



Zawiera CD

wydanie

3.

Teresa_Pamuła

Aplikacje w Delphi.

< Przykłady >

Projektuj wspaniałe aplikacje
z pomocą środowiska **Delphi 2010!**

- Wygląd i podstawowe funkcje aplikacji, czyli od czego zacząć pracę
- Wprowadzanie danych i okienka komunikatów, czyli zapewnianie dialogu między programem a użytkownikiem
- Tworzenie nowych komponentów, czyli jak nadać aplikacji niepowtarzalny styl

Hellen



» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2010

Aplikacje w Delphi. Przykłady. Wydanie III

Autor: Teresa Pamuła
ISBN: 978-83-246-2851-3
Format: 158×235, stron: 416



Projektuj wspaniałe aplikacje z pomocą środowiska Delphi 2010!

- Wygląd i podstawowe funkcje aplikacji, czyli od czego zacząć pracę
- Wprowadzanie danych i okienka komunikatów, czyli zapewnianie dialogu między programem a użytkownikiem
- Tworzenie nowych komponentów, czyli jak nadać aplikacji niepowtarzalny styl

Środowisko Delphi służy do szybkiego tworzenia aplikacji działających w systemie Windows. Zawiera bogate biblioteki komponentów, mechanizmy Plug and Play oraz Code Insight, a także palety komponentów i narzędzia ułatwiające ich wyszukiwanie. Dzięki Delphi IDE można w prosty sposób zaprojektować interfejs użytkownika nowej aplikacji, określić jej wygląd oraz sposób działania w oparciu o istniejące kontrolki i biblioteki klas, w dużym stopniu zdając się na automatyczne generowanie kodu.

Jeśli chcesz poznać, dogłębnie zrozumieć i wykorzystać do swoich celów sposób działania Delphi 2010, powinieneś koniecznie sięgnąć po książkę „Aplikacje w Delphi. Przykłady”. Znajdziesz tu wszelkie informacje na temat obsługi samego środowiska, podstawowych składników każdej tworzonej aplikacji, wprowadzania i formatowania danych, list, tabel, grupowania i projektowania nowych komponentów, wykorzystania technologii OLE do zapisu i modyfikacji danych w formatach .doc i .xls, możliwości graficznej prezentacji danych, ich drukowania i współpracy Twojej aplikacji z multimediami. Wszystko to oraz wiele innych zagadnień pokazano tu na praktycznych, konkretnych przykładach, ułatwiających zrozumienie i gotowych do zastosowania w Twoich własnych projektach.

- Podstawowe składniki aplikacji, menu główne i podręczne, pasek narzędzi
- Wprowadzanie danych, formatowanie i wyświetlanie na ekranie
- Okienka komunikatów i okienka dialogowe z karty Dialogs
- Listy wyboru i prosty edytor
- Grupowanie i dynamiczne tworzenie komponentów
- Komponenty do wyboru daty i czasu, odmierzanie czasu
- Zakładki TTabControl i TPageControl
- Grafika w Delphi i wyświetlanie obrazów
- Tabelaryzacja danych i ich graficzna prezentacja danych
- Współpraca programu z plikami dyskowymi
- Drukowanie w Delphi i programy z wieloma oknami
- Posługiwanie się wieloma komponentami tego samego typu
- Definiowanie nowych klas komponentów i wykorzystanie mechanizmu OLE
- Podstawowe operacje na bazach danych
- Delphi i multimedia

Odkryj fantastyczne możliwości Delphi!

Spis treści

Wprowadzenie	7
Rozdział 1. Projektowanie aplikacji w Delphi	9
Środowisko zintegrowane — Delphi IDE	10
Elementy projektu aplikacji	10
Standardowe właściwości komponentów	14
Standardowe zdarzenia	14
Nowości w Delphi 2010	17
Jak przekształcić interfejs aplikacji z Delphi 7 na nowocześniejszy, zgodny z nowszymi wersjami Windows?	20
Jak sprawdzić, czy aplikacja jest już uruchomiona?	21
Rozdział 2. Podstawowe składniki aplikacji	23
Okno aplikacji	23
Ikona aplikacji	28
Wyświetlanie napisów	29
Rodzaje przycisków, podobieństwa i różnice	33
Etykiety i przyciski	37
Linki	42
Rozdział 3. Menu główne i podręczne, pasek narzędzi	45
Wielopoziomowe menu główne	46
Przyporządkowanie poleceń opcjom menu	48
Menu podręczne	52
Polskie litery w nazwach poleceń menu	54
Pasek narzędzi TToolBar	54
Rozdział 4. Wprowadzanie danych, formatowanie i wyświetlanie na ekranie	57
Liczby — funkcje konwersji i formatowanie	58
Daty — funkcje konwersji i formatowanie daty oraz czasu	60
Przecinek czy kropka?	62
Systemowe separatory liczb i daty	63
Wprowadzanie danych za pomocą okienek edycyjnych TEdit	64
Wprowadzanie danych za pomocą okienek InputBox i InputQuery	71
Sposoby zabezpieczania programu przed błędami przy wprowadzaniu danych	72
Maskowanie danych wejściowych	72
Blokowanie możliwości wprowadzania niektórych znaków, np. liter lub cyfr	74
Korzystanie z funkcji konwersji StrToIntDef (z wartością domyślną)	75

Zmiana zawartości okienka TEdit za pomocą suwaka TScrollBar.....	75
Zmiana zawartości okienka TEdit za pomocą komponentu TUpDown.....	76
Stosowanie instrukcji obsługi wyjątków	77
Obliczenia. Wybrane funkcje modułu Math	79
Rozdział 5. Okienka komunikatów	83
Wyświetlanie komunikatów z napisami stałymi w języku systemowym	
— MessageBox	84
Wyświetlanie komunikatów za pomocą funkcji ShowMessage, MessageDlg, MessageDlgPos	86
Rozdział 6. Okienka dialogowe z karty Dialogs.....	91
Odczyt i zapis plików z wykorzystaniem okien dialogowych	94
Wyświetlanie nazwy wybranego koloru za pomocą okna TColorDialog	96
Zmiana czcionki na etykiecie za pomocą okna TFontDialog	96
Rozdział 7. Listy wyboru — TListBox i TComboBox	99
Dodawanie elementów do listy.....	101
Wybieranie elementów z listy	102
Sortowanie elementów listy.....	105
Sposoby wyświetlania elementów listy	106
Lista z nazwami czcionek.....	107
Blokowanie edycji dla listy TComboBox.....	108
Czytanie i zapisywanie zawartości listy do pliku dyskowego	109
Konfigurator	110
Rozdział 8. Prosty edytor — komponent TMemo	113
Kopiowanie, wycinanie i wklejanie tekstu	115
Czytanie i zapisywanie tekstu do pliku.....	116
Wyświetlanie informacji o położeniu kursora	117
Automatyczne kasowanie linii niezawierających liczb lub wybranych znaków	118
Wyświetlanie współrzędnych kursora w polu TMemo i zegara na pasku TStatusBar... ..	119
Dodawanie danych do TMemo z komponentu TEdit, TComboBox.....	121
Rozdział 9. Grupowanie komponentów	123
Pola opcji i pola wyboru.....	123
Komponenty grupujące.....	124
Ramka TBevel.....	130
Rozdział 10. Komponenty do wyboru daty i czasu TDateTimePicker i TMonthCalendar	133
Rozdział 11. Zakładki TTabControl i TPageControl	137
Rozdział 12. Odmierzanie czasu — komponent TTimer	145
Rozdział 13. Grafika w Delphi — korzystanie z metod obiektu TCanvas	149
Wyświetlanie prostych figur geometrycznych i tekstu	150
Rysowanie „trwałe” — zdarzenie OnPaint.....	157
Rysowanie myszą po formularzu.....	159
Rysowanie myszą po komponencie TImage.....	160
Przykłady animacji w Delphi.....	162
Rozdział 14. Wyświetlanie obrazów — komponent TImage	169
Rysowanie po obrazie.....	171
Binaryzacja obrazu	172
Skalowanie obrazów .BMP, .JPG	175

