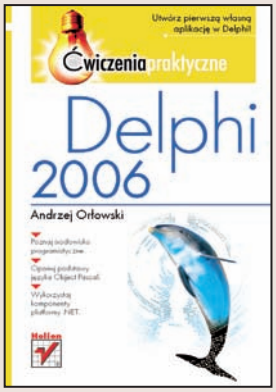
Delphi 2006. Ćwiczenia praktyczne



Autor: Andrzej Orłowski
ISBN: 83-246-0524-X
Format: A5, stron: 128

Delphi 2006 to najnowsza wersja niezwykle popularnego środowiska programistycznego, które jest połączeniem języka programowania Object Pascal z możliwością budowania aplikacji z „klocków" — gotowych komponentów. Taka filozofia pracy pozwala na tworzenie nawet najbardziej złożonych programów szybko i efektywnie. W najnowszej wersji Delphi możliwe jest także korzystanie z elementów platformy .NET opracowanej przez firmę Microsoft. Za pomocą tego środowiska można kreować zarówno proste aplikacje konsolowe, jak i rozbudowane systemy informatyczne wykorzystujące mechanizmy sieciowe, bazy danych i zaawansowane biblioteki graficzne.

Książka „Delphi 2006. Ćwiczenia praktyczne" to wprowadzenie do programowania z wykorzystaniem tego właśnie środowiska. Czytając ją i wykonując kolejne ćwiczenia, nauczysz się zasad korzystania z narzędzi oferowanych przez Delphi i przyswoisz sobie reguły pisania programów w języku Object Pascal, będącym fundamentem środowiska. Poznasz różnice między aplikacjami wykorzystującymi biblioteki Win32 i .NET, utworzysz własne programy oraz skompilujesz je i uruchomisz.

- Interfejs użytkownika Delphi 2006
- Podstawowe elementy języka Object Pascal
- Definiowanie parametrów projektu
- Aplikacje konsolowe
- Tworzenie aplikacji okienkowych w oparciu o komponenty VCL i .NET
- Korzystanie z bibliotek WinAPI i OpenGL

Poznaj profesjonalne narzędzie programistyczne
i korzystaj z niego podczas pracy

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

Spis treści

Przykładowe aplikacje

W tej części książki przedstawione zostaną przykłady tworzenia podstawowych aplikacji dla *Win32* i platformy *.NET*. Środowisko programistyczne *Delphi 2006* umożliwia tworzenie aplikacji dla *Win32* oraz dla platformy *.NET*. Ponieważ aplikacje tworzone dla platformy *.NET* nie mogą być uruchomione na komputerze, na którym nie zainstalowano odpowiednich składników, lub komputer nie pracuje pod kontrolą systemu *Windows 2003*, istnieje potrzeba oznaczenia, dla której platformy utworzona została dana aplikacja. Istnieją przynajmniej dwa (2) sposoby rozróżnienia, dla której platformy utworzona została dana aplikacja:

1. Nadanie programowi nazwy z sekwencją *_NET* w nazwie programu, np. *Klawisz_NET*.
2. Rozróżnienie, na podstawie ikony programu, tak jak pokazuje to rysunek 4.1.

Rysunek 4.1. *Ikony programów dla Win32 i platformy .NET*

Różnice są widoczne (szachownica dla *Win32* i skrzyżowane strzałki dla *.NET*). Aplikacje tworzone dla *Win32* i *.NET* różnią się nie tylko ikoną programu, ale głównie składnią poszczególnych poleceń, komponentami i ich obsługą, oraz innymi elementami, które zostaną pokazane w przykładach poszczególnych aplikacji tworzonych dla tych platform.

Aplikacje konsolowe

Aplikacje konsolowe mają w środowisku programistycznym *Delphi 2006* ograniczone zastosowanie, głównie ze względu na brak interfejsu graficznego tego typu aplikacji, do którego jesteśmy już przyzwyczajeni. Zaletą takich aplikacji jest natomiast, dla początkujących programistów, możliwość szybkiego sprawdzenia działania pisanych procedur lub funkcji, które muszą być pisane samodzielnie przez programistę.

Aplikacja konsolowa dla Win32

ĆWICZENIE

4.1 Nowy projekt aplikacji konsolowej dla Win32

Z utworzeniem projektu nowej aplikacji przy użyciu pola *File*/*New* Czytelnik został już zapoznany. Poniżej zilustrowany zostanie ten sam proces z wykorzystaniem strony powitalnej (*Welcome Page*), co jest moim zdaniem znacznie prostsze i wygodniejsze (droga „na skróty").

Aby utworzyć aplikację konsolową dla *Win32* należy kolejno:

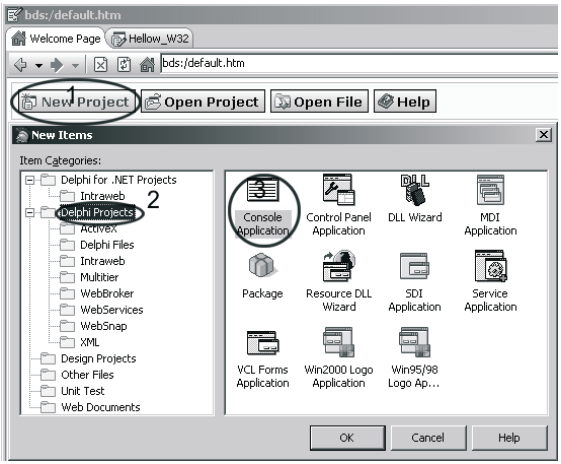
1. Wybrać zakładkę *New Project*.
2. Zaznaczyć platformę, dla której będzie tworzona aplikacja (*Delphi Projects*).
3. Zaznaczyć ikonę *Console Application* i zatwierdzić wybór podwójnym kliknięciem lewego klawisza myszki lub kliknąć na przycisku *OK*.

Kolejne zaznaczane opcje tworzenia nowego projektu aplikacji pokazuje rysunek 4.2.

Kod źródłowy nowego projektu aplikacji konsolowej dla *Win32* przedstawia listing 4.1.

Listing 4.1. *Kod źródłowy nowego projektu aplikacji konsolowej dla* Win32

```
program Project1;
{$APPTYPE CONSOLE} // Typ aplikacji
uses
  SysUtils;
begin
  { TODO -oUser -cConsole Main : Insert code here }
end.
```



Rysunek 4.2. *Wybór aplikacji konsolowej dla Win32*

Na przedstawionym listingu kod źródłowy nie zawiera żadnych instrukcji, nie będzie więc nic wykonywał, a na dodatek, po uruchomieniu, natychmiast zakończy swoje działanie.

1. Zapisać nowy projekt na dysku (w oddzielnym katalogu), wykorzystując polecenie *Save Project As…*, nadając mu nową unikatową nazwę, tak jak opisane to zostało już w punkcie „Własne aplikacje/Tworzenie aplikacji/Zachowanie projektu (plików) na dysku/Zachowanie na dysku plików z możliwością zmiany nazwy (Save Project As...)”, np. w katalogu *F:/D_2006/Konsola/ App_Con_W32/* pod nazwą *App_Con_W32*. Po wykonaniu powyższych czynności kod źródłowy naszego projektu będzie wyglądał tak jak na listingu 4.2.

Listing 4.2. *Kod źródłowy programu po zapisaniu na dysku, z nadaniem mu żądanej unikatowej nazwy*

```
program App_Con_W32;
{$APPTYPE CONSOLE}
uses
  SysUtils;
begin
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

Na tym etapie programowania mamy już utworzony nowy projekt, zapisany na dysku, co zabezpiecza nas przed utratą dotychczas wpro-

wadzonych zmian. Podstawowy sposób uruchomienia programu został już opisany wcześniej, w punkcie *Pole Run*. Na przykładzie aplikacji konsolowej dla *Win32* pokażę inny sposób uruchamiania programu, przydatny szczególnie wtedy, kiedy kompilator nie zgłasza żadnych błędów, a program nie działa tak jak oczekujemy, lub nie działa w ogóle (np. zawiesza się). Uzupełnijmy nasz program do postaci przedstawionej na listingu 4.3.

Listing 4.3. *Program aplikacji konsolowej po zmianie nazw domyślnych*

```
program App_Con_W32;
{$APPTYPE CONSOLE}
uses  SysUtils;
var
 I, X : Integer;
 procedure Suma (X, Razy : Integer);
 var
  Powtarzaj : Integer;
 begin
  Powtarzaj := 0;
  while Powtarzaj < Razy do
   begin
     I := I + X;
   end;
 end;

