. in einer Unit, die benutzt wird, oder einer Objektklasse, von der geerbt wurde.

Werden lokale Namen vergeben, die schon global vereinbart waren, dann *überschreiben* die lokalen Bezeichner die globalen, ersetzen diese also. ***Alle Variablen existieren nur solange, wie der Programmteil, in dem sie vereinbart wurden, aktiv ist.***

In Objekten werden Methoden und Eigenschaften des Objekts angesprochen, indem der Methode oder Eigenschaft der Objektname - durch einen Punkt getrennt - vorangestellt wird.

Objekteigenschaft:

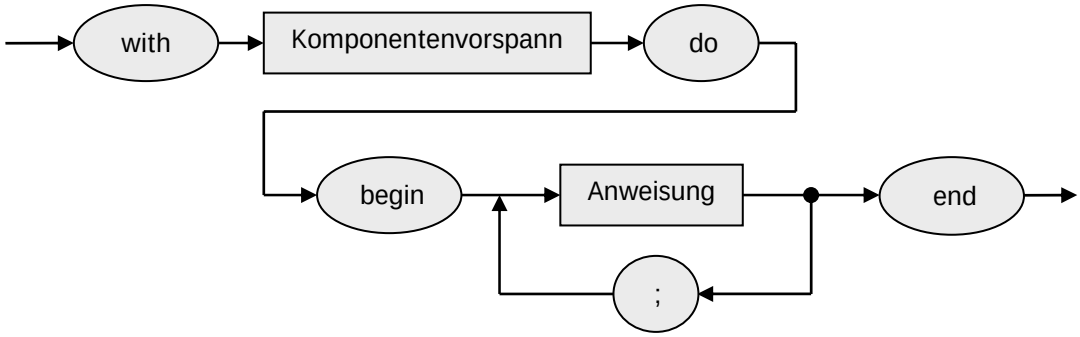


Da Objekte wieder Objekte enthalten können, und diese wieder Objekte usw., können Namen recht lang werden:

- *Form1.Canvas.Pen.Color* ist die Stiftfarbe des Pens der Leinwand von *Form1*.
- *Zeichenbrett.Bild.Canvas.Brush.Style* gibt die Art an, wie der Hintergrund auf der Canvas des Bildobjekts gezeichnet wird, das auf dem Zeichenbrett enthalten ist.

Werden mehrere Komponenten eines Objekts nacheinander angesprochen, dann kann der Vorspann in der Bezeichnungskette durch eine *with*-Anweisung zusammengefasst werden. In den folgenden Anweisungen wird dann bei allen Bezeichnern versucht, Namen mit diesem Vorspann zu finden, bevor sie ohne Vorspann benutzt werden.

WITH-Anweisung:



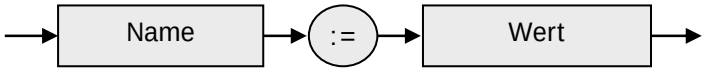
```
WITH Zeichenbrett.Bild.Canvas DO
      BEGIN
      Pen.Color   := clRed;
      Pen.Width   := 3;
      Pen.Mode    := pmNOTXOR;
      Brush.Style := bsClear
      END;
```

ersetzt also

```
Zeichenbrett.Bild.Canvas.Pen.Color   := clRed;
Zeichenbrett.Bild.Canvas.Pen.Width   := 3;
Zeichenbrett.Bild.Canvas.Pen.Mode    := pmNOTXOR;
Zeichenbrett.Bild.Canvas.Brush.Style := bsClear;
```

Der *Zuweisungsoperator* „:=“ weist einer Eigenschaft (oder einer Variablen) den rechts angegebenen Wert zu. Er ist zu lesen als „... *erhält den Wert* ...“.