Przeglądanie wczytanych obrazów	178
Wyświetlanie zawartości listy obrazków TImageList	179
Zamiana formatu obrazów z .JPG, .GIF, .PNG na .BMP	181
GIF animowany	182
Zaznaczanie i wycinanie prostokątnego obszaru z obrazu	183
Rozdział 15. Tabelaryzacja danych — komponenty TStringGrid, TDrawGrid i TValueListEditor	187
Ustalanie podstawowych parametrów tabeli	190
Wypełnianie tabeli danymi	192
Wybieranie komórek tabeli	194
Filtrowanie wprowadzanych danych	197
Niestandardowe przejście do kolejnej komórki — klawisz Enter	199
Zmiana koloru i wyrównania tekstu w wybranych komórkach	200
Zmiana koloru wierszy tabeli	204
Wyświetlanie tekstu w komórce w dwóch wierszach	206
Totolotek	208
Tabela i lista	210
Wyświetlanie listy obrazów i tekstu w tabeli TDrawGrid	211
Wyświetlanie obrazu pobranego z pliku w komórkach tabeli TDrawGrid	213
Wprowadzanie tekstu do komórek tabeli TDrawGrid	217
Tabela TValueListEditor	218
Rozdział 16. Graficzna prezentacja danych — komponent TChart	223
Rysowanie wykresów z wykorzystaniem komponentu TChart	223
Opis wybranych właściwości, metod i zdarzeń komponentów TChart i TChartSeries	226
Wykresy kołowe	229
Wykresy kolumnowe	232
Wykresy funkcji matematycznych	235
Formatowanie i skalowanie wykresów	239
Posługiwanie się wieloma wykresami	245
Wykres Gantta	248
Rozdział 17. Współpraca programu z plikami dyskowymi	251
Wybór foldera plików	251
Wyszukiwanie plików	254
Zapisywanie danych z okienek TEdit i tabeli TStringgrid do pliku tekstowego	256
Czytanie danych do okienek TEdit i tabeli TStringgrid z pliku tekstowego	258
Zapisywanie i odczytywanie danych z tabeli do pliku *.csv	259
Zmiana nazw grupy plików	261
Korzystanie ze strumieni	262
Rozdział 18. Drukowanie w Delphi	269
Drukowanie napisów i tekstu z okienek edycyjnych	270
Drukowanie tabeli	271
Drukowanie obrazu	273
Drukowanie tekstu sformatowanego za pomocą komponentu TRichEdit	274
Drukowanie za pomocą Rave Reports	280
Rozdział 19. Programy z wieloma oknami	293
Wymiana danych i metod między modułami	293
Program z hasłem	296
Wyświetlanie tytułu programu	299

Aplikacje typu MDI.....	301
Test wyboru.....	303
Rozdział 20. Posługiwanie się wieloma komponentami tego samego typu.	
Operatory Is i As	305
Wprowadzanie i kasowanie danych dla kilku okienek edycyjnych	306
Przypisywanie grupie komponentów tej samej procedury obsługi zdarzenia	308
Wyświetlanie informacji o numerach kontrolek, ich nazwach i klasach	311
Ankieta	312
Rozdział 21. Przykłady wykorzystania mechanizmu OLE w Delphi.....	315
Komponent TOLEContainer	316
Zapisywanie tekstu, grafiki i tabeli do dokumentu w formacie .DOC	318
Zapisywanie danych z aplikacji w Delphi w formacie .XLS	321
Czytanie, modyfikacja i zapisywanie pliku w formacie .XLS.....	324
Rozdział 22. Dynamiczne tworzenie komponentów	327
Wyświetlanie kontrolek i przypisywanie zdarzeniom procedur obsługi.....	328
Przykłady dynamicznego tworzenia wykresów	333
Tworzenie menu w czasie działania programu	336
Tablice dynamiczne.....	338
Rozdział 23. Definiowanie nowych klas komponentów.....	341
Klasa tabel z wyrównaniem zawartości komórek do prawej strony	341
Klasa okienek z właściwością Alignment.....	343
Instalowanie nowych komponentów na palecie komponentów	345
Instalacja nowego komponentu w Delphi 2006.....	349
Nowy komponent do ankiety.....	351
Nowy komponent złożony z komponentów standardowych.....	353
Instalacja nowego komponentu w Delphi 2010	356
Rozdział 24. Podstawowe operacje na bazach danych.....	357
Przeglądanie istniejących baz danych w formacie .dbf.....	359
Tworzenie własnej bazy danych.....	361
Modyfikowanie bazy	363
Filtrowanie rekordów bazy danych.....	365
Wyszukiwanie rekordów	367
Sortowanie.....	368
Rysowanie wykresów na podstawie danych z bazy.....	368
Obliczanie średniej ze wszystkich wartości danego pola.....	370
Biblioteka — przykład relacyjnej bazy danych	370
Logiczne połączenie tabel	373
Drukowanie danych za pomocą programu Rave Reports.....	374
Rozdział 25. Delphi i multimedia	383
Komponent TAnimate	383
Komponent TMediaPlayer	384
Playlista	389
Literatura.....	393
Skorowidz	395

Rozdział 13.

Grafika w Delphi — korzystanie z metod obiektu TCanvas

Niektóre komponenty mają właściwość typu obiektowego Canvas (tzw. płótno). Są to m.in.: TForm, TImage, TPaintBox, TBitmap, TComboBox, TStringGrid, TListBox, TPrinter.

Właściwość Canvas zawiera metody, które pozwalają na rysowanie na tych komponentach za pomocą linii różnych figur, kolorowanie powierzchni oraz wyświetlanie tekstu. Możliwa jest również zmiana koloru i grubości linii, koloru i wzoru wypełnienia, atrybutów czcionki itd.

Rysowanie za pomocą metod obiektu Canvas różnych obiektów może być przydatne do zmiany cech niektórych komponentów, np. TStringGrid czy TChart, a także przy drukowaniu formularza i tekstu.

