

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

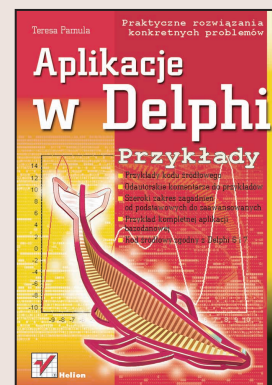
ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# Aplikacje w Delphi. Przykłady

Autor: Teresa Pamula  
ISBN: 83-7361-212-2  
Format: B5, stron: 260



Delphi jest narzędziem do programowania obiektowego w systemie Windows, opartym na języku Object Pascal, którego składnia jest zgodna ze składnią języka Turbo Pascal. Środowisko Delphi IDE (ang. Integrated Development Environment) umożliwia zaprojektowanie w prosty sposób interfejsu użytkownika, generując część kodu programu automatycznie.

Istnieje wiele książek poświęconych Delphi, ta jednak różni się od pozostałych. Przedstawia ona przykłady kodu źródłowego pokazującego, jak w praktyce używać Delphi. Jest więc uzupełnieniem książek omawiających sam język Object Pascal i prezentuje wykorzystanie wiedzy teoretycznej w praktyce programistycznej. Przykładowym tekstom programów towarzyszą komentarze autora wyjaśniające dlaczego użyto takiego, a nie innego rozwiązania.

Zagadnienia omówione w tej książce obejmują szeroki zakres tematyczny:

- Projektowanie interfejsu użytkownika, tworzenie menu
- Formatowanie tekstu i liczb
- Tworzenie okien dialogowych, list wyboru
- Pisanie prostego edytora tekstu
- Prezentacja danych w tabelach i za pomocą wykresów
- Pisanie aplikacji graficznych
- Korzystanie z zasobów dyskowych i drukarek
- Posługiwanie się wieloma komponentami tego samego typu
- Dynamiczne tworzenie komponentów, tworzenie nowych klas komponentów
- Pisanie aplikacji opartych na bazach danych

Nie odkrywaj Ameryki: sprawdź najpierw, jak inni rozwiązali problemy, które napotkałeś programując w Delphi. Ta książka zaoszczędzi Twój czas i pozwoli pisać programy znacznie efektywniej a przy okazji sprawi, że poznasz wiele nowych możliwości wspaniałego narzędzia jakim jest Delphi.

Wydawnictwo Helion  
ul. Chopina 6  
44-100 Gliwice  
tel. (32)230-98-63  
e-mail: helion@helion.pl



# Spis treści

<b>Wprowadzenie .....</b>	<b>7</b>
<b>Rozdział 1. Projektowanie aplikacji w Delphi .....</b>	<b>9</b>
Środowisko zintegrowane — Delphi IDE .....	9
Elementy projektu aplikacji .....	10
Standardowe właściwości komponentów .....	13
Standardowe zdarzenia .....	14
<b>Rozdział 2. Podstawowe składniki aplikacji .....</b>	<b>17</b>
Okno aplikacji .....	17
Ikona aplikacji .....	22
Wyświetlanie napisów .....	22
Rodzaje przycisków, podobieństwa i różnice .....	27
Etykiety i przyciski .....	30
<b>Rozdział 3. Menu główne i podręczne .....</b>	<b>35</b>
Wielopoziomowe menu główne .....	35
Przyporządkowanie poleceń opcjom menu .....	37
Menu podręczne .....	40
„Polskie litery” w nazwach poleceń menu .....	41
<b>Rozdział 4. Wprowadzanie danych, formatowanie i wyświetlanie na ekranie .....</b>	<b>43</b>
Liczby — funkcje konwersji i formatowanie liczb. Przecinek czy kropka? .....	44
Daty — funkcje konwersji i formatowanie daty i czasu .....	46
Systemowe separatory liczb i daty .....	49
Wprowadzanie danych za pomocą okienek edycyjnych TEdit .....	49
Wprowadzanie danych za pomocą okienek InputBox i InputQuery .....	54
Sposoby zabezpieczenia programu przed błędami przy wprowadzaniu danych .....	55
Obliczenia. Wybrane funkcje modułu Math .....	61
<b>Rozdział 5. Okienka komunikatów .....</b>	<b>65</b>
Wyświetlanie komunikatów z napisami stałymi w języku systemowym — MessageBox .....	66
Wyświetlanie komunikatów za pomocą funkcji ShowMessage, MessageDlg, MessageDlgPos .....	67