begin
  { TODO -oUser -cConsole Main : Insert code here }
 I := 0;
 Suma(5, 3);
 Writeln(I);
 Readln;
end.
```

W trakcie kompilacji nie zostanie zasygnalizowany żaden błąd. Program pod względem składniowym jest bowiem prawidłowy. Jego uruchomienie spowoduje jednak efekt przedstawiony na rysunku 4.3.

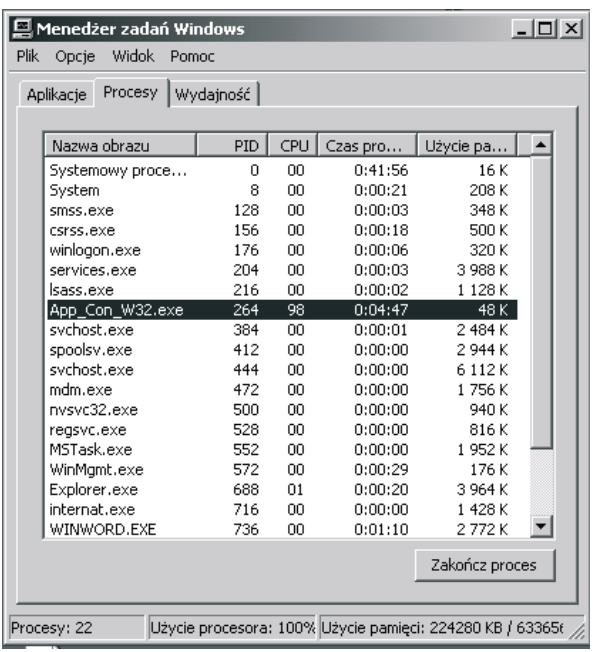
Rysunek 4.3. *Ekran aplikacji, która nie reaguje na komendy (zawieszonej)*

ĆWICZENIE

4.2 Zakończenie pracy zawieszonego programu

W sytuacji zawieszenia się programu należy wykonać następujące czynności:

1. Nacisnąć kombinację klawiszy *Ctrl+Alt+Del*. Po wyświetleniu okienka ***Zabezpieczenia systemu Windows*** kliknąć na przycisk *Menedżer Zadań*.
2. W oknie ***Menedżer zadań Windows*** zaznaczyć zawieszoną aplikację (rysunek 4.4).



Rysunek 4.4. *Wybór zawieszonej aplikacji w celu jej zakończenia*

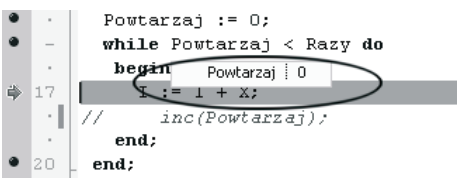
3. Kliknąć na przycisk *Zakończ proces*. Po tej operacji powrócimy do środowiska *Delphi 2006*.

ĆWICZENIE

4.3 Szukanie błędów w programie (praca krokowa)

Teraz najważniejszym zagadnieniem jest znalezienie przyczyny zawieszenia się programu. W tym celu należy:

1. Z menu *Run* wybrać polecenie *Trace Into F7* lub nacisnąć klawisz funkcyjny *F7* (polecam ten drugi sposób).
2. W kolejnych krokach (naciskając każdorazowo klawisz *F7*) kontrolować zachowanie się programu (wartości zmiennych).
3. Po dojściu (w tym konkretnym przypadku) do pętli *While* w kolejnych obiegach pętli kontrolować wartości wszystkich zmiennych.
4. Sprawdzać szczególnie wartość zmiennej wyliczeniowej *Powtarzaj*, tak jak pokazuje to rysunek 4.5.



Rysunek 4.5. *Wartość zmiennej wyliczeniowej Powtarzaj, nie zmienionej, po kilku przebiegach pętli While*

Sprawdzenie wartości najprościej można przeprowadzić, naprowadzając kursor myszki na daną zmienną (*Powtarzaj*).

Jeżeli po kilku przebiegach pętli jej wartość nie ulega zmianie, oznacza to, że pominięta została instrukcja inkrementacji (dekrementacji) tej zmiennej, co oczywiście skutkuje niemożliwością spełnienia warunku zakończenia jej działania (nieskończenie wielka liczba powtórzeń), a tym samym zawieszeniem się programu. Po dodaniu instrukcji inkrementacji (`inc()`) w przykładzie instrukcja ta zawarta jest w linii komentarza (`// inc(Powtarzaj)`). Po usunięciu symbolu komentarza (`//`) program zakończy pracę zgodnie z oczekiwaniem, a na ekranie zostanie wyświetlony wynik zgodny z parametrami wywołania procedury `Suma()`.

Zakończenie pracy aplikacji konsolowej możliwe jest poprzez:

- Naciśnięcie przycisku *Close (X)*.
- Naciśnięcie kombinacji klawiszy *Ctrl+C*.
- Naciśnięcie kombinacji klawiszy *Ctrl+Break*.
- Programowe zakończenie działania aplikacji poprzez naciśnięcie dowolnego klawisza, jeżeli ostatnią instrukcją w programie będzie funkcja `Readln`.

Aplikacja konsolowa dla platformy .NET

Utworzenie nowej aplikacji konsolowej dla platformy *.NET* przebiega identycznie jak dla platformy *Win32*, z tą jednakże różnicą, że należy wybrać inne opcje w okienku ***New Items*** po wyborze zakładki *New Project* na stronie powitalnej. Przykład nowego projektu aplikacji konsolowej dla platformy *.NET* po jego utworzeniu, nadaniu unikatowej nazwy i zapisaniu na dysku, przedstawia listing 4.4.

Listing 4.4. *Nowa aplikacja konsolowa dla platformy .Net*

```
program App_Con_NET;
{$APPTYPE CONSOLE}
uses
  SysUtils;
begin
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

W przypadku automatycznie wygenerowanego szkieletu projektu nie otrzymujemy niestety żadnej informacji, dla jakiej platformy jest utworzony dany projekt, co może stwarzać pewne problemy na dalszym etapie programowania. Dlatego przypominam o nadaniu projektowi nazwy sugerującej jej środowisko pracy lub zastosowaniu zapisu nazw modułów charakterystycznych dla platformy *.NET* w sekcji `uses`, czyli np. *Borland.Vcl.SysUtils*. Zmodyfikowany projekt przedstawia listing 4.5.

Listing 4.5. *Projekt nowej aplikacji konsolowej dla platformy .NET po modyfikacji*

```
program App_Con_NET;
{$APPTYPE CONSOLE}
uses
  Borland.Vcl.SysUtils, Borland.Vcl.Math;
```

```
var
 Kwadrat : Double;
 Kat     : Single;
begin
  { TODO -oUser -cConsole Main : Insert code here }
 Kat := DegToRad(45); // Funkcja z modułu Math
 Kwadrat := tan(Kat); // Funkcja z modułu Math
 Writeln(Kwadrat:6:2); // Tekst formatowany
 Console.ReadLine;
end.
```

W powyższym przykładzie zastosowałem typy `Single` i `Double` dla zmiennych rzeczywistych. Takie zadeklarowanie typów jest zalecane w miejscu typu `Real`. Zalecenie to wynika ze zgodności tych typów z formatem liczb akceptowanym przez koprocesor (procesor arytmetyczny — *FPU*, czyli *Floating-Point Unit*), bez potrzeby ich konwertowania, co znacznie poprawia wydajność obliczeń. W sekcji uses dodany został moduł *Math*, który udostępnia znacznie szerszy zakres funkcji matematycznych.

VCL Form Applications — Delphi for Win32

Aplikacje *VCL Form Application — Delphi for Win32* będą najprawdopodobniej najczęściej tworzonymi aplikacjami przez początkujących programistów, zaczynających swoją przygodę z *Delphi 2006*. Powodem tego będzie prawie identyczna konstrukcja tych programów, jak w poprzednich wersjach *Delphi*. Co to jest biblioteka *VCL* (*Visual Component Library*) nie będę teraz wyjaśniał, bo najlepiej jest zapoznać się z tym pojęciem na praktycznym przykładzie. Ponadto w punkcie tym zajmiemy się również importem gotowych już projektów napisanych dla *Delphi 7* do *Delphi 2006*. W pierwszej kolejności jednak zaczniemy od nowego projektu aplikacji *VCL Forms*.

Nowy projekt aplikacji typu VCL Forms dla Win32

Aby utworzyć nowy projekt aplikacji *VCL Form* dla *Win32*, należy postępować według wskazówek podanych w punkcie *Aplikacja konsolowa dla Win32*, wybierając oczywiście inny rodzaj aplikacji (*VCL Forms Application — Delphi for Win32*). W oknie edycyjnym wyświetlony zostanie kod źródłowy modułu nowego projektu, przedstawiony na listingu 4.6.

Listing 4.6. *Kod źródłowy modułu nowego projektu*

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```

Po nadaniu projektowi i modułowi nowej, unikatowej nazwy, i zapisaniu na dysku w oddzielnym katalogu (np. program — *KL_VCL_32* i moduł — *KlawiszVCL32*), kod źródłowy naszego nowego projektu powinien wyglądać tak, jak na listingu 4.7.