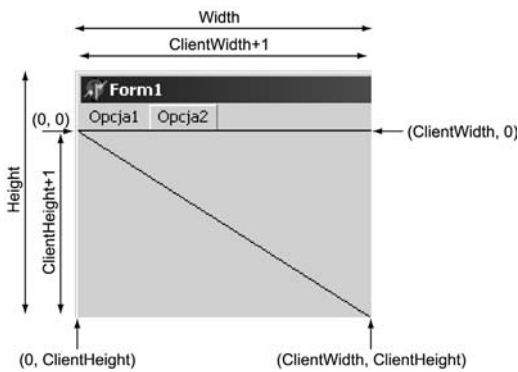
Rysowanie po obrazie wczytanym do komponentu TImage jest możliwe wtedy, kiedy jest to obraz w formacie *.BMP* (bitmapa), ponieważ bitmapa ma właściwość Canvas. Jeśli do komponentu TImage wczytany zostanie obraz w formacie *.JPG*, *.GIF* lub *.PNG*, to należy go przekształcić na bitmapę. W przykładzie 13.14 przedstawiono zastosowanie tej możliwości.

Wybrane właściwości klasy TCanvas:

- ◆ Brush — określa wzór lub kolor wypełnienia figur (tzw. pędzel).
- ◆ Font — krój czcionki dla wyświetlanych napisów.
- ◆ Pen — określa cechy kreślonych linii: grubość, styl, kolor (tzw. pióro).
- ◆ PenPos — określa współrzędne kursora graficznego.

Podstawowymi parametrami większości procedur i funkcji graficznych są współrzędne punktu na komponencie, po którym rysujemy. Lewy górny róg ma współrzędne (0, 0), a prawy dolny najczęściej (Width, Height). Na rysunku 13.1 przedstawiono współrzędne okna formularza, które wykorzystano w zadaniach z tego rozdziału. Współrzędne liczone są względem punktu o współrzędnych (0, 0).

Rysunek 13.1.
Formularz z zaznaczonymi wartościami współrzędnych wierzchołków (x, y)



Wyświetlanie prostych figur geometrycznych i tekstu

Proste figury i tekst możemy wyświetlić na formularzu, korzystając z procedur i funkcji obiektu typu TCanvas — tabela 13.1. Właściwości takiego obiektu umożliwiają m.in. zmianę grubości i stylu rysowanych linii, zmianę koloru i wzoru wypełnienia figur oraz wybór kroju i stylu czcionki dla tekstu.

Tabela 13.1. Wybrane metody obiektu TCanvas

Metoda	Znaczenie
Kolor:=Canvas.Pixels[x,y]	Za pomocą funkcji Pixels można odczytać kolor piksela w miejscu o współrzędnych (x, y) — zmienna Kolor jest typu TColor.
Canvas.Pixels[10,20]:=clRed	Ta sama funkcja wywołana w ten sposób powoduje wyświetlenie na formularzu czerwonego punktu w miejscu o współrzędnych [10, 20] — współrzędną poziomą (x) liczymy od lewej do prawej, a współrzędną pionową od góry w dół. Współrzędne lewego górnego wierzchołka to (0, 0).
MoveTo(x,y: integer)	Przenosi kursor graficzny do punktu o współrzędnych x, y.
LineTo(x,y:integer)	Rysuje linię od bieżącej pozycji kursora graficznego do punktu o współrzędnych x, y.
Rectangle(x1, y1, x2, y2: Integer)	Procedura rysuje prostokąt wypełniony standardowym kolorem pędzla (Canvas.Brush.Color).
Ellipse(x1, y1, x2, y2: Integer)	Procedura rysuje elipsę (lub koło) — parametrami są współrzędne dwóch przeciwległych wierzchołków prostokąta (kwadratu), w który elipsa jest wpisana.

Tabela 13.1. Wybrane metody obiektu TCanvas — ciąg dalszy

Metoda	Znaczenie
Polyline(Points: array of TPoint)	Procedura rysuje linię łamaną lub wielokąt. Parametrami są współrzędne punktów, które zostaną połączone linią. Jeśli współrzędne punktu pierwszego i ostatniego są takie same, to rysowany jest wielokąt; w przeciwnym razie linia łamana, np. procedura: <pre>Polyline([Point(40, 10), Point(20, 60), Point(70, 30), Point(10, 30), Point(60, 60), Point(40, 10)])</pre> narysuje gwiazdę pięcioramienną (patrz pomoc dla polyline).
Polygon(Points: array of TPoint)	Procedura umożliwia narysowanie wielokąta wypełnionego bieżącym kolorem i stylem pędzla. Przykładowo instrukcje: <pre>Canvas.Brush.Color = clRed; Canvas.Polygon([Point(10, 10), Point(30, 10), Point(130, 30), Point(240, 120)]);</pre> spowodują narysowanie czworokąta wypełnionego kolorem czerwonym. Współrzędne punktu pierwszego i ostatniego nie muszą się pokrywać, ponieważ procedura i tak łączy na końcu punkt ostatni z punktem pierwszym.
Refresh	Odświeżanie formularza — procedura kasuje wszystkie obiekty rysowane za pomocą metod obiektu Canvas i nieumieszczone w procedurze obsługi zdarzenia OnPaint.
Draw(x, y: integer; Graphic: TGraphic)	Rysuje obraz określony parametrem Graphic w miejscu o współrzędnych x i y (przykład 13.14).
Arc(x1,y1, x2,y2, x3,y3, x4,y4: integer)	Rysuje krzywą eliptyczną w prostokącie o współrzędnych (x1, y1; x2, y2) od punktu o współrzędnych (x3, y3) do punktu (x4, y4).
TextOut(x,y: integer; const Text: string)	Wyświetla tekst od punktu o współrzędnych x, y — lewy górny róg prostokąta zawierającego tekst; Text to parametr w postaci tekstu stałego w apostrofach, np. 'Ala ma kota', lub zmienna zawierająca łańcuch znaków, np. a:='Ala ma kota' (const w nagłówku procedury oznacza podobne wywołanie jak w przypadku wartości, lecz umożliwia bardziej efektywne wykorzystanie pamięci).
CopyRect(const Dest: TRect; Canvas: TCanvas; const Source: TRect)	Kopiuje część obrazu z jednego płótna na inne płótno.
FillRect(const Rect: TRect)	Rysowanie prostokąta wypełnionego bieżącym kolorem i wzorem.
FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle)	Wypełnianie, tzw. powodziowe, obiektów.
FrameRect(const Rect: TRect)	Rysowanie obwodu prostokąta.
Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);	Rysowanie wycinka koła.
RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer)	Rysowanie prostokąta z zaokrąglonymi narożnikami.
StretchDraw(const Rect: TRect; Graphic: TGraphic)	Dopasowanie rysunku do obszaru danego prostokąta.

Tabela 13.1. Wybrane metody obiektu *TCanvas* — ciąg dalszy

Metoda	Znaczenie
<code>TextHeight(const Text: string): Integer</code>	Funkcja zwraca wysokość tekstu w pikselach.
<code>TextOut(X, Y: Integer; const Text: string)</code>	Procedura wyświetla napis na komponencie mającym właściwość <code>TCanvas</code> .
<code>TextRect(Rect: TRect; X, Y: Integer; const Text: string)</code>	Procedura wyświetla napis w prostokącie, którego współrzędne są podane w postaci typu <code>TRect</code> (pierwszy parametr). Procedura była wykorzystywana przy formatowaniu komórek tabeli.
<code>TextWidth(const Text: string): Integer</code>	Funkcja zwraca szerokość tekstu w pikselach.