<b>Rozdział 6. Okienka dialogowe z karty Dialogs .....</b>	<b>73</b>
<b>Rozdział 7. Listy wyboru — TListBox i TComboBox.....</b>	<b>79</b>
Dodawanie elementów do listy .....	81
Wybieranie elementów z listy .....	82
Sposoby wyświetlania elementów listy.....	84
Blokowanie edycji dla listy TComboBox .....	86
Czytanie i zapisywanie zawartości listy do pliku dyskowego .....	87
<b>Rozdział 8. Prosty edytor — komponent TMemo.....</b>	<b>89</b>
Kopiowanie, wycinanie i wklejanie tekstu.....	90
Czytanie i zapisywanie tekstu do pliku .....	91
Wyświetlanie informacji o położeniu kursora .....	92
Automatyczne kasowanie linii nie zawierających liczb lub wybranych znaków .....	93
<b>Rozdział 9. Grupowanie komponentów.....</b>	<b>95</b>
Pola opcji i pola wyboru.....	95
Komponenty grupujące .....	96
Ramka TBevel.....	100
<b>Rozdział 10. Tabelaryzacja danych — komponent TStringGrid .....</b>	<b>101</b>
Ustalanie podstawowych parametrów tabeli.....	104
Wypełnianie tabeli danymi.....	106
Wybieranie komórek tabeli .....	108
Filtrowanie wprowadzanych danych.....	110
Niestandardowe przejście do kolejnej komórki — klawisz Enter .....	112
Zmiana koloru i wyrównania tekstu w wybranych komórkach .....	114
Wyświetlanie tekstu w komórce w dwóch wierszach .....	117
Totolotek .....	119
Tabela i lista .....	121
<b>Rozdział 11. Graficzna prezentacja danych — komponent TChart.....</b>	<b>123</b>
Rysowanie wykresów z wykorzystaniem komponentu TChart .....	123
Opis wybranych właściwości, metod i zdarzeń komponentów TChart i TChartSeries ..	125
Wykresy kołowe.....	128
Wykresy kolumnowe.....	131
Wykresy funkcji matematycznych .....	134
Formatowanie i skalowanie wykresów .....	138
Posługiwanie się wieloma wykresami.....	142
<b>Rozdział 12. Odmierzanie czasu — komponent TTimer.....</b>	<b>147</b>
<b>Rozdział 13. Grafika w Delphi — korzystanie z metod obiektu TCanvas.....</b>	<b>149</b>
Wyświetlanie prostych figur geometrycznych i tekstu .....	150
Rysowanie „trwałe” — zdarzenie OnPaint .....	156
Przykłady animacji w Delphi .....	158
<b>Rozdział 14. Wyświetlanie obrazów — komponent TImage.....</b>	<b>163</b>
Rysowanie po obrazie .....	165
Binaryzacja obrazu.....	166
<b>Rozdział 15. Współpraca programu z plikami dyskowymi .....</b>	<b>169</b>
Wybór foldera plików .....	169
Wyszukiwanie plików .....	171
Zapisywanie danych z okienek TEdit i tabeli do pliku tekstowego.....	174
Czytanie danych z pliku tekstowego.....	175
Zapisywanie i odczytywanie danych z tabeli do pliku *.csv .....	177

---

<b>Rozdział 16. Drukowanie w Delphi .....</b>	<b>179</b>
Drukowanie napisów i tekstu z okienek edycyjnych .....	180
Drukowanie tabeli i wykresu .....	181
Drukowanie obrazu .....	184
<b>Rozdział 17. Programy z wieloma oknami .....</b>	<b>187</b>
Wymiana danych i metod między modułami .....	187
Program z hasłem .....	190
Wyświetlanie tytułu programu .....	192
<b>Rozdział 18. Posługiwanie się wieloma komponentami tego samego typu.</b>	
<b>Operatory Is i As .....</b>	<b>195</b>
Wprowadzanie i kasowanie danych dla kilku okienek edycyjnych .....	196
Przypisywanie grupie komponentów tej samej procedury obsługi zdarzenia .....	197
Wyświetlanie informacji o numerach kontroltek, ich nazwach i klasach .....	200
<b>Rozdział 19. Tablice dynamiczne .....</b>	<b>203</b>
<b>Rozdział 20. Dynamiczne tworzenie komponentów .....</b>	<b>207</b>
Wyświetlanie kontroltek i przypisywanie zdarzeniom procedur obsługi .....	208
Przykłady dynamicznego tworzenia wykresów .....	212
Tworzenie menu w czasie działania programu .....	215
<b>Rozdział 21. Definiowanie nowych klas komponentów .....</b>	<b>219</b>
Klasa tabel z wyrównaniem zawartości komórek do prawej strony .....	219
Klasa okienek z właściwością Alignment .....	221
Instalowanie nowych komponentów na palecie komponentów .....	223
Nowy komponent do ankiety .....	227
<b>Rozdział 22. Podstawowe operacje na bazach danych .....</b>	<b>231</b>
Przeglądanie istniejących baz danych w formacie .dbf .....	233
Tworzenie własnej bazy danych .....	235
Modyfikowanie bazy .....	237
Filtrowanie rekordów bazy danych .....	238
Wyszukiwanie rekordów .....	240
Sortowanie .....	241
Rysowanie wykresów na podstawie danych z bazy .....	242
Obliczanie średniej ze wszystkich wartości danego pola .....	243
Biblioteka — przykład relacyjnej bazy danych .....	244
<b>Literatura .....</b>	<b>249</b>
<b>Skorowidz .....</b>	<b>251</b>