Listing 4.7. *Kod źródłowy programu i modułu po modyfikacji*

```
program KL_VCL_32;
uses
  Forms,
  KlawiszVCL32 in 'KlawiszVCL32.pas' {Okno_Klawisz},
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TOkno_Klawisz, Okno_Klawisz);
  Application.Run;
end.
```

```
unit KlawiszVCL32;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```

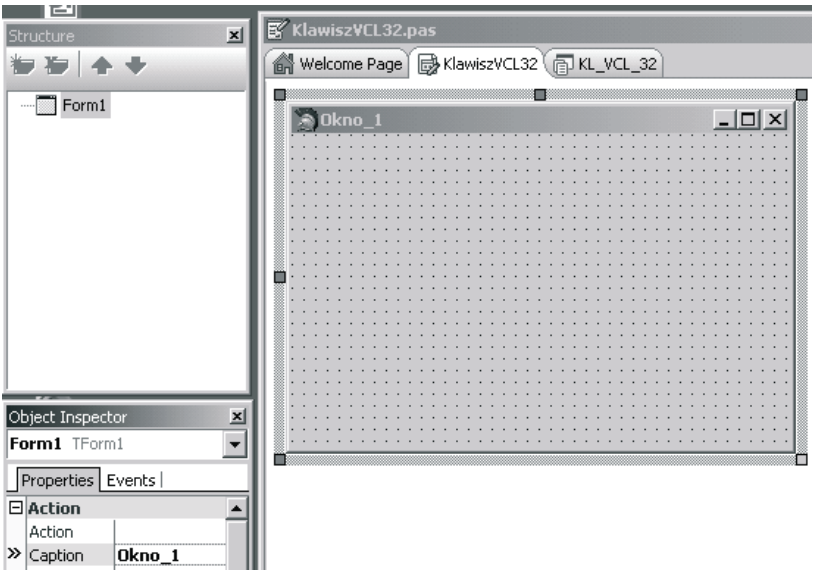
Po uruchomieniu programu na ekranie zobaczymy nasze okno, takie jak na rysunku 4.6.

Rysunek 4.6. *Okno programu KL_VCL_W32*

Pora, aby naszemu oknu nadać nowy tytuł (*Caption*) odpowiedni dla danej aplikacji, oraz nową unikatową nazwę (*Name*), tak by przy większej liczbie formularzy (okien) nie utracić kontroli nad projektem. Wymienione powyżej czynności przeprowadzamy w oknie ***Object Inspector*** środowiska *Delphi 2006*.

Zmiana tytułu (Caption) okna

Nadanie nowego tytułu okna przeprowadzamy tak, jak pokazuje rysunek 4.7.



Rysunek 4.7. *Nadanie nowego tytułu oknu aplikacji*

Wpisywanie nowego tytułu okna (*Caption*) jest automatycznie odzwierciedlane na pasku tytułowym okna.

Zmiana nazwy (Name) formularza

Zmianę nazwy formularza ilustruje rysunek 4.8.

Rysunek 4.8. *Zmiana nazwy formularza*

Wprowadzone zmiany nazwy (*Name*) formularza są automatycznie wprowadzane do kodu źródłowego, co ilustruje listing 4.8.

Listing 4.8. *Zmiana nazwy formularza*

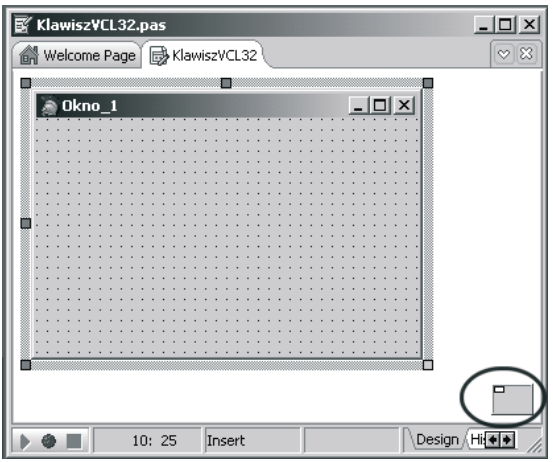
```
unit KlawiszVCL32;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs;
type
  TOkno_Klawisz = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Okno_Klawisz: TOkno_Klawisz;
implementation
{$R *.dfm}
end.
```

Na powyższym listingu widać, że nastąpiła (automatycznie) zmiana nazwy typu `class(TForm)` z `TForm1` na `TOkno_Klawisz` oraz zmiana nazwy zmiennej typu `TOkno_Klawisz` z `Form1` na `Okno_Klawisz`. Taki program nie robi na razie nic, poza wyświetleniem okna, i jest oczywiście do niczego nieprzydatny, ale stanowi szkielet, który teraz można obudować funkcjami, procedurami, komponentami itd., tworząc w pełni użyteczną aplikację. Wyświetlone na ekranie okno ma rozmiary domyślne i, co oczywiste, nie zawsze odpowiada potrzebom tworzonej aplikacji, dlatego najczęściej będziemy zmuszeni do zmiany jego rozmiarów oraz położenia na ekranie.

Zmiana położenia okna na ekranie

Położenie okna na ekranie kontrolujemy w oknie edycyjnym (zakładka *Design*), jak pokazuje to rysunek 4.9.

Aby zmienić pozycję okna na ekranie, wystarczy uchwycić kursorem myszki biały prostokąt (okno) widoczny na tle ekranu (szary prostokąt w zaznaczonym polu) i przesunąć go w dowolne miejsce szarego prostokąta, tak jak na rysunku 4.10.



Rysunek 4.9. *Pozycja okna aplikacji na ekranie*

Rysunek 4.10. *Zmiana pozycji okna za pomocą okna edycji kodu*

Metoda ta, chociaż wygodna, jest jednak mało precyzyjna i niekiedy niewystarczająca, dlatego lepiej skorzystać z zakładki *Properties* (właściwości) okna ***Object Inspector***. Odpowiednie pola tej zakładki pokazuje rysunek 4.11.

Rysunek 4.11. *Pola do zmiany położenia okna na ekranie*

Wartości w polach *Left* i *Top* odnoszą się do lewego górnego narożnika okna.

W oknie ***Object Inspector*** wyświetlona jest nazwa obiektu poddawanego modyfikacjom (tutaj *Okno_Klawisz*), co jest szczególnie ważne, gdy mamy do czynienia z wieloma obiektami.

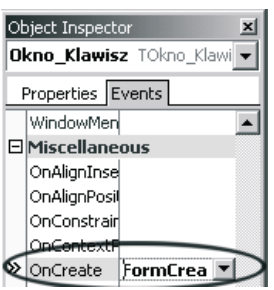
Przedstawione sposoby nie są jedynym możliwymi, istnieje oczywiście również możliwość usytuowania okna na ekranie na drodze programowej, która jest nadrzędna w stosunku do ustawień ***Object Inspector***, co pokaże kolejne ćwiczenie.

ĆWICZENIE

4.4 Zmiana usytuowania okna na pulpicie

Aby programowo ustawić pozycję okna na pulpicie, należy:

1. W oknie ***Object Inspector*** wybrać zakładkę *Events*.
2. Wybrać pole *OnCreate* i dwukrotnie kliknąć w jego polu edycyjnym (rysunek 4.12).



Rysunek 4.12. *Utworzenie procedury obsługi zdarzenia OnCreate*

3. W utworzonej automatycznie procedurze obsługi tego zdarzenia wpisać dwie linie kodu:

```
procedure TOkno_Klawisz.FormCreate(Sender: TObject);
begin
  Okno_Klawisz.Left := 500; // Object Inspektor : 37
  Okno_Klawisz.Top := 300;  // Object Inspektor : 269
end;
```

Po uruchomieniu programu okno zostanie wyświetlone w miejscu określonym w procedurze obsługi zdarzenia *OnCreate*.

Dodajemy komponenty

Projektowanie graficznego interfejsu użytkownika (*GUI*) jest prawdopodobnie jedną z ważniejszych czynności, jakie ma do wykonania programista. Pomijając względy estetyczne, czytelność i intuicyjność

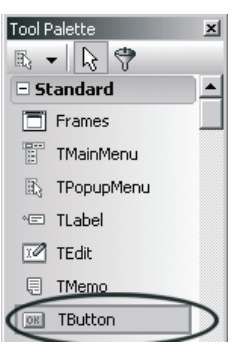
interfejsu, w znacznej mierze przyczynia się do atrakcyjności całej aplikacji. W zakresie czytelności całego interfejsu decydującą rolę odgrywają umieszczone na nim elementy sterujące pracą aplikacji (komponenty). Ponadto komponenty realizują cały szereg zadań, odciążając programistę od pisania wielu linii kodu, co znacznie przyspiesza pracę nad całą aplikacją. Biblioteka komponentów (*VCL*) jest tak obszerna w *Delphi 2006*, że omówienie ich wszystkich jest w tej książce całkowicie niemożliwe. Do tego dochodzą komponenty tworzone przez niezależnych programistów, których jest prawdopodobnie jeszcze więcej. W tej części książki zajmiemy się więc tylko najważniejszymi komponentami, umożliwiającymi sprawne sterowania naszą aplikacją. W kolejnych ćwiczeniach będziemy dodawać do naszej aplikacji kolejne komponenty, tak aby przybliżyć Czytelnikowi ich możliwości i przydatność w tworzonej aplikacji. W celu dodania jakiegokolwiek komponentu należy oczywiście skorzystać z palety komponentów zawartej w oknie ***Tool Palette***. W zależności od typu tworzonej aplikacji musimy wybrać odpowiednie pole tego okna, które zwykle rozwija się automatycznie dla danego typu aplikacji. Dodawanie różnych (najczęściej używanych) komponentów pokażę w kolejnych ćwiczeniach.

ĆWICZENIE

4.5 Dodanie do formularza komponentów Button (klawisz) i Edit (pisz)

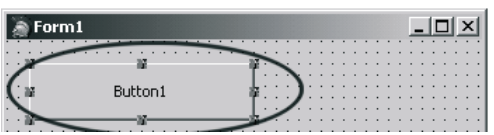
Aby dodać do formularza komponent (np. *Button*), należy:

- wybrać dany komponent z listy komponentów,
- kliknąć na wybranym komponencie (rysunek 4.13),



Rysunek 4.13. *Wybór komponentu do umieszczenia na formularzu*

- przenieść kursor w żądane miejsce posadowienia komponentu na formularzu,
- oznaczając obszar umieszczenia komponentu, kliknąć lewym przyciskiem myszki,
- wybrany komponent zostanie umieszczony na formularzu w wyznaczonym miejscu i otoczony ramką do zmiany wymiarów (rysunek 4.14), oraz opatrzony domyślnym tytułem (*Button1*).



Rysunek 4.14. *Umieszczenie komponentu (Button1) na formularzu*

Tak zainstalowany komponent nie będzie nic wykonywał do czasu jego oprogramowania.

Dodanie dalszych komponentów (*Edit1* i *Edit2*) odbywa się identycznie. Po dodaniu nowych komponentów kod źródłowy aplikacji będzie wyglądał jak na listingu 4.9.

Listing 4.9. *Kod źródłowy po dołączeniu komponentów Button1, Edit1 i Edit2*

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```

Po uruchomieniu wyświetlone okno będzie wyglądać tak, jak na rysunku 4.15.

Rysunek 4.15. *Okno po wstawieniu nowych komponentów (nazwy i tytuły komponentów domyślne)*

Wcześniej wspomniane było, że nazwy (tytuły) domyślne są praktyczne, ale mało przydatne w rzeczywistości. Szczególnie trudno bowiem powiązać nazwę komponentu z jego rzeczywistym przeznaczeniem.

ĆWICZENIE

4.6 Nadanie tytułów i nazw komponentom

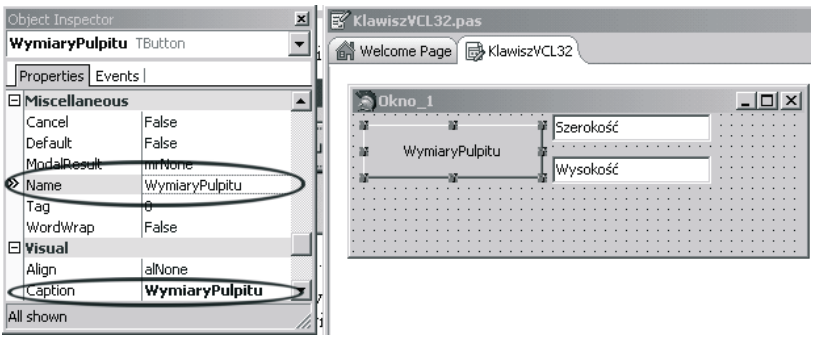
Nadawanie nazwy formularzowi było już opisane, natomiast nadawanie tytułu i nazwy komponentom przebiega w identyczny sposób dla wszystkich komponentów, i dlatego przedstawiony zostanie sposób przeprowadzenia tej czynności tylko na przykładzie przycisku *Button1*.

Aby nadać nazwę (*Name*) i tytuł (*Caption*) komponentowi, najprościej jest:

1. Zaznaczyć wybrany komponent (zostanie otoczony ramką).
2. W oknie ***Object Inspector*** wybrać pole *Miscellaneous*/*Name* i wpisać nazwę komponentu (rysunek 4.16).

Po nadaniu nazwy (*Name*) komponentowi następuje automatyczne przypisanie tego tekstu właściwości *Caption*, ale nigdy odwrotnie. Właściwość *Caption* można dowolnie zmieniać.

Treść tekstu we właściwości *Name* nie może zawierać polskich znaków **diakrytycznych**.



Rysunek 4.16. *Nadanie nazwy (Name) i tytułu (Caption) komponentowi Button1*

Zmiana rozmiarów okna

Rozmiary okna naszej aplikacji (szerokość i wysokość), jeżeli nie są to wartości wymagane z innych względów, najlepiej pozostawić do ustalenia *Delphi 2006*.

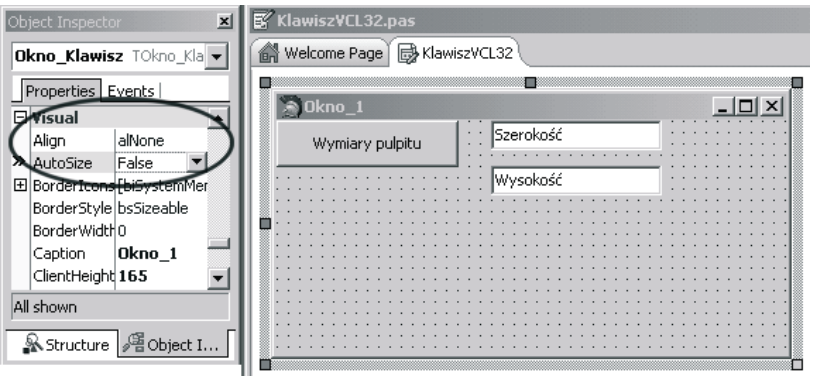
ĆWICZENIE

4.7 Dopasowanie automatyczne (AutoSize)

Aby okno automatycznie dostosowywało się od niezbędnych rozmiarów, należy:

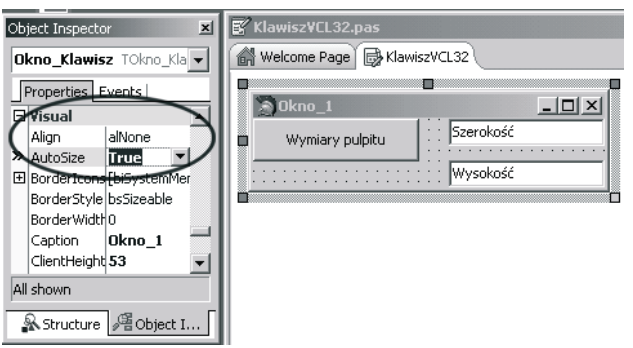
1. Wyróżnić formularz (klikając na obszarze formularza). Wokół formularza pojawi się ramka do zmiany rozmiarów formularza.
2. W oknie ***Object Inspektor*** wybrać zakładkę *Properties* i odszukać pole *Visual*/*AutoSize*. W oknie edycyjnym tego pola widnieje napis *False*. Oznacza to, że rozmiary okna ustalane są przez programistę. Widok formularza przy takim ustawieniu ilustruje rysunek 4.17.
3. Zmienić wartość pola *Visual*/*AutoSize* na *True*. Obraz na ekranie ulegnie zmianie, co widać na rysunku 4.18.

Jeżeli na formularzu nie będzie żadnego komponentu, to okno zostanie wyświetlone w postaci belki zadań.



Rysunek 4.17. *Ustawienia pola Visual/AutoSize, przy ustalaniu rozmiarów okna przez programistę*

Rysunek 4.18. *Dopasowanie rozmiarów okna do zainstalowanych komponentów*



ĆWICZENIE

4.8 Zmiana rozmiarów okna przy zmianie rozdzielczości ekranu

Projektując jakąś aplikację, nie mamy pewności, że zawsze będzie pracowała przy jednakowej, ustalonej przez nas rozdzielczości ekranu. W przypadku pracy przy innych rozdzielczościach może dojść do sytuacji, że nasze okno nie będzie mieściło się na ekranie lub będzie nie tam, gdzie być powinno. Aby móc ustawić właściwe rozmiary (i jego położenie) w zależności od rozdzielczości, należy posiadać informację o aktualnych wymiarach pulpitu (*Desktop*). W tym celu należy:

1. Dodać w sekcji var dwie zmienne: L (szerokość) i H (wysokość).

```
var
  Okno_Klawisz: TOkno_Klawisz;
  L, H : Integer; // Muszą być zmiennymi typu Integer
```

2. Zmodyfikować kod źródłowy procedury obsługi zdarzenia *OnCreate* do postaci:

```
procedure TOkno_Klawisz.FormCreate(Sender: TObject);
begin
 L := Screen.DesktopWidth;
 H := Screen.DesktopHeight;
  Okno_Klawisz.Left := L div 2;
  Okno_Klawisz.Top := H div 3;
  Okno_Klawisz.Width := L div 3;
  Okno_Klawisz.Height := H div 4;
end;
```

ĆWICZENIE

4.9 Zmiana rozmiarów okna ramką wymiarową

Zmiana rozmiarów okna przy pomocy ramki otaczającej formularz:

Uchwycić kursorem za krawędź ramki lub wydzielone kwadraciki i przeciągając nim, nadać formularzowi wymagane rozmiary.

Dodajemy pasek menu

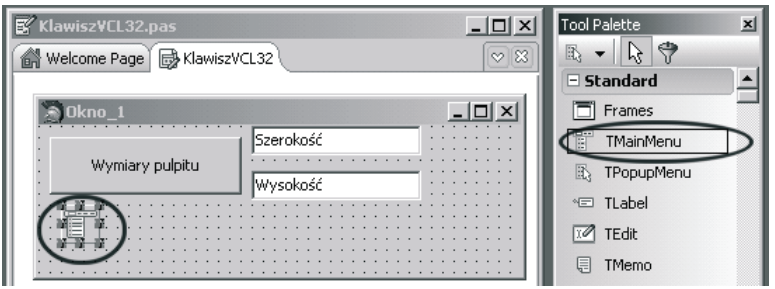
Pasek *menu* to obok belki tytułowej jeden z najczęściej występujących elementów sterujących okna aplikacji. Pasek *menu* powinien zawierać praktycznie wszystkie podstawowe polecenia (pola), umożliwiające sterowanie pracą całej aplikacji.

ĆWICZENIE

4.10 Pasek Menu

Aby dodać pasek *Menu* do naszego formularza, należy:

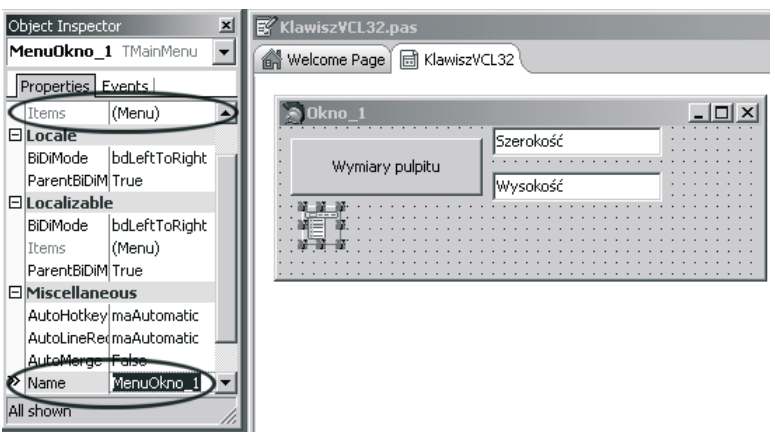
1. W oknie ***Tool Palette*** wybrać zakładkę *Standard* i zaznaczyć komponent *TMainMenu*, a następnie naprowadzić kursor w dowolne miejsce na formularzu i kliknąć lewym przyciskiem myszki. Efekt działania przedstawia rysunek 4.19.



Rysunek 4.19. *Umieszczanie komponentu TMainMenu na formularzu*

Komponent *TMainMenu* jest komponentem niewidocznym.

2. Przy zaznaczonym komponencie *TMainMenu* (otoczony ramką), w oknie ***Object Inspektor***, w zakładce *Properties*, w polu *Name* nadajemy własną nazwę nowemu menu np. *MenuOkno_1*. Przechodzimy do pola *Items*, aby uruchomić okno kreatora menu poprzez dwukrotne kliknięcie na tym polu. Wszystkie w/w czynności ilustruje rysunek 4.20.



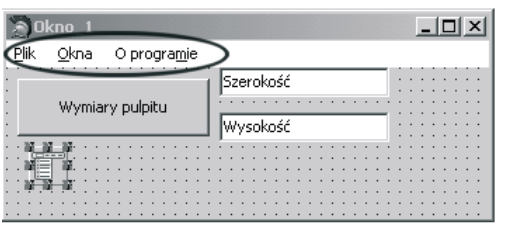
Rysunek 4.20. *Zmiana nazwy komponentu TMainMenu i uruchomienie kreatora menu*

3. Po otworzeniu okna kreatora menu wpisujemy żądany napis, który po zatwierdzeniu pojawi się jako opis danego pola menu. Jednocześnie za polem pojawi się nowy prostokąt (zakreślony linią przerywaną) kolejnego pola menu, co pokazuje rysunek 4.21.

Rysunek 4.21. *Tworzenie menu*

4. Dodajemy podmenu każdego pola poprzez kliknięcie na wybranym polu. Po zakończeniu dodawania pól menu i podmenu zamykamy okno kreatora, a na formularzu pojawi się pasek menu, co ilustruje rysunek 4.22.

Rysunek 4.22. *Formularz z dodanym paskiem Menu*



Dodanie paska *Menu* niczego jeszcze w działaniu programu nie zmienia. Teraz należy wypełnić programem określone zdarzenia (*Events*) tych pól. Zaczniemy od najprostszego pola, jakim jest pole *Plik*/*Zakończ* {*Zakocz1*}. Nazwa w nawiasie klamrowym to nazwa nadana przez *Delphi 2006*.

ĆWICZENIE

4.11 Oprogramowanie pola Plik/Zakończ{Zakocz1}

Zadaniem tego pola jest zakończenie działania aplikacji.

Aby oprogramować dowolne polecenie podmenu (tutaj pola *Plik*), należy:

1. Dwukrotnie kliknąć na komponencie *TMainMenu*.
2. Wybrać żądane pole i jego polecenie (*Plik*/*Zakończ*).
3. W oknie ***Object Inspector***, na zakładce *Events* wybrać zdarzenie, np. *OnClick*.
4. Dwukrotnie kliknąć na tym zdarzeniu. W oknie edycyjnym wyświetlony zostanie kod źródłowy modułu, w którym dodana została procedura obsługi tego zdarzenia.

5. Wpisujemy odpowiednie instrukcje, tak aby procedura wykonała założone zadanie. W tym wypadku treścią procedury będzie jedna linia kodu. Całą procedurę ilustruje poniższy fragment kodu:

```
procedure TOkno_Klawisz.Zakoczl1Click(Sender: TObject);
begin
 Application.Terminate;
end;
```

Po uruchomieniu programu będzie można zakończyć pracę aplikacji klikając na polecenie *Plik/Zakończ*.

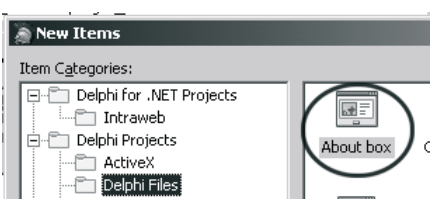
Pozostałe polecenia poszczególnych pól uzupełnia się właściwymi instrukcjami kodu w procedurach obsługujących żądane zdarzenia. Pewnym wyjątkiem jest pole *O programie*. Jego obsługa różni się nieco od opisanej poprzednio, a to ze względu na wywołanie specjalnego okna ***About***.

ĆWICZENIE

4.12 Dodajemy okno About

Aby dodać okno ***About*** do pola *O programie*, należy:

1. W oknie ***New Items*** (*File/New/Other…*) wybrać ikonę modułu *About Box*, tak jak na rysunku 4.23.



Rysunek 4.23. *Wybór modułu About Box*

2. W dolnej części okna oznaczyć opcję *Use*, która automatycznie dołączy moduł do naszego projektu (rysunek 4.24).

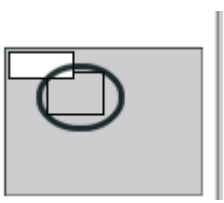
Rysunek 4.24. *Zaznaczenie opcji automatycznego dołączenia modułu do projektu*

3. Zatwierdzić wybory (*OK*). Nowy moduł zostanie włączony do pliku programu, co pokazuje listing 4.10.

Listing 4.10. *Uzupełniony automatycznie kod źródłowy programu po dodaniu modułu*

```
program KL_VCL_32;
uses
  Forms,
  KlawiszVCL32 in 'KlawiszVCL32.pas' {Okno_Klawisz},
  About in 'f:\borland\bds\4.0\ObjRepos\DelphiWin32\About.pas'
{AboutBox};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TOkno_Klawisz, Okno_Klawisz);
  Application.CreateForm(TAboutBox, AboutBox);
  Application.Run;
end.
```

Tak dodany moduł jest również automatycznie uruchamiany razem z całą aplikacją. Widać to na kontrolce formularzy w oknie edycji, co pokazuje rysunek 4.25.



Rysunek 4.25. *Formularze uruchamiane automatycznie wraz ze startem aplikacji podstawowej*

4. Otworzyć do edycji plik *About.pas* (*View*/*Units*…) i zapisać go na dysku w naszym katalogu (*File*/*Save As*…), nadając mu inną, unikatową nazwę, np. *OProgramie*.

Po takiej operacji nasz plik programu przyjmie postać:

```
program KL_VCL_32;
uses
  Forms,
  KlawiszVCL32 in 'KlawiszVCL32.pas' {Okno_Klawisz},
  OProgramie in 'OProgramie.pas' {AboutBox}; // AboutBox to nie
  // zmieniona nazwa formularza
{$R *.res}
begin
  Application.Initialize;
```

```
  Application.CreateForm(TOkno_Klawisz, Okno_Klawisz);
  Application.CreateForm(TAboutBox, AboutBox);
  Application.Run;
end.
```

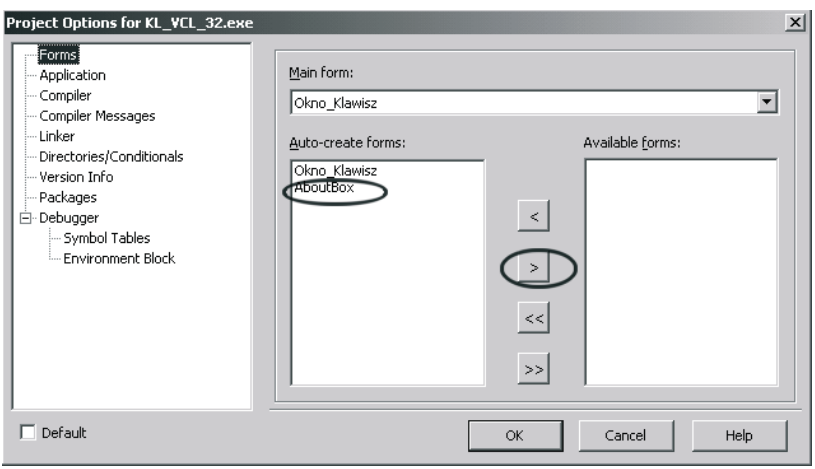
Rozwiązanie takie jest co prawda wygodne, ale zajmuje bardzo dużo pamięci, co nie zawsze jest pożądane. Sytuacji takiej można zapobiec, tworząc daną aplikację (*Application.CreateForm(TAboutBox, AboutBox)*) tylko wtedy, kiedy jest potrzebna.

ĆWICZENIE

4.13 Tworzenie aplikacji (okna) na życzenie użytkownika

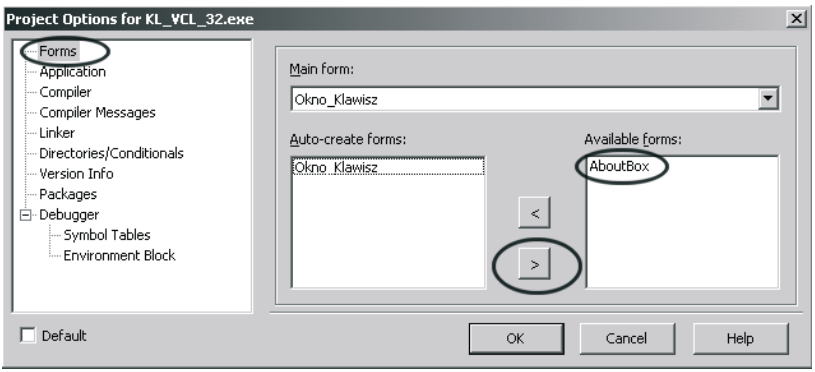
Aby zapobiec tworzeniu aplikacji pomocniczej jednocześnie z tworzeniem aplikacji głównej, należy:

1. Otworzyć okno ***Project Options*** (*Project*/*Options*…), widoczne na rysunku 4.26.



Rysunek 4.26. *Wybór formularza do ręcznego uruchomienia*

2. Po wybraniu formularza, który nie ma być tworzony automatycznie w czasie startu programu, naciskamy klawisz (>), co powoduje przeniesienie zaznaczonego formularza do okienka *Available forms*, jak jest to pokazane na rysunku 4.27.



Rysunek 4.27. *Przeniesienie wybranego formularza (AboutBox) z okna Auto-create forms: do okna Available forms: (nie uruchamiany przy starcie programu)*

Działanie takie skutkuje usunięciem określonego modułu z pliku kodu źródłowego programu, co ilustruje listing 4.11.

Listing 4.11. *Anulowanie automatycznego tworzenia nowego okna AboutBox*