Oprócz wymienionych metod zdefiniowane są te, które korzystają z tzw. mechanizmów niskopoziomowych i właściwości `Handle` komponentu, np. instrukcja:

```
kol:=GetNearestColor( Form1.Canvas.Handle, RGB(125,67,22));
```

spowoduje przypisanie zmiennej `kol` koloru najbardziej zbliżonego do podanego — w przypadku gdy bieżący tryb graficzny nie ma koloru typu RGB.

Przykład 13.1.

Wyświetl na etykiecie współrzędne prawego dolnego wierzchołka formularza — lewy górny ma współrzędne (0, 0).

Rozwiązanie

Wstaw etykietę `TLabel`. Współrzędne prawego dolnego wierzchołka formularza możemy odczytać, korzystając z właściwości `ClientWidth` i `ClientHeight` formularza. Należy wpisać np. w procedurze obsługi zdarzenia `OnClick` etykiety instrukcję:

```
Label1.Caption:=IntToStr(ClientWidth)+' ',' '+IntToStr(ClientHeight);
```

lub użyć funkcji `GetClientRectangle`, która zwraca wartość typu `TRect` określającą współrzędne dwóch przeciwległych wierzchołków formularza:

```
R:=Form1.GetClientRectangle; //R typu TRect można zadeklarować jako zmienną lokalną
Label1.Caption:=Inttostr(R.Right)+' ',' '+ Inttostr(R.Bottom);
```

Przykład 13.2.

Na środku formularza wyświetl punkt koloru czerwonego, przy czym nie może w tym miejscu znajdować się inny obiekt (np. przycisk), bo wyświetlony piksel zostanie przez ten obiekt przesłonięty.

Rozwiązanie

Poniższą instrukcję wpisz np. w procedurze obsługi przycisku:

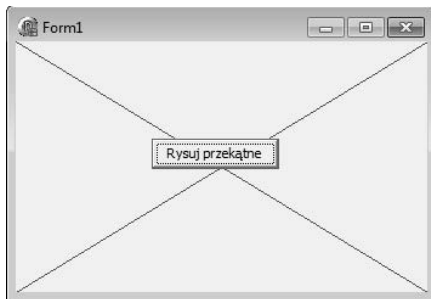
```
Canvas.Pixels[ClientWidth div 2, ClientHeight div 2]:=clRed;
```

Przykład 13.3.

Narysuj linie koloru czerwonego będące przekątnymi formularza — rysunek 13.2.

Rysunek 13.2.

*Formularz
z przekątnymi
pozostającymi
po zmianie
jego rozmiaru*

**Rozwiązanie**

Poniższe instrukcje wpisz np. w procedurze obsługi przycisku.

Pierwsza przekątna:

```
Canvas.Pen.Color:=clRed; //zmiana koloru pióra na czerwony
//przesunięcie kursora graficznego do punktu o współrzędnych (0,0)
Canvas.MoveTo(0,0);
//narysowanie linii od bieżącego położenia kursora graficznego do punktu z prawego
//dolnego wierzchołka
Canvas.LineTo(ClientWidth, ClientHeight);
```

Narysuj drugą przekątną.

Aby przekątne pozostały na formularzu podczas zmiany jego rozmiaru, należy wykonać dwa zdarzenia: `OnPaint` i `OnResize`. W procedurach obsługi tych zdarzeń powinny znaleźć się instrukcje jak w procedurach poniżej:

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Pen.Color:=clRed;
  Canvas.MoveTo(0,0);
  Canvas.LineTo(ClientWidth, ClientHeight);
  Canvas.MoveTo(ClientWidth,0);
  Canvas.LineTo(0, ClientHeight);
end;
```

i

```
procedure TForm1.FormResize(Sender: TObject);
begin
  Refresh; // przy zmianie rozmiaru okna
           // kasowane są poprzednie przekątne
end;
```

Przykład 13.4.

Wyświetl na formularzu punkty rozmieszczone losowo i o losowych kolorach.

Rozwiązanie

Wstaw przycisk i w procedurze obsługi zdarzenia `OnClick` wpisz odpowiednie instrukcje:

```
//Losowe punkty
procedure TForm1.Button2Click(Sender: TObject);
var i:integer;
begin
for i:=1 to 10000 do
    Canvas.Pixels[Random(ClientWidth), Random(ClientHeight)]:=
    RGB( Random(255),Random(255), Random (255 ) );
end;
```

Przykład 13.5.

Wyświetl na formularzu trzy różne prostokąty — ramkę, prostokąt wypełniony kolorem `Brush.Color`, prostokąt z zaokrąglonymi brzegami.

Rozwiązanie

W procedurze obsługi przycisku wpisz instrukcje jak poniżej:

```
procedure TForm1.Button3Click(Sender: TObject);
var
prost: TRect;
begin
    prost:= Rect(200,10,300,100);
    Canvas.Brush.Color := clBlack;
    //ramka
    Canvas.FrameRect(prost);
    Canvas.Brush.Color := clGreen;
    //prostokąt wypełniony
    Canvas.Rectangle(200,120,300,210);
    //prostokąt z zaokrąglonymi brzegami
    Canvas.RoundRect(200,230,300,320,20,20);
end;
```

Przykład 13.6.

Wyświetl na środku formularza napis `Zadania z Delphi` w kolorze niebieskim, o rozmiarze czcionki równym 36 pkt, bez tła — rysunek 13.3.

Rysunek 13.3.

Napis na środku formularza



Rozwiązanie

W procedurze wykorzystano funkcje zwracające szerokość i wysokość napisu oraz rozmiary formularza — i na tej podstawie obliczono współrzędne lewego górnego wierzchołka wyświetlanego napisu:

```
procedure TForm1.Button2Click(Sender: TObject);
var x,y:integer;
begin
  Canvas.Font.Name:='Arial';
  Canvas.Font.Color:=clBlue;
  Canvas.Font.Size:=24;
  Canvas.Brush.Style:=bsClear;
  x:=ClientWidth-Canvas.TextWidth('Zadania
    z Delphi');
  y:=ClientHeight-Canvas.TextHeight('Z');
  Canvas.TextOut(x div 2, y div 2,'Zadania z Delphi');
end;
```

Przykład 13.7.

Narysuj elipsę o maksymalnych wymiarach na formularzu.

Rozwiązanie

W procedurze obsługi przycisku wpisz instrukcję:

```
//elipsa wpisana w prostokąt o rozmiarach formularza
Canvas.Ellipse(0,0, ClientWidth, ClientHeight);
```

Przykład 13.8.

Narysuj na formularzu trójkąt o zielonym obwodzie i żółtym wypełnieniu.

Rozwiązanie

I sposób — z wykorzystaniem procedury PolyLine i FloodFill:

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  Canvas.Brush.Color:=clYellow;
  Canvas.Pen.Color:=clGreen;
  //rysowanie trójkąta
  Canvas.Polyline([Point(20,20),Point(200,20),Point(110,100),Point(20,20)]);
  Canvas.Floodfill(100,25,clgreen,fsborder); //procedura wypełnia obiekt narysowany
  //kolorem zielonym, wewnątrz którego znajduje się punkt o współrzędnych (100,25)
end;
```