Wertzuweisung:

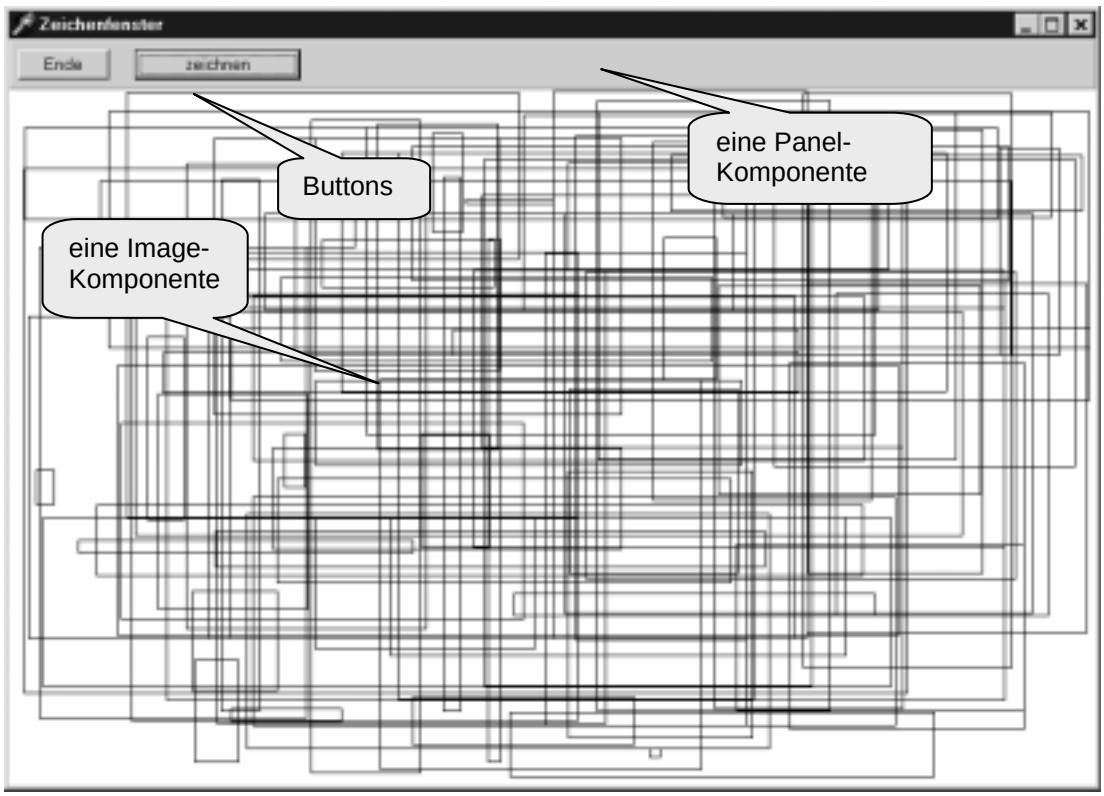


3.3 Zufallsgrafik

Wir wollen als erstes ein Programm entwickeln, mit dem Rechtecke auf der Zeichenfläche erzeugt werden, deren Eckpunkte zufällig verteilt sind.

Dazu benötigen wir einen Button namens *Zeichnen*, der ein Zufallsrechteck erzeugt, wenn er angeklickt wird. Wir müssen also die Zeichenbefehle in der Ereignisbehandlungsroutine für das *OnClick*-Ereignis unterbringen. Diese wird automatisch mit *ZeichnenClick* bezeichnet, wenn wir den Zeichnen-Button entsprechend benannt haben. Zusätzlich erzeugen wir den schon bekannten Ende-Button zum Abbruch des Programms. Beide Buttons (und später weitere) wollen wir in einem Container unterbringen, der diese Knopfleiste übersichtlich zusammenfasst. Für solche Zwecke ist die *Panel*-Komponente geeignet, deren *Align*-Eigenschaft wir im Objektinspektor auf *alTop* setzen. Damit befindet sich das Panel immer am oberen Rand des Fensters.