```
program KL_VCL_32;
uses
  Forms,
  KlawiszVCL32 in 'KlawiszVCL32.pas' {Okno_Klawisz},
  OProgramie in 'OProgramie.pas' {AboutBox};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TOkno_Klawisz, Okno_Klawisz);
  Application.Run;
end.
```

Na powyższym listingu widać, że w sekcji uses deklaracja modułu pozostała, natomiast usunięta została linia `Application.CreateForm(TAboutBox, AboutBox);`, co oznacza, że określony formularz (okno), nie będzie tworzony automatycznie przy starcie programu. Ponieważ dodatkowe okno nie zostało utworzone automatycznie przy starcie programu, zachodzi potrzeba utworzenia go w trakcie działania programu, co pokaże kolejne ćwiczenie.

ĆWICZENIE

4.14 Utworzenie okna w trakcie działania programu

Aby utworzyć nowe (dodatkowe) okno w trakcie działania programu głównego, należy:

1. Dodać nową sekcję uses w module głównym (*KlawiszVCL32*), wpisując tam nazwę dołączanego modułu (*OProgramie*).
2. Dwukrotnie kliknąć na komponencie *TMainMenu*, co spowoduje otworzenie okna kreatora menu.
3. Wybrać właściwe pole (*O programie*), i w oknie ***Object Inspector*** w zakładce *Events* wybrać polecenie obsługi zdarzenia *OnClick*, po czym dwukrotnie kliknąć na tym zdarzeniu.
4. W utworzonej procedurze obsługi tego zdarzenia wpisać następujące linie kodu:

```
procedure TOkno_Klawisz.Oprogramie1Click(Sender: TObject);
begin
  AboutBox := TAboutBox.Create(Application);
  AboutBox.Show;
end;
```

ĆWICZENIE

4.15 Zamknięcie okna dodatkowego (O programie)

Aby zamknąć okno ***O programie***, utworzone w trakcie działania programu, należy:

1. Otworzyć do edycji plik modułu dodatkowego (*About.pas*).
2. Wybrać klawisz *OK* i w zakładce *Events* okna ***Object Inspector*** dwukrotnie kliknąć na zdarzeniu *OnClick*.
3. W utworzonej procedurze obsługi zdarzenia wpisać:

```
procedure TAboutBox.OKButtonClick(Sender: TObject);
begin
 AboutBox.Free;
end;
```

Dodajemy formularze

W tym punkcie omówione zostaną różne metody dodawania dodatkowych formularzy do tworzonego programu. Dodatkowym celem jest pokazanie różnych formularzy (okien), jakie mogą być tworzone przez *Delphi 2006*. W celu zademonstrowania powyższych zagadnień wykorzystamy utworzony już wcześniej projekt *Kl_VCL_32* i przypiszemy do pola *Okna* jego menu odpowiednie rodzaje formularzy (okien). Możliwe rodzaje formularzy dostępne są w oknie ***Object Inspector*** w zakładce *Properties*, co pokazuje rysunek 4.28.

Rysunek 4.28. *Dostępne style formularza (okna)*

ĆWICZENIE

4.16 Dodajemy formularze (okna) różnych stylów

Często zachodzi potrzeba użycia okna w innym niż domyślny (*bsSizeable*) stylu. W tym ćwiczeniu pokazane zostaną różne, oferowane przez *Delphi 2006* style okien (formularzy). Przedstawiony zostanie jeden ze sposobów dołączania nowego formularza do projektu, wspólny dla formularzy wszystkich stylów.

Aby dołączyć nowy formularz do projektu, należy:

1. Otworzyć nowy moduł (*File*/*New*/*Form — Delphi for Win32*). W oknie edycyjnym pojawi się nowa zakładka: *Unit1*, co pokazuje rysunek 4.29.

Rysunek 4.29. *Nowy moduł w oknie edytora kodu*

2. Zmienić domyślną nazwę (*Form1*) na odpowiednią dla danego okna, np. *Okno_bsDialog*. Nastąpi teraz jednocześnie zmiana właściwości *Caption*, która, jeżeli nam odpowiada, może pozostać. Wprowadzone zmiany ilustruje rysunek 4.30.

Rysunek 4.30. *Nadanie nowej nazwy i tytułu formularzowi (oknu)*

3. Zapisać na dysku nowy plik w katalogu naszego projektu (*File/Save As…)*, nadając mu unikatową nazwę np. *OknobsDialog*. Z chwilą wykonania tej czynności zmianie uległy (zostały automatycznie zaktualizowane) zapisy w pliku źródłowym programu (*KL_VCL_32*) oraz nazwa nowego modułu (*OknobsDialog*).
4. W „naszej" sekcji uses dopisujemy nazwę nowego modułu:

```
uses
 OProgramie, OknobsDialog;
```

Na tym etapie nasz nowy moduł jest już dołączony do projektu.

ĆWICZENIE

4.17 Wyświetlenie nowego okna (Okno_bsDialog)

Ponieważ nasze nowe okno zostało utworzone przy starcie programu, aby je wyświetlić, wystarczy w kodzie źródłowym modułu głównego (*KlawiszVCL32*) umieścić procedurę obsługi zdarzenia *OnClick* pola menu (*Okna/bsDialog*).

Aby umieścić żądaną procedurę i wyświetlić okno ***Okno_bsDialog***, należy:

1. Na głównym formularzu projektu dwukrotnie kliknąć na komponencie *TMainMenu*.

2. Na wyświetlonym oknie kreatora menu kliknąć na polu *Okna*, a następnie na polu *bsDialog*, tak jak na rysunku 4.31.

Rysunek 4.31. *Wybór pola menu do wyświetlenia okna Okno_bsDialog*

3. Wybrać w oknie ***Object Inspektor*** w zakładce *Properties* zdarzenie *OnClick* i dwukrotnie kliknąć lewym przyciskiem myszki.
4. W edytowanej procedurze umieścić następujący kod:

```
procedure TOkno_Klawisz.BsDialog1Click(Sender: TObject);
begin
  Okno_bsDialog.Left := 0; // Lewa krawędź pulpitu
  Okno_bsDialog.Top := 0; // Górna krawędź pulpitu
  Okno_bsDialog.Show; // Pokaż okno
end;
```

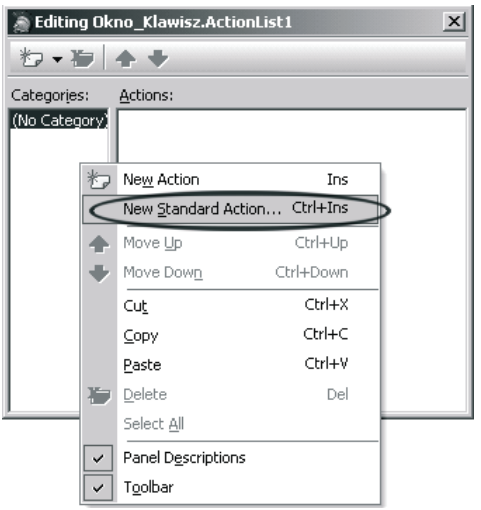
Zamknięcie dodatkowego okna może odbyć się na kilka sposobów. Najprostszym jest oczywiście skorzystanie z przycisku **X** (*Zamknij*).

Action — tajemnicza właściwość

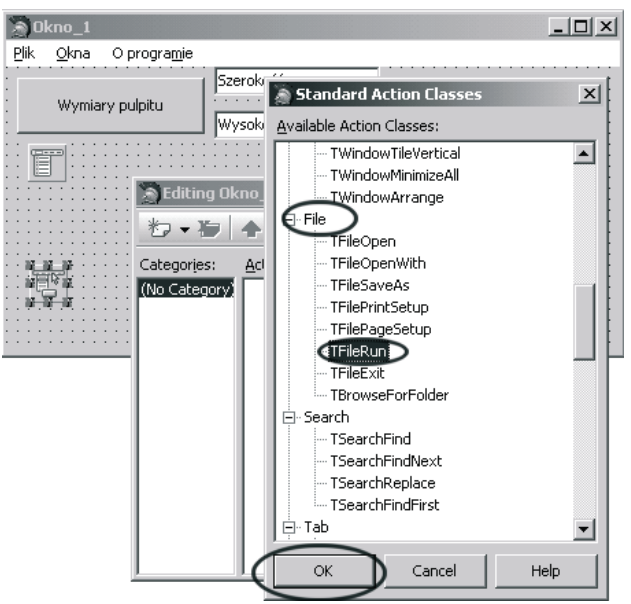
Niekiedy zdarza się, że nasza aplikacja musi umożliwić uruchomienie innych aplikacji. W prezentowanym od początku tego rozdziału projekcie, w polu menu *Plik* znajduje się polecenie *Uruchom*. Najprostszym sposobem wykonania tego zadania jest skorzystanie z właściwości *Action* tego polecenia.

Aby wykorzystać właściwość *Action*, należy:

1. Umieścić na formularzu komponent *TActionList*. Komponent ten znajduje się na karcie *Standard* okna ***Tool Palette***.
2. Kliknąć dwukrotnie na tym komponencie, a następnie kliknąć prawym klawiszem w okienku *Categories:*. Zobaczymy okno, takie jak na rysunku 4.32.
3. Kliknąć dwukrotnie na tym komponencie. Zobaczymy okno, takie jak na rysunku 4.33.



Rysunek 4.32. *Okno edycji ActionList*



Rysunek 4.33. *Wybranie kategorii File i akcji TFileRun*

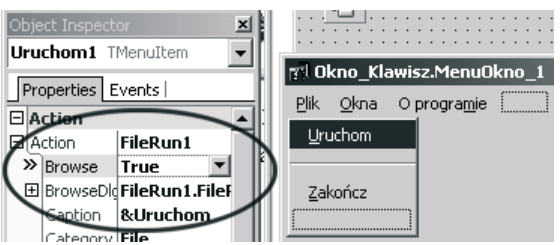
4. Zatwierdzić wybór (*OK*), a okno ***Editing*** zmieni swój wygląd (w polu *Categories:* pojawi się napis *File*). Po kliknięciu na tym napisie w polu *Actions:* pojawi się napis *FileRun1*, co pokazuje rysunek 4.34.

Rysunek 4.34. *Uzupełniona lista ActionList1*

Tak uzupełnioną listę można już wykorzystać do wypełnienia właściwości *Action* pola *Plik*/*Uruchom*.

5. Kliknąć dwukrotnie na komponencie *MenuOkno_1* (*TMainMenu*), a następnie wybrać polecenie *Plik*/*Uruchom*.
6. Na zakładce *Properties* kliknąć na polu *Action* i z rozwiniętej listy dostępnych akcji (teraz tylko jedna) wybrać tę właściwą (*FileRun1*).
7. Zmienić we właściwości *Action*/*Browse* zapis *False* na *True*.
8. Zmienić we właściwości *Action*/*Caption* zapis domyślny *&Run…* na *&Uruchom*.

Czynności 6,7 i 8 ilustruje rysunek 4.35.



Rysunek 4.35. *Ustawianie wartości właściwości Action polecenia Plik/Uruchom*

Podsumowanie

W tym miejscu przedstawione zostaną kompletne listingi kodów źródłowych i modułów tworzonej przykładowej aplikacji KL_VCL_32. Kod źródłowy programu przedstawia listing 4.12.

Listing 4.12. *Kod źródłowy programu KL_VCL_32*