II sposób — z wykorzystaniem procedury Polygon, rysującej wielokąt wypełniony bieżącym kolorem pędzla (Brush). Współrzędne ostatniego punktu nie muszą pokrywać się ze współrzędnymi punktu pierwszego wielokąta, ponieważ pierwszy punkt jest automatycznie łączony z ostatnim:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.Brush.Color:=clYellow;
```

```

Canvas.Pen.Color:=clGreen;
Canvas.Polygon([Point(20,20),Point(200,20),Point(110,100)]);
end;

```

Przykład 13.9.

Wyświetl na formularzu linie rysowane różnymi stylami.

Rozwiązanie

Wstaw przycisk TButton. W procedurze obsługi zdarzenia OnClick przycisku wpisz instrukcje jak w poniższej procedurze:

```

//style linii
procedure TForm1.Button1Click(Sender: TObject);
var x,y:integer;
begin
  x := 210;
  y := y+10;//y - zmienna globalna
  Canvas.MoveTo(x,y);
  x := Random(ClientWidth - 10);
  y := Random(ClientHeight - 10);
  Canvas.Pen.Color := RGB(Random(256),Random(256),Random(256));
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.LineTo(x+200, y);
end;

```

Przykład 13.10.

Wyświetl na formularzu prostokąt malowany różnymi stylami pędzla po każdym kliknięciu przycisku.

Rozwiązanie

Wstaw przycisk TButton. W procedurze obsługi zdarzenia OnClick przycisku wpisz instrukcje jak w poniższej procedurze:

```

//style pędzla
procedure TForm1.Button2Click(Sender: TObject);
begin
  Refresh; //kasuje poprzedni prostokąt
  Canvas.Brush.Color :=RGB(Random(256),Random(256),Random(256)); //kolorem pędzla
                                                                    //malowane są wzory
  case Random(7) of
    0: Canvas.Brush.Style := bsClear;
    1: Canvas.Brush.Style := bsSolid;
    2: Canvas.Brush.Style := bsBDiagonal;
    3: Canvas.Brush.Style := bsFDiagonal;
    4: Canvas.Brush.Style := bsCross;

```

```
5: Canvas.Brush.Style := bsDiagCross;  
6: Canvas.Brush.Style := bsHorizontal;  
7: Canvas.Brush.Style := bsVertical;  
end;  
Canvas.Rectangle(0,0, 200,100);  
end;
```

Rysowanie „trwałe” — zdarzenie OnPaint

Instrukcje zawierające metody obiektu Canvas można umieszczać w procedurach obsługi zdarzenia `OnClick` dla przycisków, dla formularza i innych komponentów. Można również korzystać z innych zdarzeń komponentów. Jednak tylko niektóre z nich umożliwiają tzw. „trwałe” rysowanie, czyli rysowanie odnawiane po każdej zmianie, np. po zmianie rozmiaru okna i przykryciu w ten sposób części obiektów graficznych. Dla okna formularza korzysta się w tym celu ze zdarzenia `OnPaint`. Dla innych komponentów podobne zdarzenia mają inne nazwy. Przedstawiono je w tabeli 13.2.

Tabela 13.2. Zdarzenia umożliwiające rysowanie „trwałe”

Zdarzenie	Znaczenie
<code>OnPaint</code>	Zdarzenie dla formularza generowane każdorazowo, gdy zawartość okna formularza wymaga odświeżenia. Sytuacja taka ma miejsce przy tworzeniu okna formularza, a także wtedy, gdy np. jedno okno zostanie przesłonięte innym oknem lub gdy następuje zmiana jego rozmiaru.
<code>PaintBoxPaint</code>	Odpowiednik zdarzenia <code>OnPaint</code> dla komponentu <code>PaintBox</code> .
<code>OnDrawCell</code>	Zdarzenie występujące dla komponentu typu <code>TDrawGrid</code> i <code>TStringGrid</code> — umożliwia „trwałe” rysowanie obiektów i wyświetlanie tekstu w komórkach.
<code>OnAfterDraw</code>	Zdarzenie dla komponentu typu <code>TChart</code> , odpowiednik zdarzenia <code>OnPaint</code> .

Rysowanie po komponentcie typu `TImage` nie wymaga odnawiania. Jest „trwałe”.

Przykład 13.11.

Narysuj na formularzu prostokąt koloru czerwonego, tak aby nie kasował się po przykryciu okna formularza innym oknem. Prostokąt powinien rysować się po kliknięciu przycisku i kasować po kliknięciu drugiego przycisku — rysunek 13.4.

Rozwiązanie

Wstaw dwa przyciski `TButton`.

Gdyby instrukcję rysującą prostokąt umieścić w procedurze obsługi zdarzenia `OnPaint`, to prostokąt byłby na formularzu bezpośrednio po uruchomieniu programu. Dlatego procedurę obsługi tego zdarzenia z nową instrukcją należy wywołać za pomocą przycisku.

Rysunek 13.4.

*Rysowanie
i kasowanie
prostokąta
na formularzu*



Szkielet procedury `FormPaint` można uzyskać, klikając dwukrotnie w oknie Inspektora Obiektów z prawej strony zdarzenia `OnPaint`. Później trzeba jednak wykasować w Inspektorze Obiektów tę nazwę, aby instrukcje w procedurze obsługi zdarzenia `OnPaint` nie wykonywały się bezpośrednio po uruchomieniu programu.

W przykładzie pokazano, jak wykonać takie zadanie.

```
//procedura obsługi zdarzenia OnPaint dla formularza rysuje prostokąt.
// wywoływana programowo może mieć inną nazwę
procedure TForm1.FormPaint(Sender: TObject);
begin
    Canvas.Rectangle(100,100,ClientWidth-100,ClientHeight-100);
end;

//procedura rysuje prostokąt koloru czerwonego i przypisuje procedurze obsługi
//zdarzenia OnPaint procedurę FormPaint
procedure TForm1.Button1Click(Sender: TObject);
begin
    Canvas.Pen.Color:=clRed;
    Canvas.Rectangle(100,100,ClientWidth-100,ClientHeight-100);
    OnPaint:=FormPaint;//przypisanie procedury obsługi zdarzenia procedury rysującej
                        //prostokąt
end;

// odłączenie procedury FormPaint od zdarzenia OnPaint – wykasowanie prostokąta
procedure TForm1.Button2Click(Sender: TObject);
begin
    OnPaint:=nil; //ta instrukcja spowoduje, że rysunek prostokąta nie będzie odnawiany
    Refresh;      //procedura ta kasuje prostokąt
end;
```

Przykład 13.12.

Wypełnij tło formularza bitmapą, np. *wzorek.bmp*.

Rozwiązanie

W procedurze obsługi zdarzenia `OnPaint` dla formularza wpisz instrukcje jak w procedurze poniżej.

Zadeklaruj zmienną globalną lub pole klasy TForm1 (w sekcji public):