Den Rest des Fensters soll eine *Image*-Komponente namens *Bild* ausfüllen, auf der gezeichnet wird. Deren Align-Eigenschaft setzen wir auf *alClient*. Das Image passt sich so automatisch der restlichen Fensterfläche an.



Wie erzeugt man nun Zufallszahlen? Für diesen Zweck benutzen wir die *random*-Funktion, die (Pseudo)-Zufallszahlen aus einem vorgegebenen Bereich berechnet. Sie kann auf zwei Arten aufgerufen werden:

```
x := random(Obergrenze);
```
liefert ganze Zahlen

oder

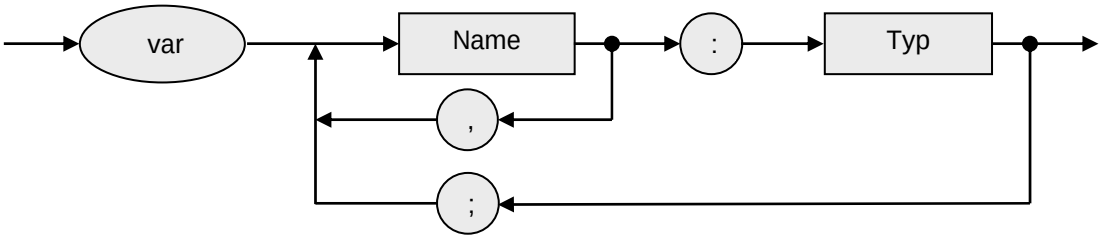
```
x := random;
```
liefert Gleitpunktzahlen

Der *Parameter* *Obergrenze* gibt an, wie viele verschiedene Zufallszahlen als Ergebnis möglich sind. Die niedrigste Zahl ist die Null. Der Aufruf von *random(100)* liefert also eine von 100 verschiedenen Zufallszahlen, beginnend bei 0: damit ist das größtmögliche Ergebnis 99 (denn zwischen – einschließlich - 0 und 99 liegen 100 Zahlen). Wird die *random*-Funktion ohne Parameter aufgerufen, dann liefert sie Zufallszahlen zwischen 0 und 1. Der Zufallsgenerator kann auf einen zufälligen Startwert gesetzt werden indem an beliebiger Stelle einmal der Befehl *randomize* aufgerufen wird.

Ganze Zahlen werden in Delphi meist als Werte gespeichert, für die jeweils entweder 16 oder 32 Bits zur Verfügung stehen (je nach Delphi-Version). Vereinbart man die eine Hälfte als positiv, die andere als negativ, dann hat dieser *Datentyp* ***integer*** (wobei „integer“ eben „ganze Zahl“ bedeutet) einen *Wertebereich* von –32768 bis +32767 bzw. von –2147483648 bis +2147483647. Solche Zahlen benötigen Speicherplatz, der von Delphi bereitgestellt werden muss. Erreichbar wird dieser Speicher über einen Namen, den wir uns entsprechend den Namenskonventionen ausdenken. Da wir diesen Speicher mit unterschiedlichen Werten füllen können, spricht man von *Variablen* mit den entsprechenden *Variablen-Namen*.

Bevor man Variable benutzt, müssen sie vereinbart werden, und das muss vor dem ersten Befehl geschehen, also entweder im vorderen Teil einer Unit (im Interface-Abschnitt) oder später im Implementation-Teil. Die *Variablenvereinbarung* hat einen Aufbau, der sich aus dem entsprechenden Syntaxdiagramm ergibt. Wie schon bei den Bezeichnern sind alle Vereinbarungen erlaubt, die einem möglichen Durchlauf durch das Diagramm entsprechen.

Variablenvereinbarung:



Mögliche Variablenvereinbarungen sind z. B.