```
program KL_VCL_32;
uses
  Forms,
```

```
  KlawiszVCL32 in 'KlawiszVCL32.pas' {Okno_Klawisz},
  OProgramie in 'OProgramie.pas' {AboutBox},
  OknobsDialog in 'OknobsDialog.pas' {Okno_bsDialog};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TOkno_Klawisz, Okno_Klawisz);
  Application.CreateForm(TOkno_bsDialog, Okno_bsDialog);
  Application.Run;
end.
```

Ten kod tworzony jest automatycznie przez Delphi 2006, i na razie nie ma potrzeby ingerencji w jego zawartość.

Na listingu 4.13 przedstawiony jest kod źródłowy głównego modułu projektu — *KlawiszVCL32*.

Listing 4.13. *Kod źródłowy modułu KlawiszVCL32*

```
unit KlawiszVCL32;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls, Menus, StdActns, ExtActns, ListActns, ActnList;
type
  TOkno_Klawisz = class(TForm)
    Szerokosc: TEdit;
    Wysokosc: TEdit;
    Wymiary: TButton;
    MenuOkno_1: TMainMenu;
    Plik1: TMenuItem;
    Zakocz2: TMenuItem;
    Okna1: TMenuItem;
    BsDialog1: TMenuItem;
    BsSingle1: TMenuItem;
    BsSizeToolWin1: TMenuItem;
    BsNone1: TMenuItem;
    BsSizeAble1: TMenuItem;
    BsToolWin1: TMenuItem;
    Oprogramie1: TMenuItem;
    N2: TMenuItem;
    Uruchom1: TMenuItem;
    ActionList1: TActionList;
    FileRun1: TFileRun;
    procedure BsDialog1Click(Sender: TObject);
    procedure Oprogramie1Click(Sender: TObject);
    procedure Zakocz1Click(Sender: TObject);
```

```
    procedure WymiaryClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Okno_Klawisz: TOkno_Klawisz;
  L, H        : Integer;
implementation
{$R *.dfm}
  uses // Nasze moduły
   OProgramie, OknobsDialog;
procedure TOkno_Klawisz.BsDialog1Click(Sender: TObject);
begin
  Okno_bsDialog.Left := 0;
  Okno_bsDialog.Top := 0;
  Okno_bsDialog.Show;
end;

procedure TOkno_Klawisz.FormCreate(Sender: TObject);
begin
 L := Screen.DesktopWidth;
 H := Screen.DesktopHeight;
   Okno_Klawisz.Width := L div 3;
   Okno_Klawisz.Height := H div 4;
 // Okno na środku pulpitu
  Okno_Klawisz.Left := (L div 2) - Okno_Klawisz.Width div 2;
  Okno_Klawisz.Top := (H div 2) - Okno_Klawisz.Height div 2;
end;

procedure TOkno_Klawisz.Oprogramie1Click(Sender: TObject);
begin
  AboutBox := TAboutBox.Create(Application);
  AboutBox.Show;
end;

procedure TOkno_Klawisz.WymiaryClick(Sender: TObject);
begin
 Szerokosc.Text := IntToStr(L);
 Wysokosc.Text := IntToStr(H);
end;

procedure TOkno_Klawisz.Zakocz1Click(Sender: TObject);
begin
 Application.Terminate;
end;
end.
```

Windows Forms Application — Delphi for .NET (WinForm)

Aplikacje typu *WinForm* znacząco różnią się od aplikacji *VCL*. Podstawowymi różnicami pomiędzy projektami *VCL* a *WinForm* są miedzy innymi różne:

1. Nazwy komponentów.
2. Właściwości komponentów.
3. Funkcje i procedury.
4. Nazwy modułów.
5. Nazwy zdarzeń.

W tej części spróbujemy zaprojektować podobną do poprzedniej (`program KL_VCL_32;`), chociaż trochę uproszczoną pod względem funkcjonalności aplikację, jaka była utworzona w części *VCL Form Appilcation — Delphi for Win32*.

Nowy projekt aplikacji typu Windows Forms Application — Delphi for .NET

ĆWICZENIE

4.18 Tworzymy nowy projekt WinForm

Aby utworzyć nowy projekt typu *WinForm*, należy:

1. Wybrać z menu polecenie *File/New/Windows Forms Application — Delphi for .NET*. Utworzone zostaną podstawowe pliki projektu (nazwy domyślne: *Project1 i Unit1*).
2. Po jego utworzeniu zapisać nowy projekt w oddzielnym katalogu, z jednoczesnym nadaniem unikatowych nazw projektowi (np. *KLWinForms*) i modułowi głównemu (np. *KL_WinForm*). Kody źródłowe tak utworzonego projektu przedstawiają listingi 4.14 i 4.15.

Listing 4.14. *Kod źródłowy programu KLWinForms*

```
program KLWinForms;
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.dll'}
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.Data.dll'}
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.Drawing.dll'}
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.Windows.Forms.dl
l'}
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.XML.dll'}
{$R 'KL_WinForm.TWinForm.resources' 'KL_WinForm.resx'}
uses
  System.Reflection,
  System.Runtime.CompilerServices,
  System.Runtime.InteropServices,
  System.Windows.Forms,
  KL_WinForm in 'KL_WinForm.pas' {KL_WinForm.TWinForm:
System.Windows.Forms.Form};
{$R *.res}
[STAThread]
begin
  Application.Run(TWinForm.Create);
end.
```

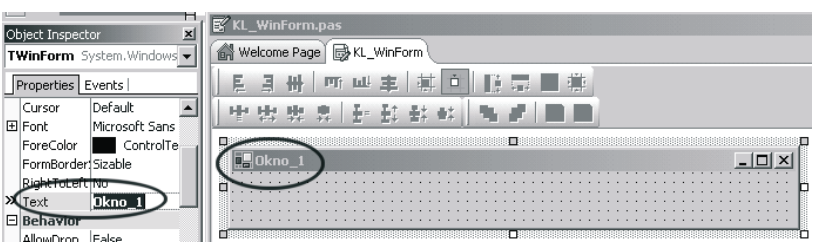
Listing 4.15. *Kod źródłowy modułu KL_WinForm*

```
unit KL_WinForm;
interface
uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data;
type
  TWinForm = class(System.Windows.Forms.Form)
  strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
  private
    { Private Declarations }
  public
    constructor Create;
  end;