```
var Bitmap: TBitmap;

procedure TForm1.FormPaint(Sender: TObject);
var x, y: Integer;
begin
  y := 0;
  while y < Height do
  begin
    x := 0;
    while x < Width do
    begin
      Canvas.Draw(x, y, Bitmap);
      x := x + Bitmap.Width;
    end;
    y := y + Bitmap.Height;
  end;
end;
```

W metodzie FormCreate (po dwukrotnym kliknięciu w formularz) dopisz instrukcje:

```
Bitmap:=TBitmap.Create;
Bitmap.LoadFromFile(wzorek.bmp');
```

Obraz, który jest powielany jak tło formularza, może mieć też inny format, np. *.JPG*. Należy wtedy zamienić format *.JPG* na *.BMP* (bitmapa ma właściwość Canvas). Wtedy treść metody FormCreate powinna być następująca:

```
procedure TForm1.FormCreate(Sender: TObject);
var obraz:TImage; //trzeba zadeklarować moduł ExtCtrls
begin
  Bitmap:=TBitmap.Create;

  obraz:=TImage.Create(Self);
  obraz.Picture.LoadFromFile('wzorek.jpg');
  //zamiana formatu obrazu JPG na TBitmap
  Bitmap.Assign(obraz.Picture.Graphic);
end;
```

Aby program prawidłowo działał, należy jeszcze w sekcji Uses zadeklarować moduł *JPEG* zawierający definicję klasy *TJPEGImage* oraz *ExtCtrls* zawierający definicję klasy *TImage*. Przykład 13.12A na płycie prezentuje sposób uzyskania tła z obrazu typu *.JPG*.

Rysowanie myszą po formularzu

Przykład 13.13.

Napisz program umożliwiający rysowanie po formularzu po naciśnięciu lewego przycisku myszy.

Rozwiązanie

Wykorzystano zdarzenie `OnMouseDown` występujące po naciśnięciu przycisku myszy na komponencie i zdarzenie `OnMouseMove` występujące przy przesuwaniu kursora myszy nad komponentem.

Wpisz instrukcje jak w poniższych procedurach obsługi zdarzeń `OnMouseDown` i `OnMouseMove`.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift:
  TShiftState; X, Y: Integer);
begin
  Canvas.MoveTo(x,y);
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then //czy lewy przycisk myszy wciśnięty
    Canvas.LineTo(x,y);
end;
```

Rysowanie myszą po komponencie TImage

Przykład 13.14.

Napisz program umożliwiający wyświetlenie w losowych miejscach białych pikseli na obrazie po kliknięciu przycisku i białych kółek, również losowo, po kliknięciu drugiego przycisku.

Rozwiązanie

Na formularzu wstaw komponent `TScrollBar` i wewnątrz niego komponent `TImage` z zakładki *Additional* tak, aby lewe górne wierzchołki obu komponentów pokrywały się. Dla komponentu `TImage` ustaw właściwość `Autosize` na `true` i za pomocą właściwości `Picture` załaduj do niego obraz typu *.BMP* (nie *.JPG*). Dodaj do formularza jeszcze dwa przyciski `TButton`.

W procedurze obsługi zdarzenia `OnClick` dla przycisku `Button1` wpisz instrukcje jak w procedurze poniżej.

```
procedure TForm1.Button1Click(Sender: TObject);
var i: integer;
begin
  for i:=1 to 100 do
    Image1.Canvas.Pixels[Random(Image1.Width),
      Random(Image1.Height)]:=clWhite;
end;
```

Dla przycisku `Button2` — rysowanie losowych kółek:

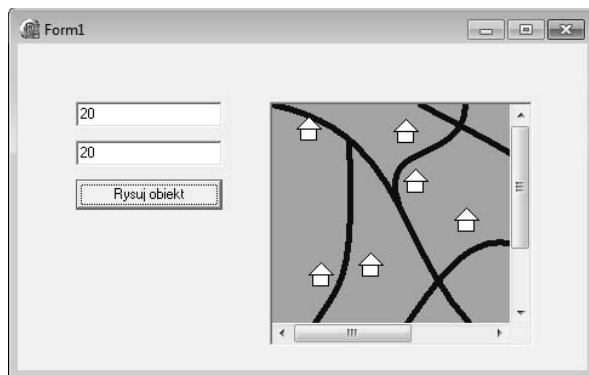
```
procedure TForm1.Button2Click(Sender: TObject);
var i,k,t: integer;
begin
  for i:=1 to 100 do
  begin
    k:=Random(Image1.Width);
    t:=Random(Image1.Height);
    Image1.Canvas.Ellipse(k,t,k+10,t+10);
  end;
end;
```

To samo można wykonać również dla obrazów typu *.JPG* oraz od wersji Delphi 2009 dla obrazów *.GIF* i *.PNG*. Należy jednak wcześniej wykonać konwersję do formatu *.BMP*, ponieważ obiekt tego typu ma właściwość *Canvas*. Sposób konwersji do typu *.BMP* pokazano w przykładach 14.13 oraz 14.7 i 14.8.

Przykład 13.15.

Napisz program umożliwiający po kliknięciu przycisku wyświetlenie obrazu przedstawiającego np. mapę obiektów (domków). Obiekty rysowane są na mapie po podaniu współrzędnych i kliknięciu przycisku lub po kliknięciu na mapie — rysunek 13.5.

Rysunek 13.5.
*Rysowanie obiektów
na obrazie*



Rozwiązanie

Na formularzu wstaw przycisk *TButton*, dwa komponenty *TEdit*. Komponent *TScrollBar* i w jego wnętrzu *TImage*. Do komponentu *TImage* załaduj obraz.

Obiekt (domek) zostanie narysowany za pomocą następującej procedury:

```
procedure TForm1.Rysuj_domek(x,y:integer);
begin
  with Image1.Canvas do
  begin
    //daszek
    Polygon([Point(x,y),Point(x+10,y-10),Point(x+20,y)]);
    //reszta
    Rectangle(x+3,y,x+17,y+10);
  end;
end;
```

Parametry x,y określają współrzędne lewego wierzchołka trójkąta (daszku).

Nagłówek procedury należy zadeklarować w sekcji `public` definicji klasy `TForm1` w sposób jak poniżej:

```
procedure Rysuj_domek (x,y:integer);
```

A treść procedury `Rysuj_domek` wpisać w sekcji `implementation`.

W procedurze obsługi zdarzenia `OnClick` dla przycisku wpisz instrukcje jak w procedurze poniżej.

```
procedure TForm1.Button1Click(Sender: TObject);
var x,y: integer;
begin
  x:=StrToIntDef(Edit1.Text,10);
  y:=StrToIntDef(Edit2.Text,10);
  //rysowanie obiektu - domek
  Rysuj_domek(x,y);
end;
```

Funkcja `StrToIntDef` umożliwia zabezpieczenie programu przed błędami. Jeśli w okienku *Edit* nie będzie liczby całkowitej, to nie wystąpi błąd, tylko domyślnie zmiennej x lub y przypisana zostanie wartość 10.

Do rysowania obiektów na obrazie po kliknięciu przycisku można wykorzystać zdarzenie `OnMouseDown` dla komponentu `TImage`. Procedura obsługi tego zdarzenia zwraca współrzędne punktu kliknięcia liczone względem lewego górnego wierzchołka (0, 0).