- `var i: integer;`
- `var i,j,k: integer;`
- `var Nummer: integer; Farbe: tColor;`

Für unser Programm vereinbaren wir für die Koordinaten der gegenüberliegenden Eckpunkte eines Rechtecks entsprechende Variable

```
var x1, y1, x2, y2: Integer;
```

und weisen diesen als Wert jeweils eine Zufallszahl zu. Deren Bereich ergibt sich aus den momentanen Maßen der Image-Komponente:

```
x1 := random(width);
x2 := random(width);
y1 := random(height);
y2 := random(height);
```

Danach setzen wir den Füllstil des Brush auf *bsClear* (Fläche nicht füllen) und „bitten“ die Zeichenfläche, ein Rechteck mit den angegebenen Maßen zu zeichnen.

```
canvas.brush.style := bsClear;
canvas.rectangle(x1,y1,x2,y2);
```

Insgesamt erhalten wir die folgende Unit:

```
unit uZufall1;

interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls,
     Forms, Dialogs, StdCtrls, ExtCtrls;

type TZeichenfenster = class(TForm)
       Panel1: TPanel;
       Bild: TImage;
       zeichnen: TButton;
       Ende: TButton;
       procedure zeichnenClick(Sender: TObject);
       procedure EndeClick(Sender: TObject);
       end;

var Zeichenfenster: TZeichenfenster;

implementation
{$R *.DFM}

procedure TZeichenfenster.zeichnenClick(Sender: TObject);
var x1,y1,x2,y2: integer;
begin
with Zeichenfenster.Bild do
   begin
   x1 := Random(Width);  x2 := Random(Width);
   y1 := Random(Height); y2 := Random(Height);
   Canvas.Brush.Style := bsClear;
   Canvas.Rectangle(x1,y1,x2,y2)
   end
end;

procedure TZeichenfenster.EndeClick(Sender: TObject);
begin Halt end;

initialization
randomize
end.
```