  [assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]
```

```
implementation

{$AUTOBOX ON}

procedure TWinForm.Dispose(Disposing: Boolean);
begin
  if Disposing then
  begin
    if Components <> nil then
      Components.Dispose();
  end;
  inherited Dispose(Disposing);
end;

constructor TWinForm.Create;
begin
  inherited Create;
  // Required for Windows Form Designer support
  InitializeComponent;
  // TODO: Add any constructor code after InitializeComponent call
end;
end.
```

Zmiana tytułu (Text) okna

Zmiany tytułu okna dokonuje się w oknie ***Object Inspector***, w zakładce *Properties*, we właściwościach *Appearance*/*Text*, co pokazuje rysunek 4.36.



Rysunek 4.36. *Zmiana tytułu formularza (okna)*

Dodajemy komponenty

Proces dodawania komponentów na formularzu jest identyczny jak w przypadku innych rodzajów projektów. Różnica polega jedynie na innych nazwach komponentów i nazwach ich właściwości. Przykładowo:

- zamiast komponentu *TButton* jest *Button*,
- zamiast komponentu *TEdit* jest *TextBox*,
- właściwość *Caption* to *Text*,
- zdarzenie *OnClick* zastąpiło samo *Click*.

ĆWICZENIE

4.19 Dodajemy komponenty

Aby dodać do naszego formularza nowe komponenty, należy:

1. W oknie ***Tool Palette*** wybrać żądany komponent (z odpowiedniej palety), np. *Button*.
2. Kliknąć na wybranym komponencie, następnie przenieść kursor w wybrane dla danego komponentu miejsce i zaznaczyć obszar, w którym zostanie on umieszczony.

   Postępując w ten sam sposób, umieśćmy na formularzu jeszcze:

   - dwa komponenty *TextBox* (*Edit*) z palety *Windows Forms*,
   - dwa komponenty *Label* (*Label*) z palety *Windows Forms*,
   - komponent *MainMenu* (*MainMenu*) z palety *Components*,
   - komponent *OpenFileDialog* (*OpenDialog*) z palety *Dialogs*.

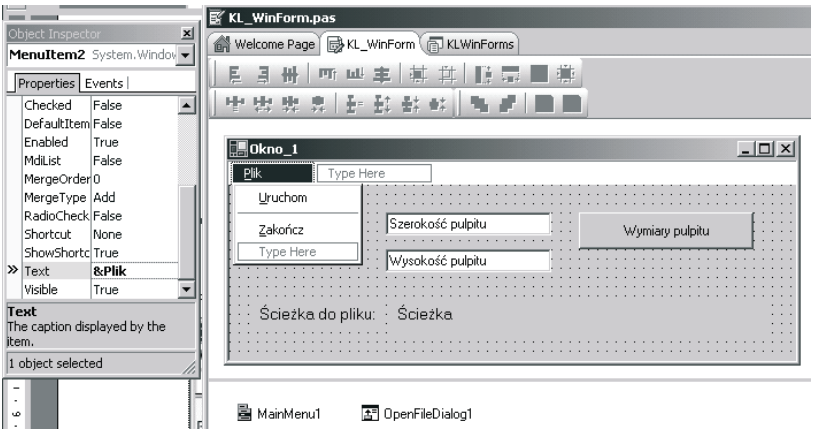
W nawiasach podano nazwy komponentów formularzy *VCL* dla *Delphi*.

3. Nadać komponentom *Label* i *TextBox* nowe nazwy (*Name*) i tytuły (*Text*).

Formularz z dodanymi komponentami przedstawia rysunek 4.37.

Pasek menu — pola i polecenia

Pasek *menu* to obok belki tytułowej jeden z najczęściej występujących elementów sterujących okna aplikacji. Pasek *menu* powinien zawierać praktycznie wszystkie podstawowe polecenia (pola), umożliwiające sterowanie pracą całej aplikacji.



Rysunek 4.37. *Formularz aplikacji typu WinForm z dodanymi komponentami*

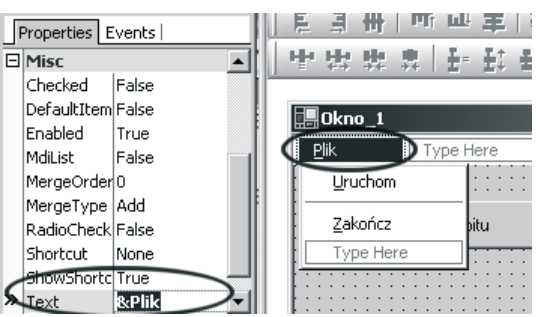
ĆWICZENIE

4.20 Opis pola i poleceń menu

Aby opisać pola i polecenia menu, należy:

1. Kliknąć na pierwszym polu menu (zaznaczony prostokąt z napisem *Type Here*).
2. W oknie ***Object Inspector***, zakładce *Properties* i polu *Misc*/*Text* wpisać nazwę pola np. *&Plik*.
3. Wpisać nazwy kolejnych poleceń pola *Plik*.

Opisane pole i polecenia menu przedstawia rysunek 4.38.



Rysunek 4.38. *Nadawanie nazw poszczególnym polom i poleceniom menu*

Na tym etapie tworzenia nowej aplikacji uruchomienie jej spowoduje jedynie wyświetlenie formularza z rozmieszczonymi komponentami, które niestety nie będą nic robić. Pora więc na oprogramowanie ich tak, aby spełniały swoją rolę.

Oprogramowanie komponentów

Większość komponentów, aby mogła poprawnie działać, musi obsługiwać określone zdarzenia, zgłaszane przez użytkownika aplikacji. Najczęściej będzie to kliknięcie na danym komponencie, co wywoła odpowiednią reakcję na to zdarzenie. Oprogramowanie zainstalowanych komponentów rozpoczniemy do polecenia *Plik*/*Zakończ*, które kończy pracę naszej aplikacji.

ĆWICZENIE

4.21 Polecenie Plik/Zakończ

Aby kliknięcie na poleceniu *Plik*/*Zakończ* kończyło pracę aplikacji, należy:

1. Rozwinąć pole *Plik* (kliknąć na tym polu).
2. Kliknąć na poleceniu *Zakończ*.
3. W oknie ***Object Inspector*** przejść do zakładki *Events*.
4. Kliknąć dwukrotnie na zdarzeniu *Misc*/*Click*.
5. W utworzonej automatycznie procedurze obsługi zdarzenia wpisać poniższy kod:

```
procedure TWinForm.MenuItem5_Click(sender: System.Object; e:
System.EventArgs);
begin
  Application.Exit; // Odpowiada instrukcji Application.Terminate
  dla VCL
end;
```

Mamy teraz możliwość wybrania stylu okna (właściwości *FormBorderStyle*) jako *None*, co oznacza brak belki tytułowej, a co za tym idzie brak przycisków sterujących.

ĆWICZENIE

4.22 Polecenie Uruchom

Polecenie *Uruchom* pozwoli na uruchomienie programu typu *.*exe*).

Aby polecenie *Uruchom* działało, należy:

1. Kliknąć na komponencie *OpenFileDialog*.
2. Na zakładce *Properties* w polu *Misc*/*Filter* wpisać tekst: `Pliki typu (*.exe)|*.exe`.
3. Rozwinąć pole *Plik* i kliknąć na poleceniu *Uruchom*.
4. Na zakładce *Events* kliknąć dwukrotnie na zdarzeniu *Click*.
5. Uzupełnić procedurę poniższym kodem:

```
procedure TWinForm.MenuItem3_Click(sender: System.Object; e:
System.EventArgs);
var
 Plik : String;
begin
 OpenFileDialog1.ShowDialog; // Otwarcie okna dialogowego
 Plik := OpenFileDialog1.FileName; // Przypisanie do zmiennej Plik
 // pełnej nazwy wybranego pliku (łącznie ze ścieżką dostępu)
  Label2_Sciezka.Text := Plik; // Wyświetlenie pełnej nazwy wybranego
  // pliku
  WinExec(Plik, SW_SHOW); // Uruchomienie wybranego pliku
end;
```

Funkcja *WinExec()* nie działa w 32-bitowych programach VCL tworzonych w Delphi 2006.

ĆWICZENIE

4.23 Przycisk Wymiary pulpitu

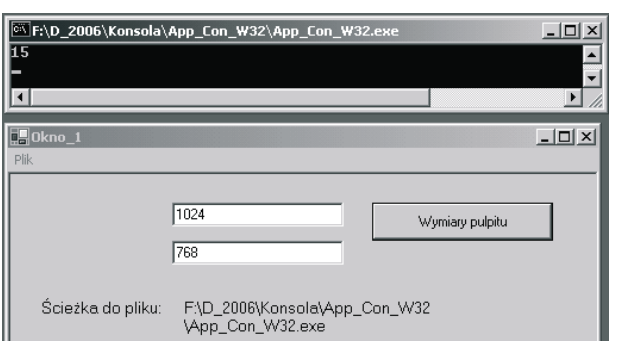
Zadaniem tego klawisza jest dostarczenie informacji o aktualnych wymiarach (w pikselach) pulpitu. Informacja taka może być przydatna np. do proporcjonalnej zmiany wymiarów okna w zależności od aktualnej rozdzielczości ekranu (pulpitu). W oprogramowaniu tego przycisku wykorzystane zostały pewne funkcje, na których szczegółowe omówienie ze względu na ograniczoną objętość książki nie ma miejsca, a Czytelnik musi je przyjąć „na wiarę".

Aby przycisk *Wymiary pulpitu* dostarczył żądanej informacji, należy:

1. W sekcji uses dodać dwa moduły: *SysUtils* i *Windows*.
2. Kliknąć na przycisku *Wymiary pulpitu*, a następnie dwukrotnie na zdarzeniu *Click*.
3. Procedurę obsługi zdarzenia uzupełnić kodem:

```
procedure TWinForm.WymiaryPulpitu_Click(sender: System.Object; e:
System.EventArgs);
var
 L, H       : Integer;
 WymiarOkna : TRect; // Struktura zawierająca pola Left, Top, Right
 // i Bottom
 h_Wnd      : HWND; // Uchwyt okna
begin
  h_Wnd := GetDesktopWindow; // Funkcja GetDsktopWindow zwraca uchwyt
  // głównego okna systemu (Desktop), czyli ekranu
  GetWindowRect(h_Wnd, WymiarOkna); // Pobranie danych ekranu
  // i przekazanie ich zmiennej, WymiarOkna
    L := WymiarOkna.Right - WymiarOkna.Left; // obliczenie szerokości
    // okna
    H := WymiarOkna.Bottom - WymiarOkna.Top; // obliczenie wysokości
    // okna
   WysokoscPulpitu.Text := IntToStr(H); // Konwersja danych
   // i wyświetlenie
   SzerokoscPulpitu.Text := IntToStr(L); // Konwersja danych
   // i wyświetlenie
end;
```

Po dodaniu wszystkich procedur obsługi zdarzeń zainstalowanych komponentów gotowy program można skompilować i uruchomić. Efekt końcowy przedstawia rysunek 4.39.



Rysunek 4.39. *Końcowy efekt działania programu KLWinForms*