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Rysuj_domek(X,Y);
end;
```

Przykłady animacji w Delphi

W programowaniu stosuje się różne techniki animacji. Jednym z prostszych sposobów jest rysowanie obiektu, następnie kasowanie i ponowne rysowanie w innym miejscu. Wadą tego rozwiązania jest trudność w uzyskaniu płynności ruchu obiektów.

Inna metoda polega na zastosowaniu dwóch obszarów, na których rysujemy. W danej chwili widoczny jest tylko jeden z nich. Drugi jest wówczas modyfikowany i wyświetlany w miejsce pierwszego dopiero po zakończeniu operacji.

W zadaniach przykładowych zastosowano pierwszy sposób animacji. Udało się uzyskać odpowiednią płynność ruchu obiektów, dlatego nie wykorzystano sposobu z użyciem dwóch obszarów rysowania.

Przykład 13.16.

Wykonaj następującą animację: kółko o średnicy 30 pkt przesuwa się od lewego do prawego brzegu formularza i z powrotem.

Rozwiązanie

W procedurze obsługi przzerwania od Timera wpisz:

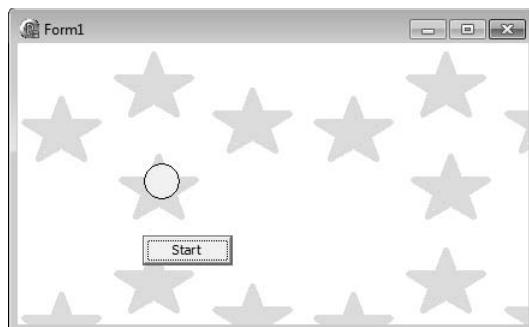
```
{ $J+ }  
procedure TForm1.Timer1Timer(Sender: TObject);  
const x1:integer=0;  
      y1:integer=100;  
      krok:integer=5;  
begin  
    //kasowanie obiektu  
    Canvas.Brush.color:=Color; //kolor formularza  
    Canvas.Pen.color:=Color;   //kolor pióra  
    Form1.Canvas.Ellipse(x1,y1,x1+30,y1+30);  
    //rysowanie kółka kolorem czerwonym  
    Canvas.Brush.color:=clRed;  
    x1:=x1+krok;  
    Canvas.Ellipse(x1,y1,x1+30,y1+30);  
    if x1+30>= Clientwidth then krok:=-krok;  
    if x1<=0 then krok:=-krok;  
end;
```

Dyrektywa { \$J+ } przed treścią procedury włącza opcję kompilatora umożliwiającą zmianę wartości stałych typowanych (ang. *Assignable typed constants*). Opcja ta powinna być standardowo włączona, ale jeśli nie mamy pewności, lepiej dodać dyrektywę { \$J+ }.

Przykład 13.17.

Wykonaj animację tak jak w zadaniu poprzednim, gdy formularz jest wypełniony wzorem — rysunek 13.6.

Rysunek 13.6.
Animacja z tłem



Rozwiązanie

Na formularzu umieść przycisk TButton i komponent TTimer. Właściwość Interval ustaw na 200 ms, a właściwość Enabled na false. Treść procedur obsługi przycisku i przzerwania od Timera przedstawiono poniżej.

Zadeklaruj zmienną globalną:

```

var Bitmap, Bitmap1: TBitmap;
//załadowanie obrazu .BMP do zmiennej Bitmap
procedure TForm1.FormCreate(Sender: TObject);
begin
//W procedurze FormCreate należy przeczytać bitmapę z pliku.
    Bitmap:=TBitmap.Create;
    Bitmap.LoadFromFile('c:\windows\kawa.bmp');
end;

// procedura pobiera prostokątny fragment formularza i uruchamia Timer
procedure TForm1.Button1Click(Sender: TObject);
var x,y:integer;
begin
    // utworzenie obiektu Bitmap1
    Bitmap1 := TBitmap.Create;
    Bitmap1.Width:=ClientWidth;
    Bitmap1.Height:=30;
//pobranie prostokątnego wycinka formularza - obszaru, po którym będzie się
//poruszało kółko
    for x:=0 to ClientWidth-1 do
        for y:=0 to 29 do
            Bitmap1.Canvas.Pixels[x,y]:=Form1.Canvas.Pixels[x,y+100];
        Timer2.Enabled:=true; //w Inspektorze Obiektów zablokuj Timer2
    end;

{$J+} //procedura obsługi przerwania od Timera - rysowanie i kasowanie obiektu
    //co 200 ms
    procedure TForm1.Timer2Timer(Sender: TObject);
    const x1:integer=0;
           y1:integer=100;
           krok:integer=5;
    var x,y:integer;
    begin
        //jeśli zwiększymy rozmiar formularza, to trzeba w procedurze obsługi zdarzenia
        //OnResize jeszcze raz pobrać bitmapę
        Canvas.Draw(0,y1,Bitmap1); //wyświetlenie wcześniej pobranego paska formularza,
                                   //kasowanie obiektu

        //rysowanie kółka
        Canvas.Ellipse(x1,y1,x1+30,y1+30);
        x1:=x1+krok;
        if x1+29>=Clientwidth then krok:=-krok;
        if x1<=0 then krok:=-krok;
    end;

//wypełnianie formularza bitmapą
procedure TForm1.FormPaint(Sender: TObject);
var x, y: Integer;
begin
    y := 0;
    while y < Height do
    begin
        x := 0;
        while x < Width do
        begin
            Canvas.Draw(x, y, Bitmap);

```

```

        x := x + Bitmap.Width;
    end;
    y := y + Bitmap.Height;
end;

// procedura FormDestroy zwalnia pamięć
// zajmowaną przez bitmapy
procedure TForm1.FormDestroy(Sender: TObject);
begin
    Bitmap.Free;
    Bitmap1.Free;
end;

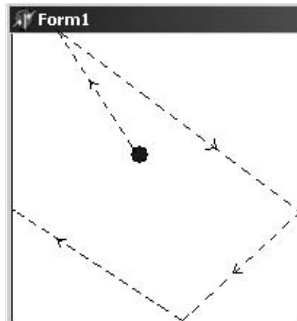
```

Przykład 13.18.

Wykonaj animację polegającą na przemieszczaniu się kulki w losowych kierunkach w prostokątnym obszarze o wymiarach (0, 0, 200, 200). Wykorzystaj komponent TPaintBox z zakładki *System* — rysunek 13.7.

Rysunek 13.7.

*Animacja
niebieskiej kulki*



Rozwiązanie

Na formularzu umieść komponent TPaintBox i TTimer. Komponent TPaintBox jest stosowany do wyświetlania (kreślenia) grafiki, która ma być ograniczona do obszaru prostokątnego. Korzystając z komponentu TPaintBox, programista nie musi kontrolować, czy obszar ten nie został przekroczony — jeśli narysowany obiekt nie mieści się wewnątrz komponentu TPaintBox, to zostaje obcięty. Dodatkowo zawarty w nim rysunek możemy przesuwając po formularzu, zmieniając właściwości Left i Top tego komponentu. Procedura przedstawiona poniżej działa poprawnie z komponentem TPaintBox i bez niego — wtedy kulka przesuwa się po formularzu.

W zadaniu można również dodać przycisk, który będzie włączał zegar (animację) po wpisaniu w procedurze obsługi instrukcji `Timer1.Enabled:=true;` (wcześniej należy zegar zablokować w okienku Inspektora Obiektów — `Enabled=true`).