## Rozdział 13.

# Grafika w Delphi — korzystanie z metod obiektu TCanvas

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

Właściwość Canvas zawiera metody, które umożliwiają 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.

Wybrane właściwości obiektu TCanvas:

Brush — określa wzór lub kolor wypełnienia figur (tzw. pędzel);

Font — krój czcionki dla wyświetlanych napisów;

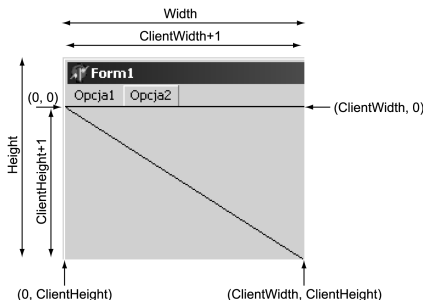
Pen — określa cechy kreślonych linii: grubość, styl, kolor (tzw. pióro);

PenPos — określa współrzędne kursora graficznego.

Podstawowymi parametrami większości procedur i funkcji graficznych są współrzędne punktu na komponentcie, 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 tego rozdziału.

**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
<code>Kolor:=Canvas.Pixels[x,y]</code>	Za pomocą funkcji <code>Pixels</code> można odczytać kolor piksela w miejscu o współrzędnych $(x, y)$ — zmienna <code>Kolor</code> jest typu <code>TColor</code> .
<code>Canvas.Pixels[10,20]:=c1Red</code>	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 $(0, 0)$ .
<code>MoveTo(x,y: integer)</code>	Przenosi kursor graficzny do punktu o współrzędnych $x, y$ .
<code>LineTo(x,y: integer)</code>	Rysuje linię od bieżącej pozycji kursora graficznego do punktu o współrzędnych $x, y$ .
<code>Rectangle(x1, y1, x2, y2: Integer)</code>	Procedura rysuje prostokąt wypełniony standardowym kolorem pędzla ( <code>Canvas.Brush.Color</code> ).
<code>Ellipse(x1, y1, x2, y2: Integer)</code>	Procedura rysuje elipsę (lub koło) — parametrami są współrzędne dwóch przeciwległych wierzchołków prostokąta (kwadratu), w który elipsa jest wpisana.
<code>Polyline(Points: array of TPoint)</code>	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 <i>polyline</i> ).

Tabela 13.1. Wybrane metody obiektu TCanvas (ciąg dalszy)

Metoda	Znaczenie
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 niemieszczone 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. 'Ała ma kota', lub zmienna zawierająca łańcuch znaków, np. a:='Ała 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.
TextHeight(const Text: string): Integer	Funkcja zwraca wysokość tekstu w pikselach.
TextOut(X, Y: Integer; const Text: string)	Procedura wyświetla napis na komponencie posiadającym właściwość TCanvas.
TextRect(Rect: TRect; X, Y: Integer; const Text: string)	Procedura wyświetla napis w prostokącie, którego współrzędne są podane w postaci typu TRect (pierwszy parametr). Procedura była wykorzystywana przy formatowaniu komórek tabeli.
TextWidth(const Text: string): Integer	Funkcja zwraca szerokość tekstu w pikselach.

Oprócz wymienionych metod zdefiniowane są metody, 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 posiada 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:

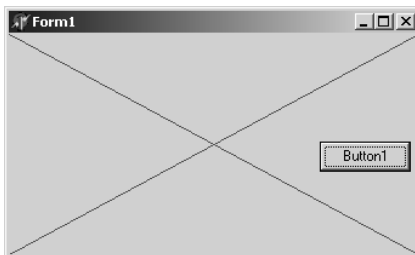
```
Canvas.Pixels[ClientWidth div 2, ClientHeight div 2]:=clRed;
```

### 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
//dołnego wierzchołka
Canvas.Lineto(ClientWidth, ClientHeight);
```

Narysuj drugą przekątną.

Aby przekątne pozostały na formularzu podczas zmiany jego rozmiaru, należy wykorzystać 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 pt, 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