Klassendefinition der neuen Form

Instanz dieser Klasse

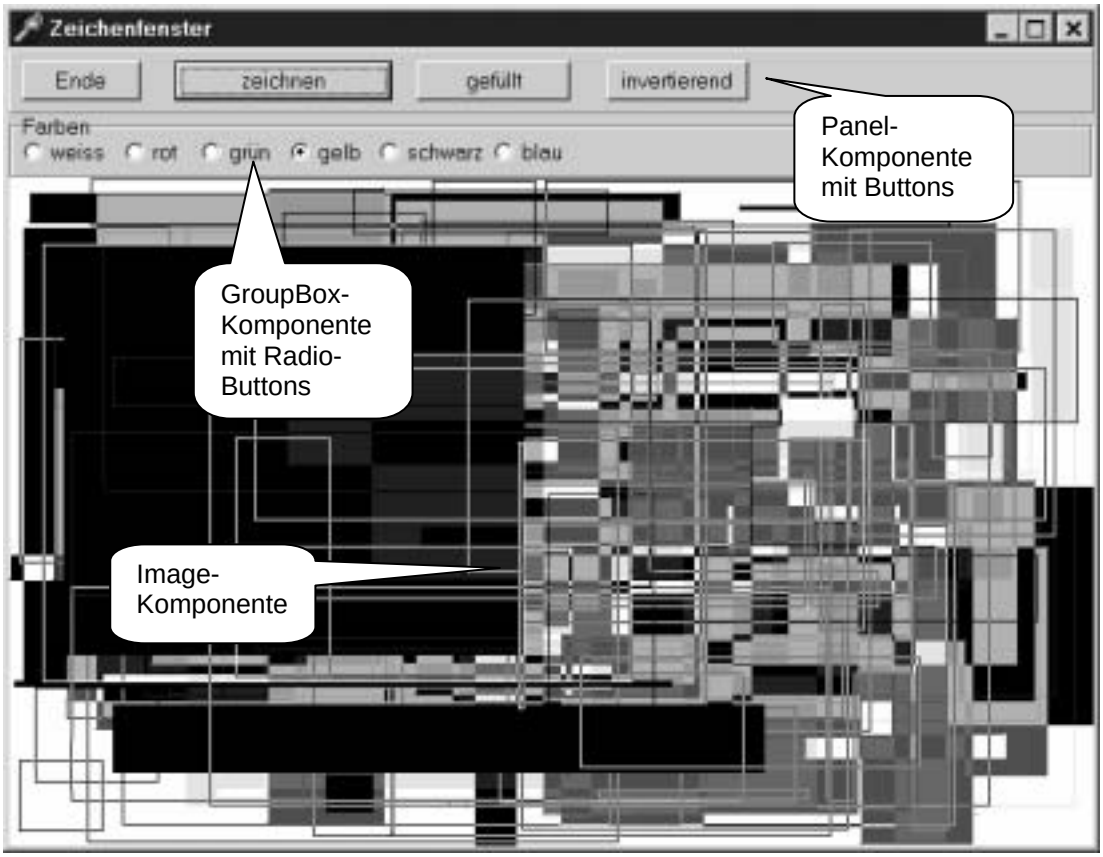
Ereignisbehand-lungsmethode

Vorbereitungsschritt: Der Zufallsgenerator wird auf einen zufälligen Anfangswert gesetzt.

Wir wollen jetzt unser Programm etwas ausbauen, also um Möglichkeiten anreichern,

- die Zeichenfarbe zu wechseln,
- die Rechteckflächen zu füllen (*bmSolid*)
- und die Rechtecke wahlweise invertierend (*pmNOTXOR*) oder überschreibend (*pmCOPY*) zu zeichnen.

Dazu ergänzen wir die vorhandene Form um zwei Buttons *Hintergrund* und *Modus* (mit denen wir die Zeichenart ändern) sowie um eine *GroupBox*, die wir *Farben* nennen. Diese platzieren wir unter die Panel-Komponente mit den Buttons. In der Groupbox ordnen wir einige Radio-Buttons *rot, schwarz, gelb,* ... zur Auswahl der Farben an. So zusammengefasste Radio-Buttons wirken wie eine Leiste von Druckschaltern, von denen immer nur einer aktiv sein kann. Da Schwarz die anfangs automatisch ausgewählte Zeichenfarbe ist, markieren wir diesen Radio-Button im Objektinspektor als aktiv, indem wir dessen *checked*-Eigenschaft auf *true* setzen.



Wir haben jetzt eine Reihe von Auswahlknöpfen – aber wie benutzt man sie?

Jeder Button soll eine Zeicheneigenschaft (Farbe, Zeichenmodus, ..) bei Bedarf auf einen vorgegebenen Wert setzen. Nach dieser Aktion soll diese Eigenschaft solange unverändert bleiben, bis sie wieder bewusst gewechselt wird. Wir benötigen deshalb *Variable*, in denen die Zeicheneigenschaften gespeichert werden. Da diese Größen während des gesamten Programmlaufs ihre Werte behalten müssen, dürfen sie nicht innerhalb einer Ereignisbehandlungsmethode vereinbart sein, sondern gehören als *globale Variable* direkt in den Haupt-Implementationsteil der Unit, am besten direkt an den Anfang:

```
implementation
{$R *.DFM}
VAR Farbe: tColor;
    Hg   : tBrushStyle;
    Art  : tPenMode;
```

Der Typ einer Variablen muss der Größe entsprechen, die gespeichert werden soll. Bei Farben handelt es sich um den Delphi-Datentyp *tColor*, bei der Art des Hintergrundes um *tBrushStyle* und beim Stift um *tPenMode*. (Üblicherweise kennzeichnet man neu definierte Datentypen durch ein vorangestelltes *t*.) Immer, wenn ein Button angeklickt wird, muss die entsprechende Variable den Wert einer geeigneten *Delphi-Konstanten* erhalten, von denen wir schon einige kennen gelernt haben. (Insgesamt gibt es Hunderte solcher Konstanten, ihre Bedeutung erfährt man am einfachsten aus der Delphi-Hilfe, die für die praktische Arbeit mit dem System unverzichtbar ist.) Als Beispiel wollen wir die Reaktion des *blau*-Buttons auf ein OnClick-Ereignis betrachten:

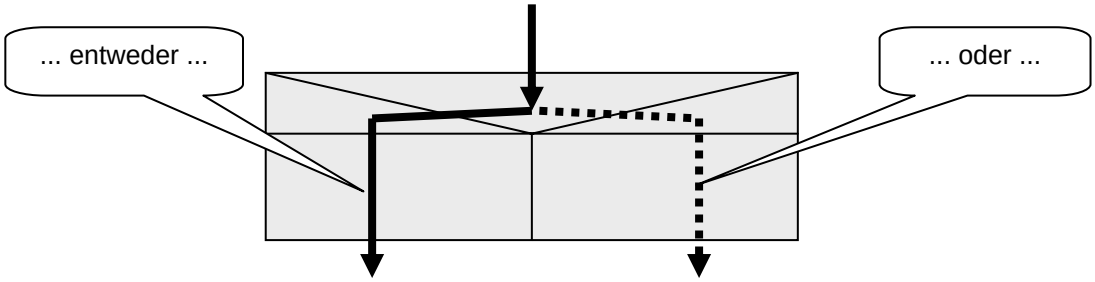
```
procedure TZeichenfenster.blauClick(Sender: TObject);
begin
farbe := clBlue
end;
```

Etwas komplizierter gestaltet sich die Angelegenheit bei den Buttons, die den Zeichenstil festlegen. Hier erhält die entsprechende Variable ja keinen festen Wert, sondern ihr Wert wird jedes Mal „umgedreht“. Dabei sollte auch gleich die Aufschrift des Buttons geändert werden, da er ja bei jedem Mausklick seine Funktion ändert. Wir haben es also mit einer *Alternative* oder *Fallunterscheidung* zu tun: je nach augenblicklicher Einstellung werden Variable und Aufschrift auf einen anderen Wert gesetzt.

der Button hat die Aufschrift ´leer´	
wahr	falsch
Hg ← bmSOLID	Hg ← bmCLEAR
Hintergrund.Caption ← ´gefüllt´	Hintergrund.Caption ← ´leer´

(Bei den Aufschriften der Buttons handelt es sich um *Zeichenfolgen* vom Datentyp *string*, die in Delphi durch Hochkommas eingeschlossen werden.)

Die Darstellung von Befehlsfolgen in Kastenform nennt man ein *Struktogramm*. Struktogramme werden von oben nach unten durchlaufen, beschreiben den Ablauf eines Programmstücks. In unserem Fall wird zuerst entschieden, welche Seite der Alternative ausgewählt wird. Danach werden die entsprechenden Befehle ausgeführt.

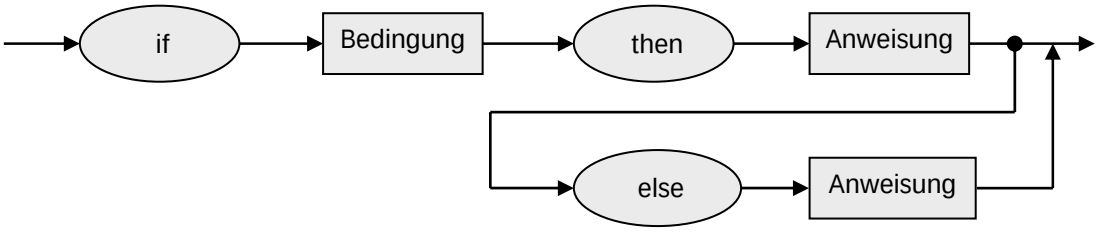


Die Übersetzung in PASCAL lautet:

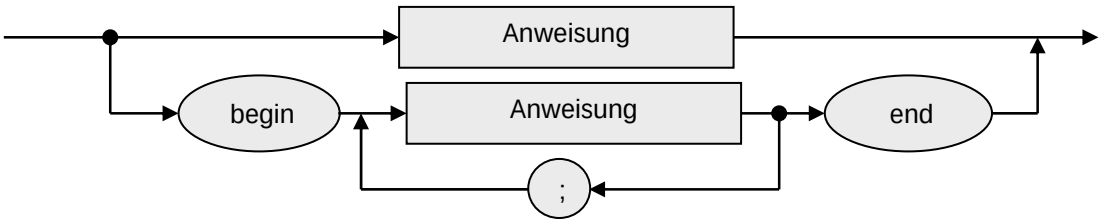
```
procedure TZeichenfenster.HintergrundClick(Sender: TObject);
begin
if Hintergrund.Caption = 'leer' then
   begin
   Hg := bsSolid;
   Hintergrund.Caption := 'gefüllt'
   end
else begin
     hg := bsClear;
     Hintergrund.Caption := 'leer'
     end
end;
```

In Object-Pascal wird der Aufbau einer Alternative durch ein Syntaxdiagramm beschrieben.

Alternative:



Anweisung:



Der *else*-Zweig kann also bei Bedarf entfallen (*einseitige Alternative*), und die auszuführende Anweisung kann aus einer einzelnen, (meist aber) aus mehreren Anweisungen bestehen, die dann mit *begin* und *end* zusammengeklammert werden.

Zulässige Alternativen sind

- ```
  if i > 0
     then i := i + 1
     else i := i – 1;
  ```
- ```
  if Farbe = clRed
     then rectangle(x1,y1,x2,y2)
     else ellipse(x1,y1,x2,y2);
  ```
- ```
  if Name = ´Meier´
     then begin
          Vorname := ´Hans´;
          Alter := 44
          end;
  ```

Ein häufiger Fehler ist das Einfügen eines Semikolons vor *else*. ***Dort darf laut Syntaxdiagramm kein Semikolon stehen!***

Globale Variable erfordern meist das Setzen von Anfangswerten als Vorgaben, mit denen das Programm startet. Werte, die sich nicht auf Delphi-Komponenten beziehen, können in den *Initialization*-Teil der Unit geschrieben werden. Dort sollte z. B. die Anweisung *randomize* untergebracht werden, die den Zufallsgenerator auf einen zufälligen Anfangswert setzt. Aber auch die Anfangsfarben und Stile können gesetzt werden. (Alternativ dazu auch in einer *OnCreate*-Routine.)

```
initialization
randomize;
Farbe := clBlack;
Hg    := bsClear;
Art   := pmCopy;
end.
```
```

Wir brauchen jetzt nur noch vor dem Zeichnen eines Rechtecks dafür zu sorgen, dass die gewählten Einstellungen auch umgesetzt, also dem *Pen* und dem *Brush* zugewiesen werden.

```
procedure TZeichenfenster.zeichnenClick(Sender: TObject);
var x1,y1,x2,y2: integer;
begin
with Zeichenfenster.Bild do
   begin
   x1 := Random(Width);
   x2 := Random(Width);
   y1 := Random(Height);
   y2 := Random(Height);
   Canvas.Pen.Color   := Farbe;
   Canvas.Brush.Color := Farbe;
   Canvas.Brush.Style := Hg;
   Canvas.Pen.Mode := Art;
   Canvas.Rectangle(x1,y1,x2,y2);
   end
end;
```

Zuweisen der gewählten Eigenschaften an den Pen und den Brush ...

... bevor das Rechteck gezeichnet wird.

3.4 Aufgaben

1. Bilden Sie jeweils fünf korrekte bzw. falsche ***Pascal-Bezeichner.***

2. a: Bilden Sie fünf korrekte ***Variablenvereinbarungen***.

   b: Bilden Sie fünf falsche Variablenvereinbarungen und geben Sie jeweils genau die Stelle an, an der Ihre Vereinbarung dem Syntaxdiagramm widerspricht.

3. Übersetzen Sie in ***Pascal-Alternativen***:

   a: Wenn die Zahl i verschieden von Null ist („i <> 0“), dann wird j zum Kehrwert von i.

   b: Wenn der Name ´Müller´ ist, dann wird die Aufschrift des Buttons1 ´getroffen´ sonst ´falsch!´.

   c: Wenn a größer als b ist, dann vertausche a und b.

4. Finden Sie alle ***Fehler***:

```
PROZEDUR Prüfknopf.Click(Sender: tObjekt)
VAR rot, grün := INTEGER;
IF Sender=rot
         i = i+1;
ELSE Color = grün
END;
```

5. Vervollständigen Sie das Programm zur ***Zufallsgrafik***

   a: durch Einführung eines Knopfes zum Umschalten zwischen den Zeichenarten „invertierend“ bzw. „überschreibend“.

   b: durch Einführung eines Knopfes zum Umschalten zwischen „Rechtecke zeichnen“ bzw. „Ellipsen zeichnen“.

   c: durch weitere Farbknöpfe.

   d: durch Möglichkeiten zur Auswahl anderer Hintergründe (*bsCROSS*, *bsHORIZONTAL*, *bsVERTICAL*, *bsDIAGONAL*, ...).

6. Informieren Sie sich im Hilfesystem über die folgenden Grafikmethoden der Canvas. Suchen Sie dafür mit der Option „Suche über Schlüsselwort“ nach der Klasse *tCanvas* und dann nach den angegebenen Begriffen.

   a: Rechtecke mit abgerundeten Ecken (Methode *RoundRect*).

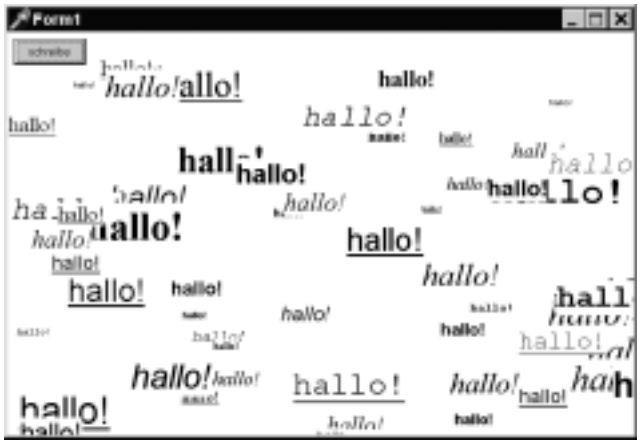
   b: Punkte (Eigenschaft *Pixels*).

   c: Torten- und Bogenstücke (Methoden *Pie* und *Arc*)

   d: Polygone (Methoden *PolyLine* bzw. *Polygon)*

   Verwenden Sie dann diese Möglichkeiten in der Zufallsgrafik.

7. Informieren Sie sich über die *TextOut*-Methode der Canvas sowie über die Einstellmöglichkeiten für *tFont.* Erzeugen Sie dann „Zufallstexte“ z. B. wie nebenstehend.



8. a: Starten Sie die erstellten Programme mit Hilfe des ***Debuggers***, indem Sie entweder im Startmenü die Einzelschritt-Option wählen oder die Taste F7 drücken. Verfolgen Sie so schrittweise den Programmablauf und die Reaktionen des Programms auf unterschiedliche Benutzereingaben (z. B. Mausklicks).

   b: Lassen Sie im Programmtext die Maus kurze Zeit auf einer Variablen ruhen. Es wird Ihnen dann deren momentaner Wert angezeigt.

   c: Wählen Sie alternativ dazu im Startmenü die Option „Ausdruck hinzufügen“. Geben Sie Variablennamen an und verfolgen Sie im Debugger die Änderungen der Werte.

   d: Informieren Sie sich im Hilfesystem über den Begriff „Haltepunkt“. Setzen Sie dann Haltepunkte ein, um Programme im Debugger zu testen.