```

{$J+}
procedure TForm1.Timer1Timer(Sender: TObject);
const
    x: integer=6;
    y: integer=6;
    krokx: integer=6;
    krocky: integer=6;

```

```

begin
  with PaintBox1.Canvas do
    begin
      //czyszczenie prostokąta
      Brush.Color:=clWhite;
      Rectangle(0,0,200,200);
      //obliczenie współrzędnych
      x:=x+kroxx;
      y:=y+kroky;
      //rysowanie koła w kwadracie o boku
      // równym 6 pikseli
      Brush.Color:=clBlue;
      Ellipse(x-6, y-6, x+6, y+6);
      if (x>Paintbox1.Width-6) then
        begin
          kroxx:=6+Random(5);
          kroxx:=-kroxx;
        end;
      if (y>Paintbox1.Height-6) then
        begin
          kroky:=6+Random(5);
          kroky:=-kroky;
        end;
      if (x<=6) then kroxx:=-kroxx;
      if (y<=6) then kroky:=-kroky;
    end; //with
end;

```

Korzystając z Inspektora Obiektów, właściwości `Interval` komponentu `Timer1` przypisz wartość 50.

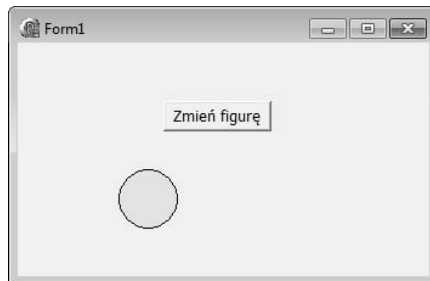
Przykład 13.19.

Umieść na formularzu komponent typu `TButton` i `TShape`. Zadaniem przycisku jest wyświetlanie po każdym kliknięciu na przemian kółka lub prostokąta.

Po naciśnięciu klawiszy strzałek komponent `Shape` przesuwa się zgodnie z kierunkiem strzałki — rysunek 13.8.

Rysunek 13.8.

*Przesuwanie koła
za pomocą
klawiszy strzałek*



Uwaga

Aby klawisze strzałek nie były przechwytywane przez komponent `Button1`, należy ustawić dla niego właściwość `TabStop` na `false`.

Rozwiązanie

Wstaw komponenty TButton i TShape. Dla komponentu TShape ustaw właściwość Shape na stCircle i właściwość Brush\Color na clYellow. W procedurze obsługi kliknięcia przycisku wpisz instrukcje jak poniżej:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Shape1.Shape=stCircle then Shape1.Shape:=stRectangle
    else Shape1.Shape:=stCircle;
    Form1.ActiveControl:=nil;
end;
```

W celu sprawdzenia klawiszy strzałek wykorzystaj zdarzenie OnKeyDown dla formularza. Treść procedury obsługi tego zdarzenia przedstawiono poniżej:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
    case Key of
        vk_Right: Shape1.Left:=Shape1.Left+10; //strzałka w prawo
        vk_Left:   Shape1.Left:=Shape1.Left-10; //strzałka w lewo
        vk_Up:     Shape1.Top:=Shape1.Top-10; //strzałka w górę
        vk_Down:  Shape1.Top:=Shape1.Top+10; //strzałka w dół
    end;
end;
```

W przykładzie wykorzystano kody klawiszy wirtualnych. Parę przykładowych kodów podano w tabeli 13.3.

Tabela 13.3. Wybrane klawisze i ich kody

Kod klawisza	Klawisz
vk_F1, vk_F2, ... vkF24	F1, F2, ..., F24
vk_LBUTTONDOWN, vk_RBUTTONDOWN	Lewy przycisk myszy, prawy przycisk myszy
vk_INSERT	Ins
vk_DELETE	Del
vk_ESCAPE	Esc
Vk_BACK	Backspace
Vk_TAB	Tab

Wszystkie kody można znaleźć w Systemie Pomocy Delphi, wystarczy umieścić kursor na słowie vk_Left i nacisnąć F1 lub wyszukać hasło Virtual Key Codes.

Środowisko Delphi służy do szybkiego tworzenia aplikacji działających w systemie Windows. Zawiera bogate biblioteki komponentów, mechanizmy Plug and Play oraz Code Insight, a także palety komponentów i narzędzia ułatwiające ich wyszukiwanie. Dzięki Delphi IDE można w prosty sposób zaprojektować interfejs użytkownika nowej aplikacji, określić jej wygląd oraz sposób działania w oparciu o istniejące kontrolki i biblioteki klas, w dużym stopniu zdejść się na automatyczne generowanie kodu.

Jeśli chcesz poznać, dogłębnie zrozumieć i wykorzystać do swoich celów sposób działania Delphi 2010, powinieneś koniecznie sięgnąć po książkę „Aplikacje w Delphi. Przykłady”. Znajdziesz tu wszelkie informacje na temat obsługi samego środowiska, podstawowych składników każdej tworzonej aplikacji, wprowadzania i formatowania danych, list, tabel, grupowania i projektowania nowych komponentów, wykorzystania technologii OLE do zapisu i modyfikacji danych w formatach .doc i .xls, możliwości graficznej prezentacji danych, ich drukowania i współpracy Twojej aplikacji z multimediami. Wszystko to oraz wiele innych zagadnień pokazano tu na praktycznych, konkretnych przykładach, ułatwiających zrozumienie i gotowych do zastosowania w Twoich własnych projektach.

- Podstawowe składniki aplikacji, menu główne i podręczne, pasek narzędzi
- Wprowadzanie danych, formatowanie i wyświetlanie na ekranie
- Okienka komunikatów i okienka dialogowe z karty Dialogs
- Listy wyboru i prosty edytor
- Grupowanie i dynamiczne tworzenie komponentów
- Komponenty do wyboru daty i czasu, odmierzanie czasu
- Zakładki TTabControl i TPageControl
- Grafika w Delphi i wyświetlanie obrazów
- Tabelaryzacja danych i ich graficzna prezentacja
- Współpraca programu z plikami dyskowymi
- Drukowanie w Delphi i programy z wieloma oknami
- Posługiwanie się wieloma komponentami tego samego typu
- Definiowanie nowych klas komponentów i wykorzystanie mechanizmu OLE
- Podstawowe operacje na bazach danych
- Delphi i multimedia

Odkryj fantastyczne możliwości Delphi!

(w katalogu: 3748)



Księgarnia Internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:
@ <http://helion.pl/promocje>
Kupuj najtaniej! cytując:
@ <http://helion.pl/taimotory>
Zamów informacje o nowościach:
@ <http://helion.pl/nowosci>

Helion Sp. z o.o.
ul. Kalichowska 1c, 44-100 Gliwice
tel.: 32 230 56 63
e-mail: helion@helion.pl
<http://helion.pl>

helion.pl
Księgarnia
Internetowa

Cena: 59,00 zł

ISBN 978-83-246-2651-3



Informatyka w najlepszym wydaniu