```
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;
```

### 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(Seender: TObject);  
var x,y:integer;  
begin  
  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, 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.

### 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

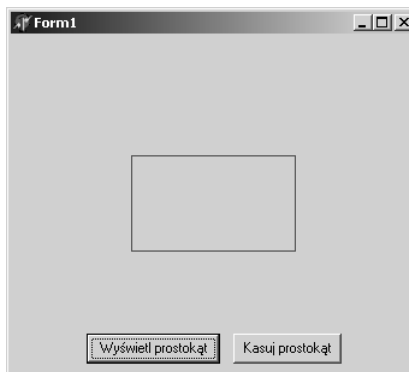
Wstaw dwa przyciski `TButton`.

Tabela 13.2. Zdarzenia umożliwiające rysowanie „trwale”

Zdarzenie	Znaczenie
OnPaint	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.
PaintBoxPaint	Odpowiednik zdarzenia OnPaint dla komponentu PaintBox.
OnDrawCell	Zdarzenie występujące dla komponentu typu TDrawGrid i TStringGrid — umożliwia „trwale” rysowanie obiektów i wyświetlanie tekstu w komórkach.
OnAfterDraw	Zdarzenie dla komponentu typu TChart, odpowiednik zdarzenia OnPaint.

**Rysunek 13.4.**

Rysowanie  
i kasowanie  
prostokąta  
na formularzu



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.

W przykładzie pokazano, jak wykonać takie zadanie.

```

procedure TForm1.MojaProc(Sender: TObject);
begin
    Canvas.Rectangle(100,100,ClientWidth-100,ClientHeight-100);
end;

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

// odłączenie procedury MojaProc 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 formularz bitmapą, np. *kawa.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 `Form1.FormCreate` dopisz instrukcje:

```
Bitmap:=TBitmap.Create;  
Bitmap.LoadFromFile('C:\WINNT\kawa.bmp');
```

## 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 sposobu 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 dopiero po zakończeniu operacji w miejsce pierwszego.

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

### Przykład 13.13.

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

## Rozwiązanie

W procedurze obsługi przerwania 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;
```

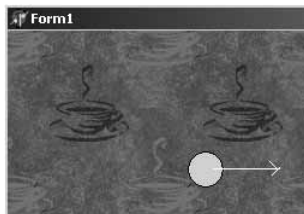
Dyrektywa {\$J+} przed treścią procedury włącza opcję kompilatora umożliwiającą zmianę wartości stałych typowanych (ang. *Assignable typed constants*). Opcja ta powinna być standardowo włączona, ale jeśli nie mamy pewności, lepiej dodać dyrektywę {\$J+}.

### Przykład 13.14.

Wykonaj animację tak jak w zadaniu poprzednim, gdy formularz jest wypełniony wzorem — rysunek 13.5.

#### Rysunek 13.5.

*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 przerwania od *Timera* przedstawiono poniżej.

Zadeklaruj zmienną globalną:

```
var Bitmap,Bitmap1: TBitmap;

// 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;

// 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;
    end;
end;

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

```

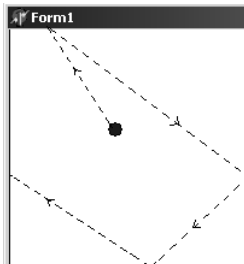


**Przykład 13.15.**

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 13.6.

**Rysunek 13.6.**

*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 przesuwać 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;
      kroky:integer=6;
begin
  with PaintBox1.Canvas do
  begin
    //czyszczenie prostokąta
    Brush.Color:=clWhite;
    Rectangle(0,0,200,200);
    //obliczenie współrzędnych
    x:=x+krokx;
    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>194) then
    begin
      krokx:=6+Random(5);
```

```

        krokx:=-krokx;
    end;
    if (y>194) then
    begin
        kroky:=6+Random(5);
        kroky:=-kroky;
    end;
    if (x<=6) then krokx:=-krokx;
    if (y<=6) then kroky:=-kroky;
end; //with
end;

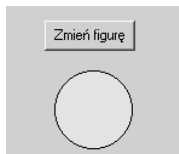
```

### Przykład 13.16.

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 13.7.

**Rysunek 13.7.**  
Przesuwanie koła  
za pomocą  
klawiszy strzałek



Aby klawisze strzałek nie były przechwytywane przez komponent *Button1*, należy ustawić dla każdego z nich właściwość *TabSet* na *false*.

### Rozwiązanie

Wstaw komponenty TButton i TShape. 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;
        vk_Left:  Shape1.Left:=Shape1.Left-10;
        vk_Up:    Shape1.Top:=Shape1.Top-10;
        vk_Down: Shape1.Top:=Shape1.Top+10;
    end;
end;

```